Segmenteur Étiqueteur Markovien (SEM)

Table des matières

1 Préface

1.1 Présentation de SEM

Segmenteur Étiqueteur Markovien (SEM) [10] 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 [3]. 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 reconnaitra 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 pas modifier le contenu de ce dossier car cela pourrait causer des problèmes si l'on souhaite le mettre à jour.

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 [7] 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.

3 Corpus, annotations et ressources linguistiques

3.1 French Treebank (FTB)

SEM a été appris automatiquement sur le French Treebank (FTB) [1].

3.2 Jeu d'annotation PoS

L'annotation PoS se base sur le jeu d'étiquettes défini par [3] :

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

3.3 Annotation en chunks

Le chunking utilise les annotations définies dans [11] :

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 [8] :

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) [2] 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 en 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

4.3 fichiers SEM

SEM peut utiliser deux formats de fichiers en interne : un format XML et un json. Les deux donnent les mêmes informations et représentent au plus près le type Document utilisé en interne.

Dans les fichiers SEM, nous trouvons diverses informations, comme le nom du document, son contenu et des métadonnées. D'autres informations peuvent également être présentes, comme les segmentations (en tokens, phrases et paragraphe) et les annotations (en parties-du-discours, chunks, et entités nommées).

4.3.1 Exemples

Un exemple de fichier XML-SEM est donné dans la figure 1. Le même fichier au format JSON présentant les mêmes informations, il ne sera pas donné.

5 Utilisation

SEM dispose de module 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\_opti
```

```xml
<? xml version="1.0" encoding="UTF-8" ?>
<document name="exemple.txt">
  <metadata encoding="utf-8" />
  <content>SEM est un programme bien documenté.

SEM est écrit par Yoann Dupont.</content>
  <segmentation name="tokens">
    <s s="0" l="3" />
    <s s="4" l="3" />
    <s s="8" l="2" />
    <s s="11" l="9" />
    <s s="21" l="4" />
    <s s="26" l="9" />
    <s s="35" l="1" />
    <s s="38" l="3" />
    <s s="42" l="3" />
    <s s="46" l="2" />
    <s s="49" l="9" />
    <s s="59" l="5" />
    <s s="65" l="3" />
    <s s="69" l="5" />
    <s s="75" l="6" />
    <s s="81" l="1" />
  </segmentation>
  <segmentation name="sentences" reference="tokens">
    <s s="0" l="7" />
    <s s="7" l="9" />
  </segmentation>
  <segmentation name="paragraphs" reference="sentences">
    <s s="0" l="1" />
    <s s="1" l="1" />
  </segmentation>
</document>
```

FIGURE 1 – exemple de fichier XML-SEM.

Les différents modules seront détaillés un par un.

5.1 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).

5.2 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_info 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 tagging

description

arguments

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 (sérialise) 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 (désérialise) 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 echaînement 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 séquencage 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'entrainer un nouveau modèle.

arguments
resources (facultatif)
le dossier de resources 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 2. 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'entrainer 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="chaine" : le nom de la *feature*. Obligatoire pour les *features* racines.

— action="chaine" : 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="chaine" : 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 chaine 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 2 – 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 3, 4, 5 et 6. 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 chaine en entrée en minuscules ;
- substring (*string feature*) : fournit une sous-chaine de la chaine donnée en entrée. Définit les options suivantes :
  - from="[entier]" : l'indice de début de la sous-chaine (défaut : 0)
  - to="[entier]" : l'indice de fin de la sous-chaine. Si 0 est donné, "to" sera la fin de la chaine (défaut : 0)

Le deuxième type de *token feature* est la *unary*. Elle prend un unique argument. Un

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

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

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

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

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

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

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

FIGURE 6 – exemple de la feature *nullary* "substring".

exemple est illustré sur la figure 7. Les différentes actions sont :

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

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

FIGURE 7 – 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 8. Les différentes actions sont :

— substitute : substitute une chaine par une autre. Le premier argument est *pattern* et le second est *replace*.

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

FIGURE 8 – exemple de la feature *binary* "substitue".

Le quatrième type de *token feature* est la *n-ary*. Elle prend un nombre arbitraire d'argument. Un exemple est illustré sur la figure 9. Les différentes actions sont :

— sequencer : effectue une séquences 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*.

6.1.2 Les features *boolean*

Les *boolean features boolean* définissent des expressions booléennes. Un exemple de feature *boolean* est donné dans la figure 10. 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.

```
<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 9 – exemple de la feature *n-ary* "sequencer". L'exemple ici implémente les classes de caractères (appelées *word shapes* dans [6]).

