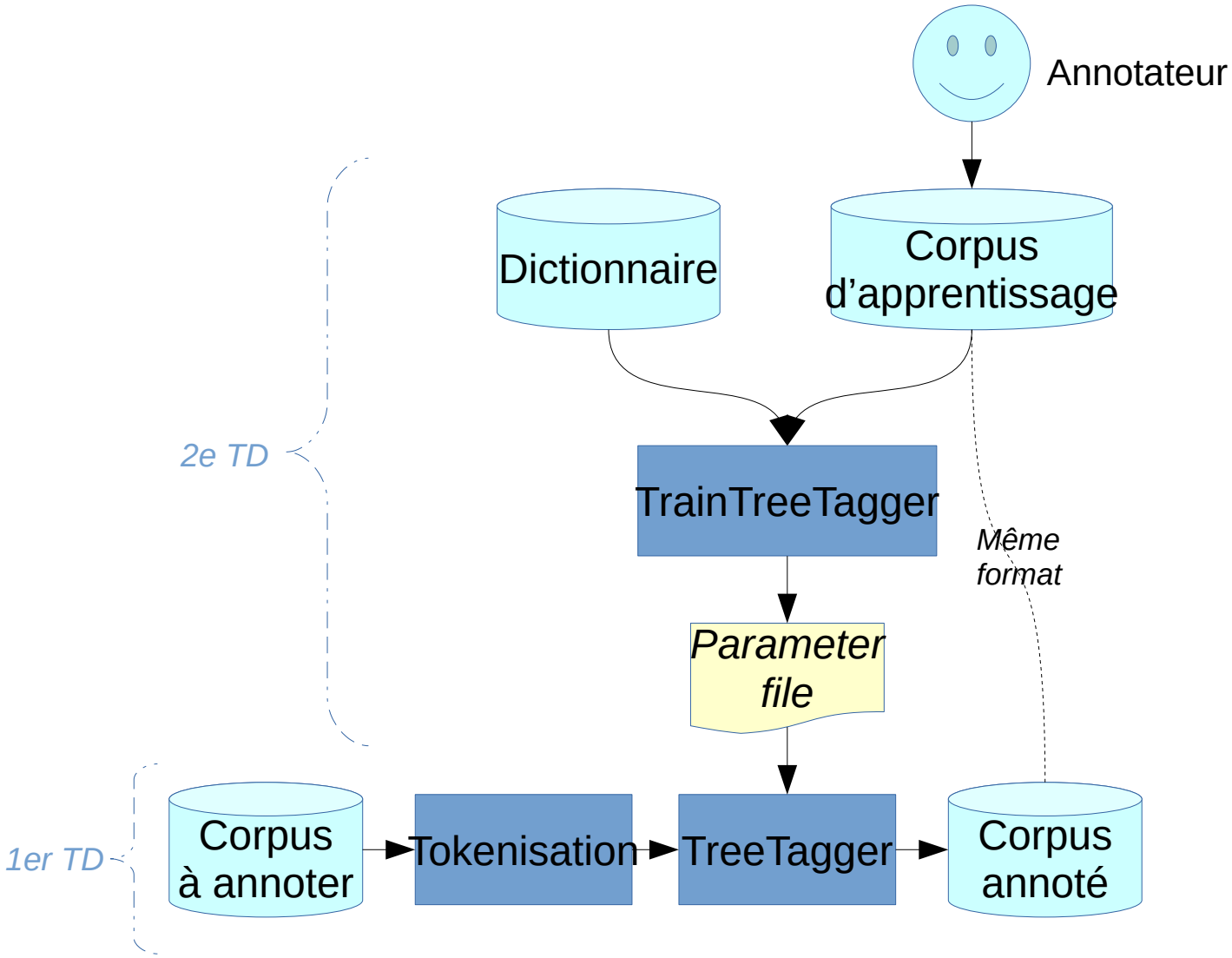
Introduction

Ce TD fait suite au TD précédent, et suppose que vous avez toujours votre dossier TDCORPUS (sinon, reportez-vous au TD précédent), et le logiciel LibreOffice Calc (Microsoft Office ne fait pas l'affaire).

Nous allons introduire la partie « haute » du schéma ci-dessous, tout en continuant à en utiliser la partie « basse » (1er TD).



Ressources pour la création d'un modèle de langage

Nous allons commencer par re-télécharger PERCEO :
https://www.ortolang.fr/market/corpora/perceo

