

Rapport Projet TAL

Evaluation de deux plateformes open source d'analyse
linguistique

Polytech Paris-Saclay – cycle ingénieur – 3^{ème} année



Equipe Composée de : **Alya BEN OUADDAY, LOK Tshanon**
Responsable du module : **SEMMAR Nasredine**

Table des matières

Le Traitement automatique du Langage	3
Introduction.....	3
Pipeline pour du traitement automatique de la langue	4
Métriques d'évaluation	5
Les plateformes d'analyse linguistique.....	6
NLTK.....	6
Stanford CoreNLP	7
Evaluation de l'analyse morpho-syntaxique (POStagging).....	8
Corpus d'évaluation.....	8
Métriques et Résultats	10
Evaluation de la reconnaissance d'entités nommées (NE recognition)	12
Corpus d'évaluation.....	12
Métriques	13
Analyse de chaque plateforme	14
NLTK :.....	14
Stanford Core NLP:	15
Limitations et difficultés rencontrées.....	16
Organisation	17
Bibliographie	18

Le Traitement automatique du Langage

Introduction

Le Traitement Automatique du Langage (ou TAL) est une discipline qui a pour objectif de modéliser, grâce à l'informatique, le langage (naturel), qu'il soit écrit ou parlé. Il s'agit d'une discipline pluridisciplinaire qui mêle la linguistique à l'informatique et même à présent à l'intelligence artificielle. Cette discipline est couramment utilisée et subit depuis quelques années, de grandes transformations. Le traitement automatique de la langue a longtemps été connu pour les correcteurs orthographiques et les traducteurs automatiques (de qualité laissant à désirer il fut un temps). En revanche, il existe maintenant des applications de plus en plus efficaces et opérationnelles, l'extraction et l'analyse de données (textuelles entre autres) étant d'un intérêt majeur pour l'industrie d'aujourd'hui. Évidemment, il subsiste toujours certains problèmes, mais la qualité du traitement automatisé est de plus en plus satisfaisante, que ce soit pour la traduction de langues, les conversations avec agents artificiels, les moteurs de recherches qui deviennent plus précis et efficaces et même l'extraction et l'analyse de données issues des réseaux sociaux (sentiments des tweets, publications facebook..)

Le TAL présente toutefois plusieurs difficultés. A priori, une machine n'a aucune connaissance préalable de la langue, et nécessite donc des indications, des règles, tout comme un langage informatique (lexique, syntaxe ..). Il faut pouvoir lui indiquer ce qu'est un mot, une phrase, et lui apprendre donc les subtilités d'une langue qui n'est peut-être pas assez codifiée pour être apprise par une machine.

En effet, le langage naturel est bien souvent complexe et ambiguë. Il faut par exemple pouvoir déterminer si une apostrophe marque une séparation entre deux mots ou s'il s'agit d'un même mot, comme pour *l'aube*, et *aujourd'hui*. Il s'agit d'ailleurs d'un des grands problèmes que nous avons rencontré durant ce projet, lorsque nous essayons de comparer deux fichiers qui ne séparent pas les mots de la même manière. En y réfléchissant, presque chaque mot d'une phrase pourrait potentiellement poser problème. En dehors de la séparation des mots "tokenisation", s'ajoute également la sémantique pour pouvoir analyser un mot. Comment, par exemple, déterminer automatiquement dans quels contextes le mot « avocat » se rapporte au domaine juridique ou au domaine alimentaire ?

Finalement, la principale difficulté est donc de traiter par un langage binaire et donc **non ambigu** - c'est-à-dire l'informatique - un **langage ambigu** – les langues.

L'objectif de notre projet, et de ce rapport, consiste alors à évaluer deux plateformes open source d'analyse linguistique : NLTK et Stanford CoreNLP.

Il s'agit d'une comparaison de ces deux plateformes déjà découvertes et manipulées lors des TP, afin d'apercevoir les différences, points forts et limites/améliorations de chacune.

Pour cela, nous comparerons les résultats obtenus à une référence issue de la plateforme LIMA, en développant les scripts nécessaires à l'exploitation de résultats. Il faudra ainsi passer par plusieurs étapes de TAL qui seront explicitées dans les parties ci-après, et rendre les résultats comparables avec la référence (fichier evaluate).

Ce travail aura été réalisé par Alya Ben Ouadday et Tshanon Lok.

Pipeline pour du traitement automatique de la langue

Le TAL est décomposable en plusieurs outils linguistiques : l'analyse morphologique, l'analyse morpho syntaxique (POS tagger), l'analyse syntaxique et la reconnaissance d'entités nommées (NE recognizer).

Il est possible de faire du TAL en suivant cette pipeline :

La **tokenisation** permet de découper les chaînes de caractères, les phrases, en un sous ensemble de chaînes de caractères – des mots, ponctuations et autres – en prenant en compte le contexte ainsi que des règles de découpage. Généralement, des règles de segmentation ainsi que des automates d'états finis sont utilisés.

