Rapport Traitement Automatique de la Langue

1. **Introduction**

Le traitement automatique de la langue (TAL) est un sous-domaine de la linguistique, de l'informatique et de l'intelligence artificielle qui s'intéresse aux interactions entre les ordinateurs et le langage humain, en particulier à la programmation des ordinateurs pour traiter et analyser de grandes quantités de données en langage naturel.

Les premiers travaux ont débuté en 1950 aux Etats-Unis dans le contexte de la guerre froide. De nos jours ce domaine à toujours autant d’importance avec notamment la mondialisation et la volonté de pouvoir comprendre de et se faire comprendre tout en ne parlant pas la même langue.

Pour traduire un texte, plusieurs étapes sont nécessaires :

* Le découpage des chaînes de caractère en texte
* L’analyse morphologique
* L’analyse morpho-syntaxique
* L’analyse syntaxique
* La reconnaissance d’entités nommées

Durant ce cours de TAL nous avons utilisé plusieurs outils pour découvrir leurs fonctionnements et les comparer pour ainsi avoir un point de vue critique sur leurs caractéristiques respectives.

1. **Présentation des plateformes d’analyse linguistique**

2.1. Stanford Core NLP

Stanford Core NLP toolkit est un framework d’annotation qui est développé sous Java et qui réunit la majorité des fonctionnalités nécessaires au traitement des langues. Il peut traiter des textes sous plusieurs langues et sous différents types d’encodage.

Lors du traitement d’un texte brut, Stanford crée une instance de l’objet Annotation dans laquelle ce texte est enregistré. Ensuite, un ensemble d’Annotateurs modifie ce texte en y ajoutant des informations analytiques supplémentaires.

Chaque annotateur représente une étape et l’ensemble des étapes est organisé pour former un pipeline. Au cours de chaque étape, chaque annotateur ajoute des informations supplémentaires au texte et actualise à chaque fois le texte enregistré dans l’instance d’Annotation.

Les étapes les plus importantes qui composent le pipeline de Stanford sont: la tokenisation -> formation des phrases à partir des tokens -> l'étiquetage morpho-syntaxique -> l’extraction des entités nommées -> l’analyse des sentiments.

Pour le framework Stanford Core NLP, certains annotateurs utilisent des modèles entraînés avec des méthodes statistiques d’apprentissage supervisé à partir de corpora déjà annotés, tandis que d’autres sont basés sur des règles. Par exemple:

L’extraction des entités nommées est basée sur une approche statistique de type CRF.

L’extraction des entités numériques est faite à base de deux systèmes de règles: un pour l’argent et les chiffres et un autre pour l’extraction des informations temporelles.

2.2. NLTK

NLTK comme son nom l'indique: Natural Language Toolkit ,est une plate-forme open-source de traitement automatique de la langue pour des programmes fait en Python. Il fournit des interfaces faciles à utiliser pour plus de 50 corpus et ressources lexicales ainsi qu'une suite de bibliothèques de traitement de texte pour la classification, la tokenisation, la recherche de racines, l’étiquetage morpho-syntaxique, l'analyse et l’affichage en arborescence, le raisonnement sémantique et l’identification d’entités nommées.

L’outil dispose d’une documentation complète sur les API, ce qui en fait un outil convenable à tous.

NLTK incorpore le traitement symbolique et statistique du langage naturel :

Le traitement symbolique étant le codage sous forme de grammaire à l’aide de règles et de bases de données lexicales souvent développées manuellement.

