PROJET TAL

TRAITEMENT AUTOMATIQUE DES LANGUES

ESTEBAN NERAUDAU – MAXIME GOURION

ET5 INFO

ANNÉE 2023/2024

Table des matières

1 Introduction

Le domaine du traitement automatique des langues (TAL) constitue un pilier fondamental de la révolution numérique actuelle. Il fait appel à des techniques avancées issues de la linguistique computationnelle et de l'intelligence artificielle dans le but de traiter et d'analyser le langage naturel. L'évolution rapide des données disponibles en ligne accentue le besoin d'outils performants capables de décoder la complexité inhérente du langage humain. Les tâches de désambiguïsation morpho-syntaxique et de reconnaissance d'entités nommées se placent au cœur de cette problématique, jouant un rôle crucial dans la compréhension des textes et la structuration des données linguistiques.

La désambiguïsation morpho-syntaxique, notamment le marquage des parties du discours (POS tagging), est essentielle pour attribuer correctement la fonction grammaticale de chaque mot dans son contexte. La reconnaissance d'entités nommées (NER) complète cette analyse en identifiant et en classifiant les termes spécifiques tels que les noms de personnes, de lieux ou d'organisations, ce qui permet d'extraire des informations précises et de structurer le contenu textuel.

Ce rapport propose une évaluation comparative de deux plateformes d'analyse linguistique reconnues : Stanford NLP et NLTK. Stanford NLP est réputé pour son framework robuste basé sur des modèles d'apprentissage profond et des algorithmes complexes, tandis que NLTK est largement utilisé dans le milieu académique et industriel pour sa simplicité d'usage et ses approches hybrides qui marient ressources linguistiques traditionnelles et techniques d'apprentissage automatique. Nous analyserons la manière dont chaque plateforme aborde le POS tagging et le NER, en mettant en lumière leurs atouts et leurs faiblesses. En utilisant le fichier evaluate.py fourni, nous aspirons à dresser un tableau exhaustif de leurs performances respectives, ouvrant ainsi des perspectives pour des améliorations futures susceptibles d'impacter significativement le domaine du TAL.

2 Présentation des plateformes d'analyse linguistique

2.1 Stanford

Stanford CoreNLP est une infrastructure intégrée conçue pour l'analyse linguistique de textes écrits dans plusieurs langues. Cette plateforme s'appuie sur une variété d'approches computationnelles pour traiter le langage, incluant des techniques à base de règles, des méthodes statistiques et des modèles basés sur le traitement neuronal. En combinant ces différentes méthodes, Stanford CoreNLP est capable d'offrir des fonctionnalités telles que le POS tagging, la lemmatisation, la reconnaissance d'entités nommées, la résolution de coréférence et l'analyse du sentiment.

Pour le POS tagging et la reconnaissance d'entités nommées, par exemple, Stanford CoreNLP utilise typiquement des modèles statistiques conditionnels, comme les Conditional Random Fields (CRF), qui sont entraînés sur de grands corpus annotés. Ces modèles apprennent des caractéristiques complexes des mots et de leur contexte pour prédire la catégorie grammaticale ou l'entité à laquelle ils appartiennent. En ce qui concerne les approches neuronales, la plateforme intègre des réseaux de neurones profonds qui permettent de capturer des relations sémantiques plus subtiles à travers de vastes ensembles de données d'entraînement, améliorant ainsi la précision de l'analyse linguistique.

Stanford CoreNLP se distingue par sa modularité, permettant aux utilisateurs de sélectionner et de chaîner différents composants de traitement selon les besoins de leur application. L'architecture de la plateforme est conçue pour être extensible, facilitant l'intégration de nouveaux modèles et l'adaptation à de nouvelles tâches. Par ailleurs, il fournit une interface de programmation (API) accessible et bien documentée, ce qui facilite son intégration dans des systèmes plus larges et son utilisation par la communauté des développeurs et des chercheurs en TAL.

2.2 NLTK

Le Natural Language Toolkit, ou NLTK, est une suite d'outils conçue pour le traitement de la langue naturelle pour le langage Python. Renommée pour son approche éducative et pratique, elle permet une entrée en matière aisée dans le domaine du traitement automatique du langage (TAL) tout en offrant des fonctionnalités robustes pour des utilisateurs plus avancés. NLTK intègre une diversité d'approches de TAL, notamment des méthodes à base de règles, statistiques et certaines formes d'apprentissage automatique, mais moins sur les approches neuronales qui sont plus récentes par rapport à l'existence de NLTK.