L'**analyse morphologique** permet d'identifier les mots (tokens) avec leur(s) catégorie(s) grammaticale(s)

L'**analyse morpho-syntaxique** permet de desambiguïser certains tokens. En effet, cette étape permet d'affecter à chaque mot (token) la bonne catégorie grammaticale.

L'**analyse syntaxique (parsing)** permet de déterminer le lien entre chaque groupe de mots. C'est-à-dire reconnaître les composants / les constituants syntaxiques de la phrase.

La **reconnaissance d'Entités Nommées** permet d'identifier les personnes, lieux, dates, organisations (etc...) présentes dans le corpus sur un ensemble de tokens.

Le langage naturel étant complexe et ambiguë pour une machine – d'une part à cause de la complexité pour déterminer comment tokeniser un corpus et d'autre part pour analyser la sémantique, le POS d'un token – plusieurs approches peuvent être employées pour traiter automatiquement le langage. On a des méthodes statistiques, des méthodes à base de règles ainsi que des méthodes neuronales.

Le principe des méthodes statistiques est d'appliquer différentes formules probabilistes, tel que la formule de Bayes, et statistiques aux tokens rencontrés. On peut par exemple s'intéresser à la fréquence d'apparition d'un token dans un corpus pour en déduire le thème principal des corpus rencontrés.

Le principe des méthodes à base de règles est différent. Le but est de s'intéresser aux différents niveaux d'analyse linguistique tels que la sémantique ou bien la phonétique. Ces méthodes sont difficiles à mettre en place car elle nécessite tout d'abord l'expertise d'un linguiste. De plus, cette approche prend plus de temps à être développée dû à un temps de constitution du lexique ou/et de la grammaire. Enfin, il est compliqué de généraliser cette approche pour toutes les langues puisque chacune a un lexique et une grammaire différente.

Actuellement, plusieurs plateformes permettant de mettre à bien ces outils sont à disposition et permettent de pratiquer cette discipline. Pour ce projet, nous allons comparer la plateforme NLTK, ayant une approche hybride, ainsi que CoreNLP, ayant une approche en apprentissage automatique, afin de déterminer leurs points forts et points faibles.

Nous suivrons la pipeline évoqué précédemment et utiliserons les différents outils à l'aide des plateformes NLTK et Stanford CoreNLP.


Métriques d'évaluation

Afin de pouvoir évaluer la performance des outils utilisés en TAL, il est impératif d'avoir des mesures universelles permettant de déterminer la précision et la pertinence des résultats obtenus.

Une des métriques les plus utilisées est la **précision**.

Notons **TP** le nombre de mots correctement tagués comme la référence (les experts), et **FP** le nombre de mots qui n'ont pas été étiquetés correctement, voici le calcul de la précision **P** :

$$P = \frac{TP}{TP + FP}$$

Précision = 

La précision est donc la proportion de mots taggués de façon pertinente parmi les mots tagués par le candidat.

Utiliser une seule métrique pour interpréter les résultats ne serait pas suffisant, c'est pour cela qu'on utilise également une autre métrique, le **rappel**.

Soit **FN** le nombre de mots qui n'ont été étiquetés par aucune étiquette (appelé aussi silence), le calcul du rappel est le suivant :

$$R = \frac{TP}{TP + FN}$$

Rappel = 

Ainsi, le rappel mesure la proportion de mots correctement étiquetés au regard du nombre total de mots qui auraient pu être étiquetés. Les silences (FN), sont les mots de la référence qui aurait dû être étiquetés mais en raison d'une mauvaise segmentation, certains mots sont passés sous silence et restent sans étiquettes. Par exemple, "My car's red" peut être segmenté par la référence comme *My - car - 's - red* et pourtant étiqueté comme *My - car's -red*, ainsi, ni *car* ni *'s* n'auront réellement été étiquetés et comparables à la référence.

Ces quantités sont également liées au score **F-measure**, qui est défini comme la moyenne harmonique de la précision et du rappel:

$$F = 2 \cdot \frac{P \cdot R}{P + R}$$

Les métriques que nous avons définies ci-dessous font référence au "tagging" des mots, mais il existe les mêmes métriques, relatives cette fois à la tokenisation, pour évaluer la manière dont la segmentation des phrases (évoquée ci-dessus) a été faite en regard à la référence.

Soit **TP** le nombre de mots correctement segmentés comme les experts, **FP** le nombre de mot segmentés en plus de la référence (Bruit) et **FN** le nombre de mots non segmentés (Silence) par le candidat.

Les formules sont alors les mêmes qu'au-dessus (P, R, F).

Stanford CoreNLP

La plateforme Stanford CoreNLP est un framework développé en JAVA par Christopher D. Manning permettant d'effectuer des traitements automatiques du langage. Actuellement, le module permet d'être utilisé pour des scripts en anglais, chinois, français, allemand et arabe.

