EVIJV - utilisation de NLTK

François Bouchet

2018-2019

1 Installation et test de NLTK

1.1 Sur votre machine personnelle

1. S'assurer au préalable que Python 2.5+ ou 3.3+ est présent sur la machine (par exemple en tapant `python --version` dans l'invite de commande/une fenêtre de terminal).
   Note : si Python est installé mais que la commande `python` produit un message d'erreur disant qu'elle introuvable, vérifier si le chemin où se trouve `python.exe` (généralement `C:\Python`) a bien été ajouté à la variable d'environnement `PATH`.
2. Installer easy_install en récupérant puis en exécutant le fichier suivant :
   `https://bitbucket.org/pypa/setuptools/raw/bootstrap/ez_setup.py`
3. Installer pip en tapant `easy_install pip`
   Note : si easy_install a été correctement installé mais que la commande `easy_install` produit un message d'erreur disant qu'elle est introuvable, vérifier si le chemin où se trouve `easy_install.exe` (généralement `C:\Python\Scripts`) a bien été ajouté à la variable d'environnement `PATH`.
4. Installer Numpy (utile pour certaines fonctions de NLTK) :
   – sous Windows, en téléchargeant puis en exécutant le fichier suivant :
   `http://sourceforge.net/projects/numpy/files/latest/download`,

   – sous Linux, avec la commande `pip install -U numpy`
5. Installer NLTK en saisissant la commande suivante : `pip install -U pyyaml nltk`

1.2 Sur les machines de l'UPMC

— Récupérer l'archive `http://fbouchet.vorty.net/classes/evijv/2017/nltk.tar.gz`
— Décompresser dans un dossier local (e.g. ∼/EVIJV)
— Ajouter les packages contenus au PYTHONPATH :
`export PYTHONPATH="$PYTHONPATH:/fullPath/votreNom/EVIJV/"`

1.3 Tester NLTK et récupérer les données fournies

Lancez une invite de commande python en tapant `python`, puis

```
>>> import nltk
```

Pour récupérer les ensembles de données disponibles avec NLTK :

```
>>> nltk.download()
```

2 Analyse lexicale

Exercice 1

Considérons le corpus de chat (`text5`) :

(a) Trouver tous les mots de 4 lettres.

(b) En utilisant la distribution fréquentielle, afficher ces mots dans l'ordre décroissant de leur fréquence.

Exercice 2

Trouver les expressions permettant de trouver tous les mots de `text6` remplissant les conditions suivantes :

(a) Finissant en *hat*

(b) Contenant la lettre *z*

(c) Contenant la séquence de lettres *pt*

Le résultat devra être donné sous la forme d'une liste de mots `['mot1', 'mot2', ...]`.

Exercice 3

On s'intéresse maintenant à la fréquence non plus des mots mais des caractères.

(a) Compter la fréquence de chaque lettre de l'alphabet dans plusieurs textes du corpus. Qu'en penser ?

(b) NLTK vient également avec des textes dans d'autres langues, notamment la déclaration universelle des droits de l'homme dans plusieurs centaines de langues. Pour l'importer, saisir : `from nltk.corpus import udhr`
Parmi les langages disponibles, nous allons considérer par exemple `'French_Francais-Latin1'`, `'German_Deutsch-Latin1'`, `'English_Latin1'`, et `'Spanish_Espanol-Latin1'`.
Compter la fréquence de chaque lettre de l'alphabet dans ces nouvelles listes de mots. Qu'en penser ?

Exercice 4

Afficher pour chaque texte constituant le corpus Gutenberg :

(a) le nombre moyen de caractères par mots

(b) le nombre de mots par phrases

(c) la proportion de mots dans ce texte par rapport à tous les mots du corpus (i.e. quelle est la richesse linguistique de ce texte ?)

Exercice 5

Considérons les catégories `news`, `religion`, `hobbies`, `science_fiction`, `romance` et `humor` du corpus Brown :

(a) Déterminer les 10 bigrammes les plus fréquents dans chaque catégorie

(b) Afficher la fréquence de ces bigrammes dans chacune de ces catégories

Exercice 6

Nous allons maintenant travailler sur un corpus personnalisé :

(a) Récupérer 5 articles sur un site internet (e.g. Le Monde) dans une catégorie donnée (e.g. international), à sauvegarder dans 5 fichiers `.txt` séparés

(b) Faire de même pour 5 articles d'une autre catégorie ou d'un autre site

(c) Charger les 2 ensembles comme 2 corpus séparés

(d) Comparer les mots et bigrammes les plus fréquents

3 Étiquetage morpho-syntaxique

Lors de l'utilisation de tags, faites attention à l'ensemble que vous utilisez : les 36 tags de Penn Treebank (par défaut) ou les 19 de l'ensemble simplifié (chargement avec l'option `simplify_tags=true`). Les noms de certains tags peuvent varier légèrement (e.g. MOD ou MD, DET ou DT).

