TP1 : Expressions régulières, tokenisation et évaluation sur corpus[1]

L'objectif de cette première séance de TP est de vous faire découvrir une des tâches les plus fondamentales du TAL : le *découpage en mots*. Cette exploration nous permettra également d'introduire les notions de corpus et d'évaluation.

Mise en place de l'environnement de travail

Cette séance et les suivantes seront réalisées avec le langage Python. Historiquement les talistes ont travaillé avec le langage Perl qui a été originalement conçu pour l'extraction d'informations depuis des fichiers textes. Le choix de Python se justifie par la disponibilité de nombreuses bibliothèques scientifiques dont, notamment, `nltk` que nous utiliserons par la suite.

L'écriture des scripts Python s'effectue facilement dans n'importe quel éditeur de texte qui offre une bonne coloration syntaxique et éventuellement de la complétion de code. Pour vérifier la version de Python installée sur votre machine, lancez un interpréteur dans un terminal :

```
$ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more
    information.
>>>
```

Les ressources telles que l'énoncé des travaux pratiques et les exemples de code sont disponibles sur l'espace madoc du site web du cours.

Préparation du corpus

En TAL, le travail commence presque toujours par la constitution d'un corpus. Nous ne dérogerons pas à la règle dans cette séance. Toutefois afin de gagner du temps, un corpus brut vous est fourni. Ce dernier a été construit à partir de transcriptions issues de la reconnaissance d'écriture manuscrite sur des dépêches de l'agence Reuters. Ces transcriptions peuvent ainsi contenir des erreurs, par rapport aux dépêches initiales, dues à des erreurs de transcription du système de reconnaissance. Le corpus est composé de plusieurs documents au format texte (extension `.txt`).

La première étape de votre travail va consister à annoter ce corpus pour l'évaluation d'un découpage en mots. L'objectif d'une telle tâche est de passer d'un texte présenté sous la forme d'une longue chaîne de caractères à une liste de sous-chaînes de caractères correspondant chacune à un mot.

La notion de mot n'est pas réellement arrêtée en linguistique puisque le mot n'a ni réalité phonétique (contractions lors de la prononciation, ...), ni réalité sémantique (mots composés, contractions d'articles, ...). La réalité du mot est principalement graphique (écriture) et l'on

1. D'après un sujet de Florian Boudin et Fabien Poulard.

se contente bien souvent de définir un mot comme étant un groupe de symboles graphiques séparés par des « blancs ».

Votre premier travail consiste à récupérer les fichiers de l'archive `txt.zip` présents sur Madoc, dans le répertoire `TP 1` et à découper leur contenu en mots. Dans ce but, vous utiliserez l'espace classique pour marquer la séparation entre deux mots. Par exemple, étant donné cet extrait du fichier `fich176.txt` :

```
the Turkish ore 1 bulk 1 oil vessel Obo En gin, 78,078 tonnes dwt, had an explosion in its boiler yesterday, Lloyds shipping Intelligence Service reported the vessel has retained some power and yesterday evening was in position lat. 25.57 N., Gong. 7506 W. It is dilating to Jacksonville, Florida, with its cargo of 58,000 tons of coal the nard was bound for Iskenderun, Turkey from Lake Charles.
```

Vous devriez l'annoter pour obtenir le découpage suivant :

```
the Turkish ore 1 bulk 1 oil vessel Obo En gin , 78,078 tonnes dwt , had an explosion in its boiler yesterday , Lloyds shipping Intelligence Service reported the vessel has retained some power and yesterday evening was in position lat. 25.57 N. , Gong. 7506 W . It is dilating to Jacksonville , Florida , with its cargo of 58,000 tons of coal the nard was bound for Iskenderun , Turkey from Lake Charles .
```

Découpage en mots à l'aide de règles

La première grande famille de méthode utilisée en TAL est celle de l'écriture de règles sur la base d'observations en corpus. Dans cette démarche, les expressions régulières fournissent souvent un bon moyen d'écrire mais également d'appliquer les règles.

En Python, les expressions régulières ne sont pas disponibles par défaut, il est nécessaire d'importer le module `re`. La façon d'exprimer les expressions régulières reste très similaire à ce qui se fait en Perl. La documentation du module `re` est directement consultable en ligne (`http://docs.python.org/2/library/re.html`). Le fichier `example1.py` vous montre comment extraire les mots d'un texte à l'aide d'une expression :

```
# Lecture du contenu du fichier
import codecs

fh = codecs.open("data/txt/fich1.txt", "r", "utf-8")
content = fh.read()
fh.close()

# Extract words
import re

regexp = re.compile("\w+", re.U)
for word in regexp.findall(content):
  print word
```

Vous pouvez lancer le script en mode interactif, de sorte que vous vous retrouviez dans

l'interpréteur Python une fois le script exécuté et que vous puissiez ainsi accéder au contexte d'exécution :

```
python -i example1.py
```

Sur la base des annotations effectuées sur le sous-ensemble du corpus, proposez des expressions régulières permettant de découper un texte en mots. Implémentez ensuite votre tokeniseur sous la forme d'un script Python et faites-le s'exécuter sur le corpus. Repartez du code proposé dans le fichier `regexp-tokenizer.py`.

Évaluation du découpage automatique

Lorsque l'on commence à expérimenter des méthodes automatiques, la question de l'évaluation de ces méthodes se pose rapidement. Une bonne évaluation nécessite :

- des données de référence auxquelles confronter les données calculées automatiquement ;
- des métriques qui mesurent le taux de succès et d'erreur pour la tâche.

Nous allons utiliser une méthode d'évaluation classiquement utilisée dans les tâches de catégorisation et qui a l'avantage d'être simple à comprendre et à mettre en œuvre. La méthode consiste à classer les mots produits par le tokeniseur en quatre catégories :

- **les vrais positifs (VP)** : les mots identifiés par le tokeniseur et qui sont également présents dans les données de référence ;
- **les faux positifs (FP)** : les mots identifiés par le tokeniseur mais qui ne sont pas présents dans les données de référence ;
- **les vrais négatifs (VN)** : les mots non identifiés par le tokeniseur et qui ne sont pas non plus présents dans les données de référence (toujours vide dans notre cas) ;
- **les faux négatifs (FN)** : les mots non identifiés par le tokeniseur mais qui sont présents dans les données de référence.

Cette répartition en catégories nous permet d'utiliser deux métriques :

- **la précision = VP / (VP + FP)** : la proportion de mots que le tokeniseur a identifié correctement ;
- **le rappel = VP / (VP + FN)** : la proportion de mots que le tokeniseur n'a pas réussi à identifier.

Réalisez l'évaluation de votre tokeniseur en utilisant cette méthode. Il vous est nécessaire en premier lieu d'avoir des données de référence pour les comparer à la sortie de votre tokeniseur. Adaptez vos expressions rationnelles de manière à améliorer les performances de votre tokeniseur.

Le script `eval.py` implémente le cadre décrit précédemment. Il prend simplement en entrée deux fichiers :

- le découpage en mots calculé par votre tokeniseur
- le découpage de référence

Par exemple, en utilisant l'annotation de référence `data/tokens-ref/fich1.txt`

```
python -i eval.py data/tokens-auto/fich176.txt data/tokens-ref/
   fich176.txt
Precision: 0.9610
```

```
Recall: 0.9610
>>> fp
[u'lat', u'N.,', u'.']
>>> fn
[u'lat.', u'N.', u',']
```