

Segmenteur Étiqueteur Markovien (SEM)

Table des matières

1	Préface	3
1.1	Présentation de SEM	3
2	Installation	3
2.1	Si GIT est installé	3
2.2	Si GIT n'est pas installé	4
2.3	Wapiti	4
2.3.1	Erreurs de compilation	4
3	Corpus, annotations et ressources linguistiques	5
3.1	French Treebank (FTB)	5
3.2	Jeu d'annotation PoS	5
3.3	Annotation en chunks	5
3.4	Annotation en entités nommées	6
3.5	Lexique des Formes Fléchies du Français (LeFFF)	6
4	Formats des fichiers	6
4.1	fichiers linéaires	6
4.1.1	Exemples	6
4.2	fichiers vectorisés	7
4.2.1	Exemples	7
5	Utilisation	7
5.1	wapiti_label	8
5.2	chunking_fscore	8
5.3	clean	9
5.4	enrich	10
5.5	export	10
5.6	label_consistency	11
5.7	annotate	12
5.8	segmentation	13
5.9	compile	14

5.10	decompile	14
5.11	tagger	15
5.12	gui	15
5.13	annotation_gui	16
6	Fichiers de configuration	16
6.1	Pour le module enrich	16
6.1.1	Les features <i>arity</i>	18
6.1.2	Les features <i>boolean</i>	19
6.1.3	Les features <i>dictionary</i>	20
6.1.4	Les features <i>directory</i>	20
6.1.5	Les features <i>list</i>	20
6.1.6	Les features <i>matcher</i>	21
6.1.7	Les features <i>rule</i>	21
6.1.8	Les features <i>string</i>	22
6.1.9	Les features <i>trigger</i>	22
6.2	Pour le module tagger	23

1 Préface

1.1 Présentation de SEM

Segmenteur Étiqueteur Markovien (SEM) [Tellier et al.2012b] est un logiciel d'annotation syntaxique du français.

Il permet la segmentation de texte brut en phrases, elles-même découpées en unités lexicales, mais il est tout-à-fait en mesure de traiter un texte pré-segmenté. Les unités multi-mots peuvent être gérées de deux manières différentes : soit comme une seule unité lexicale où chaque mot est relié par le caractère '_', soit comme une suite de mots ayant une annotation particulière précisant que les mots sont reliés entre eux et possèdent globalement la même fonction syntaxique.

SEM propose trois niveaux d'annotation : le premier est une annotation morpho-syntaxique de chaque unité lexicale du texte selon le jeu d'étiquettes défini par [Crabbé and Candito2008]. Le deuxième est une annotation en chunks selon le modèle BIO (Begin In Out), le programme permettant d'obtenir un étiquetage selon un chunking complet ou bien partiel, auquel cas il ne reconnaîtra que les groupes nominaux (le chunking partiel étant soumis à des règles différentes que le chunking complet, l'un n'est donc pas un sous-ensemble de l'autre). Le troisième est une annotation en entités nommées.

Toutes les commandes du manuel sont mises entre guillemets pour les distinguer clairement du reste du texte, mais elle doit être écrite sans eux.

2 Installation

Sur la page suivante se trouvent toutes les informations nécessaires :

<https://github.com/YoannDupont/SEM>

SEM doit être téléchargé pour être installé, l'installation se lance de la façon suivante :

```
python setup.py install --user
```

qui se chargera d'installer SEM avec tous les prérequis.

Il existe deux possibilités pour télécharger la dernière version.

2.1 Si GIT est installé

Il faut alors aller dans un terminal et taper la commande suivante :

```
git clone https://github.com/YoannDupont/SEM.git
```

Cela va créer un dossier de SEM dans le répertoire où est tapée la commande.

Il s'agit de la branche GIT (dépôt), qui sert à gérer les différentes versions du logiciel. Il ne faut en AUCUN cas modifier le contenu de ce dossier (c'est surtout vrai si on prévoit de mettre-à-jour la branche, mais c'est une habitude à prendre immédiatement). Pour utiliser SEM, il faut copier les différents fichiers et dossiers dans un autre répertoire. Un dossier `.git` est présent : étant caché il ne sera pas copié si on n'active pas l'affichage des fichiers cachés, sinon il faut le désélectionner. C'est ce dossier qui contient les informations de versionnement.

L'intérêt ici est de pouvoir mettre-à-jour simplement le logiciel en tapant la commande "git pull" dans la branche. Cela mettra à jour uniquement les fichiers qui doivent l'être, ce qui est pratique quand (comme ici) le contenu est assez lourd.

2.2 Si GIT n'est pas installé

Sur <https://github.com/YoannDupont/SEM> cliquer sur "clone or download" puis "Download ZIP".

L'avantage de cette méthode est qu'il s'agit des fichiers non-versionnés, il n'est donc pas nécessaire d'être aussi précautionneux avec le contenu du dossier. L'inconvénient est que pour mettre à jour, il faut tout retélécharger.