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

FIGURE 10 – 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 11 et 12.

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

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

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

FIGURE 12 – 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 [5]. 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 13 – exemple de la feature *directory*.

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

FIGURE 14 – exemple de la feature *directory* "fill".

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 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 15 – 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 16, 17 et 18 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 16 – exemple de la feature *matcher* "check".

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

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

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

FIGURE 18 – 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 reconnaitre une règle parmi plusieurs au choix. Un exemple de feature rule est illustré dans la figure 19.

```
<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 19 – exemple de la feature *rule*.

6.1.8 Les features *string*

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

— equal (*boolean feature*) : vérifie l'égalité de la chaine en entrée par rapport à la chaine 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">0</string>
```

FIGURE 20 – 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 21.

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

FIGURE 21 – 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 22.

7 Réentraîner SEM

Les modèles proposés par SEM ne sont pas adaptés à tous les cas d'usage. Afin d'adapter SEM, ce dernier propose d'entraîner de nouveaux modèles qu'il pourra utiliser par la suite.

Dans la suite de la section, nous présenterons la procédure pas à pas pour entraîner de nouveaux modèles dans SEM. Nous nous baserons pour cela sur un cas d'usage très simple : reconnaître les personnes et les logiciels. Pour cela nous traiterons deux cas : un premier avec un fichier déjà annoté de type BRAT et un second où le fichier doit être annoté manuellement.

```
<? 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 22 – Spécification d'un pipeline de SEM. Les pipelines permettent de définir une séquence de traitements ainsi que certaines options globales.

7.1 Réentraîner SEM depuis des fichiers déjà annotés

7.1.1 Lancer la GUI de SEM

Pour lancer l'interface graphique de SEM, lancer dans un terminal :

