Programmation de Modèles Linguistiques (I)

(L5SOPROG L3 Sciences du Langage)

2024-2025

TD5 Évaluer des outils de reconnaissance d'entités nommées (REN)

Caroline Koudoro-Parfait, Sorbonne Université.

Objectifs

- Utiliser des outils de Reconnaissance d'entités nommées
- Annoter manuellement les entités nommées dans un texte
- Tokenizer un texte
- Annoter automatiquement les entités nommées dans un texte
- Évaluer l'annotation automatique : Calcul de similarité

Un outil de reconnaissance d'entités nommées (REN) sert à reconnaître automatiquement les noms de personnes, de lieux, d'organisations, de monnaies, d'évènements etc. dans des textes.

La majorités des outils de REN sont entraînés sur des corpus de langue moderne, souvent issus d'articles de journaux et nativement numériques.

Aujourd'hui, nous allons d'abord annoter manuellement des textes romanesques du 19ème siècle, puis les annoter de manière automatique en utilisant la bibliothèque *spaCy*. Nous nous demanderons si ces outils automatiques sont robustes à la variabilité car notre corpus de textes n'est pas du même genre (littéraire) et de la même époque que les corpus utilisés pour leur entraînement.

Exercice 1 : Annoter manuellement un texte littéraire du 19ème s.

L'annotation des entités nommées est utile pour déterminer qui sont les protagonistes, où se déroule l'action . . . Par ailleurs des chercheurs ont montré que la majorité des requêtes dans les moteurs de recherches contiennent une entité nommée.

Il est donc très utile de repérer les EN pour pouvoir indexer un texte et faire des recherches via un moteur de recherche.

La REN permet une indexation qui fait que nous n'avons pas à parcourir toute une base de données afin de trouver des documents pertinents pour chaque requête (et donc aller "lire" tous les documents à chaque requête), l'index est utilisé pour accéder directement à un ensemble de documents qui contiennent les mots de la requête.

En Python, un index peut être représenté par un dictionnaire avec les mots comme clé, puis en valeur un ensemble (ou une liste) de documents qui contient ce mot.

Etape 1: Préparer son document d'annotation

- Tokeniser

- stocker dans un document .csv

Pour annoter les textes manuellement, nous allons stocker les tokens dans des tableaux au format .csv qui comprendront une colonne "Token" et une colonne pour chaque type d'entité nommée qui existe dans le schéma d'annotation de *spaCy* pour le français : LOC (lieux), PER (personne), ORG (organisation) et MISC (divers). Pour faciliter le stockage, un token par ligne, dans le tableau csv nous remplacerons les sauts de lignes, les doubles espaces et les tabulations par un "$", en utilisant la bibliothèque *re*.

```
import glob
import re
for path_file in glob.glob(path_corpus):
    liste_tok=[]#initialise une liste pour stocker les tokens
    print(path_file)
    filename=path_file.XXX#quelle est la solution pour
    # récupérer le nom du fichier ?
    texte=lire_fichier(path_file)
    for ponctuation in ["\n","  ","\t"]:#On peut en ajouter d'autres
        texte=re.sub(ponctuation,"\$",texte)
```

Maintenant on va utiliser le tokenizer de *spaCy* plus efficace que le simple split() : `https://spacy.io/usage/linguistic-features#tokenization`

Installation de spaCy sur votre machine

- Lire attentivement le site web `https://spacy.io/usage`
- Écrire une note d'une quinzaine de ligne pour résumer ce que permettent les différentes fonctionnalités de la bibliothèque *spaCy* et quelles sont les données d'entraînement pour la REN.

