Projet syntaxe et sémantique

Cours "Syntaxe et sémantique" - Master 2 IdL - S. Aït-Mokhtar

07/12/2022

Projet individuel à rendre au plus tard le **10 janvier 2023**.

1 Introduction

La tâche consiste à écrire un programme en python 3 qui utilise l'analyse syntaxique et les plongements lexicaux pour extraire les mentions de type *nourriture*, *boisson* et *service*, à partir d'avis sur des restaurants écrits par des clients.

On utilisera spaCy pour l'analyse syntaxique et la librairie gensim de Python pour l'accès au plongements lexicaux.

Les sections suivantes définissent plus précisément ce qu'il faut extraire et détaillent la méthode à suivre. Les versions des librairies à installer sont également précisées. Il est très important d'utiliser les mêmes versions car l'évaluation compare automatiquement les extractions de votre programme à des extractions de référence produites avec des versions spécifiques des librairies et modèles utilisés – si vous n'utilisez pas les mêmes versions, cela peut causer des différences supplémentaires entre les extractions de votre programme et celles de référence et ainsi diminuer le score d'évaluation.

Le rendu sera évalué en prenant en compte les éléments suivants donnés par ordre d'importance :

— La précision et le rappel (couverture) des extractions.

— Son efficacité computationnelle (vitesse d'exécution et mémoire requise).

Des points de pénalités seront appliqués sur la note finale en cas de retard dans l'envoi du rendu de projet ou si le programme ne fonctionne pas comme spécifié.

2 Extraction de triplets

Le projet consiste à écrire un programme python qui extrait des triplets (a, t, c) à partir de textes, où :

— a est un des trois aspects du restaurant : *nourriture*, *boisson* ou *service*.

— t est le terme qui dénote l'aspect (on prendra seulement le mot principal si le terme est multi-mots). Par exemple, dans *"les tomates farcies étaient délicieuses"*, l'aspect est *nourriture* et le terme est *tomates*.

— c est une expression qui donne une caractérisation objective ou subjective du terme d'aspect et donne ainsi une information ou un jugement sur l'aspect a. Dans la phrase précédente, c est le mot *délicieuses*.

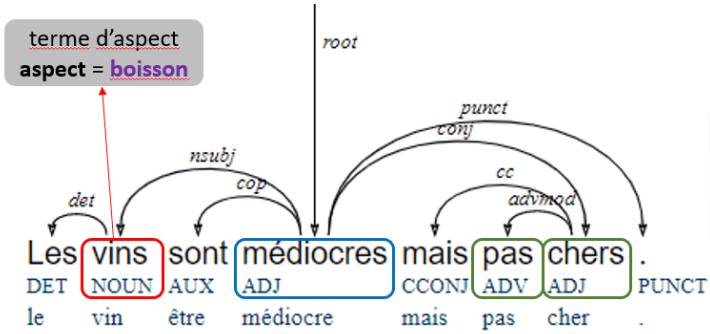
Dans la phrase *"les tomates farcies étaient délicieuses"*, il faudrait donc extraire le triplet *(nourriture, tomates, délicieuses)*.

Pour identifier les termes d'aspect, une méthode python est déjà fournie pour que vous l'utilisiez directement. Elle se fonde sur les similarités entre plongement lexicaux des tokens des phrases avec ceux des mots définis comme exemples de chacun des trois aspects.

Il ne s'agit pas d'extraire tous les cas de mentions des trois aspects (*nourriture*, *boisson* ou *service*), ce serait trop difficile et même impossible. On se limitera par conséquent à un petit sous-ensemble de cas, définis à travers **les contraintes syntaxiques suivantes** :

