Algorithms for Speech and NLP TD 4 : Construction d'un Parser probabiliste pour le Francais

Pirashanth RATNAMOGAN
MVA ENS Paris-Saclay
pirashanth.ratnamogan@ens-paris-saclay.fr

1. But du TD

Ce quatrième TD a pour but de créer un parser probabiliste pour l'étiquetage morpho-syntaxique. Cela consiste à associer aux différents mots d'une phrase les éléments syntatiques auxquels ils se réfèrent implicitement. On veut par exemple à un haut niveau détecter les verbes, les noms (etc) mais on veut également pouvoir faire des regroupement plus importants. Cet algorithme sera basé sur la base de données SEQUOIA treebank v6.0. La majorité des préceptes utilisés seront justifiés dans l'ouvrage référence du cours [1]

Exemple de phrase :

Au_cours_de la cérémonie d'inauguration. $\xrightarrow{\text{parse}}$
((SENT (PP (P Au_cours_de) (NP (DET la) (NC cérémonie) (PP (P d') (NP (NC inauguration))))) (PONCT .)))

2. L'apprentissage du modèle

Le but du TD est de crée un parser basé sur l'algorithme CYK (Cocke-Younger-Kasami) et le modèle PCFG (Probabilistic Context-Free Grammar). Afin d'appliquer l'algorithme CYK nous devons construire les règles et les probabilités associées à chacune de ses règles pour notre PCFG. Afin de pouvoir effectuer cette opération plusieurs étapes ont été nécessaires.

2.1. Lire le fichier SEQUOIA

Lire le fichier SEQUOIA est surement la partie la plus longue et la plus complexe du TD. Afin de lire le fichier et d'extraire les règles présentes dans chaque phrases plusieurs options sont possibles. La première serait d'utilier le module `Tree` de NLTK qui extrait directement les règles dans une phrase donnée. Mais puisque le but du TD et de créer le parser il me paraissait plus judicieux de tout faire de zéro, cela permettait également d'apporter de la flexibilité et d'ajuster le script de lecture en fonction des besoins. L'algorithme de lecture que j'ai créé est basé sur : le repérage des symboles non-terminaux grâce au caractère '(', et sur la création d'un niveau associé à chaque symbole dans la phrase en faisant un décompte des parenthèses de la gauche vers la droite ('(' rajoute +1 et ')' -1) l'idée était de faire l'association $A \rightarrow BCDE$ que si B,C,D et E ont un niveau immédiatement supérieur à celui de A (=niv$(A) + 1$ et si aucun autre symbole du niveau égale à celui de A n'est présent entre A et les éléments non-terminaux $BCDE$. En ayant associé un niveau à chaque symbole non-terminal j'arrive à récupérer toutes les règles nécessaires dans les lignes SEQUOIA. J'ai beaucoup jonglé avec les structures de données présentes dans python dict, set et tuple. Dans ma lecture j'ai crée un `dict` de règles qui a un symbole non-terminal (clé) associe un `set` de tuple contenant toutes les manières de séparer le symbole clé du dictionnaire. De la même manière j'ai créé un dictionnaire qui à toutes les ancres (vocabulaires) associe un `set` de symboles terminaux avec lesquelles ils ont été directement associés.

2.2. Créer les probabilités pour le PCFG

Afin d'associer des probabilités aux différentes règles précedemment extraites on utilise la méthode simple décrite dans [1]. On aura la règle suivante :

$$P(\alpha \rightarrow \beta) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\sum_\gamma \text{Count}(\alpha \rightarrow \gamma)}$$

où α, β, γ correspond à des symboles quelconques.

2.3. Transformer dans la forme normale de Chomsky

Pour appliquer la version probabilisée de CYK, il faut que toutes nos règles soit dans la forme normale de Chomsky. Je gère les 'units production' ([1] chap 12) directement lors de ma lecture de fichier : puisque je fais ma lecture des règles de gauche à droite, lorsque je vois que la règle sera une unit production je n'ajoute pas la règle et je remplace la chaine $\alpha \rightarrow \beta$ (=unit production), $\beta \rightarrow \gamma$ par la chaine $\alpha \rightarrow \gamma$ et tout cela se repercute dans les associations suivantes dans la phrase parcourue (pour le vocabulaire en particulier).

Je gère les règles avec plus de 2 éléments en sortie en faisant des fusions itératives tel que : $A \rightarrow BCD$ avec probabilité p devient $A \rightarrow$ '$B + C$'D avec probabilité p et j'ajoute la règle '$B + C$' $\rightarrow BC$ avec probabilité 1. Je traite tous les cas avec des sorties plus grandes que 2 ainsi, introduire le symbole '+' permet de repérer les règles ajoutés que l'on ne voudra pas afficher à la fin.