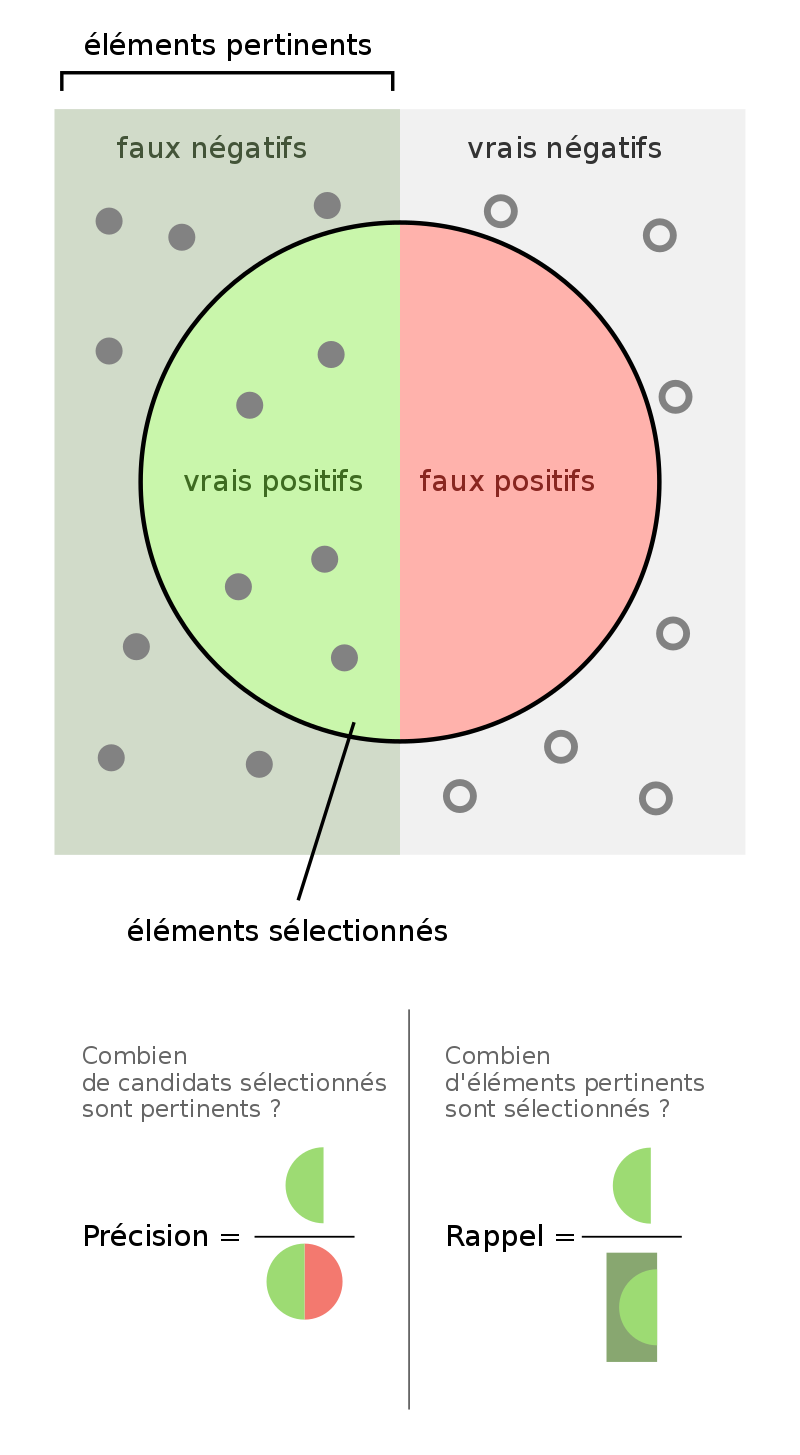
Quant au traitement statistique, il est basé sur des algorithmes et non pas des listes de mots. Ainsi le texte est transformé en input numérique autrement dit en vecteurs en comptant les occurrences des mots ou groupes de mots.

**Approches utilisées :**

La partie analyse morpho-syntaxique est faite grâce à la fonction **pos\_tag()** fournie par NLTK cette fonction prend en paramètre un texte tokenisé. Afin de tokenizer le texte nous avons utilisé la fonction **word\_tokenize()** elle aussi fournie par NLTK. L'étiquetage fait par **pos\_tag()** est réalisé grâce à un modèle entraîné disponible dans la librairie.

**Description du corpus utilisé :**

Pour l'évaluation de la reconnaissance d’entités nommées nous avons utilisé un corpus (ne\_reference.txt.conll) formé de 430 phrases, pour un total d'environ 10 500 mots.

**Description des métriques de mesure:**

**La précision** pour une classe donnée est le nombre de mots correctement tagués rapporté au nombre total de tags généré pour cette classe .

En [statistique](https://fr.wikipedia.org/wiki/Statistique), la précision est appelée [valeur prédictive positive](https://fr.wikipedia.org/wiki/Valeur_pr%C3%A9dictive_positive).

**Le rappel** pour une classe donnée est défini par le nombre de mots correctement tagués au regard du nombre de tags pour cette classe dans la référence.

En statistique, le rappel est appelé [sensibilité](https://fr.wikipedia.org/wiki/Sensibilit%C3%A9_et_sp%C3%A9cificit%C3%A9).

