Joel Mbouwe
jores-jores.mbouwe-nanmou@student.ecp.fr

Lab session # 2
SPEECH & NLP 2020

03/09/20

Objectif : Il s'agissait tout au long de ce 2^e TP de développer un parser probabiliste pour l'étiquetage morpho-syntaxique de la langue française. Ceci en implémentant un modèle CYK et un modèle PCFG (Probabilistic Context Free Grammar) qui soient robustes aux mots absents des données d'entraînement.
Remarque : Afin d'éviter des éventuels soucis de sparcité, j'ai ignoré les termes après les traits d'union dans les POS non terminaux.

1 Construction de la pipeline

1.1 Création d'un Probabilistic Context Free Grammar

Le premier module que j'ai implémenté *proba_contextfg* a pour objectif d'extraire du corpus une grammaire probabiliste non contextuelle.
Pour se faire, la première étape a été de transformer les règles de grammaire présentes dans notre base de données en forme normale de Chomsky. On fait cela parce que l'algorithme CYK que l'on souhaite implémenter ne peut traiter que les règles comportant deux POS non terminaux à droite. Le processus de normalisation de Chomsky consiste ainsi à binariser la grammaire en remplaçant :

- La règle binaire $A_0 \rightarrow A_1 \dots A_n$ ($n > 2$ POS non terminaux) par les $n-1$ règles suivantes : $A_0 \rightarrow A_1 B_1$, $B_1 \rightarrow A_2, B_2$, … , $B_{n-2} \rightarrow A_{n-1}, A_n$. De la même maniere la règle $A \rightarrow B, c$ par $A \rightarrow B, C$ avec $C \rightarrow c$.
- La règle unitaire $A \rightarrow B$ par $A \rightarrow B_0, B_1$ avec $B \rightarrow B_0, B_1$

pour ce faire, j'ai utilisé la libraire NLTK qui, à partir d'une règle de grammaire génère la forme normale de chomsky associée.
PCFG : A partir des règles sous forme normale de chomsky, l'algorithme PCFG consiste à calculer les probabilités des règles de la grammaire : $P(A \rightarrow B) = \frac{count(A \rightarrow B)}{\sum_C count(A \rightarrow C)}$ J'ai préféré stocker ces probabilités dans des dictionnaires puisqu'ils permettent un temps d'accès constant.

1.2 Gestion des mots absents du vocabulaire

J'ai ensuite créé une classe *OOV_module* permettant la gestion des mots jamais rencontrés dans la base d'entraînement. Cette gestion se base sur deux principales techniques :

- La correction orthographique via la distance de Leveinstein. Pour un mot OOV, je considère un certain nombre ($c = 20$) de mots candidats, situés à une distance (au sens de Leveinstein) d'au plus **3** du mot original. L'un de ces candidats pouvant correspondre au bon orthographe du mot initial.
- La proximité semantique via la mesure de similarité entre les embeddings *polyglot-fr* des mots. Pour un OOV, les c mots les plus proches (au sens du cosine similarity) du mot initial seront considérés comme mots alternatifs.

Ainsi, chaque OOV aura au maximum $2c$ potentiels mots de remplacement. Par ailleurs, à l'aide du corpus d'entraînement, j'ai construit les unigrams et les bigrams des mots. Enfin j'ai implémenté un décodeur de type beam-search (un décodeur greed-search aurait considérablement ralenti le module) afin de trouver le mot parmi les $2c$ candidats qui maximise la probabilité de la phrase à parser.

1.3 Algorithme CKY

Pour l'implémentation du probabilistic CKY contenu dans le module *proba_cky*, je me suis basé sur la formulation de l'algorithme contenu dans le chapitre 12 de [1]. En effet, soit une phrase à parser

de longueur n, notant P(s, l, X) la probabilité que l'arbre le plus probable permettant de parser la sous-phrase de longueur $l+1 \in [1, n]$, commençant à la position du mot $s \in [0, n-l-1]$, soit de POS de tête X; on a :

$$P(s, l, X) = \max_{0 \leq c \leq l-1, X \to Y,Z} P(X \to YZ) P(s, c, Y) P(s+c+1, l-c-1, Z)$$

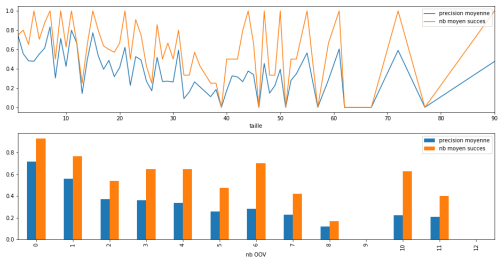
et de plus $P(s, 0, X) = P(X \to s)$
J'ai ainsi implémenté un algorithme de programmation dynamique calculant ces différentes probabilités et si $P(0, n-1, SENT) = 0$ alors l'opération de parsing sera considérée comme échouée. Le cas contraire je retrace les différents POS qui ont permis d'atteindre $P(0, n-1, SENT)$. Enfin j'utilise NLTK pour annuler la forme normale de Chomsky de ma sortie afin d'estimer la précision de la méthode.

2 Analyse des résultats

J'ai entraîné l'intégralité de la pipeline avec 90% de la base de données et j'ai testé le modèle sur les 10% restants soit 310 phrases. Sans parallélisation, le modèle a pris 17 minutes pour traiter les phrases des données tests. J'ai utilisé la librairie **evalb** pour l'évaluation de la précision (en terme de POS correctement trouvés) du modèle. Les figures ci-après récapitulent 1a les résultats du modèle sur les données d'entraînement et donnent des informations plus détaillées 1b par rapport à l'influence de la longueur des phrases et du nombre de OOV.

parsing	nbre de phrases	taille moyenne des phrases	précision moyenne	nb moyen de OOV
échec	117	25.000000	0.000000	4.700935
succès	203	18.551724	0.646712	2.463054

(a) statistiques globales



(b) statistiques locales

L'on constate que le modèle peine à parser les phrases très longues (On constate une décroissance du nombre moyen de phrase parsé en fonction de la longueur des phrase) et également celles contenant beaucoup de mots OOV. Ce qui implique que le module OOV bien que déjà consistant perd en efficacité lorsque le nombre de OOV augmente. En effet le module OOV arrive dans certains cas à corriger les fautes d'orthographe et même à remplacer un mot inconnu par un mot alternatif sémantiquement proche : *"je suis à l' int**e**rieur l' h**o**pital . "* va être corrigée par *"je suis à l' int**é**rieur l' h**ô**pital . "* ou encore *"Ils **eurent** des enfants"* par *"ils **ont** des enfants ."*

L'on constate également, que la précision moyenne de l'algorithme en décroit en fonction de la longueur des phrases. Cela peut se justifier par le fait quen nous avons utilisé la forme normale de Chomsky qui fait en sorte que les règles $A \to B \to C$ ne soient pas récupérables via l'algorithme de CYK ce qui impacte sur la précision.
En effet la phrase *(SENT (NP (NC Amélioration) (PP (P de) (NP (DET la) (NC sécurité)))))* sera parsée comme *(SENT (NP (NC Amélioration) (PP (P de) (NP (DET la) (NC sécurité)))))*

Pistes d'amélioration : Il pourrait être intéressant d'entraîner l'algorithme avec un plus gros volume de données afin de renforcer le OOV notamment les N-grams utilisés dans le décodeur. En effet la gestion des OOVs est le levier le plus actionnable dans cet algorithme. Une question autour de la vitesse d'exécution de l'algorithme se pose également.

References

[1] Daniel Jurafsky James H. Martin. Speech and language processing 2018.