TP de désambiguïsation lexicale basée sur des connaissances
Master IC2A - spécialité WIC
2015-2016

Jérôme Goulian et Didier Schwab
LIG (Laboratoire d'Informatique de Grenoble)
GETALP (Groupe d'Étude en Traduction Automatique/Traitement Automatisé des Langues et de la Parole)
Université Grenoble Alpes
http://getalp.imag.fr/WSD
{Jerome.Goulian,Didier.Schwab}@imag.fr

Ce TP va servir à illustrer et à mieux comprendre les concepts de désambiguïsation lexicale basée sur des connaissances vus en cours. L'ensemble des exercices seront réalisés avec des corpus, des termes, des ressources en anglais mais aucune connaissance particulière de cette langue n'est nécessaire. L'ensemble des exercices sera à coder en *Java*.

1 Notation et autres informations diverses

Ce TP est à réaliser pendant les séances et à finir pour le **mi-janvier**. Chaque groupe devra envoyer son TP (et compte-rendus) par message électronique aux adresses indiquées ci-dessus.

2 Corpus d'évaluation

Nous évaluerons les algorithmes produits sur le corpus de la tâche *gros grain* de la campagne d'évaluation *Semeval 2007* (Navigli *et al.*, 2007). Il est composé de 5 textes en anglais pour lesquels il faut annoter 2269 mots.

Texte	Genre	Nombre de phrases	Nombre de mots	Mots/phrases
d001	journalisme	35	368	10,51
d002	critique littéraire	41	379	9,24
d003	voyage	59	500	8,47
d004	informatique	76	677	8,91
d005	biographies	34	345	10,15
Total		245	2269	9,26

TABLE 1 – Les cinq textes du corpus

Cette annotation doit être faite parmi les sens proposés par WordNet.

Parmi les 2269 mots à annoter, 678 n'ont qu'un seul sens (monosémiques), c'est-à-dire qu'on est certain de trouver la bonne solution pour ceux là tandis que pour 1591 termes polysémiques, il faut faire un choix.

polysémie	Noms	Verbes	Adjectifs	Adverbes	all
monosemiques	358	86	141	93	678
polysemiques	750	505	221	115	1591
total	1108	591	362	208	2269

TABLE 2 – Statistiques sur les mots du corpus à annoter

L'évaluation se fait en grain grossiers. Par exemple, les sens considérés comme proches (par exemple, "*neige/précipitation*" et "*neige/couverture*" ou "*porc/animal*" et "*porc/viande*") sont groupés. Les sens annotés sont jugés corrects s'ils sont dans le bon groupement, une sorte d'erreur acceptable. Les résultats sont analysés par les formules classiques :

Précision $P = \frac{sens\ correctement\ annotés}{sens\ annotés}$ Rappel $R = \frac{sens\ correctement\ annotés}{sens\ à\ annoter}$ F-mesure $F = \frac{2 \times P \times R}{P+R}$

2.1 Connaître le corpus

Téléchargez le fichier zip **TP-WSD-WIK.zip**. Extrayez son contenu. Tous les chemins seront indiqués à partir de la racine de ce répertoire.

Allez dans *evaluation/test*. Ce répertoire comprend :

- *wsj_0105.mrg* : Le premier texte
- *wsj_0186.mrg* : Le second texte
- *wsj_0239.mrg* : Le troisième texte
- *Computer_programming.txt* : Le quatrième texte
- *Masaccio_Knights_of_the_Art_by_Amy_Steedman.txt* : Le cinquième texte
- *eng-coarse-all-words.xml* : Le corpus prêt à être annoté en un seul fichier. Les termes issus des cinq textes à annoter portent un identifiant. Ils sont lemmatisés (*lemma*) et leur partie du discours (*pos*) est indiquée.
- *coarse-all-words.dtd* : Le fichier dtd qui décrit la structure syntaxique du fichier *eng-coarse-all-words.xml*.

Vous pouvez prendre connaissance du contenu de ces textes si vous parlez anglais. Il n'est absolument pas nécessaire de connaître le contenu de ces textes pour les désambiguïser à l'aide de programmes.

Exercice 1 : Ouvrez le fichier *eng-coarse-all-words.xml*. Nous avons vu en cours la structure de ce fichier. Vérifiez que vous comprenez ses caractéristiques. Prenez la onzième phrase du cinquième texte (*d005.s011*). Pour chaque mot de la phrase, indiquez les annotations fournies. Quels sont les quatre parties du discours des mots annotés ? Pourquoi certains mots ne sont pas annotés ?