1. Le terme d'aspect t doit être un nom commun (catégorie grammaticale NOUN de UD2)
2. L'expression c est soit :
   (a) Un adjectif qualificatif qui modifie le terme t, donc il y a une dépendance syntaxique de type *amod* entre le terme t et l'expression e.
   (b) Un attribut adjectival du sujet dans une clause avec un verbe copule (être, sembler, etc.) Dans ce cas, l'analyse syntaxique produit une dépendance de type *nsubj* ou *nsubj:pass* entre l'adjectif et le terme t de l'aspect.
   (c) Un adjectif (ADJ) a_2 qui est coordonné avec un adjectif a_1 remplissant une des deux conditions précédentes : il y a donc une relation de dépendance de type *conj* entre a_1 et a_2, et a_1 est soit un adjectif qualificatif ou un attribut adjectival du terme t de l'aspect, et a_2 ne porte pas sur un autre terme que t.
   (d) Si dans l'une des trois situations précédentes, le mot extrait a un modifieur adverbial "négatif" dont le lemme est un des éléments {*pas, non, peu, guère, mal, trop*} , alors c doit être préfixé avec ce lemme. Par exemple, dans la phrase *Vin pas vraiment cher*, si l'analyseur syntaxique produit un arc de dépendance $(cher, advmod, pas)$, alors l'expression c à extraire est *pas_cher* et donc le triplet complet à extraire est *(boisson, vin, pas_cher)*.

Afin de montrer comment exploiter ces contraintes pour extraire les triplets *(a, t, c)* à partir de textes, la figure 1 ci-dessous décrit dans le détail comment elles s'appliquent sur une phrase analysée syntaxiquement avec spaCy.



Triplets (a, t, c) à extraire :

Aspect	Terme	Caractérisation
boisson	vins	médiocres
boisson	vins	pas_chers

FIGURE 1 – L'analyseur syntaxique produit ce graphe de dépendances pour la phrase *Les vins sont médiocres mais pas chers*. Grâce à la méthode *get_aspect_emb(token)* fournie dans le fichier de code distribué, et qui se fonde sur la similarité des vecteurs des plongements lexicaux des mots, on identifie le token *vins* comme un terme qui dénote l'aspect *boisson*. La contrainte 1 est satisfaite car le terme est un nom commun (NOUN). On cherche donc si ce terme remplit une des contraintes concernant la caractérisation : on trouve que le terme a un attribut adjectival (*médicores*) dans une clause à copule dont il est le sujet (dépendance *nsubj*). La contrainte 2b est donc satisfaite et on identifie *médiocres* comme une caractérisation du terme d'aspect. On vérifie alors la contrainte du modifieur négatif 2d et on trouve que *médiocres* n'est pas modifié par un adverbe négatif, et donc la caractérisation du terme d'aspect se limite à *médiocres*, sans préfixe négatif. On extrait ainsi le triplet *(boisson, vins, médicores)*. Ensuite, puisque l'on a identifié un token (adjectif *médiocres*) comme une caractérisation dans un triplet, on vérifie la contrainte 2c pour les cas de coordination : si l'on a une dépendance de type *conj* qui lie ce token à un autre token de catégorie ADJ, et qui n'est pas rattaché syntaxiquement (avec *nsubj*) à un autre terme, alors on doit le considérer comme une autre caractérisation du terme *vins* de l'aspect *boisson* et donc extraire un autre triplet. Le token *chers* satisfait cette contrainte. On doit vérifier alors la contrainte du modifieur négatif 2d et on trouve en effet que dans le graphe de dépendances, il y a bien une relation *advmod* entre *chers* et l'adverbe négatif *pas*. On préfixe donc *chers* avec *pas*, pour obtenir *pas_chers* comme la caractérisation du terme *vins* de l'aspect *boisson*, et on extrait ainsi le deuxième triplet *(boisson, vins, pas_chers)*.

La table 1 montre d'autres exemples avec moins de détails : pour chaque phrase avec un terme d'aspect identifié t, elle donne les contraintes satisfaites portant sur le terme d'aspect et sa caractérisation c, et enfin le triplet à extraire quand les contraintes sont satisfaites.

Il est important de noter que même un programme qui implémente parfaitement les contraintes syntaxiques définies ici va rater des extractions supposés être couvertes par ces définitions, et produira également des extractions incorrectes. La raison en est simple : la méthode d'identification des termes d'aspect n'est pas infaillible, et, surtout, l'analyseur spaCy commet des erreurs de segmentation en phrases des textes ainsi que des erreurs d'analyse syntaxique (catégories grammaticales des mots et/ou relations de dépendances incorrectes). Dans tous les cas, **il ne faut pas essayer de corriger les erreurs de spaCy ou de la méthode d'identification des termes d'aspect** : la performance des extractions sera évaluée en supposant que les analyses de spaCy sont correctes.