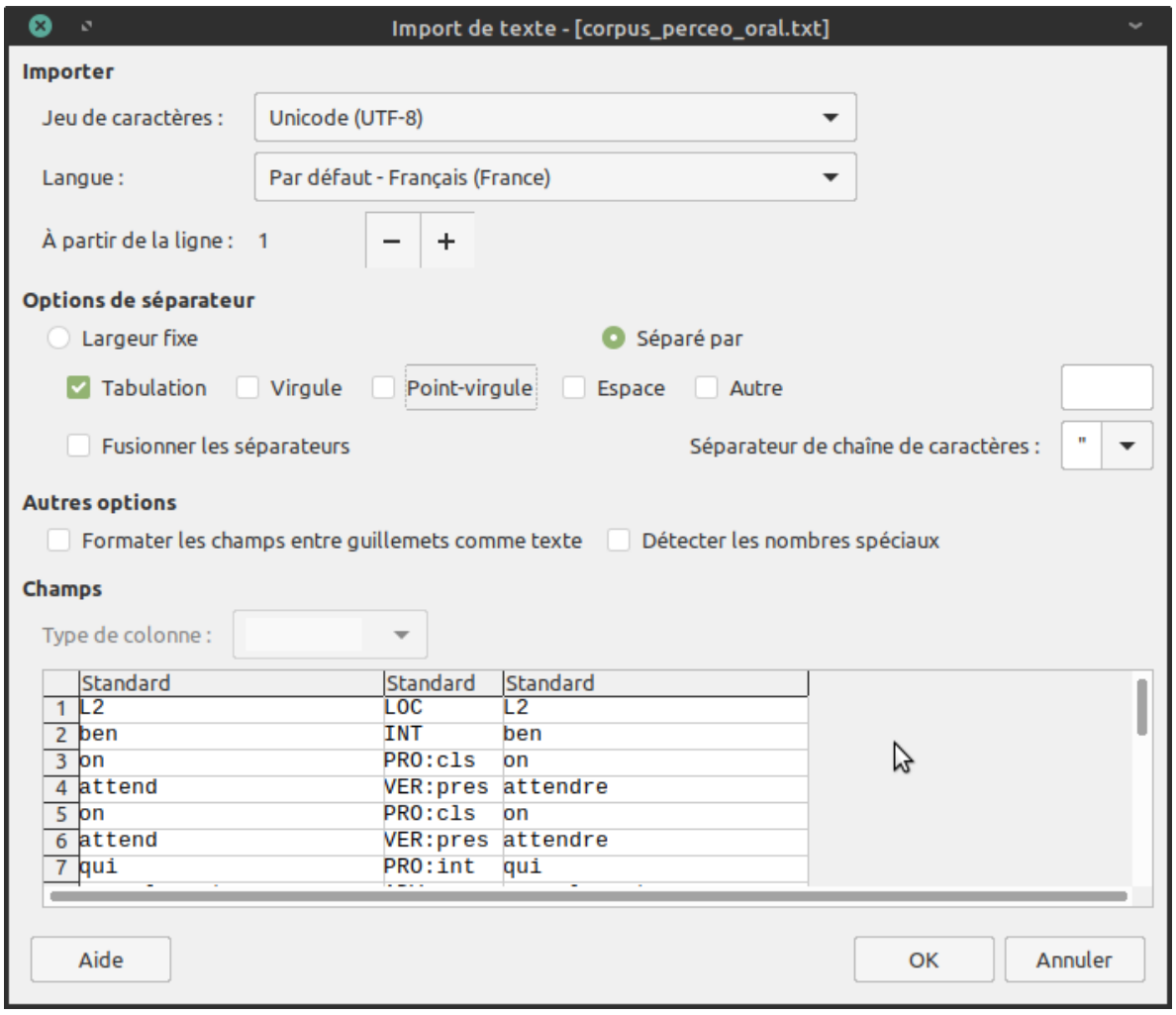
La dernière fois, nous n'avions récupéré que le modèle de langage (fichier *.par*) et sa documentation. Un truc appréciable avec PERCEO, c'est que les données ayant servi à le fabriquer (corpus d'apprentissage et dictionnaire) sont aussi fournis. Nous allons donc le re-fabriquer à partir de ces données, dans un premier temps pour voir comment on s'y prend, et dans un second temps pour « l'améliorer ».

Dans l'archive téléchargée, nous avons besoin de :

- perceo_oral/corpus_perceo_oral.txt
- perceo_oral/lexique_morphalou_perceo_oral.txt

… que nous allons déposer dans notre dossier TDCORPUS.

Ouvrons ces fichiers avec LibreOffice Calc. Pour rappel, il faut préciser qu'il est codé en Unicode (UTF-8), avec la Tabulation comme séparateur :



Corpus d’apprentissage

On voit que le corpus d’apprentissage ressemble à une sortie d’annotation de TreeTggaer :

	A	B	C
1	L2	LOC	L2
2	ben	INT	ben
3	on	PRO:cls	on
4	attend	VER:pres	attendre
5	on	PRO:cls	on
6	attend	VER:pres	attendre
7	qui	PRO:int	qui
8	normalement	ADV	normalement
9	L3	LOC	L3
10	ben	INT	ben
11	¤P1¤	NAM	¤P1¤
12	L4	LOC	L4
13	non	FNO	non
14	L1	LOC	L1
15	/tu les as eu(e)s,***/	MLT	/tu les as eu(e)s,***/
16	L3	LOC	L3
17	ouais	FNO	ouais
18	ben	INT	ben
19	je	PRO:cls	je
20	l'	PRO:clo	le
21	ai	AUX:pres	avoir
22	eu	VER:pper	avoir
23	au	PRP:det	au
24	tel	NOM:trc	téléphone
25	tout à l'heure	ADV	tout à l'heure
26	au	PRP:det	au
27	téléphone	NOM	téléphone

Dans ce corpus, les tokens L2, L3, etc. indiquent un changement de locuteur (c’est l’identifiant du locuteur qui parle). On notera que ce corpus comporte beaucoup de formes propres à une certaine manière de transcrire l’oral (voir la doc du corpus), et on peut se douter qu’un modèle de langage fabriqué à partir de ce corpus, ne sera vraiment adapté qu’à cette manière de transcrire la parole.

Dictionnaire

Ouvrons maintenant le dictionnaire (capture d’écran ci-après) : le format peut paraître un peu plus déroutant. Il s’agit d’un dictionnaire de formes fléchies.