2.3 Wapiti

Wapiti [Lavergne et al.2010] est un logiciel implémentant les CRF, il permet d'apprendre des annotateurs à partir de corpus annotés et d'effectuer l'annotation.

La dernière version de Wapiti compatible avec SEM est disponible dans le dossier ext. Les consignes d'installation sont disponibles avec. SEM est prévu pour fonctionner avec le Wapititel qu'il est fourni dans le dossier ext, il faut le compiler pour pouvoir appeler Wapiti avec SEM. Depuis la version 3.0.0 de SEM, Wapiti est automatiquement compilé à l'installation.

2.3.1 Erreurs de compilation

Wapiti est un logiciel ayant recours à certaines spécificités du matériel et du système d'exploitation pour améliorer ses performances. En conséquence, il est possible d'avoir des erreurs dues à l'absence de ces spécificités sur votre machine.

La plus fréquente semble être due à la fonction "`__sync_bool_compare_and_swap`" présente dans le fichier "`gradient.c`". Si la commande `make` provoque une erreur et vous affiche des messages relatifs à cette fonction, la procédure est très simple.

Dans le fichier "`wapiti.h`", cherchez la ligne :
`//#define ATM_ANSI`
Et supprimez la chaîne "`//`" en début de ligne pour obtenir :
`#define ATM_ANSI`
Sauvegardez et reprenez la procédure d'installation.

3 Corpus, annotations et ressources linguistiques

3.1 French Treebank (FTB)

SEM a été appris automatiquement sur le French Treebank (FTB) [[Abeillé et al.2003](#)].

3.2 Jeu d'annotation PoS

L'annotation PoS se base sur le jeu d'étiquettes défini par [[Crabbé and Candito2008](#)] :

ADJ : adjectif	P : préposition
ADJWH : adjectif	P+D : préposition
ADV : adverbe	P+PRO : préposition
ADVWH : adverbe	PONCT : ponctuation
CC : conjonction de coordination	PREF : préfixe
CLO : clitique objet	PRO : pronom
CLR : clitique réfléchi	PROREL : pronom
CLS : clitique sujet	PROWH : pronom
CS : conjonction de subordination	VINF : verbe à l'infinitif
DET : déterminant	VPR : verbe au participe présent
DETH : déterminant	VPP : verbe au participe passé
ET : mot étranger	V : verbe à l'indicatif
I : interjection	VS : verbe au subjonctif
NC : nom commun	VIMP : verbe à l'impératif
NPP : nom propre	

3.3 Annotation en chunks

Le chunking utilise les annotations définies dans [[Tellier et al.2012a](#)] :

AP : groupe adjectival
AdP : groupe adverbial
NP : groupe nominal
VN : noyau verbal

CONJ : conjonction
UNKNOWN : chunk inconnu
PP : chunk prépositionnel

3.4 Annotation en entités nommées

Pour effectuer la reconnaissance des entités nommées, SEM se base sur les annotations définies par [Sagot et al.2012] :

Person : les personnes
Location : les lieux
Organization : toute organisation ou association à but non lucratif
Company : les entreprises
POI : Point Of Interet (exemple : l'Opéra)
FictionCharacter : les personnages fictifs
Product : les produits

3.5 Lexique des Formes Fléchies du Français (LeFFF)

Le Lexique des Formes Fléchies du Français (LeFFF) [Clément et al.2004] est un lexique du Français riche fournissant des information morphologiques et syntaxiques. SEM utilise le LeFFF en tant que dictionnaire afin d'améliorer la qualité de son annotation PoS.

4 Formats des fichiers

SEM permet de traiter deux types de fichiers en entrée : les fichiers dits linéaires et les fichiers dits vectorisés.

4.1 fichiers linéaires

Un fichier linéaire est un fichier dans lequel les mots sont (souvent) séparés par un espace. Ils représentent la majorité des textes (texte brut). SEM considère qu'un retour à la ligne termine une phrase, lorsqu'il fournit en sortie un fichier linéaire, chaque phrase sera séparée par un retour à la ligne. Si un fichier en entrée est un fichier linéaire, SEM pourra le segmenter en tokens et phrases.

SEM ne peut traiter en entrée que les fichiers de texte brut.

4.1.1 Exemples

exemple 1 : texte brut
Le chat dort.

exemple 2 : texte annoté en PoS

Le/DET chat/NC dort/V ./PONCT

exemple 3 : texte annoté en PoS et en chunks

(NP Le/DET chat/NC) (VN dort/V) ./PONCT

4.2 fichiers vectorisés

Un fichier vectorisé est un fichier où chaque mot est sur une ligne, les phrases étant séparées par une ligne vide. Dans un fichier vectorisé, chaque token peut contenir plusieurs informations, ces informations sont séparées par des tabulations. Chaque information est donc sur une « colonne » qui lui est spécifique.