Stanford CoreNLP est un module (API) open source composé de plusieurs outils de reconnaissance du langage comme on peut le voir sur le schéma ci-dessus.

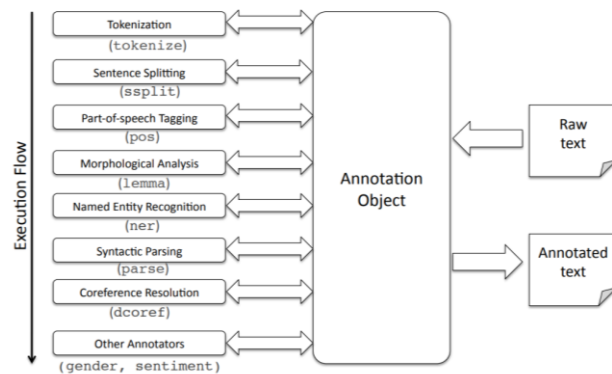


Figure 5: Différents modules de Stanford CoreNLP

Il est en effet composé d'un tokenizer qui permet de découper une chaîne de caractères en un ensemble de tokens. Il est d'ailleurs possible de regrouper les tokens en phrases. Il possède un tagger qui associe à chaque token une catégorie grammaticale (Part-Of-Speech). Il peut retrouver la forme canonique d'un mot avec une analyse morphologique ou bien identifier des informations à partir d'un texte simple en extrayant la relation entre un ensemble de tokens. Il peut par ailleurs faire de la reconnaissance d'entités nommées et résoudre des problèmes de coréférence. Le module peut par ailleurs faire d'autres types d'annotations comme assigner à chaque token des points positifs ou négatifs selon la négativité ou positivité de ce token.

Plusieurs approches sont possibles pour faire du traitement automatique du langage. Dans le cas de Stanford CoreNLP, la plateforme est entièrement à base d'apprentissage automatique au contraire de NLTK qui a une approche hybride. Cette approche probabiliste, par apprentissage automatique, permet à Stanford de pouvoir s'adapter aux différents corpus étudiés. En effet, l'approche par machine learning ne nécessite pas d'écrire en brut des règles linguistiques.

L'approche par apprentissage automatique est performante pour de la classification de document ou bien du clustering de mots depuis un corpus. De manière générale, cette approche est bonne lorsqu'il y a beaucoup de mots-clés, de tokens.

Nous nous intéresserons pour ce projet à la tokenisation, à la performance de l'analyse morpho-syntaxique proposée par la plateforme ainsi qu'à la performance de la reconnaissance d'entités nommées.

Evaluation de l'analyse morpho-syntaxique (POStagging)

Corpus d'évaluation

Au début du projet, nous ne disposions pas de corpus annoté pour pouvoir le taggué avec les différentes plateformes. Nous avons seulement un fichier segmenté et annoté par la plateforme LIMA.

Ainsi, nous avons mis en place un script nommé « getOriginalCorpus » permettant de retrouver le corpus original à partir du fichier annoté :

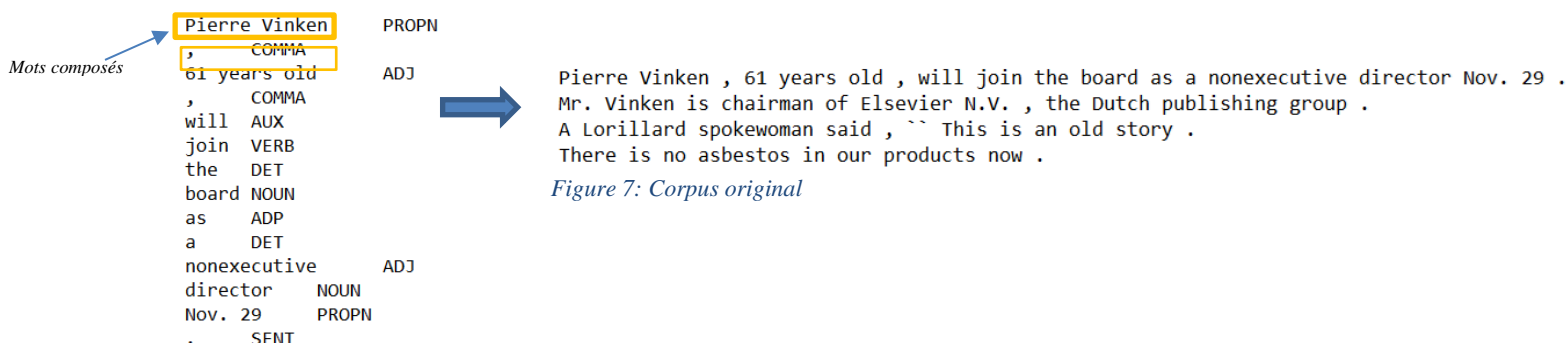


Figure 6: Référence LIMA annotée

Cependant, nous avons remarqué que les deux autres plateformes NLTK et Stanford ne tag pas les mots composés de la même manière, alors que le format standard d'un fichier annoté pour l'évaluation consiste en deux colonnes : le mot , une tabulation puis son annotation (tag).