Dans le domaine des approches basées sur des règles, NLTK permet aux utilisateurs de construire des grammaires de traitement du langage qui peuvent analyser la structure des phrases et reconnaître les entités nommées en se basant sur des patterns syntaxiques prédéfinis. En parallèle, NLTK fournit un accès à des algorithmes statistiques classiques comme les classificateurs bayésiens naïfs, les arbres de décision, et les machines à vecteurs de support pour diverses tâches de TAL comme la classification de texte et l'analyse de sentiment.

En termes de modèles statistiques, NLTK inclut des outils pour le POS tagging, le chunking, et le parsing, qui s'appuient souvent sur des modèles comme les Hidden Markov Models (HMM) et les Conditional Random Fields (CRF). Ces modèles sont entraînés sur des corpus annotés pour prédire la structure grammaticale et les rôles sémantiques dans des phrases.

Cependant, contrairement à d'autres plateformes comme Stanford CoreNLP, NLTK se concentre davantage sur la mise à disposition d'un large éventail d'algorithmes traditionnels et de ressources linguistiques, telles que des corpus de texte et des lexiques, qui sont essentiels pour l'enseignement et la recherche en TAL. En outre, NLTK offre une documentation étendue et des guides pratiques qui facilitent l'apprentissage et l'expérimentation dans le domaine du TAL.

.

3 Evaluation de l'analyse morpho-syntaxique

Afin d'évaluer les deux plateformes sur le plan de l'analyse syntaxique, le corpus utilisé est composé de 481 phrases. Le fichier pos_reference.txt montre que ce corpus est composé de 10554 mots ou chunk.

Afin d'évaluer les plateformes, le fichier pos_reference.txt est utilisé comme référentiel. Ce fichier contient une liste des chunks et leur tag. Pour comparer les deux fichiers, les tags sont convertis vers leur équivalents Universel et stockés dans un fichier séparé. Les plateformes sont ensuite utilisées sur le texte pour obtenir l'analyse morpho-syntaxique, et le même processus est appliqué à ces listes de tags.

Une fois que la liste des tags Universel est disponible, on utilise le script evalute.py pour comparer les résultats et évaluer le pourcentage de ressemblance entre notre référentiel et les plateformes d'analyse

3.1 Stanford

```
PS C:\Users\maxgo\Documents\ET5\TAL\Projet_TAL\POLYTECH_TAL_2024\Projet> python evaluate.py pos_test.txt.pos.stan.univ pos_reference.txt.univ
Warning: the reference and the candidate consists of different number of lines!
Word precision: 0.043081199126464166
Word recall: 0.043081199126464166
Tag precision: 0.043081199126464166
Tag recall: 0.043081199126464166
Word F-measure: 0.043081199126464166
Tag F-measure: 0.043081199126464166
```

3.2 NLTK

```
PS C:\Users\maxgo\Documents\ET5\TAL\Projet_TAL\POLYTECH_TAL_2024\Projet> python evaluate.py pos_test.txt.pos.nltk.univ pos_reference.txt.univ
Warning: the reference and the candidate consists of different number of lines!
Word precision: 0.1209053007742704
Word recall: 0.1209053007742704
Tag precision: 0.1209053007742704
Tag recall: 0.1209053007742704
Word F-measure: 0.1209053007742704
Tag F-measure: 0.1209053007742704
```

3.3 Conclusion

Pour conclure cette analyse, la première chose à remarquer est que les pourcentages sont très faibles. Les pistes expliquant cet écart est tout d'abord le script evaluate.py qui est potentiellement non-fonctionnel. En deuxième lieu, il faut aussi remarquer la différence du nombre de chunk entre notre référence et la liste de tags obtenus après les analyses, ce qui fausse naturellement les calculs.

Cependant, malgré ces marges d'erreur, on remarque que NLTK est 3 fois plus précis que Stanford. Mais ces mesures sont à prendre avec du recul, puisqu'avec son avance, NLTK ne ressemble qu'à 12% de notre fichier de référence.

4 Evaluation de la reconnaissance d'entités nommées

Le corpus d'évaluation contenu dans le fichier "ne_reference.txt.conll" se compose de 429 phrases, qui totalisent 10 068 mots.

