ENSAI – Traitement Automatique des Langues

[TP3 et TP4] Fouille du web et extraction d'informations

L'objectif de ces deux séances de TP, conçues comme un mini-projet, est de mettre en œuvre une application simple d'extraction d'information à partir d'un corpus acquis par écoute d'un flux RSS. L'application a pour objectif de rechercher de l'information sur les entités nommées apparaissant dans le corpus de manière à pouvoir analyser les relations qui peuvent exister entre ces entités (se situe à, dirige, etc.). Cette tâche est connue en TAL sous le nom d'extraction d'information. On s'interessera notamment aux cooccurrence des entités et à la recherche de relations du même type qu'un exemple donnée.

De manière indicative, on considère que la première séance est dédiée à la mise en place d'un programme de captation de corpus (*scrapping*), la seconde étant dédiée à son traitement. Pour la seconde partie, on utilisera le corpus scrappé `francetvinfo.json`[1].

Important : Les deux TP feront l'objet d'une remise de compte-rendu par binôme, sous la forme d'un *notebook* python largement commenté. *Cf.* instructions à la fin du sujet sur les attendus du compte-rendu. Ce sont les commentaires, l'analyse des méthodes, de leurs limites et des résultats qui nous intéressent, plus que le code lui-même.

Installation des packages et données nécessaires

En plus de `spaCy` que nous avons déjà utilisé dans les TP précédents, le TP utilise les modules `feedparser` et `newspaper` pour le *scrapping*. Dans `colab`, vous devez donc commencer par installer ces librairies qui ne sont pas présentes par défaut et récupérer les fichiers nécessaires :

```
!wget https://people.irisa.fr/Guillaume.Gravier/teaching/ENSAI/data/francetvinfo.json
!wget https://people.irisa.fr/Guillaume.Gravier/teaching/ENSAI/data/tvinfo-sources.json

!pip install feedparser
!pip install newspaper3k

!python -m spacy download fr_core_news_md
```

Après chargement de la chaîne de traitement `spaCy` pour le français `fr_core_news_md`, pensez à redémarrer l'environnement d'exécution de votre notebook sous `colab`.

Acquisition de données par écoute d'un flux RSS

Dans cette première partie, on cherche à mettre en place une application écoutant les flux RSS du site du site `francetvinfo.fr` de manière à récupérer les articles correspondant et pouvoir faire de la veille géopolitique et/ou économique à partir de ces données. L'application de *scrapping* construit donc progressivement une base de données d'articles telle que celle mise à votre disposition dans le fichier `francetvinfo.json` en lisant à intervalle régulier le flux RSS et en récupérant les articles que ne sont pas déjà présents dans la base de données. La liste des flux à écouter est donnée dans le fichier `tvinfo-sources.json`[2].

[1] `https://people.irisa.fr/Guillaume.Gravier/teaching/ENSAI/data/francetvinfo.json`, également disponible sur moodle

[2] `http://people.irisa.fr/Guillaume.Gravier/teaching/ENSAI/data/tvinfo-sources.json`

Récupération des fichiers RSS

La première étape consiste à lire les entrées d'un fichier RSS grâce à la librairie `feedparser`[3].

L'exemple ci-dessous vous montre comment récupérer un flux RSS et la liste des articles mentionnés en prenant ici comme exemple un flux du journal Le Monde.fr :

```
# (a) se premunir contre le bloquage de commande
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

# (b) recuperer le fichier RSS
import feedparser as fp
url = "https://www.lemonde.fr/rss/une.xml"
data = fp.parse(url)

# (c) visualiser les elements du fichier RSS
print(data.feed.title)
print(data.feed.published)

# (d) iterer sur les entrees du flux RSS
for item in data.entries:
  print(item.title, item.published, item.link)
```

À faire :

