08/12/2019

Rapport du projet

Modèles de Linguistique Computationnelle

Algorithme de CKY

Fintan Herlihy & GAO Shuai
SORBONNE UNIVERSITÉ

Table de matières

1. Introduction

De nos jours, le développement de la technologie et le foisonnement des outils informatiques nous facilitent grandement la vie. Entourés d'ordinateurs, de tablettes et surtout de nos smartphones, partenaires omniprésents semble-t-il, nous sommes en permanence connectés à un cyberespace, lequel, d'une façon ou une autre, interagit avec nous. Des claviers à boutons physiques, aux écrans tactiles et finalement aux assistants vocaux, la forme d'interactions homme-machine a connu de grands changements.

Il est à noter que cette interaction tend à se produire à travers la langue naturelle, laquelle est manifestement le moyen de communication le plus privilégié par l'homme. Néanmoins, la machine a toujours été basée sur des calculs du code binaire, ce qui prouve, *a priori*, son incapacité de comprendre le sens d'une langue naturelle en soi. Alors, pourquoi assistons-nous à l'essor d'applications informatiques interactives touchant à des tâches cognitives ? Comment une machine, dans une certaine mesure, comprend notre langue ?

1.1. Traitement automatique de la langue

Tout cela est dû au traitement automatique des langues (TAL). Le TAL, aussi connu sous le nom du TALN, traitement automatique de la langue naturelle, consiste à modéliser et à reproduire, à l'aide de machines, la capacité humaine à produire et à comprendre des énoncés linguistiques dans une volonté de communication. Le TAL couvre bon nombre de disciplines de recherche telles que syntaxe, sémantique, traitement du signal, extraction d'informations et bibliométrie.

1.2. Analyse syntaxique automatique

Le fonctionnement du TAL basé sur des données linguistiques que collectionnent les programmes prévus à cet effet, l'analyse syntaxique automatique y occupe une place essentielle. En effet, quelques disciplines précitées que concerne un traitement et quel que soit son but, l'analyse syntaxique constitue un passage presque obligatoire. Elle consiste en premier lieu à décomposer les énoncés écrits ou éventuellement oraux en unités syntaxiques telles que syntagme nominal, syntagme verbal et complément de phrase, etc. C'est ce que l'on appelle la segmentation. Elle constitue avec en outre l'analyse morphologique, l'étiquetage ainsi que la désambiguïsation le processus de prétraitement d'un corpus.

1.2.1. Deux méthodes d'analyse

Fondamentalement, nous avons deux types de méthodes possibles pour l'analyse syntaxique automatique.

(1) L'analyse descendante, qui tient compte de l'ensemble de la phrase tout en présumant les formes possibles, et qui vérifie ensuite si elle correspond à l'une de ces formes ;
(2) L'analyse ascendante, qui commence à partir des unités de l'énoncé et essaie de les regrouper, et qui obtient une phrase en effectuant les regroupements possibles.

1.3. Défis dans l'analyse

La langue naturelle est un système complexe et évolutif de signes, qui permet la communication de l'humain. Même nous utilisateurs de langue rencontrons parfois des difficultés à se comprendre parfaitement, rien d'étonnant donc à ce que la machine fasse des erreurs en analysant notre langue. De plus, sa tâche pourrait parfois être volumineuse. Ainsi existe-t-il certains défis dans l'analyse syntaxique automatique.

(1) Ambiguïté

Généralement, l'ambiguïté peut être engendrée lors de l'analyse syntaxique de deux différentes manières.

Premièrement, les unités lexicales posent tout de suite des problèmes car il arrive souvent qu'une seule forme ait plusieurs interprétations possibles. Si l'on s'en tient à ce seul exemple, le mot « livre » peut être un nom tant masculin que féminin, ayant évidemment deux sens différents, et il peut également être la première et la troisième personne à présent du verbe « livrer ».

Deuxièmement, la structure de la phrase est aussi un facteur d'ambiguïté. Par exemple, « Paul regarde un homme avec un télescope. », dans cette phrase, nous ne savons pas qui tient le télescope. Elle peut être interprétée différemment en fonction du contexte.

(2) Tâches répétitives

Il est à noter que le processus d'analyser un énoncé est composé de plusieurs substructures d'analyse. Ces dernières sont généralement répétitives et entraînent par conséquent des tâches répétitives, ce qui baisse inévitablement l'efficacité du programme.