Pour mesurer la qualité d'une tâche d'évaluation en traitement automatique des langues, on utilise généralement trois métriques principales : la précision, le rappel et la f-mesure. La précision est le rapport entre le nombre d'éléments pertinents retrouvés et le nombre total d'éléments retrouvés. Le rappel est le rapport entre le nombre d'éléments pertinents retrouvés et le nombre total d'éléments pertinents à retrouver. Enfin, la f-mesure est la moyenne harmonique de la précision et du rappel, fournissant une seule mesure de performance globale.

4.1 Stanford

```
espo@laptop:/mnt/c/Semestre9/POLYTECH_TAL_2024/Projet$ python3 evaluate.py pos_test.txt.pos.stan.univ pos_reference.txt.univ
Warning: the reference and the candidate consists of different number of lines!
Word precision: 0.018176400476758045
Word recall: 0.018176400476758045
Tag precision: 0.018176400476758045
Tag recall: 0.018176400476758045
Word F-measure: 0.018176400476758045
Tag F-measure: 0.018176400476758045
```

Capture d'écran des résultats de evaluate.py sur notre fichier généré avec Stanford

4.2 NLTK

```
espo@laptop:/mnt/c/Semestre9/POLYTECH_TAL_2024/Projet$ python3 evaluate.py ne_test.txt.ne.nltk.conll ne_reference.txt.conll
Warning: the reference and the candidate consists of different number of lines!
Word precision: 0.018275725069527213
Word recall: 0.018275725069527213
Tag precision: 0.018275725069527213
Tag recall: 0.018275725069527213
Word F-measure: 0.018275725069527213
Tag F-measure: 0.018275725069527213
```

Capture d'écran des résultats de evaluate.py sur notre fichier généré avec NLTK

5 Points forts, limitations et difficultés rencontrées

Nos résultats semblent honnêtement très faibles qu'importe qu'on étudie le POS ou le NER. Nous nous sommes donc penchés sur des raisons qui peuvent expliquer cela :

Pour la partie 2 on remarque tout d'abord une énorme incohérence dans le sujet. Il est indiqué d'utiliser les étiquettes CoNLL-2003 pour la reconnaissance des entités nommés :

Annexe 2 : Etiquettes CoNLL-2003 pour la reconnaissance es entités nommées

O: Words that are not named entities and referred to as 'Other'.
B-PERS: Beginning of Person Name.
I-PERS: Inside of Person Name.
B-ORG: Beginning of Organization Name.
I-ORG: Inside of Organization Name.
B-LOC: Beginning of Location Name.
I-LOC: Inside of Location Name.
B-MISC: Beginning of MiscellaneousWord.
I-MISC: Inside of MiscellaneousWord.

Ce que nous avons fait dans le fichier script_part_2.py. En revanche, lorsque l'on jette un coup d'œil au fichier .conll de référence on voit que les entités sont nommés par des entités B-PER et I-PER pour les entités qui correspondent a une Person Name. Ce qui engendre forcément des résultats très mauvais lors de la comparaison des fichiers lors de l'évaluation.

```
1   Mary    B-PER
2   Barra   I-PER
3   appointed   O
4   as  O
5   General B-ORG
6   Motors  I-ORG
7   chief   O
```

Cela étant, même en corrigeant cette incohérence et changeant nos étiquettes dans nos fichiers de tests, pour ressembler au fichier de référence, les résultats affichés sont extrêmement faibles même si un peu différent de nos premiers.

Enfin l'autre observation assez évidente vu qu'un warning est affiché: « the reference and the candidate consists of different number of lines! " Nous avons donc une nouvelle fois regardé nos fichiers et effectivement notre logique est mauvaise : des sauts de ligne sont oubliés. Dans la version du script actuelle, nous ajoutés un saut de ligne au formatted_text si il y a un point (ce qui est logique) mais il existe d'autre cas où il y a un saut de ligne (par exemple après un « '' » que l'on peut voir ligne 224) mais aussi des sauts de ligne sans logique apparente (ligne 9 à 12 on voit une date, avec un saut de ligne avant et après) et idem pour la ligne 569 ou après un « '' » nous n'avons pas de sauts de ligne.

A cause de ces différents sauts de ligne non gérés, nous perdons de l'information à la fin car nos fichiers .ne.stan.conll et .ne.ntlk.conll n'ont plus ces sauts de ligne ce qui entraine une trentaine de ligne de différence par rapport au fichier de référence.