4.2.1 Exemples

exemple 1 : texte brut vectorisé

Le
chat
dort
.

exemple 2 : texte vectorisé enrichi avec l'information « le mot commence-t-il par une majuscule ? »

Le oui
chat non
dort non
. non

exemple 3 : texte vectorisé annoté en PoS

Le DET
chat NC
dort V
. PONCT

exemple 4 : texte vectorisé annoté en PoS et en chunks

Le DET B-NP
chat NC I-NP
dort V B-VN
. PONCT 0

5 Utilisation

SEM dispose de modules indépendants les uns des autres, le programme principal faisant alors office d'aiguilleur vers le module à lancer.

Pour obtenir la liste des modules disponibles et la syntaxe générale pour les lancer :

`python -m sem (-help ou -h)`

Pour connaître la version de SEM :

`python -m sem (-version ou -v)`

Pour connaître les informations relatives à la dernière version de SEM :

`python -m sem (-informations ou -i)`

Pour lancer un module, la syntaxe générale est :

`python -m sem <nom_du_module> <arguments_et_options_du_module>`

Les différents modules seront détaillés un par un.

5.1 `wapiti_label`

description

Surcouche à la commande "wapiti label" de Wapiti.

arguments

`infile`

le fichier d'entrée (format CoNLL).

`model`

le fichier modèle utilisé par Wapiti.

`outfile`

le fichier de sortie (format CoNLL).

options

`-help` ou `-h` : switch

affiche l'aide

5.2 `chunking_fscore`

description

Calcule la f-mesure sur des données étiquetées selon un schéma BIO. Fournit une f-mesure par classe, une micro f-mesure globale et une macro f-mesure globale.

arguments

`infile`

Le fichier contenant les données étiquetées à évaluer. Ce fichier est au format tabulaire type CoNLL 2003. Ce script est similaire à `conlleval`.

options

`-help` ou `-h` : switch

affiche l'aide

`-reference-column` ou `-r` : int

l'indice de la colonne où se trouvent les étiquettes de références (défaut : -2).

- `-tagging-column` ou `-t` : int
l'indice de la colonne où se trouvent les étiquettes hypothèses fournies par le système (défaut : -1).
- `-input-encoding` : string
définit l'encodage du fichier d'entrée. Prioritaire sur la valeur de `-encoding` (défaut : `-encoding`).
- `-output-encoding` : string
définit l'encodage du fichier de sortie. Prioritaire sur la valeur de `-encoding` (défaut : `-encoding`).
- `-encoding` : string
définit l'encodage du fichier d'entrée et de sortie. Si un encodage est fourni pour un fichier, cette valeur est surchargée (défaut : UTF-8).
- `-log` ou `-l` : string
définit le niveau de log : info, warn ou critical (défaut : critical).
- `-log-file` : fichier
le fichier où écrire le log (défaut : sortie terminal).

5.3 clean

description

`clean` permet de supprimer des colonnes d'informations dans un fichier vectorisé.

arguments

- `infile` : fichier
le fichier d'entrée. Format vectorisé.
- `outfile` : fichier
le fichier de sortie. S'il existe déjà, son contenu sera écrasé.
- `ranges` : string
les colonnes à garder. Il est possible de donner soit un numéro de colonne soit une portée. Une portée se constitue de deux nombres séparés par le symbole « : ». Il est possible de fournir plusieurs valeurs en les séparant par le symbole « , ».

options

- `-help` ou `-h` : switch
affiche l'aide
- `-input-encoding` : string
définit l'encodage du fichier d'entrée. Prioritaire sur la valeur de `-encoding` (défaut : `-encoding`).
- `-output-encoding` : string
définit l'encodage du fichier de sortie. Prioritaire sur la valeur de `-encoding` (défaut : `-encoding`).
- `-encoding` : string
définit l'encodage du fichier d'entrée et de sortie. Si un encodage est fourni

pour un fichier, cette valeur est surchargée (défaut : UTF-8).
-log ou -l : string
définit le niveau de log : info, warn ou critical (défaut : critical).
-log-file : fichier
le fichier où écrire le log (défaut : sortie terminal).

5.4 enrich

description

Permet de rajouter des informations à un fichier vectorisé. Les informations rajoutées sont déclarées dans un fichier de configuration xml. Ces traits sont détaillés dans la section [6.1](#).

arguments

infile : fichier
le fichier d'entrée, format vectorisé.
infofile : fichier
fichier pour ajouter des informations, format xml.
outfile : fichier, format vectorisé.
le fichier de sortie

options

-help ou -h : switch
affiche l'aide
-input-encoding : string
définit l'encodage du fichier d'entrée. Prioritaire sur la valeur de -encoding (défaut : -encoding).
-output-encoding : string
définit l'encodage du fichier de sortie. Prioritaire sur la valeur de -encoding (défaut : -encoding).
-encoding : string
définit l'encodage du fichier d'entrée et de sortie. Si un encodage est fourni pour un fichier, cette valeur est surchargée (défaut : UTF-8).
-log ou -l : string
définit le niveau de log : info, warn ou critical (défaut : critical).
-log-file : fichier
le fichier où écrire le log (défaut : sortie terminal).