La colonne de gauche contient l’ensemble des formes de la « langue » sur laquelle on travaille, donc idéalement beaucoup plus de formes que celles qui apparaissent dans le corpus. Ici, les formes incluent des « bizarreries » qui tiennent aux conventions d’annotation de l’oral. Sur un dictionnaire de la langue écrite, on n’aurait ainsi probablement pas de formes contenant des parenthèses ou des « / », mais ces formes bizarres existent dans la « langue » que constitue cette transcription de l’oral, donc il faut les indiquer dans le dictionnaire : pour TreeTagger, ce sont des caractères comme les autres, qui peuvent faire partie d’un token.

La colonne de droite est constituée de la POS correspondant à la forme de la première colonne, suivie (après un espace) du lemme.

	A	B
1	(de)	MLT (de)
2	(ir)régulier	MLT (ir)régulier
3	(le)	MLT (le)
4	(n')	MLT (n')
5	(ne)	MLT (ne)
6	(on)	MLT (on)
7	(plus)	MLT (plus)
8	(qu')	MLT (qu')
9	(re)cherche	MLT (re)cherche
10	(vous)	MLT (vous)
11	(y)	MLT (y)
12	-ce	PRO:dem ce
13	-ci	ADV -ci
14	-il	PRO:clsi il
15	-je	PRO:cls je
16	-le	PRO:clo le
17	-les	PRO:clo le
18	-lui	PRO:clo lui
19	-là	ADV là
20	-moi	PRO:clo moi
21	-nous	PRO:cls nous
22	-on	PRO:cls on
23	-toi	PRO:clo toi
24	-tu	PRO:cls tu
25	-vous	PRO:clo vous
26	-y	PRO:clo y
27	/***,notamment/	MLT /***,notamment/
28	/,ce qui veut dire/	MLT /,ce qui veut dire/
29	/,il faut le/	MLT /,il faut le/
30	/,ordinateur/	MLT /,ordinateur/
31	/,ouais/	MLT /,ouais/
32	/,rater/	MLT /,rater/
33	/a trouvé,trouvera/	MLT /a trouvé,trouvera/

Si on scrolle vers le bas, on arrive à la forme *Antre*. Celle-ci peut correspondre à deux POS : NAM et NOM. Dans le dictionnaire, quand une forme peut correspondre à plusieurs POS, on ajoute des colonnes, comme ici pour *Antre*.

168	Alicia	NAM Alicia	
169	Allemagne	NAM Allemagne	
170	Alpes	NAM Alpes	
171	Alsacien	NOM alsacien	
172	Américains	NOM américain	
173	Angel	NAM Angel	
174	Angleterre	NAM Angleterre	
175	Annecy	NAM Annecy	
176	Antre	NAM antre	NOM antre

Avec TreeTagger, l'exhaustivité du dictionnaire est importante. Quand une forme est inconnue, TreeTagger va essayer de deviner sa POS (et mettra « <unknown> » comme lemme). Sur un mot inconnu entouré de mots connus, TreeTagger se débrouille assez bien ; mais évidemment, plus il y a de mots inconnus à la suite, plus les résultats vont devenir bizarres. Si on a une phrase de 5 mots qui sont tous inconnus, TreeTagger va juste « deviner » qu'il s'agit de la séquence de 5 tokens la plus fréquente du corpus (et donc a de fortes chances de se tromper), en s'aidant juste un peu avec les préfixes/suffixes. Mais si une forme est connue, TreeTagger s'interdit de deviner, et ne choisira que des POS qui sont données dans le dictionnaire pour cette forme, même si c'est franchement improbable étant donné le contexte (TreeTagger choisira la POS la moins improbable) ; si il manque

une POS possible pour cette forme dans le dictionnaire, TreeTagger ne l'envisagera jamais et ne regardera que les POS données par le dictionnaire pour cette forme. Bref, il est recommandé d'avoir la plupart des formes de la langue sur laquelle on travaille, mais il est beaucoup plus important d'avoir toutes les POS des formes qu'on y met. Pour dire ça autrement, il est plus important d'être exhaustif sur les colonnes (avoir toutes les POS de chaque forme) que sur les lignes (avoir toutes les formes), car quand il manque des formes (lignes) TreeTagger va essayer de deviner, mais pas quand il manque des POS (colonnes).

Liste des POS

TreeTagger a besoin d'une troisième ressource dont on parle peu, parce qu'elle est facile à construire : la liste des POS. Il ne sait pas la récupérer tout seul dans le dictionnaire ou le corpus d'apprentissage.

Pour information, on peut l'obtenir avec la commande suivante (que vous n'avez pas besoin de comprendre dans le cadre du TD) :

- `cat corpus_perceo_oral.txt | cut -f 2 | sort -u`

Et on obtient la liste suivante, que je vous suggère de copier/coller dans un fichier qu'on appellera **tags.txt** .

ADJ
ADJ:trc
ADV
AUX:cond
AUX:futu
AUX:impf
AUX:infi
AUX:pper
AUX:ppre
AUX:pres
AUX:subp
DET:def
DET:dem
DET:ind
DET:int
DET:par
DET:pos
DET:pre
EPE
ETR
FNO
INT
KON
LOC
MLT
NAM
NAM:sig
NAM:trc

NOM
NOM:sig
NOM:trc
NUM
PRO:clo
PRO:cls
PRO:clsi
PRO:dem
PRO:ind
PRO:int
PRO:pos
PRO:rel
PRO:ton
PRP
PRP:det
PRT:int
TRC
VER
VER:cond
VER:futu
VER:impe
VER:impf
VER:infi
VER:pper
VER:ppre
VER:pres
VER:simp
VER:subp
VER:trc

Génération du modèle de langage

Une dernière info à trouver, c'est la POS qui sert de délimiteur de phrase dans le corpus d'apprentissage. TreeTagger travaille dans le cadre de phrases, et a besoin de cette info. Dans notre corpus de transcription de parole, il n'y a pas à proprement parler de phrases, mais les changements de locuteur (POS LOC) constituent un équivalent tout à fait acceptable.