2. Algorithme de Coche-Younger-Kasami

Afin de remédier à l'inefficacité que peut avoir l'analyse syntaxique automatique, nous préférions employer l'algorithme de Coche-Younger-Kasami (CYK ou CKY), lequel opère par analyse ascendante et avec une stratégie dite « programmation dynamique ». Cet algorithme ne se fait que pour les grammaires non contextuelles en forme normale de Chomsky.

2.1. Programmation dynamique

La programmation dynamique consiste à éviter les tâches répétitives au moyen de l'enregistrement de solutions des substructures dans un tableur en forme de triangle.

2.2. Forme normale de Chomsky

Noam Chomsky a défini un type particulier de grammaires hors-contexte. On dit qu'une grammaire hors-contexte est en forme normale de Chomsky si et seulement si toutes ses règles de production sont conformes à la forme ci-dessous :

- $A \rightarrow B\ C$;
- ou $A \rightarrow a$;
- ou $S \rightarrow \varepsilon$

A, *B* et *C* sont des symboles non-terminaux, *a* le symbole terminal, S l'axiome de la grammaire et ε le mot vide. Toutes les grammaires hors-contexte peuvent être converties en forme normale de Chomsky, ce qui permet l'utilisation d'algorithmes efficaces pour faire l'analyse syntaxique.

2.2.1. Conversion en forme normale de Chomsky

Pour toute grammaire hors-contexte, il existe une grammaire hors-contexte équivalente sous forme normale de Chomsky. Nous effectuons en général deux étapes pour mettre une grammaire en son forme normale.

(1) Transformer les symboles terminaux en non-terminaux tant qu'ils ne sont pas seuls en partie droite.
(2) Réduire les parties droites.

Pour s'en tenir à ce seul exemple, soit une grammaire hors-contexte P :

$$P = \begin{cases} S \longrightarrow aAa & (1) \\ A \longrightarrow Sb & (2) \\ A \longrightarrow bBB & (3) \\ B \longrightarrow abb & (4) \\ B \longrightarrow aC & (5) \\ C \longrightarrow aCA & (6) \end{cases}$$

Premièrement, nous constatons que dans toutes les règles il y a des symboles terminaux qui ne sont pas seuls en partie droite. Nous ajoutons donc des symboles non-terminaux intermédiaires : X_a et X_b, ainsi que deux règles correspondantes.

$$X_a \longrightarrow a$$
$$X_b \longrightarrow b$$

Nous obtenons les résultats comme suit :

$$P = \begin{cases} S \longrightarrow X_a A X_a & (1) \\ A \longrightarrow S X_b & (2) \\ A \longrightarrow X_b B B & (3) \\ B \longrightarrow X_a X_b X_b & (4) \\ B \longrightarrow X_a C & (5) \\ C \longrightarrow X_a C A & (6) \\ X_a \longrightarrow a & (7) \\ X_b \longrightarrow b & (8) \end{cases}$$

Deuxièmement, afin que la partie droite de toutes les règles soit deux symboles non-terminaux ou un terminal, nous y ajoutons des non-terminaux intermédiaires : Y_1, Y_2, Y_3 et Y_4, ainsi que leurs règles correspondantes.

$$Y_1 \rightarrow A X_a$$
$$Y_2 \rightarrow X_b X_b$$
$$Y_3 \rightarrow C A$$
$$Y_4 \rightarrow B B$$

Finalement, nous obtenons :

$$P = \begin{cases} S \longrightarrow X_a Y_1 & (1) \\ A \longrightarrow S X_b & (2) \\ A \longrightarrow X_b Y_4 & (3) \\ B \longrightarrow X_a Y_2 & (4) \\ B \longrightarrow X_a C & (5) \\ C \longrightarrow X_a Y_3 & (6) \\ X_a \longrightarrow a & (7) \\ X_b \longrightarrow b & (8) \\ Y_1 \longrightarrow A X_a & (9) \\ Y_2 \longrightarrow X_b X_b & (10) \\ Y_3 \longrightarrow C A & (11) \\ Y_4 \longrightarrow B B & (12) \end{cases}$$

Nous constatons que la conversion d’une grammaire hors-contexte en son équivalence de forme normale de Chomsky engendre la multiplication des règles.