Ainsi, nous avons décidé de retirer tous les mots composés de la référence, et donc du corpus obtenu. Cela nous permet d'avoir un format standard, de ne pas fausser la comparaison avec

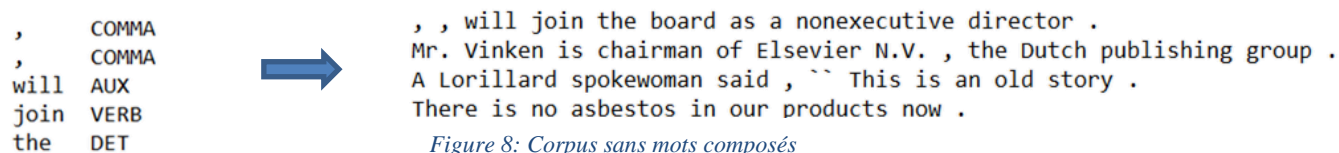


Figure 9: Référence LIMA sans mots composés

les deux autres plateformes qui n'auraient pas les mêmes mots :

Ci-dessus le corpus d'évaluation (*pos_test.wc.txt*) sur lequel nous avons exécuté le tag avec NLTK puis Stanford Core NLP.

Cependant, nous avons remarqué que les deux autres plateformes NLTK et Stanford ne tag pas les mots composés de la même manière alors que le format standard du fichier taggé pour l'évaluation consiste de deux colonnes : le mot , une tabulation puis son annotation (tag).

Ainsi, nous avons décidé de retirer tous les mots composés pour avoir un format standard, et surtout pour que cela ne fausse pas la comparaison avec les deux autres plateformes.

Il s'agit d'un fichier qui contient du texte, sans les mots composés qui étaient présents dans le fichier LIMA, avant un retour chariot « \n » à la fin de chaque phrase.

Le fichier `doPOSTag.py` permet de lancer le tag NLTK sur toutes les lignes du corpus, puis d'écrire chaque mot annoté sur une ligne (tag et mot séparés d'une tabulation), puis de sauter une ligne à la fin de la phrase.

Voici le format de sortie de NLTK après exécution du fichier `doPOSTag.py` :

```
, ,  
, ,  
will MD  
join VB  
the DT  
board NN
```

Figure 10: Format du fichier NLTK

Voici le format de sortie de Stanford après avoir lancé `stanford-postagger.sh` :

```
, , , will_MD join_VB the_DT board_NN as_IN a_DT nonexecutive_JJ director_NN ._.  
Mr._NNP Vinken_NNP is_VBZ chairman_NN of_IN Elsevier_NNP N.V._NNP , , the_DT Dutch_JJ publishing_NN group_NN ._.  
_NN 's_POS , , a_DT forum_NN likely_JJ to_TO bring_VB new_JJ attention_NN to_TO the_DT problem_NN ._.  
A_DT Lorillard_NNP spokeswoman_NN said_VBD , , ```` This_DT is_VBZ an_DT old_JJ story_NN ._.  
There_EX is_VBZ no_DT asbestos_NN in_IN our_PRP$ products_NNS now_RB ._. ````
```

Figure 11: Format du fichier Stanford

Afin de pouvoir évaluer Stanford avec la référence, il faut que ce dernier ait le même format. Ainsi, nous avons réalisé le script `post_processing_stanford.py` afin de supprimer les underscore et les remplacer par une tabulation, mettre chaque mot sur une ligne (et non toute une phrase sur une ligne) et faire un retour chariot entre deux phrase. Par ailleurs, nous avons transformé les (et { car ils n'étaient pas bien écrits et faussaient les résultats. Après exécution du fichier de post-processing sur Stanford, ce dernier a alors le même format que le fichier NLTK de la figure 10.

Métriques et Résultats

Avant de pouvoir évaluer ces deux fichiers avec la référence, nous avons dû faire face à un problème. Les fichiers candidats n'ont pas forcément tokenisé le corpus comme l'a fait la référence, ainsi, le fichier `evaluate.py` que nous avons n'est plus utilisable car ce dernier nécessite que les mêmes mots soient face à face (et donc que les deux fichiers aient le même nombre de lignes et les mêmes tokens).

Pour pallier ce problème nous avons réalisé un script `evaluate` nous-même. Nous avons remarqué que la plupart des problèmes étaient dus à la manière de séparer les « 's » . Ainsi, lorsque deux mots ne sont pas identiques dans les deux fichiers, le script effectue principalement des concaténations de deux lignes pour voir si cela correspond au mot en question, si c'est le cas, les deux mots concaténés et le mots en face sont retirés (et sont comptés soit comme du bruit *W-FP* soit comme du silence *W-FN*).