Une fois que nous avons récupéré toutes nos ressources, il ne reste plus qu'à générer le modèle de langage (fichier *.par*). Pour cela, on utilise le logiciel *train-tree-tagger* qui doit se trouver dans votre dossier TDCORPUS (cf. TD précédent).

Rappel : il faut ouvrir une ligne de commande et se positionner dans le dossier TDCORPUS. Je donne ici la syntaxe pour Unix (Linux et MacOS) ; **pour Windows**, il faut remplacer **./train-tree-tagger** par **train-tree-tagger.exe** et remplacer **./tree-tagger** par **tree-tagger.exe** .

On peut finalement lancer l'apprentissage avec la commande :

- `./train-tree-tagger -utf8 -st LOC lexique_morphalou_perceo_oral.txt tags.txt corpus_perceo_oral.txt tdcorpus.par`

Détaillons les options passées à la commande train-tree-tagger :

- `-utf8` : on travaille en Unicode (UTF-8). Il n'y a même pas à réfléchir : au XXIe siècle, on travaille toujours en Unicode UTF-8, tous les autres jeux de caractères sont des reliques d'un passé que personne ne regrette.
- `-st LOC` : la POS qui marque les fins de phrase. Dans ce corpus, on utilise LOC.
- `lexique_morphalou_perceo_oral.txt` : le dictionnaire.
- `tags.txt` : la liste des POS.
- `corpus_perceo_oral.txt` : le corpus d'apprentissage.
- `tdcorpus.par` : le nom du fichier (modèle de langage) qu'on va fabriquer. On va l'appeler *tdcorpus.par*, mais évidemment on pourrait l'appeler comme on veut.

Notez bien que **l'ordre des options est important**, sauf `-utf8` et `-st LOC`. Le premier fichier passé en paramètre est toujours le dictionnaire, le second toujours la liste des POS, etc.

Il y a aussi d'autres options qui permettent de légèrement optimiser l'apprentissage, mais ça ne fait pas une grosse différence à l'usage, donc on ne va pas les utiliser.

Si tout se passe bien, vous obtenez ceci :

```
train-tree-tagger -cl 2 -dtg 0.50 -sw 1.00 -ecw 0.15 -atg 1.20 lexique_morphalou_perceo_oral.txt tags.txt corpus_perceo_oral.txt test.par

        reading the lexicon ...
                reading the tagset ...
                reading the lemmas ...
                reading the entries ...
                sorting the lexicon ...
                reading the open class tags ...
        calculating tag frequencies ...
102000  making affix tree ...
prefix lexicon: 17028 nodes
suffix lexicon: 5820 nodes
        reading classes ...
        making ngram table ...
102139  8412
finished.
        making decision tree ...
122     saving parameters ...

Number of nodes: 123
Max. path length: 35

done.
```

Tout d'abord (première ligne), TreeTagger vous liste les paramètres qu'il utilise ; il y en a plus que ce que nous avons utilisé (-cl 2, -dtg 0.50, etc.) : il s'agit de valeurs par défaut, que l'on peut modifier. Comme dit plus haut, dans notre cas, modifier ces paramètres ne change pas grand-chose, donc ces valeurs par défaut sont très bien.

Ensuite, TreeTagger nous raconte son apprentissage. On ne va pas tout détailler, mais par exemple dans notre cas, on voit que TreeTagger a identifié 17028 préfixes, 5828 suffixes, et qu'il a créé un arbre de décision de 123 nœuds, dont la branche la plus longue fait 35 nœuds.

Et enfin on obtient notre modèle de langage, dans le fichier *tdcorpus.par*.

Évaluation du modèle

Maintenant qu'on a un modèle de langage, on va essayer de voir ce qu'il vaut ! Concrètement, ce n'est pas bien sorcier : on va prendre un bout de corpus déjà annoté à la main, le faire ré-annoter par TreeTagger, et calculer un % de différence entre les deux annotations.

En pratique, on va souvent passer beaucoup de temps à tester le modèle, chercher les erreurs les plus fréquentes, modifier les ressources, et ré-entraîner le modèle. On n'intervient jamais directement sur le fichier *.par*, c'est une boîte noire ; à la place, on intervient sur les ressources. À cette étape, il est précieux de bien connaître le fonctionnement de TreeTagger (dont on a eu un aperçu lors du premier TD), car il faut deviner, en fonction des erreurs, quelle modification des ressources pourrait améliorer les choses. Pour ce TD, on va toutefois rester à un niveau très simple.