1. En vous aidant de l'exemple ci-dessus, écrire un premier programme (une cellule dans le notebook) qui réalise les opérations suivantes :
   - lecture de la base de données existantes `francetvinfo.json` – regardez au passage la structure de ce fichier `json` et les informations qu'il contient
   - identifier les informations disponibles dans le flux RSS de l'exemple ci-dessus et les éléments correspondants dans la structure de données retournées par `fp.parse`
   - pour chacun des flux RSS listés dans le fichier `tvinfo-sources.json` :
     - lire les données du flux RSS
     - scanner les articles et repérer ceux qui ne sont pas déjà présents dans la base de données[4]

Récupération des articles

On cherche maintenant à récupérer le contenu des articles listés dans le flux RSS et n'étant pas encore dans la base de données. On utilise pour cela la libraire `newspaper`[5] qui permet de *parser* les pages HTML (de sites d'actualité) de manière à n'en récupérer que les parties utiles.

L'exemple ci-dessous illustre comment récupérer le texte d'un article ainsi que l'URL de l'image l'illustrant :

[3] Documentation sur `https://pythonhosted.org/feedparser`
[4] L'URL mentionné dans le flux RSS est utilisé comme clé pour identifier l'article dans la base de données.
[5] Documentation sur `https://newspaper.readthedocs.io`.

```
import newspaper as np

article = np.Article('http://[adresse-article-a-remplacer].html')
article.download()
article.parse()

print(article.published_parse, article.title, article.authors)
print(article.text)
print(article.top_image)
```

À faire :

2. Pour l'un des articles de votre flux RSS, regardez le code HTML de la page web. Pouvez-vous identifier les zones contenant le titre de l'article ? le texte de l'article ? Que faudrait-il faire pour récupérer le texte de l'article ? En quoi le recours à la librairie `newspaper` est-il utile ?

3. Reprendre votre programme de *scrapping* et le compléter de manière à mettre à jour votre base de données avec le texte des articles du flux RSS et le lien vers l'image illustrant l'article lorsqu'il y en a une. Vous pouvez inclure d'autres informations qui vous semblent utile pour un traitement ultérieur. On attend pour cette question une fonction qui prend en entrée les flux RSS à lire et la base de données à mettre à jour et qui retourne la base de données mise à jour.

4. Les images étant susceptibles de ne pas rester sur le site web, il faudrait idéalement en faire une copie locale dans notre base de données pour pouvoir s'y référerer ultérieurement. Comment procéderiez-vous pour ce faire ? On ne vous demande pas de faire le programme correspondant, juste de donner l'idée de comment vous vous y prendriez.

5. Donnez une idée d'une (ou plusieurs) autre(s) source(s) que l'on pourrait-on écouter pour collecter des informations complémentaires sur ces articles ? Sans donner le code, donnez une idée de ce que l'on pourrait collecter sur ces sources et de la/des librairie(s) qui pourrai(en)t nous aider en cela.

Extraction d'information

Dans cette seconde partie, l'objectif est de mettre en place une chaîne d'extraction d'informations à partir de la base de données constituée par *scrapping* des flux RSS de francetvinfo.fr `francetvinfo.json`. Vous pouvez si vous préférez utiliser les données que vous avez collectées dans la première partie dans la mesure où il y a suffisamment d'articles (au moins 2 000) ou, mieux encore, la base de données mise à jour avec les données collectées dans la première partie du projet.

L'extraction d'informations vise à rechercher des informations sur un sujet précis en fouillant les textes. On s'interessera ici, comme c'est souvent le cas, aux entités nommées de personne, d'organisation et de lieu et aux relations qui peuvent exister entre de telles entités – *e.g.*, X est PDG de la société Y, la société Y est implanté dans la ville de Z, *etc*[6]. Les utilisations typiques de ce genre de programme sont la constitution de fiches de renseignement sur les entités (*knowledge base population*), la veille économique sur des marques et produits en *marketing* ou sur des sociétés, ou encore le *fact checking* dans le monde de la presse.