NLTK

Voici les résultats que nous avons obtenus en lançant le script `evaluate` avec NLTK :

```
Word precision: 1.0
Word recall: 1.0
Tag precision: 0.9364013430772269
Tag recall: 1.0
Word F-measure: 1.0
Tag F-measure: 0.967156262749898
```

Nous remarquons que la précision et rappel pour la tokenisation est à 1.0 soit exactement identique à celui de la référence.

Après plusieurs recherches, nous avons compris que cela est dû au fait que la référence LIMA s'est basée sur la segmentation de NLTK, et possède donc exactement la même segmentation. Ci-dessous, on peut voir qu'il y en a en effet exactement le même nombre de mots dans la référence que dans le candidats, et donc qu'aucun mot de la référence n'a pas été étiqueté.

```
Nombre de mot total pour la référence : 9482
Nombre de mot total pour le candidat : 9482
Nombre de mots mal étiquetés : 644
Nombre de mots non étiquetés : 0
Nombre de mots bien tokenisés: 9482
```

En revanche, on peut voir qu'il y a 644 mots qui ont été mal étiquetés, soit environ 200 mots de plus que pour Stanford : $\frac{200}{9482} = 0.02$. Ainsi, cela explique bien la différence de 0.02% pour la précision entre Stanford et NLTK.

Le Tag-rappel est aussi de 1.0, car il évalue la proportion de mots bien évalués en comparaison au nombre de mots total qui aurait dû être évalué. Étant donné que tout a été évalué, le rappel est de 1.0.

Stanford

Voici les résultats que nous avons obtenus en lançant le script evaluate avec NLTK :

```
Word precision: 0.979500052295785
Word recall: 0.987660831048302
Tag precision: 0.9545957918050941
Tag recall: 0.9854500103928497
Word F-measure: 0.9835635141521819
Tag F-measure: 0.9697775504985937
```

Cette fois-ci, on remarque que la tokenisation est légèrement différente et donc n'est pas exactement la même que celle de la référence.

En effet, lorsque l'on a cherché à afficher le nombre de mot qui est resté sans étiquette (silence) car n'a pas été tokenisé de la même manière que la référence, nous avons trouvé que 140 mots sont restés sans étiquette :

```
Nombre de mot total pour la référence : 9482
Nombre de mot total pour le candidat : 9561
Nombre de mots mal étiquetés : 451
Nombre de mots non étiquetés : 140
Nombre de mots bien tokenizés: 9365
```

En voulant aller plus loin pour comprendre ces résultats, nous avons cherché directement à afficher ces mots pour réellement saisir ce qui diffère dans la tokenisation de Stanford pour que près de 2% des mots de la référence n'aient pas été étiquetés.

```
crude ADJ
by ADP
today's NOUN
standards NOUN
. SENT
```

Figure 12: Segmentation par LIMA

```
crude JJ
by IN
today NN
's POS
standards NNS
.
```

Figure 13: Segmentation par Stanford

Ainsi, nous avons trouvé que près de 100 mots parmi les non étiquetés sont en réalité des « 's » qui n'ont pas été segmentés de la même manière pour Stanford et la référence LIMA :

C'est un peu la même chose pour d'autres mots comme « Gotta » et « Got ta »

```
You PRON
Gotta PROPN
Have PROPN
Wa COMMA
" QUOT
```

Figure 14: Segmentation par LIMA

```
You PRP
Got VBD
ta TO
Have VBP
Wa ,
" "
```

Figure 15 : Segmentation par Stanford

Ainsi, cela explique également que Stanford ait légèrement plus de mots que la référence, Stanford sépare bien plus les mots que la référence.

En revanche, malgré plus de mots non étiquetés pour Stanford, nous devons bien sur relever que la précision au niveau des tags est plus précise pour Stanford, avec 95% (450/9482 mots mal tagués) de précision contre 93% pour NLTK.

Evaluation de la reconnaissance d'entités nommées (NE recognition)

Corpus d'évaluation

Afin d'obtenir le corpus d'évaluation de la reconnaissance d'entités nommées, il était nécessaire d'extraire à partir du fichier `out/ne_reference.txt.conll.txt` les phrases ayant servi à produire ce fichier, corpus annoté.

Dans ce corpus annoté, une ligne vide indique la fin de la phrase courante.

L'ensemble des tokens se trouvant dans la première colonne ont donc été assemblés et sauvegardés dans le fichier `src/out/ne_test.txt`

Dans ce fichier, chaque phrase a été sauvegardée sur une unique ligne.

Extrait du contenu de `ne_reference.txt.conll.txt`

Mary	B-PER
Barra	I-PER
appointed	O
as	O
General	B-ORG
Motors	I-ORG
chief	O

Le corpus `src/out/ne_test.txt` est composé de 430 lignes/ phrases. En s'intéressant au contenu, on remarque que ce corpus contient le nom de politiciens (exemple : Obama), d'organisations, de date (exemple : May 1 , 2009), de montant. On s'attend donc à ce que notre modèle les retrouve.