Nous allons évaluer notre modèle de manière quantitative, en calculant un score de *précision*. Il y a deux corpus sur lesquels ont peut calculer ces scores :

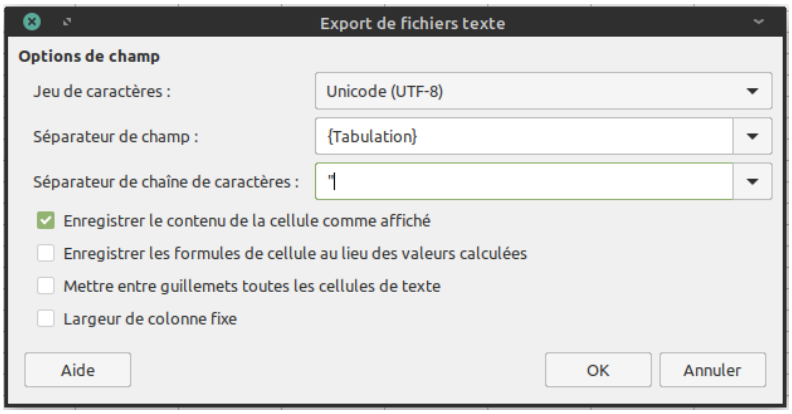
- Sur le corpus d'apprentissage : permet de déterminer si notre TreeTagger apprend bien.
- Sur le corpus que l'on souhaite annoter : permet de déterminer si le modèle est adapté à ce qu'on souhaite annoter.

Dans les deux cas, ça implique d'avoir un corpus annoté à la main qui va servir de référence. Sur le corpus d'apprentissage, on en a un (c'est le corpus qu'on utilise pour l'apprentissage !). Sur le corpus que l'on veut annoter, souvent on n'en a pas (puisqu'on veut l'annoter) ; c'est tout de même

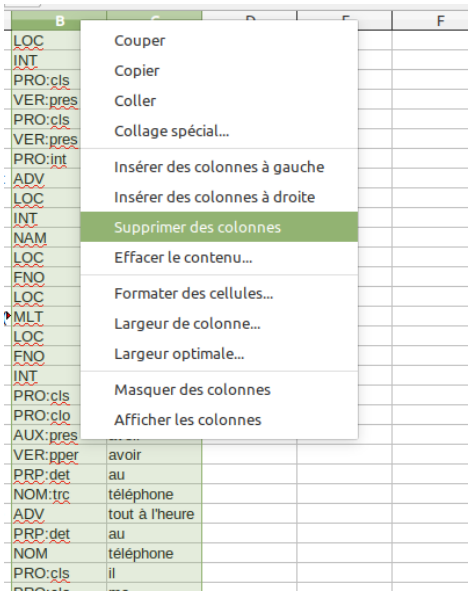
une bonne pratique d'en annoter une petite partie à la main pour pouvoir évaluer. Pour ce TD, nous ne ferons que le corpus d'apprentissage.

Comme c'est un peu trop « facile » d'évaluer l'apprentissage de TreeTagger sur exactement le même corpus qu'on utilise pour l'apprentissage, on va le couper en deux parties. En général, on prend 90 % pour l'apprentissage et 10 % pour l'évaluation. Notre corpus d'apprentissage fait 102 139 tokens, nous allons donc prendre les 10 000 premières lignes (environ) et les placer dans un autre fichier.

Cela se fait facilement avec un tableur (couper/coller). On va couper les ~10 000 premières lignes du corpus d'apprentissage (attention de terminer la découpe sur une fin de phrase -POS LOC-, afin que la partie coupée se termine par un LOC), et coller ça dans un nouveau tableur. On va ensuite sauvegarder ce nouveau tableur (qui contient ~10 000 lignes) sous le nom *tdcorpus-test-annote-manuel.csv*. Lors de la sauvegarde au format CSV, le tableur LibreOffice Calc va demander de préciser le format exact. Il faut bien indiquer *Unicode (UTF-8)* pour les caractères et *Tabulation* comme séparateur :

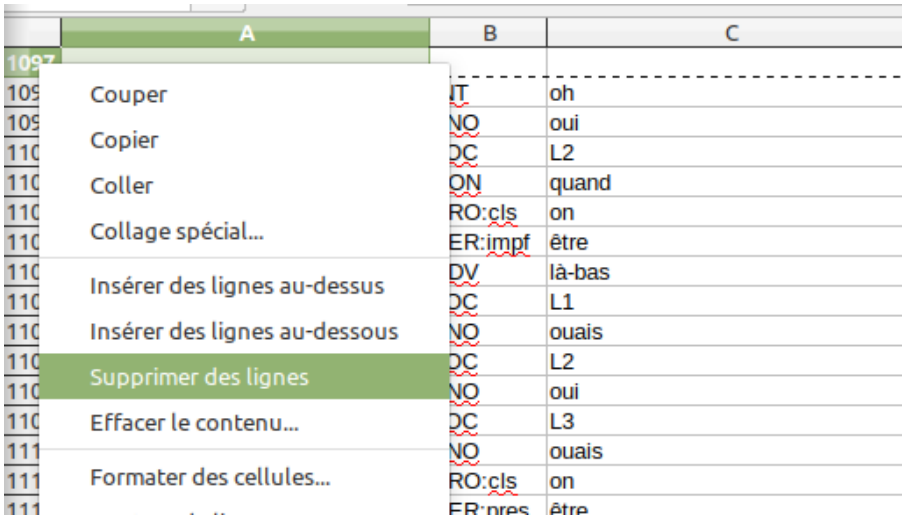


Ensuite, dans ce fichier qu'on vient de sauvegarder, on va sélectionner et supprimer les deux colonnes de droite, qui correspondent à l'annotation, pour transformer le corpus annoté en corpus non-annoté :



On va re-sauvegarder ce fichier,sous un autre nom (*Enregistrer Sous*) : *tdcorpus-test-nonannote.csv*.