5.5 export

description

Transforme des données du format CoNLL vers un autre format spécifié en argument.

arguments

infile
le fichier d'entrée au format CoNLL.

exporter_name
le nom du format d'export.

outfile
le fichier de sortie.

options

-help ou -h : switch
affiche l'aide

-pos-column ou -p
la colonne où l'information des parties du discours se trouvent.

-chunk-column ou -c
la colonne où l'information du chunking se trouvent.

-ner-column ou -n
la colonne où l'information de la reconnaissance des entités nommées se trouve.

-lang
la langue du document (défaut : fr)

-lang-style ou -s
la feuille de style CSS à utiliser pour l'export HTML (défaut : default.css)

-input-encoding : string
définit l'encodage du fichier d'entrée. Prioritaire sur la valeur de -encoding (défaut : -encoding).

-output-encoding : string
définit l'encodage du fichier de sortie. Prioritaire sur la valeur de -encoding (défaut : -encoding).

-encoding : string
définit l'encodage du fichier d'entrée et de sortie. Si un encodage est fourni pour un fichier, cette valeur est surchargée (défaut : UTF-8).

-log ou -l : string
définit le niveau de log : info, warn ou critical (défaut : critical).

-log-file : fichier
le fichier où écrire le log (défaut : sortie terminal).

5.6 label_consistency

description

Améliore la cohérence des annotations en diffusant dans le document les annotations faites par le système. Les éléments non-annotés identiques à des éléments annotés seront annotés selon la catégorie la plus fréquente.

arguments

infile
le fichier d'entrée (format CoNLL).

outfile
le fichier de sortie (format CoNLL).

options

-help ou -h : switch
affiche l'aide

-token-column ou -t
la colonne où l'information des tokens se trouve.

-tag-column ou -c
la colonne où l'information des étiquettes se trouve.

-label-consistency (choix : non-overriding, overriding)
l'heuristique de diffusion. "non-overriding" laisse les annotations du systèmes telles quelles. "overriding" écrase les annotations du système si une annotation plus longue a est trouvée.

-input-encoding : string
définit l'encodage du fichier d'entrée. Prioritaire sur la valeur de -encoding (défaut : -encoding).

-output-encoding : string
définit l'encodage du fichier de sortie. Prioritaire sur la valeur de -encoding (défaut : -encoding).

-encoding : string
définit l'encodage du fichier d'entrée et de sortie. Si un encodage est fourni pour un fichier, cette valeur est surchargée (défaut : UTF-8).

-log ou -l : string
définit le niveau de log : info, warn ou critical (défaut : critical).

-log-file : fichier
le fichier où écrire le log (défaut : sortie terminal).

5.7 annotate

description

Annote selon l'annotateur donné en argument.

arguments

infile
le fichier d'entrée.

outfile
le fichier de sortie.

annotator
le nom de l'annotateur.

location
le lieu où les informations relatives au système d'annotation se trouvent (modèle, dossier contenant des lexiques, etc).

token_field

le nom de la colonne où l'information des tokens se trouve (pas systématiquement utile)

field

Le nom de la colonne de sortie

options

–help ou -h : switch

affiche l'aide

–input-encoding : string

définit l'encodage du fichier d'entrée. Prioritaire sur la valeur de –encoding (défaut : –encoding).

–output-encoding : string

définit l'encodage du fichier de sortie. Prioritaire sur la valeur de –encoding (défaut : –encoding).

–encoding : string

définit l'encodage du fichier d'entrée et de sortie. Si un encodage est fourni pour un fichier, cette valeur est surchargée (défaut : UTF-8).

–log ou -l : string

définit le niveau de log : info, warn ou critical (défaut : critical).

–log-file : fichier

le fichier où écrire le log (défaut : sortie terminal).

5.8 segmentation

description

Prend un fichier texte linéaire et segmente le texte en phrase et tokens et donne un fichier vectorisé.

arguments

infile : fichier

le fichier d'entrée. Format texte brut.

outfile : fichier

le fichier de sortie. Format vectorisé.

options

–help ou -h : switch

affiche l'aide

–input-encoding : string

définit l'encodage du fichier d'entrée. Prioritaire sur la valeur de –encoding (défaut : –encoding).

–output-encoding : string

définit l'encodage du fichier de sortie. Prioritaire sur la valeur de –encoding (défaut : –encoding).

–encoding : string

définit l'encodage du fichier d'entrée et de sortie. Si un encodage est fourni

pour un fichier, cette valeur est surchargée (défaut : UTF-8).
-log ou -l : string
définit le niveau de log : info, warn ou critical (défaut : critical).
-log-file : fichier
le fichier où écrire le log (défaut : sortie terminal).