3 Comment procéder

D'abord, il faudrait installer les librairies python nécessaires (section 3.1) et ensuite télécharger les fichiers du projet (section 3.2). La méthode à suivre pour réaliser le programme d'extraction des triplets est présentée en section 3.3.

Phrase (terme d'aspect en gras)	Contraintes	Triplet à extraire (avec les contraintes)
1. Resto sympa et de très bonnes **grillades**.	1, 2a	nourriture, grillades, bonnes
2. **Service** sympathique et efficace.	1, 2a	service, Service, sympathique
3. **Service** sympathique et efficace.	1, 2a, 2c	service, Service, efficace
4. Il arrive souvent que les **plats** soient tièdes	1, 2b	nourriture, plats, tièdes
5. Le **plat** des coquilles Saint Jacques avec des pâtes a été correct.	1, 2b	nourriture, plat, correct
6. Les **pizzas** et les vins sont médiocres et pas chers.	1, 2b, 2c, 2d	nourriture, pizzas, pas_chers
7. Premier **plat** servi : insectes présents !	–	– (pas d'extraction)
8. L'accueil manque un peu de chaleur, le **service** qui suivra également.	–	– (pas d'extraction)
9. Le **boeuf** est d'une qualité fantastique.	–	– (pas d'extraction)
10. **Pizza** tellement fine qu'elle en est immangeable	–	– (pas d'extraction)

TABLE 1 – Exemples de phrases et les triplets à extraire si les contraintes syntaxiques sont satisfaites. Les termes d'aspect considérés sont en gras dans les phrases.

3.1 Installations à faire

Remarque : dans le cadre de ce projet, seules sont autorisées les librairies qui sont présentes dans la version de base d'anaconda et celles qui sont listées ci-dessous. On suppose que vous avez déjà installé python 3 (de préférence 3.9.x)

1. **Lancer un terminal de commandes *en mode administrateur***. La figure 2 indique comment procéder sur une machine Windows.