Ensuite, retour dans le fichier d'apprentissage du départ, *corpus_perceo_oral.txt*, qu'on a donc amputé de ses premières lignes. On va sélectionner toutes ces lignes vides, puis les supprimer avec un clic droit comme ceci :



Il ne reste plus qu'à sauvegarder ce fichier raccourci, avec *Enregistrer Sous* (pour éviter d'écraser le fichier original), et nous allons l'appeler *tdcorpus-apprentissage.csv*. Ici, le tableur Calc ne nous demande pas de préciser le format exact (Unicode + Tabulation), puisqu'il le connaît déjà.

Nous avons donc maintenant trois nouveaux fichiers :

- ***tdcorpus-test-annote.csv* qui fait environ 10 000 lignes annotées (3 colonnes), et se termine par un LOC.**
- ***tdcorpus-test-nonannote.csv* qui contient les mêmes lignes que le précédent, mais sans les annotations (une seule colonne).**
- ***tdcorpus-apprentissage.csv* qui contient environ 90 000 lignes annotées (3 colonnes), et ne commence pas par un LOC.**

On peut ensuite relancer l'apprentissage sur ce nouveau corpus d'apprentissage :

- `./train-tree-tagger -utf8 -st LOC lexique_morphalou_perceo_oral.txt tags.txt tdcorpus-apprentissage.csv tdcorpus-v2.par`

On obtient un nouveau modèle de langage tdcorpus-v2.par, que l'on va utiliser pour annoter le corpus de test non annoté (mais déjà tokénisé), *tdcorpus-test-nonannote.csv* et envoyer le résultat dans un fichier *tdcorpus-test-annote-automatique.csv* :

- `./tree-tagger -token -lemma tdcorpus-v2.par tdcorpus-test-nonannote.csv > tdcorpus-test-annote-automatique.csv`

On a donc maintenant deux fichiers de test annotés : *tdcorpus-test-annote-manuel.csv* (les ~10 000 premières lignes de notre corpus d'apprentissage), et *tdcorpus-test-annote-automatique.csv* (les mêmes lignes, mais dont on avait ôté l'annotation, avant de les reculer avec TreeTagger). On va ouvrir ces deux fichiers, copier leur contenu, et le coller côte à côte dans un nouveau tableur. À

gauche (colonnes A à C), nous allons mettre les 3 colonnes de *tdcorpus-test-annote-manuel.csv*, et à droites (colonnes E à G), nous allons mettre les 3 colonnes de *tdcorpus-test-annote-automatique.csv*. Comme ceci :

	A	B	C	D	E	F	G
1	L2	LOC	L2		L2	LOC	L2
2	ben	INT	ben		ben	INT	ben
3	on	PRO:cls	on		on	PRO:cls	on
4	attend	VER:pres	attendre		attend	VER:pres	attendre
5	on	PRO:cls	on		on	PRO:cls	on
6	attend	VER:pres	attendre		attend	VER:pres	attendre
7	qui	PRO:int	qui		qui	PRO:rel	qui
8	normalement	ADV	normalement		normalement	ADV	normalement
9	L3	LOC	L3		L3	LOC	L3
10	ben	INT	ben		ben	INT	ben
11	¤P1¤	NAM	¤P1¤		¤P1¤	NAM	¤P1¤
12	L4	LOC	L4		L4	LOC	L4
13	non	FNO	non		non	FNO	non
14	L1	LOC	L1		L1	LOC	L1
15	/tu les as eu(	MLT	/tu les as eu(e)s,***/		/tu les as eu(e)s,***/	NAM	<unknown>
16	L3	LOC	L3		L3	LOC	L3
17	ouais	FNO	ouais		ouais	FNO	ouais
18	ben	INT	ben		ben	INT	ben
19	je	PRO:cls	je		je	PRO:cls	je
20	l'	PRO:clo	le		l'	PRO:clo	le
21	ai	AUX:pres	avoir		ai	AUX:pres	avoir
22	eu	VER:pper	avoir		eu	VER:pper	avoir
23	au	PRP:det	au		au	PRP:det	au
24	tel	NOM:trc	téléphone		tel	ADJ	tel
25	tout à l'heure	ADV	tout à l'heure		tout à l'heure	ADV	tout à l'heure
26	au	PRP:det	au		au	PRP:det	au
27	téléphone	NOM	téléphone		téléphone	NOM	téléphone

Ainsi, nous pouvons comparer l'annotation espérée (à gauche) et l'annotation effective (à droite).

Pour cela, nous allons ajouter deux colonnes au tableur : une colonne H qui ne contient que des 1, et une colonne I qui contient la formule :

- `=SI(EXACT(B1;F1);1;0)`

… qui pour chaque ligne affiche 1 si les POS correspondent, et 0 si elle ne correspondent pas. Comme ceci :

	A	B	C	D	E	F	G	H	I	J
1	L2	LOC	L2		L2	LOC	L2	1	=SI(EXACT(B1;F1);1;0)	
2	ben	INT	ben		ben	INT	ben	1	1	
3	on	PRO:cls	on		on	PRO:cls	on	1	1	
4	attend	VER:pres	attendre		attend	VER:pres	attendre	1	1	
5	on	PRO:cls	on		on	PRO:cls	on	1	1	
6	attend	VER:pres	attendre		attend	VER:pres	attendre	1	1	
7	qui	PRO:int	qui		qui	PRO:rel	qui	1	0	
8	normalement	ADV	normalement		normalement	ADV	normalement	1	1	

Si vous ne savez pas « étirer » des colonnes dans un tableur, vous pouvez vous reporter à ces tutoriels :

- https://www.youtube.com/watch?v=mKHvEBxEgT4
- https://www.youtube.com/watch?v=trM3c9Gfa-A

Tout en bas de la feuille de calcul, nous allons calculer le nombre de tokens (colonne I) et le nombre de tokens corrects (colonne H), avec les formules :

- `=SOMME(I1:I1097)`
- `=SOMME(H1:H1097)`

Notez bien que dans mon cas, c'est 1097 parce que c'est la dernière ligne de mon fichier ; mais dans votre cas, si votre fichier ne fait pas tout à fait la même longueur, il vous faudra adapter cette valeur.

