Master
La Rochelle Université
GUY Adrien | GUITTON Elie |BELMOKHTAR Bilal

Information Extraction

TP1 - Named Entity Recognition Exercise (sans IA)

Input

En utilisant vos connaissances en programmation, codez un système NER qui reconnaît les noms de personnes (PERSON) et les lieux (LOCATION) à partir de l'ensemble de données fourni (europarl).

Output

Le système doit renvoyer, pour chaque fichier du dossier **europarl**, un autre fichier avec les entités détectées, au format IOB .

Organisation

Pour travailler efficacement en collaboration, nous avons établi un repository GitHub :
https://github.com/elieguitton/named-entity-recognition
Pour les différentes fonctions à coder, nous nous sommes tous attribué une partie du travail. Chacun était responsable d'une portion de l'algorithme.

Code

La première chose à faire pour notre système était de lister la totalité des fichiers à analyser en obtenant tous les chemins d'accès depuis le dossier europarl.

```
# dirname : le dossier racine pour lequel lister les chemins
# returns : la liste de chemin sous forme de String
def getListOfFiles(dirName)
```

```
# filename : nom du fichier à lire
# returns : le fichier encodé en utf-8
def readFile(filename)
```

La seconde étape était de découper chaque mot des fichiers en token. Pour cela, nous avons créé plusieurs fonctions pour retirer toutes les impuretés du texte. A la fin, nous avons donc des données que nous pouvons exploiter facilement.

```
# 1 - On enlève la ponctuation
def enlever_ponctuation(chaine):
    return re.sub(r"[^\w\s]", " ", chaine)

# 2 - On enlève les mots si ils sont dans la liste des mots vides
def enlever_mots_vides(chaine):
    return ' '.join([mot for mot in chaine.split() if mot not in
stopwords.words("french")])

# 3 - Tokenisation des mots, on les sépare en mots
def tokenisation(chaine):
    return chaine.split()

# On crée une fonction qui va appliquer les 3 fonctions précédentes
def preprocess(chaine):
    return tokenisation(enlever_mots_vides(enlever_ponctuation(chaine)))
```

Pour reconnaître les noms propres ainsi que les localisations, nous avons fait le choix d'utiliser les API suivantes :

```
LOCATIONS = "https://nominatim.openstreetmap.org"
NAMES = "https://api.genderize.io"
```

L'idée était donc d'interroger les deux API pour chaque token pour pouvoir assigner les labels avec le Named Entity Recognition.

```
# return True if the word is a name with more than 1000 occurences in the API
# mot : mot à tester
def est_nom(mot)
```

```
# return True if the word is a location stored in the api
# mot : mot à tester
def est_localisation(mot)
```

Pour gérer le cas des B-PER et I-PER, on se contente de vérifier le tag précédent pour savoir si mot actuel fait partie d'un nom propre composé. Ainsi, notre algorithme, dans son état actuel, ne peut pas détecter entièrement certains noms propres comme "La Rochelle". L'article 'La' n'étant pas un nom propre en lui-même.

```python
# Name Entity recognition (NER)
# Algorithm for a name entity recognition
# Input: a list of words
# Output: a list of tuples (word, tag)
def ner(words):
    tags = []
    for word in words:
        if word in locations:
            tags.append((word, " B-LOC"))
        elif word in persons:
            if len(tags) > 0 and tags[-1][1] == " I-PER":
                tags.append((word, " I-PER"))
            tags.append((word, "B-PER"))
        else:
            tags.append((word, "O"))
    return tags
```

Résultats

La complexité cyclomatique de notre algorithme reste assez grande. Nous ne sommes pas encore parvenus à terminer l'exécution de notre programme en moins d'une heure.
Le fichier généré sera "**ep-01-05-30-fr.iob**".

```
('2001', 'O')
('Communication', 'O')
('Présidente', 'O')
('Chers', 'O')
('collègues', 'O')
('24', 'O')
('mai', 'O')
('dernier', 'O')
('Santiago', ' B-LOC')
('Oleaga', 'O')
```

```
('également', 'O')
('rapport', 'O')
('M', 'O')
('Mayer', ' B-LOC')
('nom', 'O')
('commission', 'O')
('juridique', 'O')
('marché', 'O')
('intérieur', 'O')
```

TP2 - Named Entity Recognition Exercise (avec IA)

Input

1. En utilisant le système que vous avez créé la dernière fois, générez un fichier au format IOB pour les phrases du fichier fourni "ep-01-05-30-fr.txt" et évaluer les résultats à l'aide de Conlleval ou Nervaluate.