Extraction des entités nommées

La première étape consiste à appliquer la détection d'entités nommées sur l'ensemble des documents de la base. Vous utiliserez pour cela `spaCy` avec la chaîne de traitement `fr_core_news_md` pour le français. Pour mémoire, on rappelle dans l'exemple ci-dessous le fonctiononement de `spaCy` pour la détection d'entités nommées.

[6] Dans le monde de l'information, on se réfère souvent à la règle des 5W pour caractériser un événement : **w**ho, **w**here, **w**hen, **w**hat, **w**hy. Ce qui explique entre autres choses l'importance des entités nommées. On sait aussi qu'une grande majorité des requêtes effectuées sur un célèbre moteur de recherche concerne des entités nommées.

```
import spacy

# load pipeline and related models
process = spacy.load("fr_core_news_md")

# process text with pipeline
res = process("Jean Dupont est maire de Plouguemeur. Apple n'y a pas de locaux.")

# list e that were detected
for e in res.ents:
  print(e.text, e.start, e.end, e.label_)
```

À noter que chaque entité possède un type (`label`) qui identifie son rôle. Les attributs `start` et `end` correspondent respectivement à l'indice de début et de fin (+1) de l'entité dans la liste des *tokens* de `doc`, les *tokens* de l'entité pouvant être récupérée par `doc[e.start:e.end]`.

À faire :

7. À quoi correspondent les étiquettes IOB utilisées ? Expliquer brièvement une technique (description de la tâche, du modèle) pour la détection des entités nommées.

8. Écrire un programme qui traite l'ensemble des textes que vous aurez chargé depuis le fichier `francetvinfo.json` et stock pour chaque article les entités trouvées et les informations afférentes (type, span). On cherchera à optimiser le temps de traitement en utilisant la commande `pipe` et en n'activant dans la chaîne de traitement `spaCy` uniquement les traitemens nécessaires pour la tâche à effectuer. On rappelle que chaque article ne devra être traité qu'une et une seule fois par `spaCy` pour extraire les informations nécessaires : la suite exploite le résultat de cette analyse mais ne requiert pas de refaire l'analyse linguistique de la phrase (on sera cependant amené à reprendre cette phase d'analyse pour inclure de nouvelles informations par la suite).

Analyse des entités nommées

On cherche maintenant à analyser les entités nommées détectées à l'étape précédente, notamment leurs relations.

À faire :

9. Pour chaque type d'entité, déterminer les 20 plus fréquentes dans la collection de documents. Commentez le résultat obtenu. On mémorisera ces entités dans trois listes distinctes (une par type d'entité).

Une manière simple de détecter des relations entre deux entités consiste à regarder leur fréquence de co-occurrence au sein d'un même document, voire d'une même phrase. Une fréquence de co-occurrence élevée est dans la grande majorité des cas synonyme d'un lien entre les entités.

À faire :

10. Écrire une fonction qui prend en entrée une paire d'entités et les types correspondant et retourne le nombre de co-occurrences de ces deux entités au sein d'un même document dans la collection.

11. En partant des listes de la question 8, on utilisera la fonction de la question précédente pour chercher les paires d'entités de type différents qui apparaissent le plus souvent ensemble. Commentez les résultats retournés.

12. Reprenez l'analyse précédente en vous limitant aux co-occurrences au sein d'une même phrase pour plus de précision. Plusieurs manières simples d'accéder aux informations sur les entités et les phrases, en combinant les informations sur chacun des tokens, la segmentation en phrase et le résultat de la détection des entités en s'appuyant sur la notion de `span` (portion du document

initial). Les exemples suivants vous illustrent les informations dont vous disposez pour chaque `res` issu de l'analyse d'un texte par `spaCy`.

```
# list tokens with NER and sentence info
for token in res:
  print(token, token.ent_iob_, token.ent_type_, token.is_sent_start)

# list entities and the corresponding tokens
for e in res.ents: # e est un span du document
  print(e.text, e.start, e.end, e.label_, res[e.start:e.end])

# list sentences and related information (easiest way)
for s in res.sents: # s est un span du document
  print(s.start, s.end, [tok for tok in s], [e for e in s.ents])
```