Dans le cadre **multi-classes**, les moyennes globales de la précision et du rappel sur l'ensemble des classes i peuvent être évaluées par la macro-moyenne qui calcule d'abord la précision et le rappel sur chaque classe i suivie d'un calcul de la moyenne des précisions et des rappels sur les n classes :

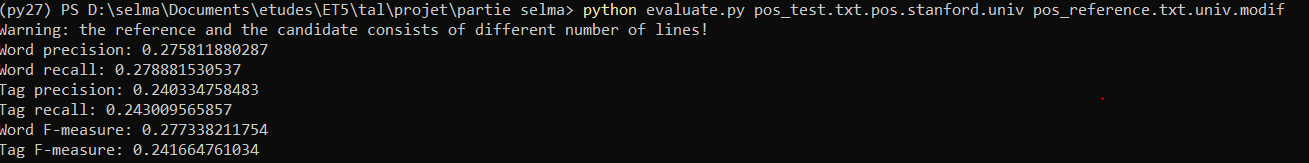
**La F-mesure** est une mesure qui combine précision et rappel, elle représente leur [moyenne harmonique](https://fr.wikipedia.org/wiki/Moyenne_harmonique). Elle est également connue sous le nom de mesure **F1** car précision et rappel sont pondérés de façon égale.

1. **Evaluation de l’analyse morpho-syntaxique**

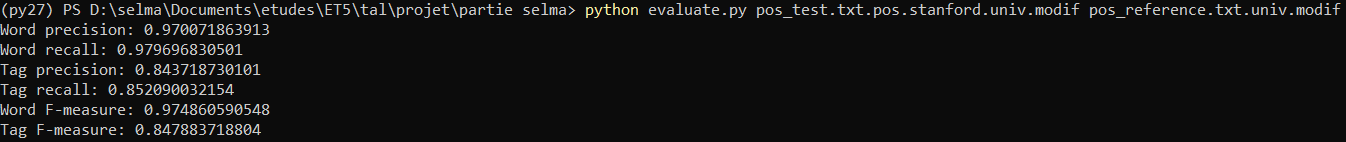
3.1. Stanford Core NLP

Tableau des résultats

Résultats avec le fichier **pos\_test.txt.pos.stanford.univ**: ce document a été généré par le pos tagger de stanford à partir du fichier pos\_test.txt puis les étiquettes ont été converties en étiquettes universelles. Comme on peut le voir avec le warning de la console, le pos tagger de stanford a causé un décalage de lignes par rapport au fichier de référence ce qui explique les résultats médiocres.

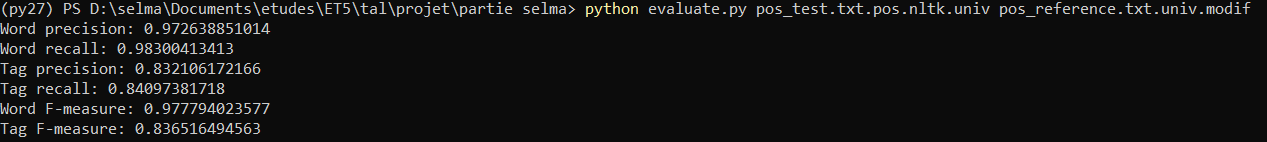


Résultats avec le fichier **pos\_test.txt.pos.stanford.univ.modif** : dans ce document nous avons corrigé le décalage de lignes ce qui donne de meilleurs résultats