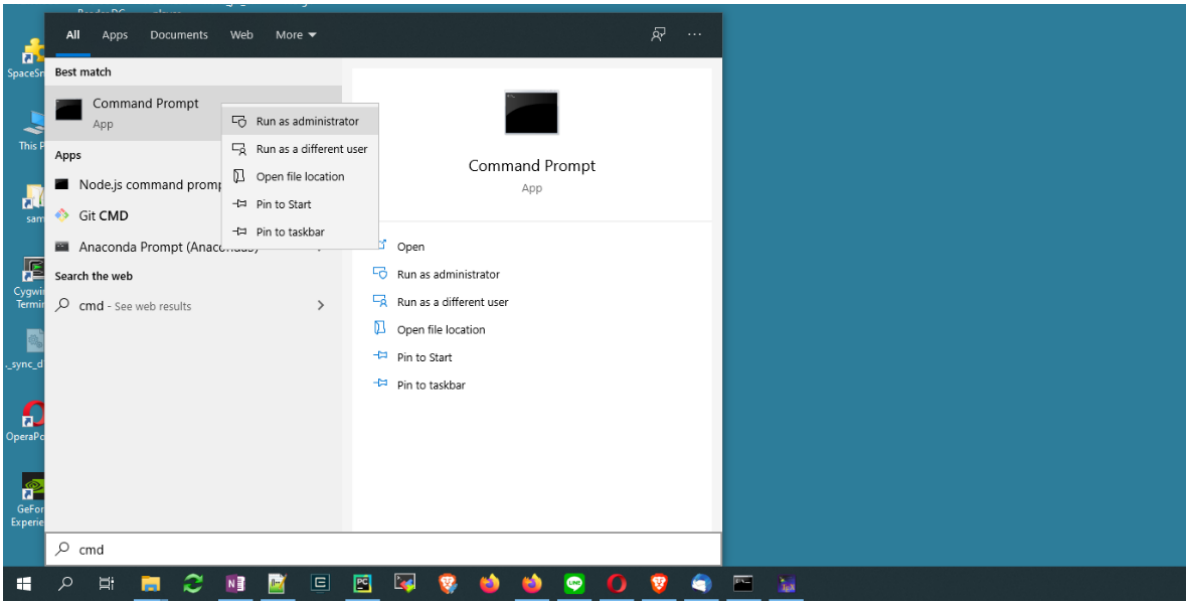


FIGURE 2 – Lancer un terminal de commandes en mode administrateur sur Windows

2. Installer gensim 4.2.0. Si vous travaillez avec Anaconda :

```
conda install gensim=4.2.0
```

Sinon :

```
pip install -U gensim==4.2.0
```

3. Installer spaCy 3.4.1. Si vous travaillez avec Anaconda :

```
conda install spacy=3.4.1
```

Sinon :

```
pip install -U spacy==3.4.1
```

4. Installer le modèle spaCy pour le français **fr_core_news_md** avec l'instruction suivante :

```
python -m spacy download fr_core_news_md
```

Attention : il faut absolument installer et utiliser la version **fr_core_news_md** du modèle et non une autre.

3.2 A télécharger

Dans le répertoire partagé du cours sur OneDrive, le sous-répertoire **projet** contient des éléments à télécharger :

— Ce document (**instructions_projet.pdf**) qui contient les instructions pour réaliser le projet.

— Un fichier de textes courts nommé **absa_rest_traindev.txt** qui contient des avis sur des restaurants. Chaque ligne contient un texte court (en général une seule phrases, parfois plusieurs). Vous pouvez tester et adapter votre programme en le lançant sur ce fichier pour faire les extractions de triplets.

— Un fichier de plongements lexicaux pour le français :

```
frWac_non_lem_no_postag_no_phrase_200_skip_cut100.bin
```

— Un fichier de code python nommé **leprogramme.py** où il faudrait écrire votre programme. Ce fichier contient déjà du code : des méthodes pour charger les plongements lexicaux avec gensim, charger le modèle de spaCy, ou pour déterminer si un token spaCy est un terme d'aspect ou non. Il faudra écrire votre programme dans ce fichier (en complétant le code). Le rendu de projet à remettre est simplement ce fichier de code, complété pour réaliser la tâche demandée. Lire les sections ci-dessous pour la méthode à suivre.

Téléchargez ces 4 fichiers et mettez-les dans un répertoire local sur votre ordinateur, que vous aurez créé pour le projet.

3.3 Méthode à suivre

1. Créer un répertoire pour le projet (par exemple **projet**) sur votre ordinateur et mettez dedans tout le contenu du sous-répertoire **projet** du dossier partagé du cours (voir section 3.2 ci-dessus)
2. Changer le nom du fichier **leprogramme.py** et le nommer **nom_prenom.py** où **nom** est votre nom de famille et *prenom* votre prénom. C'est dans ce fichier-là que vous écrirez votre programme. Il contient déjà du code, notamment pour identifier les tokens qui sont des termes d'aspect. Vous le complèterez pour tout le reste. Il est important de donner ce nom unique à votre fichier programme pour que les fichiers rendus ne soient pas confondus. Dans ce qui suit, je continuerai à l'appeller **leprogamme.py** par convenance.
3. Le programme doit s'exécuter de la manière suivante, à partir du répertoire principal du projet (qui contient aussi le fichier des plongements lexicaux) :

```
python leprogramme.py  textes.txt
```

où l'argument **textes.txt** est un fichier quelconque qui contient des textes courts, un texte par ligne. Chaque texte est constitué généralement d'une seule phrase, mais parfois de plusieurs. Le programme doit fonctionner quel que soit le nom du fichier de textes donné en argument.

4. Le programme doit extraire les triplets et sauvegarder les résultats au format json dans un fichier de sortie nommé **resultats.json** : l'objet json global est une liste qui contient les résultats des triplets extraits regroupés avec leur phrase d'origine. Le format exact est illustré par l'exemple de la figure 3.