Pour caractériser de manière simple la relation entre deux entités, on s'intéresse souvent au contexte dans lequel les deux entités apparaissent ensemble. On cherche souvent à caractériser ce contexte pour tenter de *typer* la nature de la relation – relation du type "est PDG de", "est dirigeant de", *etc.* – ou pour regrouper les relations de même type sans nécessairement les typer. On peut aussi utiliser la caractérisation entre deux entités dont la relation est connue pour recherche dans le corpus des paires d'entités entretenant la même relation : dans ce cas, on cherchera typiquement dans les contenus un patron syntaxique similaire à celui de la relation connue.

Dans un soucis de simplicité, on se limitera ici à caractériser la relation de manière un peu naïve par le verbe entre les deux entités lorsqu'il y en a un. En pratique, on utilise plutôt le résultat de l'analyse syntaxique pour caractériser la nature de la relation de manière plus fine.

À faire :

13. Parmi les relations correspondant aux paires d'entités dont la fréquence de co-occurrence est élevée (*cf.* question 9), on cherche à en trouver quelques unes qui peuvent être caractérisées par un verbe. Pour cela, écrivez une fonction qui prend en entrée deux entités et leurs types respectifs et qui analyse l'ensemble des documents où ces entités apparaissent dans la même phrase de manière à établir la liste des verbes (on prendra le lemme du verbe) qui apparaissent entre ces deux entités dans ces phrases.
14. En utilisant la fonction de la question précédente, déterminez quelques couples d'entités et un verbe associé qui vous paraissent faire sens.
15. En quoi les couples et verbes identifés à la question précédente peuvent-ils être utiles pour trouver de nouvelles relations et/ou grouper des relations de même nature ?
16. Comment pourrait-on affiner cette analyse en utilisant l'arbre de dépendance syntaxique ?

Quelques instructions pour le compte-rendu

Le compte-rendu sera fourni sous la forme d'un notebook jupyter (.ipynb) amplement commenté[7]. Pensez à préciser vos noms dans les commentaires en en-tête du notebook.

Gardez en mémoire qu'il s'agit avant tout d'un projet de NLP plus que d'un projet d'informatique : dans les commentaires, on s'attend surtout à trouver des questionnements et remarques concernant le traitement des données, les techniques de NLP mises en oeuvre, l'analyse des résultats obtenus, *etc.*, plus que des justifications de choix d'implémentation.

Vous veillerez en en-tête de chaque fonction à bien expliquer la démarche et ce que vous cherchez à faire – e.g., L'idée de la fonction ci-dessous est de bla bla bla. Pour cela, elle prend en entrée bla bla bla. Elle boucle sur l'ensemble des documents en regardant pour chacun bla bla bla.

[7] Pour exporter votre notebook Google colab au format .ipynb, `File > Download > Download .ipynb`

L'explicitation de votre démarche et de vos intentions est aussi, voire plus importante, que la qualité du code. En d'autres termes, faire du code qui marche et qui est relativement efficace (par exemple, inutile d'appeler `nlp()` plusieurs fois sur un même document), c'est nécessaire, mais ce n'est que le début d'une démarche NLP ! Nous évaluerons largement la qualité des explications fournies et des commentaires en lien avec les techniques de NLP et, de manière général, l'application développée et les résultats obtenus. N'hésitez pas à prendre un peu de recul et à élargir votre réponse en suggérant, par exemple, des améliorations de ce qui est proposé ou des biais dans la méthodologie proposée. Par ailleurs, les correcteurs doivent pouvoir comprendre ce que vous faites (ou tentez de faire) sans nécessairement avoir à faire tourner votre code, sans non plus passer trop de temps à faire du *reverse engineering* de programme.