```
python -m sem gui
```

L'interface graphique de SEM doit ressembler à la figure 23.

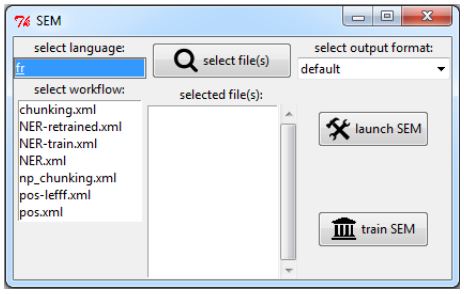

FIGURE 23 – La GUI permettant de réentraîner SEM

7.1.2 Sélectionner les données et leur prétraitement

Une fois lancée, il faut commencer par choisir les éléments suivants :
— les données qui serviront à l'entraînement ;

— la chaîne de prétraitement pour enrichir les données.

Pour choisir les données qui serviront à l'entraînement, voir les figures 24 et 25. Les différentes étapes sont :

1. cliquer sur le bouton « select file(s) » ;
2. sélectionner le(s) fichiers annotés ;
3. cliquer sur « ouvrir ».

Une fois sélectionnés, les fichiers seront listés comme indiqué dans la figure 25. Il n'est pas nécessaire que les fichiers annotés soient du même format. Ces derniers peuvent avoir n'importe quel format de fichier supporté par SEM. À l'heure où cette documentation a été écrite, les fichiers supportés sont :

— XML SEM, le format XML interne de SEM ;
— json SEM, le format json interne de SEM ;
— BRAT [9] ;
— GATE [4] ;

Une fois les fichiers annotés choisis, il faut faire le choix de la chaîne de traitement pour prétraiter les documents. SEM propose un exemple d'une telle chaîne pour réentraîner la tâche de reconnaissance des entités nommées dans la figure 24, à savoir « NER-train.xml ».

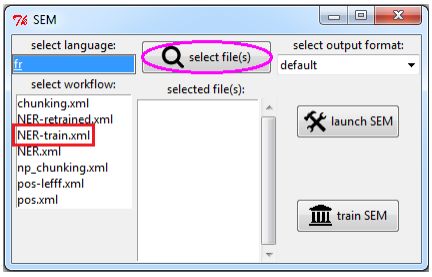

FIGURE 24 – Entouré en violet : le bouton pour sélectionner les documents à utiliser pour l'entraînement. Encadré en rouge : la chaîne de traitement à utiliser pour entraîner un nouveau modèle.

7.1.3 Lancer l'entraînement

Une fois les fichiers d'entraînement et la chaîne de traitements sélectionnés, cliquer sur le bouton « train SEM » comme illustré dans la figure 25. Cela ouvrira la fenêtre qui vous permettra de paramétrer le CRF pour réapprendre SEM, illustrée dans la figure 25. Dans cette fenêtre, vous avez notamment la possibilité de choisir un fichier pattern pour Wapiti. Si vous n'en choisissez pas, il sera automatiquement généré depuis les traits générés par

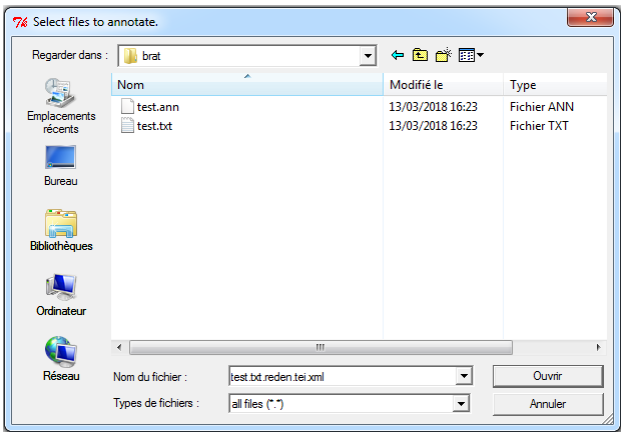

FIGURE 25 – Des exemples de fichiers annotés au format BRAT. Ne choisir que les fichiers ".ann" ou les fichiers ".txt" pour l'entraînement.

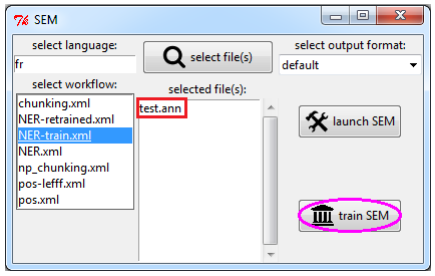

FIGURE 26 – encadré en rouge : les documents utilisés pour l'entraînement. Entouré en violet : le bouton pour entraîner SEM.

la chaîne de traitement. Lorsque tous les paramètres sont configurés, cliquer sur le bouton « train » pour lancer l'entraînement d'un nouveau modèle de SEM. Lorsque l'entraînement sera terminé, SEM indiquera où vous aurez la possibilité de récupérer les fichiers, comme illustré dans la figure 28. Pour pouvoir utiliser le modèle, il faut copier le fichier « model.txt » dans le dossier « ${SEM_DATA}/resources/models/fr/NER ».

7.2 Réentraîner SEM depuis des fichiers non annotés

Dans le cas où seulement des fichiers non annotés sont disponibles, il est nécessaire de les annoter. SEM propose d'effectuer l'annotation des fichiers en texte brut.

7.2.1 Lancer la GUI de SEM pour l'annotation manuelle

Pour lancer l'interface graphique d'annotation manuelle de SEM, lancer dans un terminal :

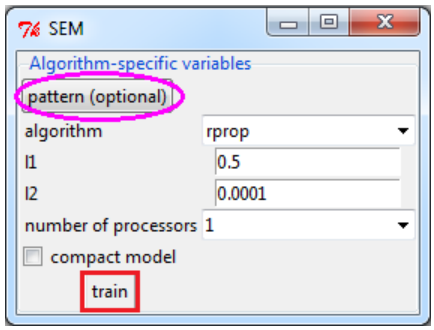

FIGURE 27 – Entouré en violet : le bouton pour sélectionner un modèle particulier. Encadré en rouge : le bouton pour lancer l'entraînement.

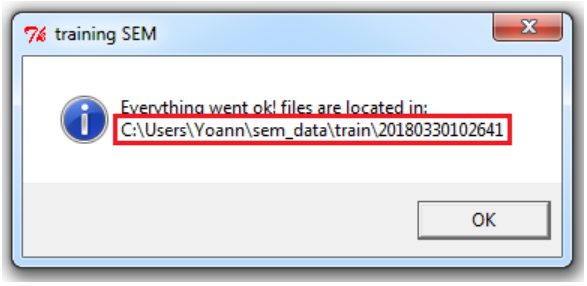

FIGURE 28 – Encadré en rouge : le chemin où trouver le fichier enregistré