. . . avez-vous bien installé le modèle de langue adéquat ?

```
import spacy

nlp = spacy.load("fr_core_news_lg")
doc = nlp(texte)
for token in doc:
    print(token.text)
```

On prépare le tableau csv pour stocker les tokens

```
import csv
with open(f'../corpus-REN_2023-2024/NOAILLES/{filename}_annot.csv',
'w', newline='') as csvfile:
    spamwriter = csv.writer(csvfile, delimiter=';',quotechar='\t',
quoting=csv.QUOTE_MINIMAL)
    spamwriter.writerow(["Token","LOC","PER","ORG","MISC"])
```

On stocke les tokens dans "liste_tok" pour qu'ensuite celle-ci apparaisse bien dans la colonne "Token" :

```
    if token.text !="\$":
        liste_tok.append(token.text)
        spamwriter.writerow([token.text])

    print(filename," : ",len(liste_tok))
```

...à vous de placer ces différents morceaux du programme aux bons endroits.

Etape 2: c'est parti, annotez !

L'annotation des entités nommées dans un texte nécessite que chaque annotateur.ice se fit à un référentiel commun afin que les annotations ne soient pas chaotiques. Il est très important de tenir compte du fait que chaque annotateur.ice ne dispose pas du même savoir culturel ou scientifique à priori et que les annotateurs.ices ne sont pas tous nativement des locuteur.ice.s de la langue à annoter. Ainsi le fait d'utiliser un guide d'annotation commun et de discuter entre pairs sont deux pratiques fondamentales de l'annotation manuelle des textes.

Les étudiant.e.s se répartissent en groupes de 4. Chaque groupe se voit attribuer un des textes figurant dans le répertoire : corpus-REN_2023-2024.zip. Ce répertoire comprends trois textes de trois auteurs : "Belle rivière" de G.Aimard, "L'éducation sentimentale" de G. Flaubert et "La nouvelle espérance" de A. de Noailles.

Le guide d'annotation est disponible sur moodle : *Guide_Named_Entities.pdf*. Il s'agit d'un guide qui définit un nouveau schéma d'annotation pour les textes littéraires. Pour notre annotation, nous tiendrons compte uniquement des grandes catégories et pas des attributs.

Chaque étudiant.e annote individuellement les 5000 premiers tokens en mettant une croix "X" (un X majuscule) dans la colonne PER si c'est une personne, LOC s'il s'agit d'un lieu etc. en se référant au guide d'annotation qui donne des exemples. Au bout de 30 minutes chaque groupe se réunit pour dialoguer autour des difficultés de l'annotation et des questions que chacun.e a pu se poser, le but de cette étape est d'harmoniser les annotations de chacun. Lisez attentivement le guide d'annotation, posez des questions, ouvrez la discussion sur ce qui vous semble plus ou moins pertinent. Félicitations, vous avez constituez une **vérité de terrain** !

Á l'issue de ce premier exercice vous devez remettre le fichier .csv avec les annotations et un compte rendu retraçant les difficultés, les questions qui vous ont traversées pendant l'annotation.

Exercice 2 : Annoter les entités nommées automatiquement

displaCy : Visualisation de la REN `https://demos.explosion.ai/displacy-ent`

Dans un premier temps, vous pouvez tester le visualiseur de spaCy, displaCy, qui permet de repérer facilement les entités dans un texte.

```
import spacy
# Importer le visualiseur displaCy de spaCy
from spacy import displacy
nlp = spacy.load("fr_core_news_lg")
path_txt="le chemin relatif vers votre fichier"
```