2.2 Évaluation d'exemples de désambiguïsation lexicale

Allez dans *evaluation/key*. Les deux fichiers importants ici sont

- **dataset21.test.key** : Le fichier comportant les annotations du corpus
- **scorer.pl** : Le scorer écrit en perl qui permet de calculer automatiquement la précision, le rappel et le F-score d'une annotation donnée en argument.

Exercice 2 : Ouvrez *dataset21.test.key* avec un éditeur de texte quelconque. Nous avons vu en cours la structure de ce fichier. Vérifiez que vous comprenez ses caractéristiques. Pourquoi y-a-t-il plusieurs synsets possible dans certain cas ?

Vous trouverez dans le répertoire *exemples-annotations*, cinq exemples d'annotation. Ce sont des fichiers qui ont été générés par des algorithmes de désambiguïsation lexicale. Vous allez utiliser le scorer pour comparer ces cinq exemples. La syntaxe d'appel du scorer est la suivante :

perl scorer.pl fichier-exemple

Exercice 3 : Pour chacun des exemples, donnez la précision, le rappel et le F-score.

Exercice 4 : N'indiquez pas de fichier au scorer. Quels sont les différentes options disponibles ?

Exercice 5 : Comparez maintenant vos exemples en fonction des documents. Pour chacun des textes, donnez la précision, le rappel et le F-score donné par chacun des exemples.

3 Évaluation de la proximité sémantique : algorithmes locaux

3.1 Mesure de Lesk et ses dérivées

3.1.1 Principe

Allez dans *Dictionnaires-Lesk*. Il y a deux dictionnaires *Dict-Lesk.xml* et *Dict-Lesk-etendu.xml*. Un troisième disponible depuis peu est à télécharger ici `http://getalp.imag.fr/static/wsd/LREC2016/dict_all_stopwords_stemming_semcor_dso_wordnetglosstag_150.zip`, nous l'appellerons *Dict-Lesk-corpus.xml* dans la suite. Tous les trois ont été générés à partir de WordNet (`http://wordnetweb.princeton.edu/perl/webwn`).

Exercice 6 : Sur le site de WordNet, cherchez le mot *cat*. Quelle sont les deux premières définitions ? Utilisez la méthode vue en cours pour transformer ces définitions en vecteurs permettant un calcul plus efficace en temps de la mesure de Lesk. Précisez chacune des étapes.

Exercice 7 : Quels sont les différentes options d'affichages disponibles ? Affichez les numéros de sens (*sense key*). Quel est le numéro du premier sens de *cat* ? Cherchez dans *Dict-Lesk.xml* les définitions que vous venez de fabriquer manuellement. Quelles sont les différences, d'où viennent-elles ?

3.1.2 L'API Java du GETALP

Vous trouverez dans le répertoire *WICWSD* une API Java créée pour ce cours à partir du projet *formica* (fourmi en italien) développé actuellement au Groupe d'Étude en Traduction Automatique/Traitement Automatisé des Langues et de la Parole du Laboratoire d'Informatique de Grenoble (GETALP-LIG). Pour l'utiliser, il faudra importer le package nommé *org.wic.wsd* et gérer les dépendances classiques. Tout ceci peut-être fait en ligne de commande ou bien avec un IDE comme *eclipse* par exemple.

3.2 Chargement des dictionnaires

Vous aurez besoin des fonctions, méthodes et classes suivantes :

```
// classe permettant la gestion d'un dictionnaire
class Dictionary

//Constructeur de Dictionnaire
public Dictionary() ;

// chargement d'un Dictionnaire où path est le chemin vers ce dictionnaire (dans notre cas, soit Dict-Lesk.xml, soit Dict-Lesk-etendu.xml, soit Dict-Lesk-corpus.xml)
public void loadDictionary(String path) ;

// Taille du dictionnaire
public int size() ;

// sens d'un lemme (entrée du dictionnaire) ;
public ArrayList<Sense> getSenses(String lemme) ;

// calcul de la proximité sémantique de deux sens
public int getSimilarity(Sense s1, Sense s2) ;

// classe permettant la gestion des sens de lemme
class Sense

// méthode d'affichage (Rappel : en java, l'appel est automatique dans un System.out.println)
public String toString() ;
```

Exercice 8 : Faites un programme java qui permet d'afficher la taille des trois dictionnaires à l'aide des informations ci-dessus.

Exercice 9 : Complétez le programme précédant pour qu'il affiche également le nombre de sens de l'entrée *cone* puis ces sens.

Exercice 10 : Complétez le programme précédant pour qu'il affiche également le nombre de sens de l'entrée *pine* puis ces sens.

