Étiquetage morpho-syntaxique
Annotation de données

Guillaume Wisniewski
guillaume.wisniewski@linguist.univ-paris-diderot.fr

novembre 2019

1 Étiqueteur morpho-syntaxique de Brill

1.1 Principe

Nous allons développer, dans ce TP, un étiqueteur morpho-syntaxique suivant le principe proposé par Eric Brill. Celui-ci repose sur une approche « par correction d'erreur » : il commence par associer chaque mot du corpus de test à son étiquette la plus fréquente (la fréquence de l'étiquette associée à un mot étant déterminée à partir d'un corpus d'apprentissage). Puis, pour chaque mot d'une phrase, des règles sont appliquées dans un ordre pré-défini afin de corriger les erreurs d'étiquetage de la première étape. Dès qu'une règle peut être appliquée, l'étiquette du mot est modifiée selon la définition de la règle et l'on recommence à appliquer les règles au mot suivant.

Les règles sont définies par cinq éléments :

— un nom (arbitraire) qui permet de les identifier ;

— un type de test (`kind`) qui permet de définir l'élément qui sera tester : soit l'étiquette du mot précédent (`tag_before`), ou du mot suivant (`tag_after`) ; soit le mot précédent (`word_before`) ou suivant (`word_after`). Il est également possible de savoir si le mot a une majuscule (`starts_with_upper`). [1]

— un argument pour le test (`argument`) qui permet de spécifier à quelle valeur doit correspondre le l'élément désigné par `kind`.

Une règle est appliquée si l'étiquette de w_i est celle spécifiée par `tag_before` et que la valeur désignée par `kind` est bien égale à `argument`.

Ainsi, une règles changeant l'étiquette `A` en `B` si l'étiquette précédente est `C` sera définie par les éléments suivants :

— **name** : `rule@41` ;

— **kind** : `tag_before` ;

1. Ces 5 types de règles couvrent (normalement) les principaux cas. Vous pouvez toutefois définir vos propres types de règles en ajoutant les fonctions python correspondantes.

— **argument** : `C` ;

— **tag_before** : `A` ;

— **tag_after** : `B` ;

1.2 Mise en œuvre

Vous trouverez sur le site du cours [2] un programme python implémentant l'approche décrite dans la section précédente. Vous pouvez installer celui-ci à l'aide des commandes suivantes :

```
# dans un répertoire de votre choix
# déplacer le fichier téléchargé sur le site du cours
> tar xvfj brill_tagger.tar.bz2
> python3 -m venv local
> ./local/bin/pip install termcolor
> ./local/bin/pip install ./dependencies/universal_dependencies-0.1-py3-none-any.whl
```

Il est intéressant de regarder le contenu de l'archive téléchargée et comment les différents types de données sont organisés dans des répertoires distincts. En particulier, l'archive contient un répertoire distinct pour les données « brutes » (c.-à-d. que vous n'avez pas modifiée) [3]. Ce répertoire contient un corpus d'apprentissage (`fr_gsd-ud-train.conllu`) qui sera utilisé pour estimer les fréquences d'utilisation des étiquettes nécessaires à la première phase de l'étiquetage et différents fichiers de test que vous utiliserez pour évaluer les performances de votre analyseur.

Une fois l'installation terminée, le programme peut-être lancé à l'aide de la commande suivante :

```
# à partir du répertoire de travail dans lequel l'archive a été
# décompressée
> ./local/bin/python brill_pos_tagger.py
```

Il est nécessaire de passer en paramètre le nom du fichier comportant les règles (option `--rules`) et le fichier sur lequel on veut tester et évaluer l'étiqueteur (option `--test`). Le programme affiche le taux d'erreur obtenu par l'analyseur morpho-syntaxique ainsi que le nombre de fois où chaque règle a été appliquée.

Il est possible en utilisant l'option `--dump` d'afficher les étiquettes morpho-syntaxiques prédites ainsi que les erreurs. L'option `--color` permet de mettre en évidence les étiquettes qui ont été mal prédites.

Les règles sont stockées dans un fichier au format `Json`, un format de données textuelles (c.-à-d. pouvant être directement modifié avec un éditeur de texte) dont la syntaxe est très proche de celle utilisée pour définir les structures de données en python. Ce fichier définit une *liste* de règles : comme le montre l'exemple de la figure 1, la liste est indiquée

2. `https://gw17.github.io/`

3. Il est fortement conseillé, dans les différentes expériences de TAL de conserver systématiquement une version non modifiée des données utilisées.

par des crochets, et les différents éléments la composant sont séparés par des virgules. Chacun de ces éléments correspond à une règle dont la définition est transparente.

```
[{
  "name": "example1",
  "kind": "tag_before",
  "argument": "A",
  "tag_before": "B",
  "tag_after": "C"
},
{
  "name": "example2",
  "kind": "tag_after",
  "argument": "E",
  "tag_before": "F",
  "tag_after": "G"
}]
```

FIGURE 1 – Exemple d'un fichier contenant deux règles.

2 Travail à effectuer

Le principal objectif de ce TP est de trouver un ensemble de règles permettant de réduire le taux d'erreur obtenu par l'étiqueteur morpho-syntaxique.

① Comment est évaluée la performance de votre analyseur morpho-syntaxique ?

② Comment pouvez-vous déterminer l'erreur la plus fréquente faite par votre analyseur ?

③ En utilisant le fichier `fr_gsd-ud-test.conllu` quelle est la plus petite erreur de prédiction que vous arrivez à obtenir ? Ce résultat est-il satisfaisant ?

④ Comment évolue cette erreur en fonction du nombre de règles que vous spécifiez ? Interprétez.

⑤ Sans changer l'ensemble de règles, évaluez les performances de votre étiqueteur sur le fichier `fr_sequoia-ud-test.conllu`. Comment interprétez-vous ces résultats ?