IFT 607 : Devoir 4
Travail individuel

Remise : 12 décembre 2014, 12h00 (**au plus tard**)

Ce devoir comporte 2 questions de programmation. Vous trouverez tous les fichiers nécessaires pour ce devoir ici : `http://info.usherbrooke.ca/hlarochelle/cours/ift607_A2014/devoir_4/devoir_4.zip`.

Veuillez soumettre vos solutions à l'aide de l'outil **turnin** :

```
turnin -c ift607 -p devoir_4 solution_analyse_syntaxique.py solution_ibm_1.py
```

1. [**5 points**] Programmez un analyseur syntaxique basé sur une PCFG et la version probabiliste de CKY.

   Le programme doit être écrit dans le langage Python. Plus spécifiquement, vous devez compléter les fonctions et méthodes du fichier `solution_analyse_syntaxique.py` disponible sur le site web du cours. Vous avez à compléter les fonctions `extraire_vocabulaire`, `remplacement_unk`, ainsi que les méthodes `entrainement` et `prediction` de la classe `AnalyseurSyntaxique`.

   Tous les détails sur ces fonctions et méthodes sont contenus dans leur *docstring*.

   L'entraînement doit utiliser un lissage *add-delta* pour l'estimation des probabilités $P(\beta|A)$. Par contre, le lissage doit s'appliquer uniquement au numérateur des β correspondant à un mot (on n'ajoute pas au numérateur des dérivations générant 2 non-terminaux B C). De plus, le lissage ne doit s'appliquer que lorsque le non-terminal A est dérivé en un mot au moins une fois dans l'ensemble d'entraînement.

   Par exemple, soit un vocabulaire $V = \{chien, chat\}$. Si on a observé les dérivations $NP \rightarrow DT\ NN$ et $NP \rightarrow chien$ dans l'ensemble d'entraînement, les probabilités seront

   $$p(DT\ NN|NP) = \frac{1}{2+\delta\times 2},\ p(chien|NP) = \frac{1+\delta}{2+\delta\times 2}\ p(chat|NP) = \frac{\delta}{2+\delta\times 2}$$

   mais si on a plutôt observé $NP \rightarrow DT\ NN$ et $NP \rightarrow NP\ NN$, les probabilités seront

   $$p(DT\ NN|NP) = \frac{1}{2},\ p(NP\ NN|NP) = \frac{1}{2},\ p(chien|NP) = 0,\ p(chat|NP) = 0\ .$$

   De plus, vous devrez utiliser la classe `Tree` de NLTK (dans `nltk.tree`). Cette classe est décrite ici : `http://www.nltk.org/_modules/nltk/tree.html`. À noter que :

   - `Tree` hérite de `list` et itérer sur un `Tree` donne accès aux enfants de la racine de l'arbre.
   - La méthode `label()` permet d'obtenir l'étiquette d'un noeud.
   - Les noeuds internes (i.e. autres que les feuilles) sont aussi des `Tree`.
   - Les feuilles d'un arbre, correspondant aux mots, doivent être des `str`.
   - La méthode `productions()` permet d'obtenir la liste des règles ayant généré l'arbre. Chaque règle est un instance de la classe `Production`, ayant des méthodes `lhs()` et `rhs()` afin d'accéder au côté droit et gauche de la règle. Il est possible de convertir l'élément `lhs()` et le ou les éléments dans `rhs()` sous la forme d'une string à l'aide de `str()`.
   - La méthode `leaves()` permet d'obtenir la phrase associée à l'arbre.

   Le script Python `analyse_syntaxique.py` importera `solution_analyse_syntaxique.py` (qui doit être dans le même répertoire) et l'utilisera afin d'entraîner un analyseur syntaxique (*parser*) sur un treebank simplifié (phrases d'au plus 10 mots, traces `-NONE-` et étiquettes fonctionnelles enlevées).

   Pour ce numéro, vous devez installer la librairie *nltk*, comme suit :

```
pip install --user nltk
```

Ensuite, vous devez télécharger le treebank. Pour ce faire, vous devez exécuter les instructions Python suivantes (par exemple via l'interpréteur) :

```
import nltk
nltk.download('treebank')
```

Voici comment utiliser le script `analyse_syntaxique.py` :

```
Usage: python analyse_syntaxique.py [mot1 mot2 ...]

Si aucun argument n'est donné, une comparaison
sera faite avec un cas pour lequel les résultats
attendus sont connus.

Optionnellement, une phrase, spécifiée mot à mot,
peut être fournie. Le programme retournera alors
son arbre syntaxique.
```

2. [**5 points**] Programmez l'algorithme d'alignement mot-à-mot basé sur le modèle IBM 1 et l'entraîment EM.

   Le programme doit être écrit dans le langage Python. Plus spécifiquement, vous devez compléter les fonctions et méthodes du fichier `solution_ibm_1.py` disponible sur le site web du cours. Vous avez à compléter les fonctions `extraire_vocabulaire`, `remplacement_unk`, ainsi que les méthodes `entrainement` et `alignemnet` de la classe `IBM1`.

   Tous les détails sur ces fonctions et méthodes sont contenus dans leur *docstring*.

   Le script Python `ibm_1.py` importera `solution_ibm_1.py` (qui doit être dans le même répertoire) et l'utilisera afin d'exécuter votre algorithme sur un sous-ensemble du corpus parallèle Europarl, de phrases en français et en anglais (fichiers `europarl-v7.smaller.tok.fr-en.fr` et `europarl-v7.smaller.tok.fr-en.en`).

   Voici comment utiliser le script `ibm_1.py` :

```
Usage: python ibm_1.py

Une comparaison sera faite avec un cas pour lequel
les résultats attendus sont connus.
```