Laboratoire nltk : Parseur de groupes nominaux

1. Introduction

Écrivez un programme d'IA pour analyser syntaxiquement des phrases et d'extraire des phrases nominales.

A rendre (déposer dans la **fad.univ-thies.sn**) au plus tard le : **30/06/2021**

2. Contexte

Une tâche courante dans le traitement du langage naturel est l'analyse syntaxique, qui consiste à déterminer la structure d'une phrase. Cette tâche est utile pour plusieurs raisons : connaître la structure d'une phrase peut aider l'ordinateur à mieux comprendre le sens de la phrase, et peut également aider l'ordinateur à extraire des informations d'une phrase. En particulier, il est souvent utile d'extraire des expressions nominales d'une phrase pour comprendre le sens de la phrase.

Dans ce problème, nous allons utiliser le formalisme de la grammaire hors contexte pour analyser des phrases anglaises afin de déterminer leur structure. Rappelons que dans une grammaire hors contexte, nous appliquons de manière répétée des règles de réécriture pour transformer des symboles en d'autres symboles. L'objectif est de commencer avec un symbole non terminal S (représentant une phrase) et d'appliquer de manière répétée les règles de la grammaire hors contexte jusqu'à ce que nous générions une phrase complète de symboles terminaux (c'est-à-dire des mots). La règle $S \rightarrow N\ V$, par exemple, signifie que le symbole S peut être réécrit comme $N\ V$ (un nom suivi d'un verbe). Si nous avons également la règle $N \rightarrow$ *"Holmes"* et la règle $V \rightarrow$ *"sat"*, nous pouvons générer la phrase complète ***"Holmes sat"***.

Bien sûr, les phrases nominales ne sont pas toujours aussi simples qu'un simple mot comme "*Holmes*". Nous pouvons avoir des phrases nominales comme "*my companion*", "*a country work*", "*the day before Thursday*", qui nécessitent des règles plus complexes pour être pris en compte. Pour tenir compte de l'expression "*my companion*", par exemple, nous pourrions imaginer une règle du genre :

```
NP -> N | Det N
```

Dans cette règle, nous disons qu'un NP (un "groupe nominal") peut être soit un simple nom (N), soit un déterminant (Det) suivi d'un nom, les déterminants comprenant des mots comme "a", "the" et "my". La barre verticale (|) indique simplement qu'il existe plusieurs façons possibles de réécrire un NP, chaque réécriture possible étant séparée par une barre.

Pour incorporer cette règle dans la façon dont nous analysons une phrase (S), nous devons également modifier notre règle $S \rightarrow N\ V$ pour permettre aux phrases nominales (NPs) d'être le sujet de notre phrase. Vous voyez comment ? Et pour prendre en compte des types de syntagmes nominaux plus complexes, nous devrons peut-être modifier notre grammaire encore davantage.

Téléchargez le code de distribution à partir de https://cdn.cs50.net/ai/2020/x/projects/6/parser.zip et dé zipper le.

Dans le répertoire *parser*, exécutez `pip3 install -r requirements.txt` pour installer la dépendance de ce projet : `nltk` for natural language processing.

3. Détails

Tout d'abord, regardez les fichiers texte dans le répertoire *sentences*. Chaque fichier contient une phrase en anglais. Votre objectif dans ce problème est d'écrire un analyseur syntaxique capable d'analyser toutes ces phrases.

Regardez maintenant le fichier *parser.py*, et remarquez les règles de grammaire hors contexte définies en haut du fichier. Nous avons déjà défini pour vous un ensemble de règles pour générer des symboles terminaux (dans la variable globale *TERMINALS*). Remarquez que *Adj* est un symbole non terminal qui génère des adjectifs, *Adv* génère des adverbes, *Conj* génère des conjonctions, *Det* génère des déterminants, *N* génère des noms (répartis sur plusieurs lignes pour plus de lisibilité), *P* génère des prépositions, et *V* génère des verbes.

Ensuite, il y a la définition de *NONTERMINALS*, qui contiendra toutes les règles de la grammaire hors contexte pour générer des symboles non terminaux. Pour l'instant, il n'y a qu'une seule règle : *S* → *N V*. Avec cette seule règle, nous pouvons générer des phrases comme *"Holmes arrived."* ou *"He chuckled."*, mais pas des phrases plus complexes que cela. C'est à vous de modifier les règles *NONTERMINALS* pour que toutes les phrases puissent être analysées !

Ensuite, regardons la fonction *main*. Elle accepte d'abord une phrase en entrée, soit à partir d'un fichier, soit via une entrée utilisateur. La phrase est prétraitée (via la fonction *preprocess*) et ensuite analysée selon la grammaire hors contexte définie par le fichier. Les arbres résultants sont imprimés, et tous les morceaux de phrases nominales (définis dans la spécification) sont également imprimés (via la fonction np_chunk).

En plus d'écrire des règles de grammaire sans contexte pour analyser ces phrases, les fonctions preprocess et np_chunk sont laissées à votre disposition !

4. Indications

Complétez l'implémentation de *preprocess* et *np_chunk*, et complétez les règles de grammaire hors contexte définies dans NONTERMINALS.