```
python -m sem annotation_gui
```

L'interface illustrée dans la figure 29 s'affichera alors.

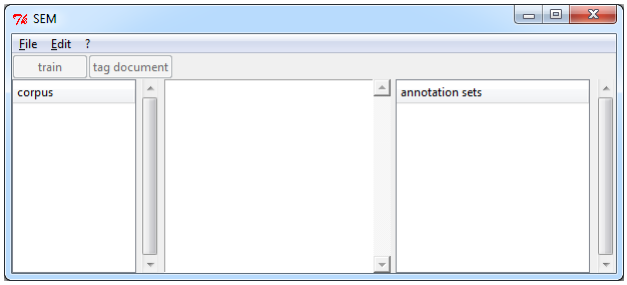

FIGURE 29 – L'interface d'annotation manuelle de SEM au démarrage.

7.2.2 Annoter manuellement avec la GUI de SEM

Dans le but de simplifier les traitements, SEM ne permet pas de modifier le contenu des fichiers via l'interface ou de modifier le jeu d'étiquettes. Ainsi, pour annoter un document il faut :

— le charger depuis un fichier ;
— charger le jeu d'étiquettes depuis un fichier ;

SEM génère des raccourcis clavier depuis le jeu d'étiquettes qui sera chargé. Un fichier contenant un jeu d'étiquettes suit un format "une étiquette par ligne" comme l'exemple suivant :

```
etiquette1

# un commentaire
etiquette2.sous-etiquette1
etiquette2.sous-etiquette2 # un autre commentaire

etiquette3
```

SEM gère les étiquettes hiérarchiques et considère le caractère "." comme le séparateur de niveaux entre étiquettes. Les lignes vides sont ignorées et le caractère "#" permet d'écrire des commentaires qui seront ignorés. Dans notre cas, nous souhaitons gérer les types *logiciel* et *personne*. Le fichier aura alors le contenu suivant :