Contenu du fichier de textes en entrée:

```
Les vins sont médiocres mais pas chers.
Il arrive souvent que les plats soient tièdes.
```

Contenu du fichier resultats.json (au format json):

```
[
    {
        "phrase": "Les vins sont médiocres mais pas chers.",
        "triplets": [
            [
                "boisson",
                "vins",
                "médiocres"
            ],
            [
                "boisson",
                "vins",
                "pas_chers"
            ]
        ]
    },
    {
        "phrase": "Il arrive souvent que les plats soient tièdes.",
        "triplets": [
            [
                "nourriture",
                "plats",
                "tièdes"
            ]
        ]
    }
]
```

FIGURE 3 – Exemple de résultats dans un fichier au format json : une liste globale dont chaque élément contient une phrase et la liste des triplets (qui peut être vide) qui en ont été extraits. L'ordre des phrases dans ce fichier est le même que dans les textes d'origine.

5. Pour réaliser ces extractions, le programme doit d'abord lire les textes du fichier donné en argument et les mettre dans une liste. Rappel : on suppose que le fichier des textes contient un texte par ligne. Il faut ensuite donner cette liste de textes à spaCy pour l'analyser et retourner une collection de documents spaCy (spaCy est déjà chargé dans le code initial que je vous fournis) :

```
1          textes = ...                          # chargement des textes dans une liste
2          docs = spacy_nlp.pipe(textes)         # analyse des textes avec spaCy
```

6. Les extractions des triplets *(aspect, terme, caractérisation)* se faisant à l'intérieur des phrases, il faut donc traiter chaque phrase de chaque document (doc = texte analysé par spaCy) de la collection de documents retournées, en extraire des triplets s'il y en a, et les stocker dans une liste :

```python
resultats = []  # liste globale des résultats par phrase
for doc in docs:
    # pour chaque document 'doc'
    for sent in doc.sents:
        # pour chaque phrase 'sent' de 'doc', extraires les triplets, s'il y en a,
        # et les stocker dans une liste initialement vide
        triplets = []
        # extraction des triplets pour cette phrase
        for token in sent:
            # pour chaque token de la pharse, on vérifie si c'est un terme d'aspect
            aspect = get_aspect_emb(token)   # méthode fournie dans le code initial
            if aspect is not None:
                term = token.text
                # si c'est un terme d'aspect, on cherche si les contraintes syntaxiques
                # d'extraction sont satisfaites et on crée éventuellement des triplets
                # (aspect, term, c)  qu'on rajoute à la liste triplets (code à compléter
                # ici)
                ...
                ...
        # sauvegarde du résultat pour cette phrase
        resulat = {'phrase': sent.text, 'triplets': triplets}
        resultats.append(resultat)
```

7. Pour vérifier les contraintes syntaxiques à partir d'un token spaCy qui est un terme d'aspect, il suffit d'accéder aux tokens qui sont sa tête syntaxique et ses dépendants syntaxiques, ainsi qu'à leur lemme et catégorie grammaticale. Pour rappel, voici les méthodes d'accès à ces informations :

token.text	la forme du token (tel qu'il apparaît dans le texe d'origine)
token.lemma_	le lemme du token
token.pos_	la catégorie grammaticale du token (part-of-speech UD2)
token.head	le token qui est la tête syntaxique de ce token
token.dep_	le type de dépendance syntaxique qui lie ce token à sa tête
token.children	la liste (possiblement vide) des tokens dépendants syntaxiques de ce token

8. Les analyses syntaxiques de spaCy contiennent des erreurs de segmentation en phrases, de catégories grammaticales, de lemmatisation et évidemment dans les relations de dépendances. N'essayez pas de les corriger : l'idée est de supposer que tout est correct et d'implémenter le code qui permet d'exploiter les contraintes syntaxiques définies pour l'extraction des triplets, même si les erreurs de spaCy vont causer des erreurs (et des oublis !) dans l'extraction des triplets.

4 Rendu du projet

En guise de rendu de projet, merci de m'envoyer à l'adresse sacours@outlook.com le fichier de code **leprogramme.py** que vous aurez complété avec votre propre code produisant des extractions à partir de n'importe quel fichier de textes d'avis ayant le même format que celui fourni en exemple (**absa_rest_traindev.txt**). Attention : merci de renommer **leprogramme.py** en **nom_prenom.py** avant de l'envoyer.

On doit pouvoir exécuter ce programme en allant dans le répertoire du projet (où il y a également le fichier de plongements lexicaux) et en tapant la commande suivante :

```
python nom_prenom.py  fichier_textes
```