5.9 compile

description

Compile (séréalise) un fichier dictionnaire qui pourra alors être utilisé en ressource dans SEM.

arguments

input : fichier dictionnaire
Le dictionnaire à compiler.
output : fichier compilé
Le dictionnaire compilé.

options

-help ou -h : switch
affiche l'aide
-k ou -kind : énumération {token, multiword}
le type de dictionnaire en entrée. token : chaque entrée représente un mot.
multiword : chaque entrée représente une suite de mots.
-i ou -input-encoding : string
définit l'encodage du fichier d'entrée. Prioritaire sur la valeur de -encoding (défaut : -encoding).
-log ou -l : string
définit le niveau de log : info, warn ou critical (défaut : critical).
-log-file : fichier
le fichier où écrire le log (défaut : sortie terminal).

5.10 decompile

description

Décompile (déséréalise) un fichier dictionnaire. Cela permet alors de modifier la ressource (changement d'encodage, ajout / suppression / modification d'entrées).

arguments

input : fichier compilé
Le dictionnaire compilé.
output : fichier dictionnaire
Le dictionnaire décompilé.

options

-help ou -h : switch

- affiche l'aide
- `-input-encoding` : string
définit l'encodage du fichier d'entrée. Prioritaire sur la valeur de `-encoding` (défaut : `-encoding`).
- `-output-encoding` : string
définit l'encodage du fichier de sortie. Prioritaire sur la valeur de `-encoding` (défaut : `-encoding`).
- `-encoding` : string
définit l'encodage du fichier d'entrée et de sortie. Si un encodage est fourni pour un fichier, cette valeur est surchargée (défaut : UTF-8).
- `-log` ou `-l` : string
définit le niveau de log : info, warn ou critical (défaut : critical).
- `-log-file` : fichier
le fichier où écrire le log (défaut : sortie terminal).

5.11 tagger

description

Il s'agit du module principal de SEM. Il permet d'effectuer une chaîne de traitements sur un fichier. Ces traitements correspondent à des modules ou à des annotations faites à l'aide de Wapiti. Les modules à enchaîner et l'ordre dans lequel cet enchaînement s'effectue est donné par un fichier de configuration xml appelé fichier de configuration maître.

arguments

- `master` : fichier xml
le fichier de configuration maître. Définit le séquençage des traitements à effectuer.
- `input_file` : fichier
le fichier d'entrée. Peut être soit un fichier de texte brut soit un fichier vectorisé.

options

- `-help` ou `-h` : switch
affiche l'aide
- `-output-directory` ou `-o` : dossier
le répertoire où les fichiers temporaires vont être créés (défaut : dossier courant).

5.12 gui

description

Interface gratuite permettant d'annoter avec SEM ou d'entraîner un nouveau modèle.

arguments

resources (facultatif)
le dossier de ressources de SEM qui contient les modèles, les lexiques, etc.
options
-help ou -h : switch
affiche l'aide

5.13 annotation_gui

description
Interface graphique pour annoter manuellement des fichiers.
arguments
aucun argument.
options
-help ou -h : switch
affiche l'aide
-log ou -l : string
définit le niveau de log : info, warn ou critical (défaut : critical).

6 Fichiers de configuration

6.1 Pour le module enrich

Le fichier de configuration du module enrich permet d'ajouter des informations à un fichier vectorisé. Il décrit d'abord les entrées présentes puis les informations à ajouter.

Le fichier d'enrichissement est un fichier XML de type de document "enrich". Il se divise en deux parties : une "entries" qui définit les entrées déjà présentes dans le fichier, une "features" qui permet d'enrichir les données.

Chaque entrée (qu'elle soit déjà présente ou ajoutée) doit être nommée (via l'attribut "*name*") et deux entrées différentes ne peuvent pas avoir le même nom. Un exemple de fichier de configuration pour le module enrich est illustré dans la figure 1. Chaque entrée a un mode, qui permet de la considérer ou non selon un certain contexte. Le mode *train* permet de n'utiliser certaines entrées que dans le but d'entraîner de nouveaux modèles. Le mode par défaut, *label*, permet d'avoir les entrées de manière systématique.

La majorité des *features* disposent des champs suivants :

- name="chaîne" : le nom de la *feature*. Obligatoire pour les *features* racines.
- action="chaîne" : pour des features de même nature (ex : évaluation d'une expression régulière), choisit le type de résultat ou un calcul différent.
- x="entier" : le décalage par rapport au mot courant (défaut : 0).

- `entry="chaîne"` : l'entrée que la feature observe (défaut : *word*, sinon *token*, sinon la première entrée définie dans la section *entries*).
- `display="(yes|no)"` : décide si la feature doit être affichée ou non. Il est possible de ne pas afficher une feature qui évalue un résultat intermédiaire pour calculer d'autres features.