Exercice 1

En utilisant la fonction `ConditionalFreqDist`, déterminer si dans les textes de la catégorie news du corpus Brown :

(a) les mots 'test', 'cut' et 'ski' sont plus fréquemment des noms ou des verbes,

(b) les 5 verbes au passé (VD) les plus fréquents.

Exercice 2

Écrire le code nécessaire pour rechercher dans le corpus Brown dans son ensemble les mots et phrases étiquetés permettant de répondre aux questions suivantes :

(a) Produire une liste alphabétique des modaux (MD).

(b) Identifier les mots qui peuvent être soit des noms pluriels, soit des verbes à la 3e personne du singulier (e.g. flies).

(c) Identifier les expressions de la forme IN + DET + NN (e.g. in the lab).

(d) Quelle est la proportion de pronoms masculins / féminins ?

Exercice 3

Nous allons maintenant travailler sur le corpus NPSchat (Naval Postgraduate School), qui contient plus de 10,000 posts de sessions de chats entre utilisateurs, afin de développer un étiqueteur morpho-syntaxique sur mesure pour celui-ci.

(a) Importer le corpus avec la commande `from nltk.corpus import nps_chat`

(b) Comme le corpus n'est pas constitué de phrases mais de posts, on utilisera la commande `nps_chat.posts()` pour récupérer les posts à tagger, et `nps_chat.tagged_posts()` pour récupérer l'ensemble de référence nécessaire pour l'évaluation de l'étiqueteur créé.

(c) Quel est le tag le plus fréquent dans ce corpus ? L'utiliser pour créer un tagger par défaut et calculer la baseline.

(d) Proposer un ensemble d'au moins une vingtaine de règles de type expressions régulières. Quel niveau de performance parvenez-vous à atteindre ?

(e) Utiliser un tagger à base de trigrammes, bigrammes et unigrammes et évaluer leur performances respectives.

(f) Combiner les trois taggers à base de N-grammes : quelle performance atteint-on ?

(g) Ajouter aux trois taggers ci-dessus celui à base d'expressions régulières et le tagger par défaut. Cela améliore-t-il les performances ?

Le résultat devra être donné sous la forme d'une liste de mots `['mot1', 'mot2', ...]`.

4 Grammaire

Exercice 1

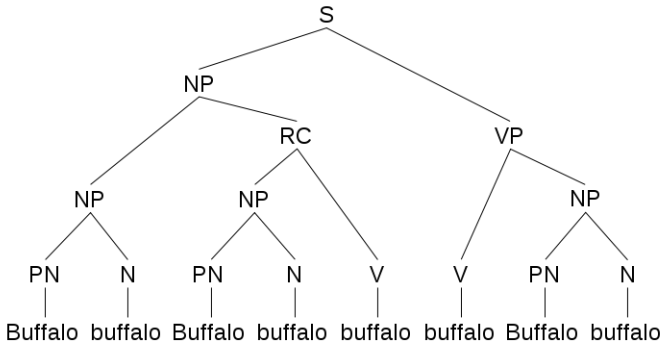
Reprendre la démonstration faite en cours du parseur avec récursion descendante :

(a) Essayer de modifier le texte (via *Edit text* dans le menu *Edit*).

(b) Essayer de modifier la grammaire (via *Edit grammar* dans le menu *Edit*), par exemple en changeant la première règle de production (NP → Det N PP) en NP → NP PP. Utiliser le bouton *Step* pour voir l'évolution de l'arbre d'analyse. Qu'arrive-t-il ?

Exercice 2

Considérons la séquence de mots : « Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo », qui contre toute apparence, est grammaticalement correcte (cf. `http://en.wikipedia.org/wiki/Buffalo_buffalo_Buffalo_buffalo_buffalo_buffalo_Buffalo_buffalo`) et dont l'arbre d'analyse est donné ci-dessous :



(a) Proposer une grammaire hors contexte permettant d'obtenir cet arbre d'analyse.

(b) Supprimer les majuscules pour simuler les problèmes à parser cette phrase à l'oral : d'autres arbres sont-ils possibles ?

(c) Comment évolue le nombre d'arbres quand la longeur de la phrase augmente ?

Exercice 3

Écrire un petit programme permettant de comparer la rapidité de l'analyse avec la récursion descendante et shift-reduce. On utilisera à cet effet le module de Python `timeit`.