1091	vous	PRO:cls	vous	vous	PRO:cls	vous	1	1
1092	étiez	AUX:impf	être	étiez	VER:impf	être	1	0
1093	bien	ADV	bien	bien	ADV	bien	1	1
1094	bien	ADV	bien	bien	ADV	bien	1	1
1095	accueillis	VER:pper	accueillir	accueillis	VER:pper	accueillir	1	1
1096	là-haut	ADV	là-haut	là-haut	ADV	là-haut	1	1
1097	L3	LOC	L3	L3	LOC	L3	1	1
1098							1097	=SOMME(I1:I1097)

Ensuite, on calcule simplement le score de précision en divisant le nombre de tokens corrects par le nombre de tokens total :

	ADV	bien	1	1	
	VER:pper	accueillir	1	1	
	ADV	là-haut	1	1	
	LOC	L3	1	1	
			1097	1061	=I1098/H1098

Ici, on obtient un score de 0,967, soit 96,7 %. Notez que puisque ce score est calculé sur environ 1000 lignes, descendre au-delà des millièmes (en dessous de 0,001 ou 0,1 %) n'a pas de sens. Avec TreeTagger, un score > 96 % est considéré comme un « bon score », c'est celui qu'on obtient dans de bonnes conditions. Or, les conditions sont ici idéales, puisqu'on a appris sur une autre partie du *même* corpus.

Annotation du corpus L'Île Mystérieuse

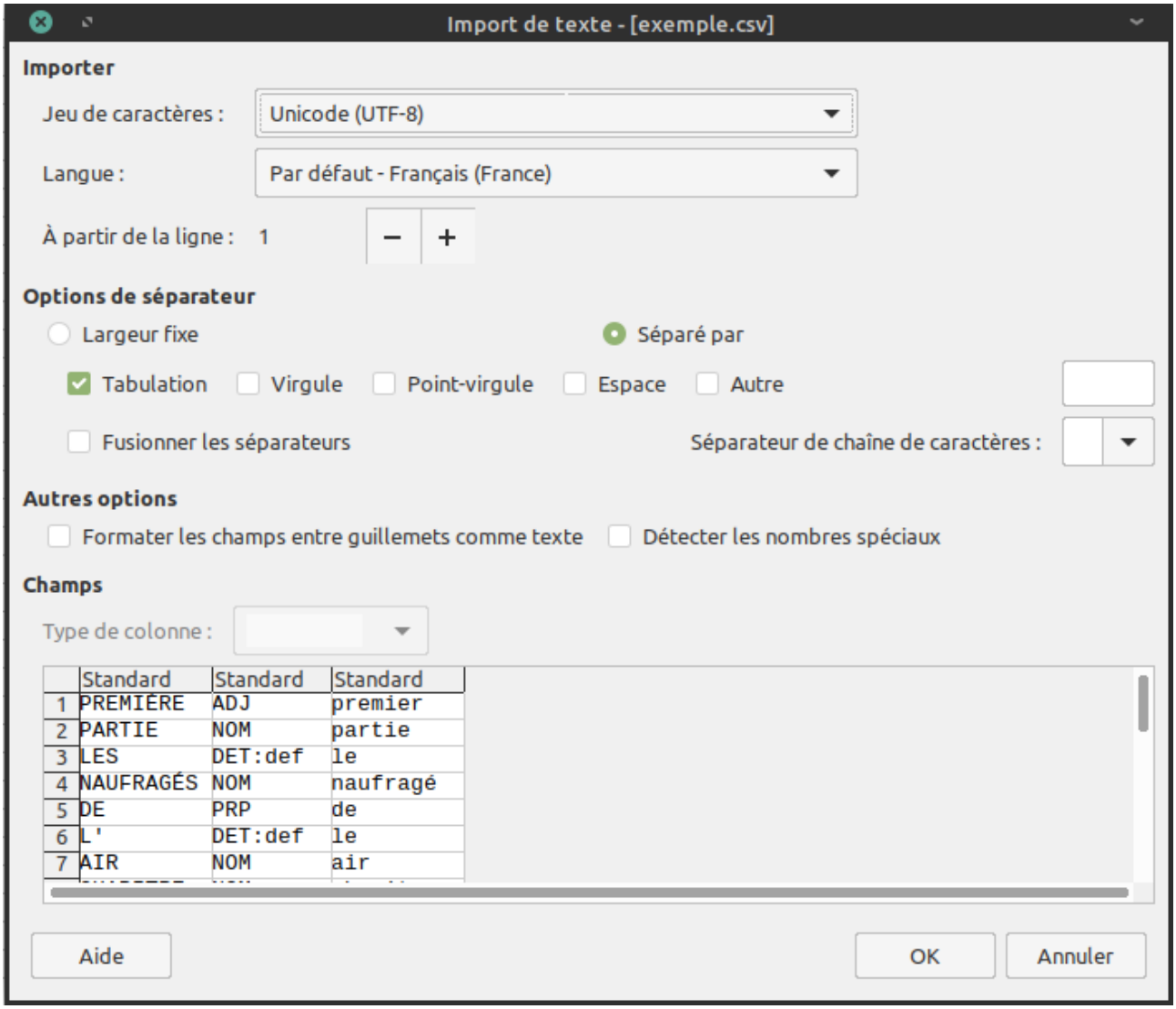
Maintenant que nous avons un modèle de langage, et que nous savons qu'il est « bon » (en tout cas sur le corpus d'apprentissage), nous allons nous en servir pour annoter un autre corpus : le texte de l'Île Mystérieuse que nous avions utilisé au TD précédent (fichier *exemple.txt*).