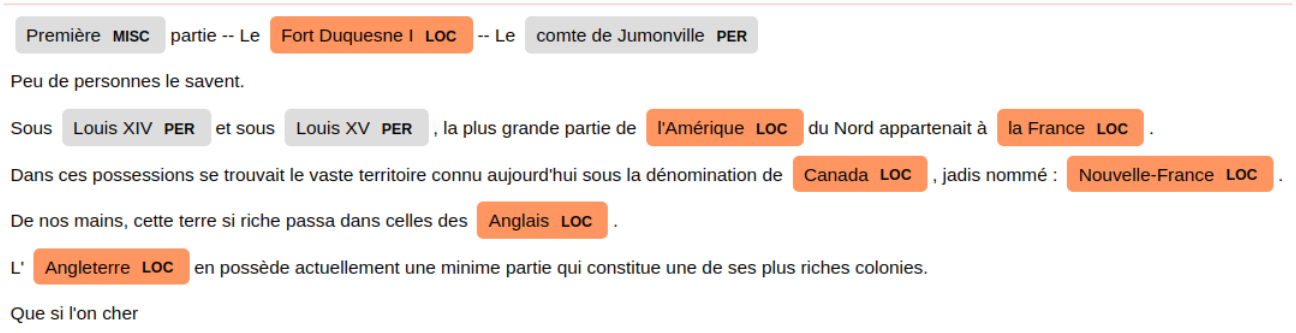

Figure 1: Exemple de sortie de DISPLACY

```
text=read_text(path_txt)
doc =  nlp(text[:500])#On analyse les 500 premiers caractères
#Afficher le visualiseur displaCy
displacy.serve(doc,style="ent")
```

Maintenant que l'on peut identifier des entités nommées, il serait bon de pouvoir les stocker. Pour ce faire vous adapterez le programme suivant : `https://spacy.io/usage/linguistic-features#named-entities`

```
import spacy

nlp = spacy.load(model_langue)
doc = nlp(text)

for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

Le programme doit permettre de :

- récupérer les entités nommées dans le texte qui vous a été assigné et leur label (PER : personne ; LOC : localisation ; MISC : Divers ; ....) en utilisant *spaCy*
- sauvegarder un dictionnaire en json, en utuilisant le code du TD4.
- d' utiliser deux modèles de langue de *spaCy*. Par exemple :
  - *fr_core_news_sm*
  - *fr_core_news_lg*
- indiquer le temps de travail de l'outil en l'affichant

On attend un fichier de sortie pour chaque modèle de langue, structuré ainsi :

```
{
"entite_0": {
  "Entite": "Paris",
  "Label": "LOC"
```

```
  }
  "entite_1": {
    "Entite": "Norine",
    "Label": "PER"
  }
  }, ...
```

L'opération peut-être longue, appliquez le programme sur au moins une partie de votre texte pour générer une sortie. Les 5000 premiers tokens par exemple.

Rédiger de manière synthétique quelques observations sur les sorties de reconnaissance d'entités nommées obtenues avec les deux modèles de langue française de votre choix.

Exercice 3 : Évaluation de la REN avec *spaCy*

Dans cet exercice vous devez disposer de votre vérité de terrain (exercice 1) et des annotations effectuées avec les deux modèles de *spaCy* (exercice 2).

Nous allons comparer les deux sorties de REN de *spaCy* avec la vérité de terrain chacune leur tour pour évaluer la pertinence de *spaCy* par rapport à une annotation manuelle. Pour ce faire nous allons utiliser deux méthodes :

- Diagramme de Venn
- Distance cosinus

Dans un premier temps il faudrait pouvoir obtenir les données dans une même structure de données par exemple en utilisant la bibliothèque *Pandas*.

```
#Récupérer les données dans le csv avec Pandas ...
import pandas as pd

df = pd.read_csv('chemin_relatif.csv',sep=";",
on_bad_lines='skip')
display(df[:100]) #permet d'afficher les 100 premières lignes

#...pour les mettre au format adéquats : par exemple un set.
data_VT=set()
```

Pour récupérer des valeurs dans des colonnes on peut utiliser la fonction .query(). Par exemple pour récupérer les LOC.

```
data_loc=df.query("LOC=='X'")
liste_loc=list(data_loc.Token)
data_VT.update(liste_loc)
print(data_VT)
```

À vous de compléter le programme pour créer un set qui comprendra toutes les entités pour chaque type de label : PER, LOC, MISC, ORG.

Ici, nous récupérons les résultats pour les versions lg et sm de nos annotations avec *spaCy* dans un set aussi

```
liste_mod=[]#pour stocker le nom du modèle
liste_entite=[]
path_data= "chemin_relatif.json"
```

```
for path in glob.glob(path_data):
#on lit fichier json et non un texte brut
    data=read_json(path)

    mod=path.split("/")[3]
    mod=mod.split("_")[-1]
    print(mod)
#on lit le dictionnaire
    for k, ssdic in data.items():
#         print(ssdic)
        for meta, val in ssdic.items():
#             print(meta)
            if meta=="entite":
#                 print(val)
                liste_entite.append(val)
```

Diagramme de Venn

La bibliothèque *Matplotlib* sera utile pour produire la visualisation du diagramme.