Il existe plusieurs types de features en fonction des résultats qu'elles calculent ou du type d'arguments qu'elles prennent en entrée :

- les *token features* : elles traitent un token à la fois, parmi lesquelles :
 - les *string features* : elles traitent un token à la fois et renvoient une chaîne de caractères ;
 - les *boolean features* : elles traitent un token à la fois et renvoient un booléen ;
- les *sequence features* : elles traitent l'ensemble de la séquence en entrée et renvoient une séquence.

```
<? xml version="1.0" encoding="UTF-8" ?>
<information>
  <entries>
    <before>
      <entry name="word" />
      <entry name="POS" />
    </before>
    <after>
      <entry name="NE" mode="train" />
    </after>
  </entries>
  <features>
    <nullary name="lower" action="lower" display="no" />
    <ontology name="NER-ontology" path="dictionaries/NER-ontology" display="no" />
    <fill name="NER-ontology-POS" entry="NER-ontology" filler-entry="POS">
      <string action="equal">O</string>
    </fill>
    <find name="NounBackward" action="backward" return_entry="word">
      <regexp action="check" entry="POS">^N</regexp>
    </find>
    <find name="NounForward" action="forward" return_entry="word">
      <regexp action="check" entry="POS">^N</regexp>
    </find>
  </features>
</information>
```

FIGURE 1 – exemple de fichier de génération de features de SEM, il est utilisé par le module enrich. Il permet de rajouter des descripteurs qui seront alors utilisés par les algorithmes par apprentissage automatique.

6.1.1 Les features *arity*

Les features de type *arity* sont définies en fonction du nombre d'arguments qu'elles prennent en paramètre. Il en existe de plusieurs types.

Le premier type de *token feature arity* est *nullary*. Ce type de feature ne prend aucun argument. Des exemples de features *nullary* sont illustrées dans les figures 2, 3, 4 et 5. Les différentes actions sont les suivantes :

- BOS (*boolean feature*) : le mot est-il en début de séquence ?
- EOS (*boolean feature*) : le mot est-il en fin de séquence ?
- lower (*string feature*) : transforme la chaîne en entrée en minuscules ;
- substring (*string feature*) : fournit une sous-chaîne de la chaîne donnée en entrée. Définit les options suivantes :
 - from="[entier]" : l'indice de début de la sous-chaîne (défaut : 0)
 - to="[entier]" : l'indice de fin de la sous-chaîne. Si 0 est donné, "to" sera la fin de la chaîne (défaut : 0)

```
<nullary name="IsFirstWord?" action="BOS" />
```

FIGURE 2 – exemple de la feature *nullary* "BOS".

```
<nullary name="IsLastWord?" action="EOS" />
```

FIGURE 3 – exemple de la feature *nullary* "EOS".

```
<nullary name="lower" action="lower" />
```

FIGURE 4 – exemple de la feature *nullary* "lower".

```
<nullary name="radical-3" action="substring" to="-3" />
```

FIGURE 5 – exemple de la feature *nullary* "substring".

Le deuxième type de *token feature* est la *unary*. Elle prend un unique argument. Un exemple est illustré sur la figure 6. Les différentes actions sont :

- isUpper : vérifie si le caractère à l'indice donné est en majuscule.

```
<unary name="StartsWithUpper?" action="isUpper">0</unary>
```

FIGURE 6 – exemple de la feature *unary* "isUpper".

Le troisième type de *token feature* est la *binary*. Elle prend exactement deux arguments. Un exemple est illustré sur la figure 7. Les différentes actions sont :

```

<binary name="To-Greek-Alpha" action="substitute">
  <pattern>alpha</pattern>
  <replace>α</replace>
</binary>

```

FIGURE 7 – exemple de la feature *binary* "substitue".

- *substitute* : substitue une chaîne par une autre. Le premier argument est *pattern* et le second est *replace*.

Le quatrième type de *token feature* est la *n-ary*. Elle prend un nombre arbitraire d'argument. Un exemple est illustré sur la figure 8. Les différentes actions sont :

- *sequencer* : effectue une séquence de traitements sur l'élément en entrée. Ces traitements sont systématiquement des *string features*, seul le dernier élément peut être une *string feature* ou une *boolean feature*.

```

<nary name="CharacterClass" action="sequencer">
  <binary action="substitute">
    <pattern>[A-Z]</pattern>
    <replace>A</replace>
  </binary>
  <binary action="substitute">
    <pattern>[a-z]</pattern>
    <replace>a</replace>
  </binary>
  <binary action="substitute">
    <pattern>[0-9]</pattern>
    <replace>0</replace>
  </binary>
  <binary action="substitute">
    <pattern>[^Aa0]</pattern>
    <replace>x</replace>
  </binary>
</nary>

```

FIGURE 8 – exemple de la feature *n-ary* "sequencer". L'exemple ici implémente les classes de caractères (appelées *word shapes* dans [Finkel et al.2004]).