On lance les deux NE recognizers sur le fichier `src/out/ne_test.txt` dans deux fichiers différents :

```
out/ne_test.txt.ne.stanford
out/ne_test.txt.ne.nltk
```

Pour pouvoir évaluer par la suite la précision, le rappel des plateformes Stanford CoreNLP et NLTK, il est nécessaire de faire du postprocessing avant de lancer l'évaluation.

Tout d'abord, on modifie le résultat des deux fichiers afin de s'assurer que le format de ces deux corpus annotés soit deux colonnes séparées par une tabulation comme dans le fichier `src/out/ne_test.txt`.

Enfin, les tags Penn TreeBank de ces corpus annotés ont été traduits en étiquettes CoNLL-2003. Les équivalences sont sauvegardées dans le fichier `data/POSTags_PTB_Universal_Linux.txt`.

Métriques

L'évaluation a été lancée avec le corpus annoté de Stanford CoreNLP puis avec celui de NLTK. Nous avons écrit le fichier d'évaluation qui permet d'harmoniser les fichiers `out/ne_test.txt.ne.stanford` et `out/ne_test.txt.ne.nltk`, c'est-à-dire que les tokens en commun sont écrit sur la même ligne dans leur fichier respectif.

Les résultats sont les suivants.

Stanford CoreNLP	NLTK
Word precision: 0.8354768028960655	Word precision: 0.8114699437934648
Word recall: 0.8702987000099236	Word recall: 0.8461309228171252
Tag precision: 0.9273028434710592	Tag precision: 0.9229006233956729
Tag recall: 0.8851897399859452	Tag recall: 0.8666494490358126
Word F-measure: 0.8525323223485953	Word F-measure: 0.8284380470725541
Tag F-measure: 0.9057570446272079	Tag F-measure: 0.8938909607529747

On remarque que les résultats de Stanford CoreNLP sont meilleurs que ceux de NLTK. Il semble donc pertinent pour la reconnaissance d'entités nommées de s'orienter vers des plateformes à base d'apprentissage automatique.

Cependant, en s'intéressant aux différences de tokenisation entre les fichiers candidats et la référence, on remarque que ces différences de fichier d'évaluation sont dues à une tabulation ce qui influence les résultats par la suite. Cependant, même sans ces soucis, nous pensons que Stanford CoreNLP sera tout de même meilleur que la plateforme NLTK.

Analyse de chaque plateforme

NLTK :

Point forts

La première chose qui nous a frappé en utilisant la plateforme NLTK est que la tokenisation était réellement identique à celle de la référence. Ainsi, l'utilisation de NLTK était bien plus simple pour nous.

La tokenization de NLTK évitait beaucoup de mots « perdus » sans étiquettes et donc cela impliquait moins de problèmes au niveau du evaluate puisque tous les mots sont bien cotés.

Par ailleurs, nous avons trouvé l'utilisation de NLTK beaucoup plus intuitive étant donné qu'il s'agit d'une librairie Python, elle était très simple à utiliser. Comme nous pourrez le constater, le pos tag a été effectué dans la fonction `doPOSTag` (et comme la Figure 2). La fonction `pos tag` était très malléable, et nous laissait une grande liberté avec la liste de tuples qui était retournée. Nous pouvions effectuer des traitements sur ce tableau, ou bien l'écrire dans un fichier, en bref NLTK est très flexible et accessible (bien plus que Stanford).

De la même manière, nous avons trouvé ça très simple d'extraire les entités nommées avec `ne_chunk`, en parcourant l'arbre syntaxique, et c'est très parlant d'avoir l'affichage graphique d'un arbre sous les yeux.

Points faibles

Un des points faibles de NLTK est qu'il est basé en partie sur des règles, ce qui implique qu'une partie de ces règles a dû être codée avec l'aide de plusieurs linguistes experts contrairement à l'apprentissage automatique qui ne nécessite que de nombreuses ressources et corpus linguistiques (en plus du modèle). Il est clair que l'avenir du TAL aujourd'hui réside dans l'essor grandissant des méthodes neuronales (apprentissage automatique) et qu'une plateforme telle que NLTK à base de règles risque d'être rapidement dépassée.

Stanford Core NLP:

Point forts

Nous avons remarqué que Stanford avait une tokenisation bien différente de la référence et cela a été très compliqué pour nous de le gérer pour pouvoir effectuer l'évaluation. En revanche, en comparant les différences de tokenisation (la plupart étant des «'s »), nous avons trouvé la tokenisation de Stanford un peu plus juste.