Exercice 11 : On va considérer que la proximité sémantique entre deux mots est la moyenne des similarités entre toutes les paires de sens possibles pour ces mots. Ajoutez dans la classe Dictionnary la fonction correspondante dont la signature est

public float getSimilarity(String word1, String word2) ;

Utilisez ensuite cette fonction pour calculer la mesure Lesk, la mesure Lesk étendue et la mesure Lesk-corpus entre *pine* et *cône*. Vous devriez trouver $0,8$ pour la première. Combien trouvez-vous pour les deux autres ?

3.3 Évaluation d'une mesure de proximité

L'évaluation d'une mesure de proximité peut se faire selon plusieurs perspectives. Évidemment, chacune d'entre elles ne capture pas les mêmes aspects et une mesure qui se révèlera bonne pour une certaine tâche ne le sera pas pour une autre. Nous verrons plus loin dans ce TD comment comparer plusieurs mesures sur une tâche de désambiguïsation lexicale. Nous allons ici comparer des mesures par rapport à un ensemble de paires de mots dont la similarité a été évaluée par des humains. Nous allons utiliser ici la collection *WordSimilarity-353*. Il s'agit d'une collection établie en 2002 qui comporte 353 paires de mots.

Allez dans *wordsim353* et ouvrez le fichier *paires.csv*. Il s'agit d'un fichier csv, vous allez pouvoir l'ouvrir avec un tableur ou avec un traitement de texte.

Exercice 12 : Que contient ce tableau ? Donnez 5 exemples choisis à différents endroits du fichier et interprétez-les.

On vous donne la classe Stats qui contient une méthode permettant de calculer la corrélation entre deux série de données. Voici sa signature :

public static double getPearsonCorrelation(double[] scores1,double[] scores2) ;

La corrélation est égale à 1 dans le cas où l'une des variables est fonction croissante de l'autre variable, à -1 dans le cas où la fonction est décroissante. Les valeurs intermédiaires renseignent sur le degré de dépendance linéaire entre les deux variables. Plus le coefficient est proche des valeurs extrêmes -1 et 1, plus la corrélation entre les variables est forte ; on emploie simplement l'expression « fortement corrélées » pour qualifier les deux variables. Une corrélation égale à 0 signifie que les variables ne sont pas corrélées.

Exercice 13 : Utilisez cette fonction pour calculer la corrélation entre les séries {1,4,8,12,2,5,4} et {4,2,1,6,6,10,4}.

Exercice 14 : Prenez les termes de la collection *WordSimilarity-353* , calculer la corrélation avec Lesk simple, Lesk étendu et Lesk corpus.

4 Algorithmes globaux

4.1 Algorithme exhaustif

Comme son nom l'indique, l'algorithme exhaustif calcule le score de chacune des combinaisons puis choisit celui qui est le plus grand.

Exercice 15 : Réalisez une fonction qui calcule le nombre de combinaisons à examiner selon un dictionnaire donné. Sa signature sera

public static long nbCombinaisons(String S, Dictionary Dict) ;

Exercice 16 : Quel est le nombre de combinaisons de sens pour les phrases lemmatisées suivantes ?

go

mouse pilot computer

dog eat bone every day

doctor be hospital last day night

pictures paint be flat round figure be very often foot do look be stand ground all point downward be hanging air

Exercice 17 : Pourriez-vous désambiguïser toutes ces phrases ? Pourquoi ?

Exercice 18 : Réalisez l'algorithme exhaustif. Sa signature sera

public static ArrayList<String> algoExhaustif(String S, Dictionary Dict) ;

qui doit renvoyer la liste des sens pour chacun des mots dans l'ordre de la phrase.

Exercice 19 : Désambiguïsez manuellement les phrases précédentes que vous considérez possible de désambiguïser. Ce sera votre corpus de référence pour l'exercice suivant.

Exercice 20 : Désambiguïsez avec l'algorithme exhaustif les phrases précédentes que vous considérez possible de désambiguïser. Vous testerez avec les deux algorithmes locaux. Calculez manuellement la précision, le rappel et la F-mesure.

4.2 Algorithmes global stochastiques

Vous trouverez sur le site Web, une archive contenant plusieurs classes permettant l'utilisation de fonctions de bases pour créer un algorithme global pour le corpus de Semeval 2007.

Exercice 21 : Ouvrez la classe Main. Décrire les différentes étapes du code de Main. En déduire la spécification des méthodes et fonctions utilisées. Exemple :

// classe permettant la création d'un objet Corpus

class Corpus ;

//Constructeur du corpus

new Corpus() ;

Exercice 22 : Quel est le score trouvé avec le fichier résultat ? En sachant ce que réalise le programme de Main, pensez-vous que ce résultat puisse être bon ?