6.1.2 Les features *boolean*

Les *boolean features* *boolean* définissent des expressions booléennes. Un exemple de feature *boolean* est donné dans la figure 9. Trois actions sont disponibles :

- *and* : "et" logique. Prend deux *boolean features* en argument.
- *or* : "ou" logique. Prend deux *boolean features* en argument.
- *not* : "non" logique. Prend une *boolean feature* en argument.

```

<boolean name="StartsWithUpper-AndNot-FirstWordOfSentence" action="and">
  <unary action="isUpper">0</unary>
  <boolean action="not">
    <>nullary action="BOS" />
  </boolean>
</boolean>

```

FIGURE 9 – exemple de la feature *boolean*.

6.1.3 Les features *dictionary*

Les *features* *dictionary* définissent des features se basant sur des lexiques. Les actions suivantes sont possibles :

- token (*boolean feature*) : vérifie l'appartenance d'un token au lexique ;
- multiword (*sequence feature*) : cherche les séquences de tokens appartenant au lexique.

Des exemples des ces features sont illustrées dans les figures 10 et 11.

```

<dictionary name="token-dictionary" action="token" path="path/to/token-dictionary" />

```

FIGURE 10 – exemple de la feature *dictionary* "token".

```

<dictionary name="multiword-dictionary" action="multiword" path="path/to/multiword-dictionary" />

```

FIGURE 11 – exemple de la feature *dictionary* "multiword".

6.1.4 Les features *directory*

Les *features* *directory* permettent d'utiliser des répertoires de lexiques comme définis dans [Dupont2017]. Deux features sont définies :

- directory (*sequence feature*) : applique un répertoire de lexiques. Les tokens non-reconnus sont remplacés par "O". La *feature* attend un champ "path" contenant le chemin vers le dossier contenant l'ensemble des lexique
- fill (*string feature*) : remplace l'élément par le contenu de l'entrée donnée dans le champ "filler-entry" si ce dernier est reconnu par la *boolean feature* donnée en argument.

```

<directory name="NER-ontology" path="../../dictionaries/fr/NER-ontology" />

```

FIGURE 12 – exemple de la feature *directory*.

6.1.5 Les features *list*

La feature *list* est une feature booléenne qui permet de définir une liste non bornée de propriétés booléennes qui seront évaluées. Une feature de type *list* dispose des actions

```
<fill name="NER-ontology-POS" entry="NER-ontology" filler-entry="POS">
  <string action="equal">O</string>
</fill>
```

FIGURE 13 – exemple de la feature *directory* "fill".

suivantes : *none* (toutes les features doivent être évaluées à faux), *some* (au moins une feature doit être évaluée à vrai) et *all* (toutes les features doivent être évaluées à vrai).

```
<list name="top-of-hierarchy" action="some">
  <string action="equal">PDG</string>
  <regexp action="check" entry="lower">^(président|directeur)$</regexp>
</list>
```

FIGURE 14 – exemple de la feature *list* "some".

6.1.6 Les features *matcher*

Les *token features regexp* évaluent des expressions régulières. Les actions suivantes, illustrées sur les figures 15, 16 et 17 sont possibles :

- action="check" (*boolean feature*) : vérifie qu'une regexp est reconnue sur l'élément en entrée.
- action="subsequence" (*string feature*) : vérifie qu'une regexp est reconnue sur l'élément en entrée et renvoie la sous-chaine reconnue.
- action="token" (*string feature*) : vérifie qu'une regexp est reconnue sur l'élément en entrée et renvoie l'élément s'il est reconnu.

```
<regexp name="only-first-upper" action="check">^[A-Z][^A-Z]*$</regexp>
```

FIGURE 15 – exemple de la feature *matcher* "check".

```
<regexp name="after-hyphen" action="subsequence">-.+$</regexp>
```

FIGURE 16 – exemple de la feature *matcher* "subsequence".

```
<regexp name="ends-with-isme" action="token">isme$</regexp>
```

FIGURE 17 – exemple de la feature *matcher* "token".

6.1.7 Les features *rule*

La *sequence feature rule* permet d'intégrer des règles en tant que features. Les arguments d'une feature rule sont systématiquement des *boolean features* ont tous un champ "card" qui indique la cardinalité de la feature. Les différentes valeurs pour "card" sont :

- ? : 0 ou 1 fois
- * : 0 ou un nombre illimité de fois
- + : 1 ou un nombre illimité de fois
- entier : exactement [entier] fois
- "min,max" : au moins min fois et au maximum max fois.

Un argument spécifique des features *rule* est "orrule" qui permet de reconnaître une règle parmi plusieurs au choix. Un exemple de feature *rule* est illustré dans la figure 18.

```
<rule name="amount">
  <list action="some">
    <regexp action="check" entry="word">^[0-9]+$</regexp>
    <regexp action="check" entry="lower">^(une?|deux|trois|quatre|cinq|six|sept|huit|neuf|dix|
onze|douze|treize|quatorze|quinze|seize|dix-sept|dix-huit|dix-neuf|vingt|trente|quarante|cinquante|soixante
|soixante-dix|quatre-vingt|quatre-vingt-dix|cents?|mille|millions?|milliards?)$</regexp>
  </list>
  <orrule>
    <regexp action="check" entry="chunking" card="+>-NP$</regexp>
    <regexp action="check" entry="chunking" card="+>-PP$</regexp>
  </orrule>
</rule>
```

FIGURE 18 – exemple de la feature *rule*.