2.4. Probabilistic CYK

Puisque notre espace des règles est probabilisé, on utilise l'algorithme de CYK probabilisé basé sur la programmation dynamique pour trouver le parsing optimal de plus grande probabilité. L'algorithme est très bien expliqué dans l'ouvrage ([1], chap 13). J'ai créé une classe `ProbabilisticCYK` qui s'appuie sur les outils précedemment décrits pour apprendre ses attributs (une grammaire de règles probabilisées), on apprend cela avec la méthode `fit`. Ensuite, la fonction predict_one_line permet fait l'algorithme CYK probabilisé à proprement parlé pour effectuer le decodage d'une phrase. Je stocke l'étiquetage morpho-syntaxique dans un objet arbre (une racine, une branche à gauche et une branche à droite, plus l'étendu sur lequel l'étiquetage est valide). La fonction finale pour avoir le parsing sous le format initial est donné par la méthode `parse_line`.

2.5. Améliorer le résultat grâce à des libraires complémentaire

Afin d'améliorer le résultat final obtenu, je laisse à l'utilisateur la possibilité d'utiliser le PoS Tagger de standford comme possible complément à mon dictionnaire qui associe un symbole terminale au vocabulaire (cet algorithme est utilisé uniquement pour trouver un symbole associable à une ancre pas présentes dans le lexique). J'offre aussi la possibilité de s'appuyer sur la librairie enchant afin de corriger les fautes d'orthographes présentes dans le texte avant de faire le parsing (cette option fonctionne moyennement car la correction n'est pas toujours très bonne).

3. Résultats et tests

L'entrainement de la grammmaire sera faite sur 80% de la base SEQUOIA (on prendra les 80% premier), et 10% sera utilisé pour les tests (comme requis).

3.1. Evaluer numériquement le modèle

Il aurait été intéressant d'obtenir une valeur numérique sur les performances de mon algorithme. Néanmoins, il est très difficile d'évaluer de manière automatique un modèle puisque notre algorithme peut par exemple créer un regroupement en trop ce qui provoquera un décalage par rapport au parsing proposé dans SEQUOIA et il faudrait donc trouver l'alignement idéal. Afin d'évaluer l'algorithme créé il me semble donc raisonnable d'analyser quelques exemples tiréd parmis la base de test.

3.2. Evaluer sur la base de test

Phrase Originale : Affaire politico-financière
Prediction : ((SENT (NC Affaire) (AP (NC politico) (PONCT -)) (ADJ financière))))
Solution : ((SENT (NP (NC Affaire) (AP (PREF politico-) (ADJ financière)))))

Phrase Originale : Wikipédia : ébauche droit.
Prediction : ((SENT (NP Wikipédia) (PONCT :)) (NP (NC ébauche) (AP droit))) (PONCT .)))
Solution : ((SENT (NP (NPP Wikipédia)) (PONCT :) (NP (NC ébauche) (NC droit)) (PONCT .)))

3.3. Evaluer sur des exemples illustratifs

Phrase Originale : Je vais au cllege.
Prediction : ((SENT (VN (CLS Je) (V vais)) (PP (P+D au) (NP (NC cllege) (PONCT .)))))

On observe que l'algorithme fonctionne dans la majorité des cas même si l'on ne prédit pas exactement la même chose que la vraie solution (on ne peut pas prédire les unit productions par exemple). La correction orthographique permet de corriger certains points (comme dans l'exemple précédent) mais elle mène à des absurdités parfois. Il est préférable de ne pas l'utiliser dans un premier temps. Le tiret est séparé de politico à cause de la manière avec laquelle je fais ma tokenization. Il faudrait mieux gérer certains cas particuliers.

3.4. Amélioration envisageable

Afin d'améliorer l'algorithme on pourrait déjà commencer par utiliser une base d'entrainement plus grande pour améliorer la qualité de notre grammaire. Utiliser une base plus grande permettrait d'ajouter les labels fonctionnelles qui sont améliorants d'après [1]. On pourrait également effectuer la correction orthographique dans les cas où l'on est certain de la correction.

4. Conclusion

Ce TD, bien que long et assez prennant lorsque l'on veut tout implémenter seul aura été une excellente occasion de manipuler les Parser. On voit que l'on peut obtenir des résultats très intéressants en utilisant les préceptes du cours. On voit aussi comment l'on pourrait combiner le TD 3 avec celui là et à quoi aurait pu servir le préprocessing.

Références

[1] D. Jurafsky and J. H. Martin. *Speech and language processing*, volume 3. Pearson London :, 2014.