Algorithmes pour le traitement de la parole et du langage naturel : TD2

Guillaume PETIT

Résumé

L'objectif de ce devoir est de développer un algorithme de parsing probabiliste (PCYK) pour l'étude des textes en NLP. Pour cela, on va créer un pcfg et un algorithme pour traiter les mots inconnus. On se basera sur le corpus SEQUIOA qu'on divisera en 80 % pour le training, 10 % pour le test et 10% pour des essais.

1 Implémentation

1.1) PCFG

Un PCFG ([1] [2]) est défini par quatre paramètres (N, Σ, R, S). N est l'ensemble des règles de vocabulaires, qu'on va appeler les noeuds non terminaux, Σ est l'ensemble des mots du vocabulaire, R est un tableau regroupant les combinaisons des règles de grammaire possibles avec leur probabilité, ie

$$A \to B \text{ et } p(A \to B) = \frac{C(A \to B)}{\sum_\gamma C(A \to \gamma)}$$

avec A et $B \in N$, et S est la racine de l'arbre.Pour pouvoir implémenter le PCYK, j'organise mon PCFG sous forme de Chomsky pendant le training. C'est-à-dire que chaque règle de grammaire mène soit à deux autres règles ou à un mot (noeud terminal). Pour faire cela, je m'aide de la librairie NLTK de Python.
Je crée ces paramètres dans ma classe PCFG. Néanmoins, pour une meilleure organisation des algorithmes, je décide de créer un tableau de probabilités pour les combinaisons de règle de grammaires ('gram' dans le code) et un lexique pour les combinaisons des noeuds non terminaux aux noeuds terminaux (lexicon dans le code). Gram et lexicon sont des dictionnaires des règles parents (lsh) et ont pour valeur un dictionnaires des règles enfants (rhs) qui ont eux-même pour valeur $p(\text{lhs} \to \text{rhs})$.

1.2) OOV

Je crée une seconde classe qui a pour objectif de faire face aux mots inconnus rencontrés pendant le parsing. Pour améliorer mes résultats, je télécharge l'embedding du Polyglot d'un lexique Français et je vais créer un modèle de langage basé sur les bigrammes. Pour trouver un mot correspondant, je vais adopter trois stratégies, une partie provenant de [3].
1ère étape : Le mot inconnu est normalisé, ie on transforme les chiffres en #, et on cherche à travers notre lexique un mot correspondant très proche, c'est-à-dire écrit en minuscule, majuscule par exemple.
2ème étape : Si la normalisation de notre mot ne se trouve pas dans le lexicon mais dans le Polyglot, alors on prend l'embedding de ce dernier et on calcule la similitude des cosinus avec les mots du lexique ayant un embedding (faire la similitude du cosinus avec l'embedding entier prenait beaucoup trop de temps, j'ai donc décidé de ne pas le faire). Cependant, il se peut que cela soit aussi dû à une erreur de grammaire. Je décide donc de générer des candidats proches selon la distance de Damereau-Levenshtein (inférieure ou égale à 2) et d'utiliser mon modèle de langage pour ajouter à leurs score la probabilité du contexte, coefficienté pour avoir un certains impact.
exemple : Dans la phrase 'Il fait à', on écrit volontairement 'Il fiat à'. Sans le modèle de langage, la seconde étape ressortirait 'Peugeot', car fiat sera compris commme Fiat après la normalisation et donc l'embedding le plus proche est celui d'une voiture. En prenant en compte le contexte avec des candidats selon Levesnhtein, le OOV ressort 'fait' qui est le mot recherché.
3ème étape Si le mot n'est pas dans le Polyglot, alors je présume que cela est sûrement dû à une erreur de grammaire. En premier lieu je génère des candidats selon Damereau-Levenshtein avec un distance inférieure ou égale 2. Si des candidats existent, alors j'utilise les probabilités du bigramme diviser par la fréquence du mot du candidat comme score. Diviser par la fréquence du mot permet d'éviter de ressortir des mots proches mais qui ressortent trop souvent. Sinon, je génère des candidats plus éloignés(jusqu'à 8) et applique le même raisonnement en pondérant par rapport à la distanc cette fois-ci . Diviser par la fréquence ici devient nécessaire, sinon beaucoup de noms propres deviendraient des ponctuations, puisque ces dernières ont une forte probabilité dans le bigramme. S'il n'y a toujours pas de candidats, alors je regarde à travers tout le lexique le mot le plus probable mais la prédiction sera forcément moins bonne.