6.1.8 Les features *string*

Les *token features string* définissent des opérations de base sur les chaînes de caractères. Les actions suivantes sont définies :

- equal (*boolean feature*) : vérifie l'égalité de la chaîne en entrée par rapport à la chaîne en argument. Définit les options suivantes :
 - casing="(sensitive|s|insensitive|i)" : définit la sensibilité à la casse de la comparaison (défaut : "sensitive").

```
<string action="equal" casing="sensitive">O</string>
```

FIGURE 19 – exemple de la feature *string* "equal".

6.1.9 Les features *trigger*

La *string feature trigger* permet de définir un déclencheur avant l'évaluation d'une autre feature. Elle a deux fils : le *trigger* qui est la propriété booléenne à vérifier avant l'évaluation du second fils, qui peut être n'importe quelle *feature* de token. Un exemple est illustré dans la figure 20.

```

<trigger name="Substitute-Numbers-Triggered">
  <regexp action="check">[0-9]</regexp>
  <binary action="substitute">
    <pattern>[0-9]+</pattern>
    <replace>0</replace>
  </binary>
</trigger>

```

FIGURE 20 – exemple de la feature *trigger*.

6.2 Pour le module tagger

Le fichier de configuration du module tagger est appelé le fichier de configuration maître. Il permet de définir une séquence de traitements (modules) ainsi que des options globales aux différents modules qui seront lancés les uns après les autres.

Le fichier maître est un fichier XML de type de document “master”. Il a deux parties : une “pipeline” qui est une séquences de modules et une “options” qui permet de définir les options globales. Un exemple de pipeline est illustré dans la figure 21.

```

<? xml version="1.0" encoding="UTF-8" ?>
<master>
  <pipeline>
    <segmentation tokeniser="fr" />
    <enrich informations="pos.xml" mode="label" />
    <annotate model="models/POS" field="POS" />
    <clean to-keep="word,POS" />
    <enrich informations="NER.xml" mode="label" />
    <annotate model="models/NER" field="NER" />
    <clean to-keep="word,POS,NER" />
  </pipeline>

  <options>
    <encoding input-encoding="utf-8" output-encoding="utf-8" />
    <log log_level="INFO" />
    <export format="html" pos="POS" ner="NER" lang="fr" lang_style="default.css" />
  </options>
</master>

```

FIGURE 21 – Spécification d’un pipeline de SEM. Les pipelines permettent de définir une séquence de traitements ainsi que certaines options globales.

Références

- [Abeillé et al.2003] Abeillé, A., Clément, L., and Toussanel, F. (2003). Building a treebank for french. In Abeillé, A., editor, *Treebanks*. Kluwer, Dordrecht.
- [Clément et al.2004] Clément, L., Sagot, B., and Lang, B. (2004). Morphology based automatic acquisition of large-coverage lexica. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC 2004*. European Language Resources Association.
- [Crabbé and Candito2008] Crabbé, B. and Candito, M. H. (2008). Expériences d'analyse syntaxique statistique du français. In *Actes de TALN'08*.
- [Dupont2017] Dupont, Y. (2017). Exploration de traits pour la reconnaissance d'entités nommées du Français par apprentissage automatique. In *24e Conférence sur le Traitement Automatique des Langues Naturelles (TALN)*, page 42.
- [Finkel et al.2004] Finkel, J., Dingare, S., Nguyen, H., Nissim, M., Manning, C., and Sinclair, G. (2004). Exploiting context for biomedical entity recognition : From syntax to the web. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, pages 88–91. Association for Computational Linguistics.
- [Lavergne et al.2010] Lavergne, T., Cappé, O., and Yvon, F. (2010). Practical very large scale CRFs. In *Proceedings the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 504–513. Association for Computational Linguistics.
- [Sagot et al.2012] Sagot, B., Richard, M., and Stern, R. (2012). Annotation référentielle du corpus arboré de paris 7 en entités nommées. In *Actes de la 19e conférence sur le Traitement Automatique des Langues Naturelles*, pages 535–542, Grenoble, France. Association pour le Traitement Automatique des Langues.
- [Tellier et al.2012a] Tellier, I., Duchier, D., Eshkol, I., Courmet, A., and Martinet, M. (2012a). Apprentissage automatique d'un chunker pour le français. In *Actes de TALN'12, papier court (poster)*.
- [Tellier et al.2012b] Tellier, I., Dupont, Y., and Courmet, A. (2012b). Un segmenteur-étiqueteur et un chunker pour le français. In *Actes de TALN'12, session démo*.