```
import matplotlib.pyplot as plt
from matplotlib_venn import venn2, venn2_circles
# taille des labels
font2 = {'size': 25}
    plt.rc('font', **font2) # sets the default font
    # changer la couleur du texte par défaut
    plt.rcParams['text.color'] = 'black'
    venn2([data_VT, set(liste_entite)],set_labels = ('EN Ref',
    'EN \%s'\%mod),set_colors=("darkgrey", "darkblue"),alpha=0.5)

    venn2_circles(subsets=(data_VT, set(liste_entite)),
linestyle="dotted", linewidth=1)
    print(f"Analyse de la version {mod}")
    plt.show()
```

- Que disent les résultats des diagrammes de Venn obtenus pour l'évaluation des modèles de REN de *spaCy* ?
- A votre avis pourquoi ?

Distance cosinus

`https://scikit-learn.org/stable/modules/generated/sklearn.metrics.DistanceMetric.html#sklearn.metrics.DistanceMetric` La similarité cosinus est une métrique souvent utilisée en Traitement automatique des Langues pour s'assurer de la qualité, évaluer les résultats d'un outil par rapport à une vérité de terrain.

Nous utiliserons un outil pour la vectorisation déjà utilisé dans le bonus du TD4 et d'autres fonctions de la bibliothèque *scikit learn* pour calculer la distance cosinus. Plus la valeur est proche de 1 plus le résultat montre que les textes comparés sont différents.

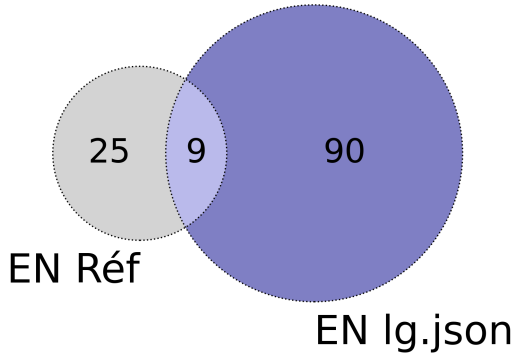

Figure 2: Type de sortie attendue

```
import sklearn
# Pour les anciennes versions
#from sklearn.neighbors import DistanceMetric
from sklearn.metrics import DistanceMetric
from sklearn.feature_extraction.text import CountVectorizer

V = CountVectorizer()
X = V.fit_transform([str(data_VT),str(liste_entite)]).toarray()
# Distance avec cosinus
distance_tab1=sklearn.metrics.pairwise.cosine_distances(X)
print(f"{La distance cosinus entre la sortie pour le modèle
{mod} et la VT est {distance_tab1[0][1]})
```

Votre travail consiste à faire varier les paramètres de CountVectorizer :

- Tester en vectorisant avec des N-grammes de caractères : analyzer="char"
- Faire varier les valeurs min et max de N: ngram range=(min,max)
- Que disent les résultats obtenus pour l'évaluation des modèles de REN de *spaCy* ?
- A votre avis pourquoi ?

Devoir

- 1 document csv annoté
- des images des diagrammes de Venn
- quelques phrases de conclusion sur les résultats (qu'est-ce qui était attendu, qu'est-ce qui est inattendu ?)

  Vous déposerez sur Moodle une archive zip nommée NUMETU.zip (où NUMETU est votre numéro d'étudiant) et contenant :

- Votre code exporté au format Python .py (et pas ipynb)
- le PDF du document que vous avez produit

  Date limite : indiquée sur le Moodle !