En effet, si l'on reprend la figure 12 et 13 de ce rapport, avec NLTK le mot « today's » reste complet avec la tokenisation et est tagué en tant que « NOUN » donc nom. Par contre, Stanford sépare « today » en tant que NOUN et « 's » en tant que possessif ce qui est correct dans ce contexte-là. D'après nous, le fait de pouvoir apporter des précisions de tag sur les contractions (qu'il s'agisse de verbe ou de possessif) est un point fort non négligeable de Stanford.

Par ailleurs, le point fort déduit des résultats et métriques est tout simplement la précision des tags qui est plus élevée.

Et enfin, à noter tout comme plus haut dans la partie NLTK que les méthodes neuronales sont de plus en plus prisées efficaces et utilisées, ainsi Stanford a le mérite d'être basé (bien qu'avec un modèle assez basique) sur une technologie qui risque d'évoluer et de faire évoluer le TAL dans les années à venir.

Points faibles

Etant en Java et un fichier .sh à exécuter, nous avons trouvé Stanford peu intuitif et assez complexe. Aucune interface n'est proposée pour interagir avec la plateforme et les différents modules, il s'agit donc d'un outil moins accessible à tous d'après nous. Il n'y a pas d'affichage graphique et il n'y a pas non plus de grande communauté ou de ressources en lignes : ainsi il est moins aisé et agréable à utiliser que NLTK.

Par ailleurs, bien qu'elle soit plus pertinente pour nous, la tokenisation de NLTK nous a posé souci pour le evaluate, et le fichier de sortie a été difficile à normaliser avec le pre processing pour obtenir comme NLTK et la référence un fichier avec deux colonnes et un mot par colonne. La sortie de Stanford est peu lisible en ligne avec des underscore partout.

Limitations et difficultés rencontrées

Plusieurs difficultés ont été rencontrées lors de ce projet. Tout d'abord, nous avons eu des problèmes liés au tag Stanford CoreNLP. En effet, lorsque le programme rencontrait les caractères '(' et '{', ces derniers avaient respectivement pour token '-LRB-' et '-LCB-'. De plus, leur tag associé était respectivement '-LRB-' et '-LCB-'. Pour palier ce problème, nous avons décidé de créer un fichier python permettant de faire du post processing permettant au final de remplacer les '-LRB-' et les '-LCB-' par les caractères '(' et '{'.

Un autre problème lié au tagging de la ponctuation était tout ce qui était lié aux apostrophes. En effet, lors du passage entre des tag PTB vers universel, nous avons eu quelques problèmes liés aux tokens suivants : ',', ' ', '`', '"'. Pour palier ce problème, nous avons ajouté au dictionnaire récupéré du TP1/2 les transformations suivantes :

```
","      .
'        .
``       .
"        .
```

(Tout est transformé en point .)

Enfin, la difficulté principale de ce projet a été le fichier python `evaluate`. Nous avons tenu à rédiger par nous même le fichier `evaluate` plutôt que de reprendre celui se trouvant dans les TP. Lorsque nous avons besoin d'évaluer les fichiers candidats et celui de la référence, il était nécessaire d'harmoniser les fichiers candidats et `nlk`. Harmoniser signifie faire en sorte que pour un même numéro de ligne, le mot du fichier candidat et le mot du fichier référence soit le même. Ainsi, les seules différences entre ces deux fichiers doivent être les tag associés aux mots taggés. Faire de l'harmonisation implique que certains mots à la fois du fichier candidat et de la référence ne soient pas inclus dans le fichier résultat de l'harmonisation – du au fait que la tokenisation n'a pas été fait de la même manière. Harmoniser doit donc faire varier le score final. C'est pourquoi nous avons créé notre propre fichier `evaluate` – pour éviter de fausser les résultats lorsque l'on fait de l'harmonisation de fichier.

L'une des difficultés de la programmation de `evaluate.py` était d'harmoniser les fichiers candidats à cause de la tokenisation. On a dû regarder le contenu de chaque fichier pour comprendre comment NLTK et Stanford CoreNLP tokenisait le corpus afin d'ajouter des règles pour harmoniser les fichiers.

Par ailleurs, certains problèmes nous ont limité pour peaufiner davantage le projet. Tout d'abord, nous n'avons eu qu'un peu plus d'une semaine de projet que l'on devait partager avec d'autres projets extérieurs à la discipline du TAL. Cela nous embête un peu car la discipline étant intéressante, nous aurions aimé avoir soit avoir plus de temps, soit pouvoir libérer un peu plus de créneau pour faire uniquement de la pratique pour du TAL et ainsi le projet.

Une autres des limites qui nous semble importantes est que les résultats auraient peut-être été meilleurs si nous avions trouvé une meilleure manière de gérer les compounds plutôt que de les supprimer.