```
logiciel
personne
```

Les figures 30 et 31 illustrent comment charger un fichier et un jeu d'étiquettes.

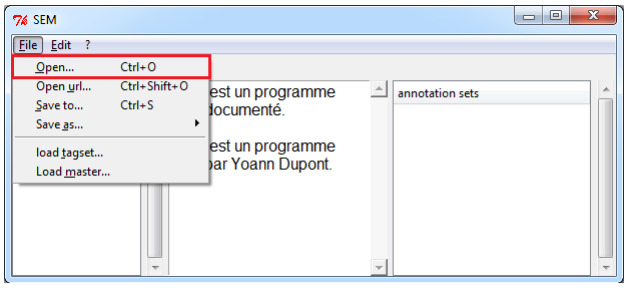

FIGURE 30 – Encadré en rouge : l'élément du menu pour charger un fichier.

Les figures 32, 33 et 34 illustrent comment annoter manuellement un corpus puis réentraîner un modèle utilisable par SEM sur ce corpus.

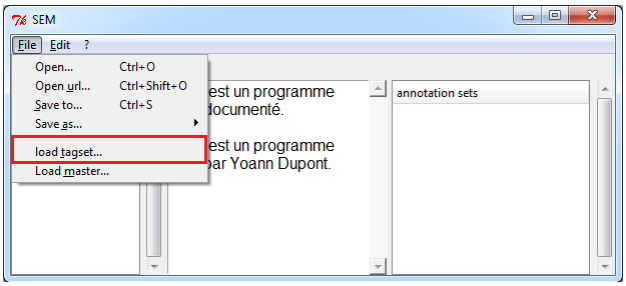

FIGURE 31 – Entouré en rouge : le bouton pour charger un jeu d'étiquettes.

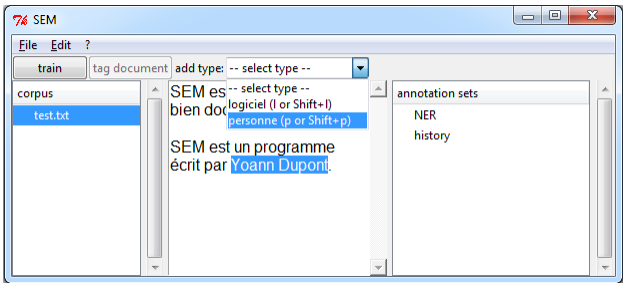

FIGURE 32 – Pour annoter un élément : on le sélectionne dans le texte et on le choisit dans la liste déroulante (indique également les raccourcis clavier).

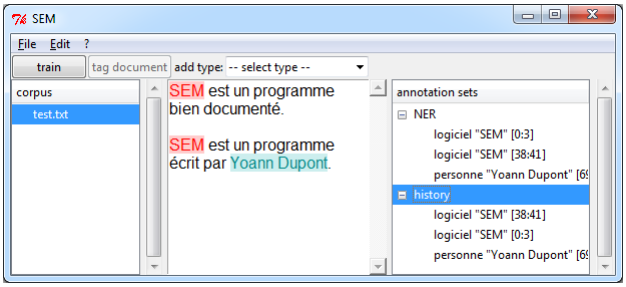

FIGURE 33 – Pour annoter toutes les occurrences du document : utiliser le raccourci $< Shift + \_ >$ où "_" est le raccourci clavier pour l'étiquette. Si l'on se réfère à la figure 32, $< Shift + l >$ permet d'annoter toutes les occurrences de "SEM" en tant que "logiciel".

7.3 Utiliser le nouveau modèle

Pour annoter des documents avec le nouveau modèle, il faut alors sélectionner la chaîne de prétraitement « NER-retrained.xml » puis cliquer sur le bouton « launch SEM » comme illustré dans la figure 35. Une fois le traitement effectué, SEM indiquera où trouver les fichiers annotés comme illustré dans la figure 36.

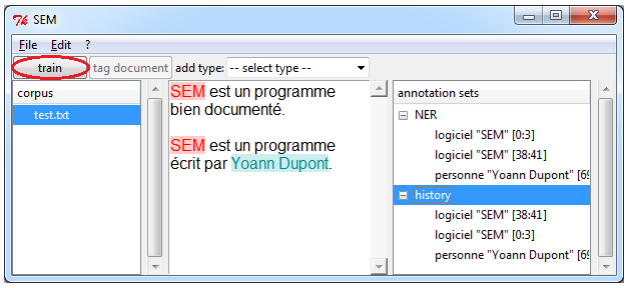

FIGURE 34 – Entouré en rouge : le bouton pour entraîner SEM avec le corpus courant selon les annotations courantes. L'interface est identique à celle décrite dans la section 7.1.3.

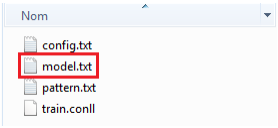

FIGURE 35 – Encadré en rouge : le fichier modèle à copier dans le dossier « ${SEM_DATA}/resources/models/fr/NER »

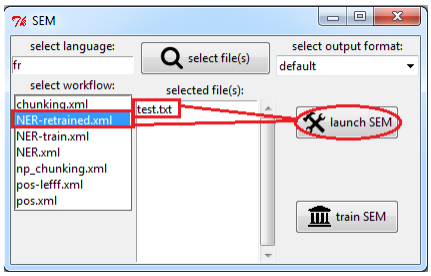

FIGURE 36 – Encadrés en rouges : la chaîne de traitement à utiliser pour annoter avec le nouveau modèle et les fichiers à annoter. Entouré en rouge : le bouton pour lancer l'annotation avec SEM.

Références

[1] A. ABEILLÉ, L. CLÉMENT et F. TOUSSENEL. "Building a treebank for French". In : *Treebanks*. Sous la dir. d'A. ABEILLÉ. Dordrecht : Kluwer, 2003.

[2] Lionel CLÉMENT, Benoît SAGOT et Bernard LANG. "Morphology Based Automatic Acquisition of Large-coverage Lexica". In : *Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC 2004*. Lisbon, Portugal : European Language Resources Association, mai 2004.

[3] B. Crabbé et M. -H. Candito. “Expériences d’analyse syntaxique statistique du français”. In : *Actes de TALN’08*. 2008.

[4] Hamish Cunningham et al. “GATE : an architecture for development of robust HLT applications”. In : *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, p. 168-175.

[5] Yoann Dupont. “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)*. 2017, p. 42.

[6] Jenny Finkel et al. “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*. Association for Computational Linguistics. 2004, p. 88-91.

[7] Thomas Lavergne, Olivier Cappé et François Yvon. “Practical Very Large Scale CRFs”. In : *Proceedings the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*. Uppsala, Sweden : Association for Computational Linguistics, juil. 2010, p. 504-513. url : http://www.aclweb.org/anthology/P10-1052.

[8] Benoît Sagot, Marion Richard et Rosa Stern. “Annotation référentielle du Corpus Arboré de Paris 7 en entités nommées”. In : *Actes de la 19e conférence sur le Traitement Automatique des Langues Naturelles*. Grenoble, France : Association pour le Traitement Automatique des Langues, juin 2012, p. 535-542. url : http://www.atala.org/taln_archives/TALN/TALN-2012/taln-2012-court-026.

[9] Pontus Stenetorp et al. “BRAT : a web-based tool for NLP-assisted text annotation”. In : *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics. 2012, p. 102-107.

[10] I. Tellier, Y. Dupont et A. Courmet. “Un segmenteur-étiqueteur et un chunker pour le français”. In : *Actes de TALN’12, session démo*. 2012.

[11] I. Tellier et al. “Apprentissage automatique d’un chunker pour le français”. In : *Actes de TALN’12, papier court (poster)*. 2012.