Nous avons créé un dossier **/test** où nous avons stocké notre fichier iob ainsi que le fichier iob de test fourni sur Moodle. Nous avons ensuite utilisé le repository de Conlleval pour récupérer les fichiers **conlleval_perl.py** et **conlleval.py**. De la même manière que précédemment, par faute de temps, nous allons seulement travailler sur le fichier "**ep-01-05-30-fr.iob**".
Nous avons réalisé une fonction **file_to_dict** permettant de convertir nos fichiers en dictionnaires. Cela va nous aider ensuite à comparer les deux fichiers, puis à écrire nos résultats dans "**comparaison.txt**".
Enfin, grâce à Conlleval, nous allons évaluer ces résultats :

```
evaluate(truth_values, predict_values)
```

Le résultat est le suivant :

```
processed 7218 tokens with 179 phrases; found: 279 phrases; correct: 96.
accuracy:  47.89%; (non-O)
accuracy:  96.09%; precision:  34.41%; recall:  53.63%; FB1:  41.92
             LOC: precision:  56.82%; recall:  67.57%; FB1:  61.73  88
             PER: precision:  24.08%; recall:  43.81%; FB1:  31.08  191

(34.40860215053764, 53.63128491620112, 41.92139737991267)
```

Le résultat ne nous surprend pas, nous pensons avoir une méthode de prétraitement différente de celle utilisée pour avoir le fichier déposé sur Moodle. De plus, sur ce fichier le texte est entièrement reconstitué alors que nous ne remettons pas les stop-words et la ponctuation dans le nôtre. Enfin sur certains cas, comme "La Rochelle", notre modèle va détecter "Rochelle" mais pas "La".

2. A l'aide de spaCy ou BERT, créez un système NER et comparez les performances avec votre système basé sur des règles.

Grâce à Spacy et à sa fonction **train**, nous avons pu entraîner notre modèle. Nous avons laissé tourner le système pour entraîner notre modèle. Nous n'avons pas eu besoin de Google Colab car nous avions assez de puissance sur l'une de nos machines. Nous arrivons à un score de **0.75** :

```
ℹ Pipeline: ['tok2vec', 'ner']
ℹ Initial learn rate: 0.001
E    #       LOSS TOK2VEC  LOSS NER  ENTS_F  ENTS_P  ENTS_R  SCORE
---  ------  ------------  --------  ------  ------  ------  ------
  0       0          0.00     36.78    2.43    2.28    2.59    0.02
  0     200         71.45   2471.87   34.56   47.20   27.26    0.35
  0     400         98.31   1976.49   42.58   52.73   35.71    0.43
  0     600        252.94   2424.08   46.90   55.20   40.77    0.47
  0     800        394.03   2673.66   50.29   56.66   45.21    0.50
  0    1000        361.98   3088.79   55.44   62.10   50.07    0.55
  0    1200        559.93   3928.40   58.73   65.25   53.39    0.59
  0    1400        663.77   4203.49   62.34   68.26   57.36    0.62
  0    1600        770.81   4866.06   60.76   66.54   55.90    0.61
  0    1800        946.47   5845.96   67.31   73.19   62.31    0.67
  0    2000       1186.43   6509.00   68.23   74.50   62.92    0.68
  0    2200       1415.23   7683.86   70.81   76.68   65.77    0.71
  0    2400       1594.44   8733.94   72.53   76.60   68.87    0.73
  0    2600       1684.48   8751.48   73.12   76.54   69.99    0.73
  0    2800       1738.34   8413.53   73.44   78.31   69.14    0.73
  0    3000       1743.35   8290.82   75.06   78.52   71.89    0.75
  0    3200       1746.25   8005.08   75.34   79.99   71.20    0.75
```

Désormais, pour l'évaluer, il va falloir que nous l'utilisons sur notre fameux fichier .iob et nous comparions le résultat avec celui trouvé précédemment.

```
!python -m spacy evaluate ./output/model-best ./spacy/test.spacy
[6]  ✓ 1m 35.3s
ℹ Using CPU

================================== Results ==================================

TOK     99.65
NER P   85.89
NER R   82.08
NER F   83.94
SPEED   13599
```

Le système basé sur des règles donne une précision moyenne de
Prules = (34,4 + 53,6 + 42)/3 = **43,3%**

Tandis que spaCy offre le résultat suivant :
Pspacy = (85,9 + 82 + 83,9)/3 = **83,9%**

Après évaluation du modèle, nous pouvons confirmer que l'outil spaCy améliore considérablement la performance du système avec une précision bien supérieure à notre système basé sur des règles.