2.3. Démonstration d'algorithme de CKY

Soit une grammaire hors-contexte en forme normale de Chomsky, et une chaîne de caractères « baaba » :

- S → AB | BC
- A → BA | a
- B → CC | b
- C → AB | a

Nous procédons à l'analyse avec la programmation dynamique :

5	S, A, C				
4	-	S, A, C			
3	-	B	B		
2	S, A	B	S, C	S, A	
1	B	A, C	A, C	B	A, C
	b	a	a	b	a

3. Implémentation et mode d'emploi du programme

Un canevas d'un script en langage Python nous a été fourni afin de mettre en pratique l'algorithme décrit précédemment. Il est composé de plusieurs classes et fonctions permettant d'implémenter certains tests et affichages par rapport à une grammaire et un langage donné, à compléter de manière à ce qu'une fonction finale puisse s'exécuter pleinement sur ces deux éléments.

La complétion du code s'est effectuée de manière linéaire et dans l'ordre où apparaissent les fonctions appelées dans la méthode *parse*.

3.1. Déterminer si une grammaire est en FNC

La première étape était la fonction *checkNFC* qui s'assure qu'une grammaire est en forme normale de Chomsky.

```
"Vérifie que la grammaire est en forme normale de Chomsky"
def checkCNF(gr):
    for r in gr.rules:
        if (len(r.rhs) > 2):
            return False
        else:
            if (((len(r.rhs) == 1) and (r.rhs[0].__str__() == r.rhs[0].__str__().lower())) or ((r.rhs[0].__str__() == r.rhs[0]
                return True
            else:
                return False
```

On initialise premièrement une boucle *for* sur la classe *Rules* contenue dans la métaclasse *Grammar* : pour chaque élément de cette classe, on s'assure que la partie droite n'a au plus que deux symboles. Ensuite, si la partie droite ne contient qu'un symbole et que ce symbole est un terminal, ou si cette même partie contient deux symboles et qu'ils sont non terminaux, la fonction retourne vrai, sinon faux.

3.2. Initialisation d'un tableau d'analyse T

Pour ensuite initialiser l'objet T qui correspond à la table dans laquelle s'effectuera l'analyse du mot par l'algorithme CKY, nous faisons appel à la fonction *init*, qui prend en argument un mot et une grammaire.

Nous avons décidé, contrairement au code qui nous a été fourni initialement qui prévoyait la création d'un dictionnaire, de créer une matrice composée d'une liste contenant des listes de listes ; notre but était donc d'avoir une manière d'accéder aux éléments contenus dans les listes de manière ordonnée, même si cela pourrait augmenter le temps que prendrait l'analyse pour s'effectuer.

La structure de cette table T est la suivante : une première liste, à l'intérieur de laquelle se trouve des listes (dans un souci de clarté nous les appellerons rangées), au sein desquelles sont disposées des listes (ou regroupements) de symboles ; par exemple, pour une partie droite d'une règle d'une grammaire donnée, on peut trouver deux symboles non terminaux, ou un terminal, correspondant à un symbole non terminal dans la partie de gauche. Nous aurons donc la possibilité, grâce aux regroupements, de pouvoir accéder à plusieurs symboles correspondant à la partie droite d'une règle.

À la suite de l'initialisation de cette matrice, nous effectuons une première analyse du mot en le parcourant indice par indice grâce à un boucle *for,* comparant chaque caractère avec la partie droite d'une grammaire (transformée en string grâce à la fonction *__str_*) contenue dans la classe *Symbol*. S'il y a correspondance dans cette comparaison, le non terminal contenu dans la partie gauche de la règle (dans laquelle il y a le symbole terminal à droite) est ajouté au sac du même indice, au niveau de la première rangée. Nous avons procédé de cette manière car dès que cette première analyse est faite il n'y a plus recours au mot, c'est à dire qu'il ne reste à priori plus de symboles terminaux à analyser.

3.3. Analyse CKY

L'initialisation de la table effectuée, il devient question d'implémenter l'algorithme d'analyse CKY ; il aurait été voulu qu'une première boucle for itère de 0 à la longueur du mot, une deuxième dans la longueur inverse (à partir de la longueur du mot mois un), et une troisième de 0 à la valeur courante de la première boucle, ceci de manière à ce que trois pointeurs (qu'on nommera A, B et R respectivement) permettent pour les deux premiers de sélectionner deux cases distinctes du tableau, qu'une boucle soient lancées sur les parties droites des règles de la grammaire et que si les valeurs concaténées des deux cases précédemment sélectionnées correspondent à une partie droite, que la partie gauche soient inscrite dans la case sélectionnée par le pointeur R, ceci toujours une rangée au-dessus de A. A monterait dans les sacs rangées par rangées, puis passerait à la prochaine colonne de sacs, tandis que B descendrait diagonalement dans les rangées et les sacs de mots, jusqu'à la longueur du mot moins un ; le mouvement de A et B est simultané et cadencé par la longueur des sous-mots à analyser.