- La fonction *preprocess* doit accepter une phrase en entrée et retourner une liste de mots en minuscules.
    - Vous pouvez supposer que la phrase sera une chaîne de caractères.
    - Vous devez utiliser la fonction *word_tokenize* de *nltk* pour effectuer la tokenisation.
    - Votre fonction doit retourner une liste de mots, où chaque mot est une chaîne de caractères en minuscules.
    - Tout mot ne contenant pas au moins un caractère alphabétique (par exemple . ou 28) doit être exclu de la liste retournée.
- La variable globale *NONTERMINALS* doit être remplacée par un ensemble de règles de grammaire hors contexte qui, lorsqu'elles sont combinées avec les règles de *TERMINALS*, permettent d'analyser toutes les phrases du répertoire *sentences*/.
    - Chaque règle doit être sur sa propre ligne. Chaque règle doit inclure les caractères → pour indiquer quel symbole est remplacé, et peut éventuellement inclure les symboles | s'il existe plusieurs façons de réécrire un symbole.

    - Il n'est pas nécessaire de conserver la règle existante *S → N V* dans votre solution, mais votre première règle doit commencer par *S →* puisque *S* (représentant une phrase) est le symbole de départ.
    - Vous pouvez ajouter autant de symboles non terminaux que vous souhaitez.
    - Utilisez le symbole non terminal *NP* pour représenter un "syntagme nominal", tel que le sujet d'une phrase.
- La fonction *np_chunk* doit accepter un arbre (***tree***) représentant la syntaxe d'une phrase, et retourner une liste de tous les morceaux de syntagmes nominaux dans cette phrase.
    - Pour ce problème, un "*groupe nominal*" est défini comme un groupe nominal qui ne contient pas d'autres groupes nominaux. De manière plus formelle, un groupe nominal est un sous-arbre de l'arbre original dont l'étiquette est NP et qui ne contient pas lui-même d'autres groupes nominaux comme sous-arbres.
        - Par exemple, si *"the home"* est un groupe nominal, alors *"the armchair in the home"* n'est pas un groupe nominal, car ce dernier contient le premier comme sous-arbre.
    - Vous pouvez supposer que l'entrée sera un objet *nltk.tree* dont l'étiquette est *S* (c'est-à-dire que l'entrée sera un arbre représentant une phrase).
    - Votre fonction doit retourner une liste d'objets *nltk.tree*, où chaque élément a l'étiquette *NP*.
    - Vous trouverez probablement la documentation de *nltk.tree* utile pour identifier comment manipuler un objet *nltk.tree*.
- Vous ne devez rien modifier d'autre dans *parser.py* que les fonctions que la spécification vous demande d'implémenter, bien que vous puissiez écrire des fonctions supplémentaires et/ou importer d'autres modules de la bibliothèque standard Python. Vous devrez modifier la définition de NONTERMINALS, mais vous ne devez pas modifier la définition de TERMINALS.

5. Indications

Il est normal que votre analyseur syntaxique génère des phrases qui, selon vous, ne sont pas syntaxiquement ou sémantiquement bien formées. Vous ne devez donc pas vous inquiéter si votre analyseur syntaxique permet d'analyser des phrases sans signification comme *"His Thursday chuckled in a paint."*

- Cela dit, vous devez éviter la sur-génération de phrases dans la mesure du possible. Par exemple, votre analyseur syntaxique ne doit absolument pas accepter des phrases du type *"Armchair on the sat Holmes."*
- Vous devez également éviter la sous-génération de phrases. Une règle comme *S → N V Det Adj Adj Adj N P Det N P Det N* réussirait techniquement à générer la phrase 10, mais pas d'une manière qui soit particulièrement utile ou généralisable.
- Les règles du code source fourni sont (intentionnellement) un ensemble de règles très simplifiées, et par conséquent peuvent souffrir de sur-génération. Vous pouvez (et devriez) apporter des modifications à ces règles pour essayer d'être aussi général que possible sans sur-générer. En particulier, considérez comment vous pourriez faire accepter à votre analyseur syntaxique la phrase "Holmes sat in the armchair." (and "Holmes sat in the red armchair." and "Holmes sat in the little red armchair."), mais qu'il n'accepte pas la phrase "Holmes sat in the the armchair."

- Il faut s'attendre à ce que votre analyseur syntaxique puisse générer plusieurs façons d'analyser une phrase. La grammaire anglaise est intrinsèquement ambiguë !
- Dans la documentation de *nltk.tree*, vous pouvez trouver les fonctions *label* et *subtrees* particulièrement utiles.
- Pour vous concentrer sur le test de votre analyseur syntaxique avant de travailler sur le découpage des syntagmes nominaux, il peut être utile de demander temporairement à *np_chunk* de retourner simplement une liste vide [], afin que votre programme puisse fonctionner sans découpage des syntagmes nominaux pendant que vous testez les autres parties de votre programme.

6. Soumission

Le dépôt se fera en ligne dans la plateforme https://fad.univ-thies.sn/. Vous êtes enrôlé dans l'espace du cours. Vous pouvez aussi partager avec moi votre espace Git contenant le projet.