Il est assez discutable d'utiliser un modèle entraîné sur des transcriptions de parole pour annoter un corpus de littérature, mais nous essayons ici de garder les choses simples, et cela va permettre de bien mettre en évidence les problèmes.

Annotons donc ce corpus (*exemple.txt*) avec notre nouveau modèle, vers le fichier *exemple.csv* :

- `cat exemple.txt | perl utf8-tokenize.perl -f -a french-abbreviations | ./tree-tagger -token -lemma tdcorpus-v2.par > exemple.csv`

Ouvrons le fichier *exemple.csv* que nous venons de créer avec le tableur Calc. Vérifiez que vous avez les bonnes options ; notamment Unicode (UTF-8), Tabulation, et que le **Séparateur de chaîne de caractères est réglé sur « rien »** (effacez le guillemet qui se trouve par défaut dans ce champ) :



Dans ce fichier, on lit alors ceci :

	A	B	C
1	PREMIÈRE	ADJ	premier
2	PARTIE	NOM	partie
3	LES	DET:def	le
4	NAUFRAGÉS	NOM	naufragé
5	DE	PRP	de
6	L'	DET:def	le
7	AIR	NOM	air
8	CHAPITRE	NOM	chapitre
9	I	NOM	<unknown>
10	L'	DET:def	le
11	ouragan	NOM	ouragan
12	de	PRP	de
13	1865	NAM	@card@
14	.	NAM	<unknown>

On voit que les résultats sont très mauvais. C'est attendu, puisqu'on est vraiment sur un type de texte totalement différent du corpus d'apprentissage, mais imaginons que nous n'ayons pas de meilleur corpus d'apprentissage à disposition et qu'il faille améliorer les choses. Et en pratique, même dans les cas moins extrêmes, il y a très souvent des ajustements à faire.

En général, on essaye de trouver en priorité les problèmes les plus récurrents. Dans notre cas, le principal problème saute aux yeux : notre corpus d'apprentissage et le dictionnaire ne contenaient aucun signe de ponctuation. Par conséquent, l'analyse se trompe sur tous les signes de ponctuation.

Pour introduire les signes de ponctuation dans les ressources d'apprentissage, nous allons ajouter une POS PONCT pour tous les signes de ponctuation :

- Dans le fichier *tags.txt*, nous ajoutons une ligne PONCT .
- Dans le fichier *lexique_morphalou_perceo_oral.txt* , nous ajoutons trois lignes :
  - . PONCT .
  - ; PONCT ;
  - , PONCT ,
  - Notez bien que la forme et la POS sont séparées par une *tabulation*, alors que la POS et le lemme sont séparés par un espace. Prenez modèle sur le reste du fichier.

Reprenons l'entraînement du modèle, que nous appellerons *tdcorpus-v3.par* :

- `./train-tree-tagger -utf8 -st LOC lexique_morphalou_perceo_oral.txt tags.txt tdcorpus-apprentissage.csv tdcorpus-v3.par`

On relance ensuite l'annotation :

- `cat exemple.txt | perl utf8-tokenize.perl -f -a french-abbreviations | ./tree-tagger -token -lemma tdcorpus-v2.par > exemple.csv`

C’est mieux ! Loin d’être parfait, mais au moins les signes de ponctuation ne sont plus inconnus.

	A	B	C
1	PREMIÈRE	ADJ	premier
2	PARTIE	NOM	partie
3	LES	DET:def	le
4	NAUFRAGÉS	NOM	naufragé
5	DE	PRP	de
6	L'	DET:def	le
7	AIR	NOM	air
8	CHAPITRE	NOM	chapitre
9	I	NOM	<unknown>
10	L'	DET:def	le
11	ouragan	NOM	ouragan
12	de	PRP	de
13	1865	DET:dem	@card@
14	.	PONCT	.

On pourrait continuer encore longtemps ; c’est un travail qui prend facilement plusieurs jours, même dans des cas moins extrêmes. Notamment, dans le cas où on introduit une nouvelle POS (ce qu’on a fait ici, mais qu’en réalité on fait très rarement !), il est souhaitable de modifier aussi le corpus d’apprentissage pour permettre à TreeTagger d’apprendre l’usage de cette nouvelle POS. On peut aussi s’apercevoir que certaines constructions (séquences de POS) sont présentes dans le corpus qu’on veut annoter, mais absentes du corpus d’apprentissage ; y ajouter quelques exemples peut dans ce cas aussi permettre de résoudre des problèmes. Enfin, on recherche systématiquement les tokens inconnus (ceux dont le token est <unknown>) qui peuvent être révélateurs de problèmes de tokenisation, ou de lacunes dans le lexique. Mais ce sera tout pour ce TD !

Conclusion

Ce qu’il faut essayer de retenir de ce TD, c’est comment on crée un modèle de langage :

- le schéma en introduction,
- savoir créer un modèle de langage à partir des fichiers ressources de base (corpus d’apprentissage, dictionnaire et liste des POS),
- savoir évaluer un modèle dans LibreOffice.