3.4. Imprimer la table générée

Pour imprimer le résultat de l'analyse, une boucle *for* est utilisée afin d'imprimer la table rangée par rangée, en convertissant les symboles.

3.5. Déterminer si le mot est généré par la grammaire

L'étape finale de l'analyse consiste à déterminer si un mot est oui ou non généré par une grammaire, ou dans d'autres mots si le symbole contenu au niveau du premier sac de la dernière rangée du tableau d'analyse correspond à l'axiome permettant de générer un mot dans la grammaire.

Une fonction *isSuccess* y est dédiée ; en prenant en argument la grammaire et la table d'analyse, nous récupérons le symbole à l'indice correspondant dans la table et le comparons avec l'axiome contenu dans la grammaire ; s'il y a égalité, la fonction retourne vrai, faux sinon.

4. Conclusion

La complétion du code s'est effectuée de manière intuitive pour ce qui est des fonctions qui créent un tableau et l'impriment, ainsi que celles qui s'assurent qu'une grammaire est en forme normale de Chomsky et que le mot est généré par cette dernière.

Le choix de créer une matrice tridimensionnelle composée de listes peut paraître contre-intuitif quant au coût en temps de l'opération, mais cela nous permet de plus facilement accéder aux divers éléments d'un sac, et ce de manière ordonnée.

Les difficultés rencontrées ont surtout été de l'ordre de l'implémentation de l'algorithme CKY au niveau du choix des indices pour les boucles for, peut-être dû à notre décision de faire une analyse séparée du mot premièrement et des symboles non terminaux générés dans un second temps.

5. Contributions par personne

5.1. Fintan Herlihy

J'ai pris en main le code, étant doté de plusieurs années d'expérience dans l'univers de la programmation, et donc des notions me permettant de comprendre la structure et le contenu du canevas qui nous a été fourni. J'ai par la suite décrit l'implémentation du programme et les difficultés auxquelles je me suis heurté.

5.2. GAO Shuai

Comme je suis nouveau en programmation, je ne suis pas très à l'aise avec le code d'autant plus que la tâche n'est pas facile même pour les anciens LFTIs. Je me suis alors chargé de la rédaction de l'introduction, de la présentation de l'algorithme de CKY et de toute la mise en forme du rapport.

6. Rapport des activités

Date	Activité	Durée
30-10-2018	Lecture de l'explication de FNC et de CKY sur Youtube	1 heure
03-11-2018	Lecture dans Jurafsky et Martin	2 heures
07-11-2018	Implémentation CheckNFC	2 heures
11-11-2018	Lecture des vidéos de Jurafsky sur Youtube	2 heures
15-11-2018	Initialisation de la table d'analyse	3 heures
17-11-2018	Rédaction de l'introduction générale	1.5 heures
22-11-2018	Tentative d'implémenter l'analyse CKY/Implémentation PrintT	5 heures
23-11-2018	Rédaction de parties 1.1 et 1.2	2 heures
25-11-2018	Rédaction de parties 1.3 et 2	3 heures
27-11-2018	Rédaction de parties 2.1	3 heures
30-11-2018	Implémentation isSuccess	2 heures
03-12-2018	Tentative d'implémenter l'analyse CKY	6 heures
04-12-2018	Rédaction de partie 2.2 et 2.3	3 heures
08-12-2018	Mise en page et la bibliographie	2 heures
08-12-2018	Mise en commun	1 heure

Bibliographie et Sitographie

- JURAFSKY, Daniel et James H. MARTIN. Speech and Language Processing, Pearson, 2018, 558 pages.
- https://fr.wikipedia.org/wiki/Programmation_dynamique
- https://fr.wikipedia.org/wiki/Algorithme_de_Cocke-Younger-Kasami
- https://bu.univ-ouargla.dz/master/pdf/DJABALLAH-Souhila.pdf?idmemoire=4357
- https://www.parisnanterre.fr/medias/fichier/anasynaut_1286466149954.pdf
- http://www.lifl.fr/~rouillar/publi/1998_Rouillard_Caelen_CAPS.PDF
- http://gurau-audibert.hd.free.fr/josdblog/wp-content/uploads/2011/12/TAL_ITCN.pdf
- http://courses.washington.edu/ling571/ling571_WIN2016/slides/ling571_class3_cnf_cky_flat.pdf
- http://www.schplaf.org/kf/pdf/FNC.zip