De même, à la suite de notre conversion (question 1) du fichier reference .conll en .txt certains « ‘’ » deviennent des « `` » qui sont deux caractères différents. Dans ce cas de figure, nous ne savons pas comment faire pour ne pas perdre d’informations sur la gestion de caractères car notre fichier est déjà au plus simple.

```python
# 1. Extraction de phrases à partir d'un fichier au format CoNLL pour la préparation des données d'entrée.
# Fonction pour convertir un fichier CoNLL en fichier texte
1 usage   Esteban *
def conll_to_txt(input_path, output_path):
    with open(input_path, 'r', encoding='utf-8') as input_file, \
         open(output_path, 'w', encoding='utf-8') as output_file:
        for line in input_file:
            if line.strip():  # Si la ligne n'est pas vide
                word = line.split('\t')[0]  # Extraire le mot
                output_file.write(word + ' ')  # Écrire le mot dans le fichier de sortie
            else:
                output_file.write('\n')  # Nouvelle ligne pour les séparations de phrases
```

Capture d’écran de la fonction conll_to_txt

Nous n’avons malheureusement pas trouvé de solution pour pallier ce problème qui explique nos résultats très bas.

En conclusion, mis à part ce souci qui nous empêche de gérer correctement certains sauts de ligne, nous avons réussi à générer des fichiers de manières satisfaisantes avec des résultats cohérents. Le souci des sauts de lignes nous empêche d’avoir une conclusion viable ce qui est très dommageable.

Par curiosité j’ai effectué un test sur un échantillon de test plus petit avec uniquement 50 lignes pour le fichier de référence .conll et j’ai pu observer quelques améliorations a faire dans le futur :

Voici mes observations (toujours pour la NER) :

- Je perds de l’information avec nltk sur les personnes car Marry Barra devrait être traité comme ‘B-PER et I-PER’ et je le récupère comme B-PER B-PER (ligne 1 du fichier ne_test.txt.ne.nltk.conll). En revanche c’est bien géré ligne 25-26 ce qui est étrange
- Pour la gestion d’une organisation, la logique doit être améliorée comme on peut le voir ci-dessous. Le fichier de reference indique que ‘The’ est un « B-ORG » et la suite une I-ORG ce qui est logique car en anglais c’est comme ça que cela fonctionne mais dans notre traitement nous détectons United State comme Localisation ce qui est faux

```
14  The B-ORG
15  United  I-ORG
16  States  I-ORG
17  's  I-ORG
18  largest I-ORG
19  car I-ORG
20  manufacturer    I-ORG
21  General I-ORG
22  Motors  I-ORG
23  today   O
24  named   O
25  Mary    B-PER
26  Barra   I-PER
```

Capture d’écran du fichier ne_reference.txt.conll

```
14  The O
15  United  B-LOC
16  States  I-LOC
17  's  O
18  largest O
19  car O
20  manufacturer    O
21  General B-ORG
22  Motors  I-ORG
23  today   O
24  named   O
25  Mary    B-PER
26  Barra   I-PER
```

Capture d’écran du fichier ne_test.txt.ne.nltk.conll

6 Organisation

Pour ce projet nous avons travaillé en collaboration sur les TPs lors des différentes séances de cours que nous avons eus.

A propos du projet, nous nous sommes réparti le travail par partie. Ainsi Maxime s'est occupé de la partie 1 et Esteban de la partie 2 sur les NE.

Le GitHub du projet est disponible en cliquant ici.

Le code de chaque partie du projet est commenté de manière à expliquer ce que chaque fonction fait et la logique de certaines parties.

7 Conclusion

Retour d'expérience d'Esteban : Ce projet a pour moi été très enrichissant. Ce n'était pas un projet simple pour moi car c'était ma première approche du traitement automatique des langues (notamment du NER) même nous avions touché à des sujets qui se rapprochaient de celui-là. D'un point de vue factuel, les résultats ne sont pas bons et les difficultés rencontrées m'ont forcé à cravacher pour essayer de trouver des explications ce qui m'a fait comprendre pas mal de concepts sans pour autant améliorer mes résultats. Des améliorations sont évidemment à faire mais finalement j'ai pu comprendre et approfondir quelques connaissances qui me serviront, pour sûr, dans ma vie professionnelle future.

Concernant les résultats, je n'ai pas bien compris comment nous pouvions comparer 2 méthodes sans tokenisation de référence, mais j'ai certainement du mal m'y prendre (voir le script projet_part_2.py).