3.2. NLTK

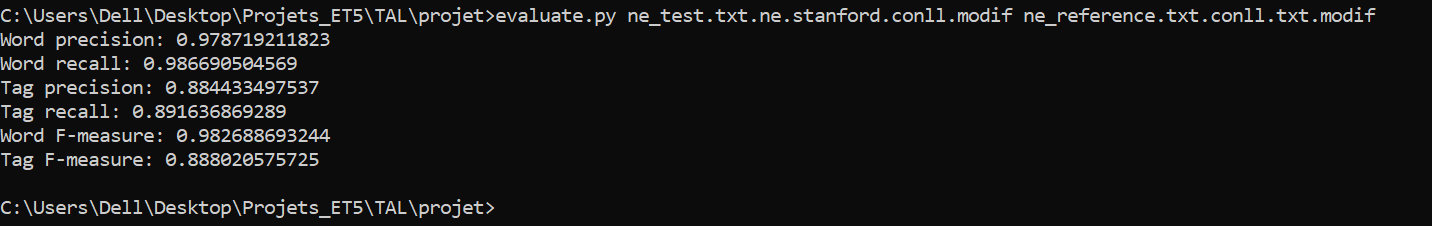
Tableau des résultats



1. **Evaluation de la reconnaissance d’entités nommées**

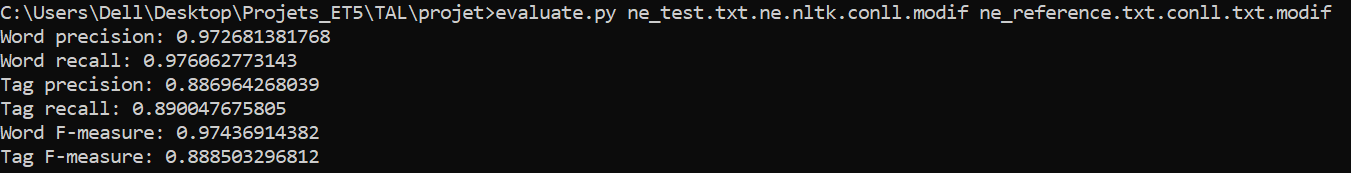
4.1. Stanford Core NLP

Tableau des résultats

****

4.2. NLTK

Tableau des résultats



**5. Points forts, limitations et difficultés rencontrées**

Analyse morpho-syntaxique

Durant notre expérimentation nous avons relevé certaines remarques concernant les deux outils, notamment concernant la tokenisation par exemple:

* “Seven-day” est reconnu comme étant une seule entité dans les lima et nltk mais l’outil de stanford le tokeinise (seven\_CD -\_HYPH day\_NN) ainsi il génére l’étiquette HYPH qui n’a pas d’équivalent universel pour régler ce souci nous avons supprimé “-\_HYPH” dans le fichier pos\_test.txt.pos.stanford.univ
* Une autre difficulté rencontré avec la tokenizer de stanford était le décalage de lignes et de certains caractères qui se retrouvaient en début de ligne alors qu’il devait être en fin de ligne
* Dans le fichier Lima les trois petits points (...) sont tokenisés et chacun se retrouve sur une ligne, ainsi , NLTK les traite comme tel mais l’outil de Stanford les regroupe sur une seule ligne.
* Il y a aussi certains cas où la tokenisation est différente de la référence pour les deux outils ( $\_$ 1.5\_CD billion\_CD pour NLTK et Stanford, $ 1.5 billion\_NOUN pour la référence).

Une difficulté que nous avons aussi rencontrée concerne la conversion des tags générés par NLTK et l’outil Stanford en tag universel car certains tags utilisés par NLTK et Stanford n’existent pas dans la table de conversion fournie. Pour cela, nous avons dû la compléter les tags manquants et qui sont donc:

-RRB- .

-LRB- .

'' .

`` .

$ X

AFX NOUN

Reconnaissance d’entités nommées

Concernant le temps d'exécution de la reconnaissance d’entités nommées :

Pour NLTK on obtient un temps d'exécution de 4 secondes (voir capture d'écran) pour le fichier ne\_test.txt sans compter les opérations de traitement de texte pour remettre en forme la sortie (arbre -> txt)

Pour Standford le traitement se fait en 1 seconde :

“CRFClassifier tagged 10150 words in 370 documents at 11105.03 words per second.”

Stanford est donc 4 fois plus rapide ce qui peut représenter beaucoup de temps quand on essaye de traiter des documents de plusieurs milliers de lignes ou plus.

En plus de cela NLTK renvoie un arbre, difficilement exploitable comparé à du texte pour faire des traitements dans un second temps.

Cependant ces arbres sont plus pratiques pour visualiser les données.

Pour la suite de cette partie, on était confronté à peu de problèmes de code mais le mode de reconnaissance des retours à la ligne de l’outil Stanford posait des problèmes. Stanford ne détecte pas le retour à la ligne lorsqu’il y a pas de point à la fin d’une phrase, cela crée une tokenisation différente entre les fichiers ref et Stanford et donc mène à une impossibilité d’évaluation.

On a résolu ce problème manuellement vu que les lignes qui ne terminent pas par un point constituent environ une vingtaine de lignes ce qui est facilement gérable manuellement.

**6. Organisation**

Durant les TP nous nous sommes répartis les différents parties des TP pour être plus productif selon la répartition ci-dessous :

TP1  
1. Selma/Haithem/Julien  
2. Selma

TP2  
1. Julien  
2. Selma  
3. Haithem

Durant le projet nous avons continué à nous répartir les tâches entre nous, ainsi Julien a réalisé les 2 premières questions des deux parties et Selma et Haithem ont continué respectivement la partie 1 et 2 du projet. Bien sûr nous nous concertions régulièrement pour se mettre à jour et se poser des questions.

Julien a réalisé le GitHub du projet.

Pour finir nous avons tous les trois rédigé le rapport des tp et du projet.

**7. Annexes**

Temps d’execution NLTK :



Temps d’execution Stanford :

