Travaux Dirigés d'informatique linguistique n°4

Cours d'Introduction à l'informatique linguistique

—Licence Informatique 3ème année—

Racinisation et étiquetage morpho-syntaxique

Ce TP consiste à utiliser un raciniseur de mots utilisant l'algorithme de Porter et un étiqueteur morpho-syntaxique `TreeTagger`.

---

1 Ressources

Pour cette séance, nous vous fournissons deux modules python (dont un avec des trous) se trouvant dans le répertoire suivant :
`http://igm.univ-mlv.fr/~mconstan/enseignement/l3/infolingu/data/`

- `tagging.py` qui permet de manipuler des textes étiquetés (classe `taggedText`).
- `textSpaceVector.py` qui permet de charger et manipuler des collections de documents textuels sous la forme de vecteurs,

2 Manipulation d'un raciniseur

1. À l'aide de la classe `Porter` (à importer de `nltk_lite.stem.porter`), implanter une fonction qui racinise un mot passé en paramètre.
   NOTE : Dans `ipython`, il existe un mode `completion` pour aider à trouver les noms des méthodes d'une classe donnée. Par ailleurs, l'attribut `__doc__` permet d'obtenir la documentation associée à une méthode donnée.
2. Écrire une fonction qui prend une liste de mots en paramètre et retourne cette même liste de mots racinisés.

3 Manipulation de TreeTagger

La classe `taggedText` dans le module `tagging.py` permet de représenter un texte étiqueté par une liste d'unités lexicales (classe `lexicalUnit`). Une instance peut être créée en chargeant directement le texte étiqueté à partir d'un fichier au format TreeTagger ou d'un fichier texte non étiqueté qui est alors automatiquement étiqueté au moyen de `TreeTagger`.

1. En utilisant la classe `taggedText` et ses méthodes, écrire une fonction qui prend un texte en entrée, fait un étiquetage morphosyntaxique au moyen de TreeTagger et l'affiche à l'écran.

2. Modifier le code précédent pour que le texte étiqueté soit sauvegardé dans un fichier au format `TreeTagger` (attention, il existe une méthode dans la classe !).
3. À quoi correspondent les étiquettes utilisées ?

4 Filtrage

1. Implanter la méthode `filter(self,posSet)` de la classe `taggedText` qui retourne un nouveau texte étiqueté : celui-ci ne contient que les unités lexicales dont la catégorie grammaticale (ou partie du discours) est présente dans l'ensemble `posSet`.
2. Écrire une fonction qui prend un texte brut en paramètre et qui renvoie la liste des noms de ce texte sous leur forme lemmatisée.
3. Modifier la fonction précédente afin que celle-ci racinise les lemmes.
4. Comment pourrait-on améliorer la représentation des textes sous la forme de vecteurs de mots ?

BONUS Modifier la classe `textVector` des précédents TDs pour tenir compte de cette amélioration

5 Evaluation

1. Implanter la méthode `eval(self,tt)` de la classe `taggedText` qui évalue le texte étiqueté (`self`) par rapport à un autre texte étiqueté `tt`. Le texte brut de `tt` correspond à celui de `self`, mais on suppose qu'il a un étiquetage morpho-syntaxique parfait. Aussi, la méthode `eval` doit retourner la précision de l'étiquetage de `self` : le rapport du nombre de bonnes étiquettes sur le nombre total d'unités lexicales.
2. Écrire une fonction qui prend un texte (`text`) et ce même texte bien étiqueté (servant de référence) en entrée. Cette fonction retourne le précision de l'étiquetage par `TreeTagger` de `text` par rapport à la référence.