1.3) PCYK

L'objectif du PCYK est de trouver l'arbre de grammaire $\hat{T}$ quand on lui donne une phrase S, ie

$$\hat{T}(S) = \underset{T \text{ s.c } S=yield(T)}{\text{argmax}} P(T)$$

Soit une phrase $x_1...x_n$ et $\pi(i, j, X)$ le score le plus élevé pour tout sous-arbre qui domine les mots $x_i...x_j$, et dont la racine est la règle X. PCYK va résoudre le problème de manière récursive en calculant [1] :

$$\pi(i,j,X) = \max_{\substack{X \to YZ \\ s \in \{i,..j-1\}}} p(X \to YZ)\pi(i,s,Y)\pi(s+1,j,Z)$$

2 Analyse

2.1) Score

Après avoir obtenu les arbres, on évalue nos scores grâce à evalb. N'ayant pu avoir accès au résumé des scores, j'ai évalué mes scores sur un échantillons des phrases que j'ai pu parser. J'obtiens une précison de 91% avec en moyenne 25% de phrases non parsées.

2.2) Limite et amélioration

Les règles du PCFG imposent une hypothèse d'indépendance sur les probabilités. Ils ne permettent pas de conditionner la probabilité d'une règle au contexte environnant et cela mène à une mauvaise modélisation des dépendances structurelles dans l'arbre. Je remarque que plusieurs erreurs se répètent. Quand l'algorithme a du mal à prédire des noms propres, ils sont remplacés par des ponctuations, ce qui induit en erreur l'arbre créer par mon PCYK.
Considérer un PCYK flexible, c'est-à-dire qui gère aussi les règles unitaires, auraient peut-être permis un algorithme plus efficace, en suivant [1] Il y a plusieurs phrases que je n'arrive pas à traduire sous forme d'arbre. En y regardant de plus près, je remarque que cela est dû soit à la structure de la phrase dans le test qui n'existe pas dans notre PCFG, ou un mauvais candidats a été assigné pour un oov, ce qui induit un mauvais parsing, conduisant à une phrase dont il est impossible de déterminer la structure.
Pour améliorer nos résultats, il existe plusieurs moyens. Agrandir notre train permettrait dans un premier temps d'avoir une plus grande richesse lexicale et grammaticale. On aurait pu aussi télécharger une base de données des erreurs grammaticales et utiliser un N-gramme plus élevé nous permettant d'améliorer notre OOV ;

$$P(x|w)P_{N-gram}(w)$$

où x est la correction et w le candidat.

3 Références

[1] Probabilistic Context-Free Grammars (PCFGs), Michel Collins
[2] Speech and Language Processing, Chapitre 14, Daniel Jurafsky & James H. Martin
[3] 'https ://nbviewer.jupyter.org/gist/aboSamoor/6046170'


```
le mot original est aventure
la normalisation est dans le polyglot. On verifie aussi les erreurs d'écriture
le remplaçant est  aventures
le mot original est coloniale
la normalisation est dans le polyglot. On verifie aussi les erreurs d'écriture
le remplaçant est  colonial
le mot original est profitable
la normalisation est dans le polyglot. On verifie aussi les erreurs d'écriture
le remplaçant est  onéreuse
le mot original est compagnies
la normalisation est dans le polyglot. On verifie aussi les erreurs d'écriture
le remplaçant est  compagnie
le mot original est privées
la normalisation est dans le polyglot. On verifie aussi les erreurs d'écriture
le remplaçant est  privée
le mot original est Indochine
la normalisation est dans le polyglot. On verifie aussi les erreurs d'écriture
le remplaçant est  Allemagne
le mot original est Descours
erreur de grammaire...
le remplaçant est  durs
le mot original est Cabaud
erreur de grammaire...
le remplaçant est  Caillaud
le mot original est Brasseries
la normalisation est dans le polyglot. On verifie aussi les erreurs d'écriture
le remplaçant est  MJC
le mot original est glacières
la normalisation est dans le polyglot. On verifie aussi les erreurs d'écriture
le remplaçant est  emprises
le mot original est Indochine
la normalisation est dans le polyglot. On verifie aussi les erreurs d'écriture
le remplaçant est  Allemagne
le mot original est etc
la normalisation est dans le polyglot. On verifie aussi les erreurs d'écriture
le remplaçant est  et
```

Exemple OOV