Exercice 23 : Modifiez la class Main pour créer le pseudo-algorithme global suivant.

Algorithme 1 Premier algorithme global stochastique

Entrée : un corpus c, un entier n
Sortie : Pour chaque texte du corpus, la configuration qui a obtenu le meilleur score

```
Partir d'une configuration quelconque
Initialiser le score maximum trouvé à 0 pour chacune des configurations
pour i = 1..n faire
    Modifier le choix de sens pour un des mots
    Calculer le score de cette nouvelle configuration
    Conserver ce score s'il est supérieur au score maximum trouvé pour ce texte
fin pour
```

Notes :

N'oubliez pas que le corpus contient plusieurs textes et que les scores d'un texte n'influencent en aucun cas le score des autres textes.

Il serait plus générique de faire en sorte que votre programme fonctionne quel que soit le nombre de textes du corpus.

Exercice 24 : Calculez le score obtenu par votre algorithme global avec chacun des deux algorithmes locaux pour 5, 10, 20, 50,100, 1000 ou 10000 itérations. Ce score s'améliore-t-il ? Pourquoi ?

Exercice 25 : Calculez le score obtenu par 3 exécutions de cet algorithme avec 100 itérations. Trouvez-vous toujours le même résultat ? Pourquoi ?

Exercice 26 : La fin de ces TPs va consister à implanter un algorithme stochastique global. La principale difficulté que doivent affronter ces problèmes est de savoir quels sont les mots pour lesquels il faudrait choisir un nouveau sens et les mots pour lesquels il faudrait conserver

ce sens. Nous avons vu en cours deux exemples de tels algorithmes : un recuit simulé et un algorithme génétique. Nous allons tester ici la méthode dite de l'*algorithme à estimation de distribution*. Wikipedia pourra vous en donner les généralités mais l'adaptation que l'on vous présente ci-dessous devrait pouvoir suffire.

L'idée est similaire à celle des algorithmes génétiques. Chaque configuration est un individu. La simulation commence avec un certain nombre d'individus pour lesquels une configuration est tirée au hasard. On ordonne les individus en fonction de leur score puis on sélectionne les m meilleurs pour réaliser l'échantillonnage. Il va s'agir pour chacun des m mots de l'échantillon de voir comment les individus se sont répartis entre les sens afin de créer de nouveaux individus respectant pour chacun des mots une répartition probabiliste similaire. Cette technique est à rapprocher du croisement entre deux individus des algorithmes génétiques (mais ici il s'agit d'une sorte de croisement entre tous les individus) mais également de la mutation à cause de la création probabiliste des nouveaux individus qui conduira nécessairement à de légères différences.

Prenons un exemple. Imaginons un mot de 3 sens et un échantillon de 100 individus : 55 ont choisi son premier sens, 15 son deuxième et 30 son troisième. On va créer n individus à partir de cette distribution. Pour chacun des individus, on va tirer une valeur au hasard (par exemple avec Math.random() en java). On va obtenir une valeur entre 0 et 1. Si la valeur est entre 0 et 0,55, l'individu aura le sens 1 pour ce mot ; si la valeur est comprise entre 0,55 et 0,7, l'individu aura le sens 2 et il aura le sens 3 sinon.

À chaque itération de l'algorithme, on conservera l'individu ayant eu le meilleur score. Si l'individu ne change pas au bout de s itérations, on pourra enregistrer le résultat dans un fichier et l'évaluer par le script vu précédemment.

Paramètres :

n : nombre d'individus

m : nombre d'individus à sélectionner pour l'échantillon ($m < n$)

s : nombre de cycles de stabilisation

Notes :

Pour plus de simplicité, nous vous conseillons de réaliser l'algorithme pour qu'il propose un résultat sur un texte à la fois.

Dans la classe Configuration, les méthodes suivantes devraient vous être utiles :

// Donne le sens sélectionné pour le mot à la position *index*

public int getSelectedSenseAt(int index) ;

// Remplace le sens du mot à la position *index* par *senseNumber*

public void setSelectedSenseAt(int index, int senseNumber) ;

Exercice 27 : Essayez d'estimer les meilleurs paramètres pour votre algorithme. Étant donnée la complexité des calculs à mettre en œuvre pour calculer les significativités des résultats, nous vous demandons simplement une estimation manuelle et grossière de ces paramètres.

Références

NAVIGLI, R., LITKOWSKI, K. C. et HARGRAVES, O. (2007). Semeval-2007 task 07 : Coarse-grained english all-words task. *In SemEval-2007*, pages 30–35, Prague, Czech Republic.