Organisation

De manière générale, la totalité du projet a été faite ensemble. Nous procédions avec une méthode appelée le “pair programming”. Le principe de cette méthode est que nous travaillions toutes les deux sur un même poste de travail : Celle qui rédige le code est appelée conductrice (*driver*), l’autre est appelée observatrice (*observer*). Le but de l’observatrice est d’assister la conductrice en décelant les imperfections, en vérifiant que le code implémente correctement le design et en suggérant des alternatives de développement. Nous échangeons ainsi les rôles régulièrement pendant la séance de programmation (environ toutes les heures)

En revanche, quelques fichiers ont été faits individuellement sur nos temps de disponibilités respectives. Le détail se trouve ci-dessous.

Alya

J’ai pu réaliser la fonction qui transforme les étiquettes des entités nommées en étiquettes coNL : `convertToCoNLL.py` . La petite difficulté ici était de détecter un changement d’étiquette pour attribuer B- ou I- . Au départ, j’avais fait un simple booléen qui détectait lorsqu’il n’y avait plus d’entités nommées et que l’on repassait à des étiquettes “O”. Finalement, en faisant des tests et après vérifications je me suis rendue compte que si deux entités nommées se suivent alors nous ne repassons pas en B-. J’ai pu modifier de sorte que cela détecte tout changement d’étiquette à présent.

J’ai aussi beaucoup apprécié me concentrer sur les calculs de métriques (fonction `evaluate()` du fichier `evaluate.py`) et c’est quelque chose que j’ai déjà pu faire lors d’un précédent projet de machine learning. Il était intéressant de voir à quel comment incrémenter les "silences" donc les mots de la référence non étiquetés, ou encore le bruit (étiquetés alors qu’ils ne devraient pas) et comparer les proportion de mots bien étiquetés par rapport au nombre de mot total.

Tshanon

Quelques fichiers ont pu être repris des TPs mais j’ai pu les réadapter pour que les fonctions répondent aux besoins du projet. Par exemple, `use_ne_chunk.py` a été repris des TP mais il était nécessaire de garder tout le texte ainsi que leur entité nommée associée lors du projet. De même pour le fichier `doPOStag.py` qui a été repris des TP et adapté en fonction des fichiers donnés en entrée et les fichiers que l’on souhaite rendre en sortie.

J’ai pu rédiger le fichier `post_processing_stanford.py` afin de faire du post processing pour les fichier stanford en particulier. Puisque l’on fait à plusieurs reprises du postprocessing avec les fichiers .stanford, plusieurs fonctions sont incluses. Il suffit d’ajouter à la commande bash une chaîne de caractères qui fera ensuite le nécessaire pour faire le bon post processing.

Bibliographie

1. Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, David McClosky, *The Stanford CoreNLP Natural Language Processing Toolkit*, 2014, disponible sur : <https://nlp.stanford.edu/pubs/StanfordCoreNlp2014.pdf>
2. Techopedia. (2014, août 2). *Natural Language Toolkit (NLTK)*. Techopedia.Com. <https://www.techopedia.com/definition/30343/natural-language-toolkit-nltk>
3. BIRD, S. G., & LOPER, E. (2004). *NLTK : The Natural Language Toolkit*. Minerva Access. https://minerva-access.unimelb.edu.au/bitstream/handle/11343/34053/66440_00001448_01_nltk.pdf?sequence=1
4. Zarour, A. (2021, 29 octobre). *Qu'est-ce que le Traitement Automatique du Langage ?* Inbenta. <https://www.inbenta.com/fr/blog/quest-ce-que-le-tal/>
5. Thierry POIBEAU, « TRAITEMENT AUTOMATIQUE DES LANGUES », *Encyclopædia Universalis* [en ligne], consulté le 25 février 2022. URL : <https://www.universalis.fr/encyclopedie/traitement-automatique-des-langues/>
6. Bird, Steven, Edward Loper and Ewan Klein (2009), *Natural Language Processing with Python*. O'Reilly Media Inc.
7. *NLTK :: Natural Language Toolkit*. (2022, 9 février). nltk.org. <https://www.nltk.org/>
8. Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, David McClosky. (2014). *The Stanford CoreNLP Natural Language Processing Toolkit*. Stanford NLP Group. Consulté le 20 février 2022, à l'adresse <https://nlp.stanford.edu/pubs/StanfordCoreNlp2014.pdf>
9. Dorash, M. (2018, 20 avril). *Machine Learning vs. Rule Based Systems in NLP - Friendly Data*. Medium. <https://medium.com/friendly-data/machine-learning-vs-rule-based-systems-in-nlp-5476de53c3b8>
10. Arthur Remaud. (2021, 2 juillet). *Utilisation d'outils de TAL pour la compréhension de spécifications de validation de données*. Actes de la 28e Conférence sur le Traitement Automatique des Langues Naturelles. Consulté le 20 février 2022, à l'adresse <https://aclanthology.org/2021.jeptalnrecital-recital.10.pdf>
11. Éric De La Clergeri. (2002, 8 mars). *Les techniques du T.A.L.* INRIA, Cours ENST – TAL. Consulté le 20 février 2022, à l'adresse <http://alpage.inria.fr/~clerger/Enseignement/ENST02/slide2.pdf>