CorpuSearch : présentation d'un outil d'extraction spécifique

Jean-Philippe Demoulin, Alda Mari, Romain Vinot
ENST – Département INFRES – CNRS URA 820
46 rue Barrault - 75013 PARIS - +33(0)145817259
{demoulin, mari, vinot}@enst.fr

Introduction

Le projet CorpuSearch est né du besoin pressant qu'exprimaient les linguistes non informaticiens de disposer d'un outil d'extraction d'informations à partir de grandes masses de données qu'on appelle corpus. Ces données sont textuelles et peuvent provenir de différentes sources: courriels, forums de discussion, pages web, textes journalistiques, littéraires… Extraire de l'information signifie rechercher, stocker, présenter sous un format donné des informations voulues : par exemple, tous les verbes apparaissant dans un texte, ou toutes les combinaisons adjectif + nom… De manière plus complexe, on peut vouloir rechercher des relations : par exemple quelle entreprise fusionne avec quelle autre entreprise et à quel moment. Cela implique que l'utilisateur doit pouvoir rentrer dans son interface une requête qui prend la forme d'une grammaire. Par exemple en demandant "ADJ + NOM", l'outil rechercherait toutes les combinaisons adjectif + nom dans le texte qu'on lui aura indiqué. Dans un souci constant de fournir à l'utilisateur final un logiciel utile, convivial et utilisable, il est essentiel de prévoir un langage de formulation de requête définissant une syntaxe à laquelle cet utilisateur est habitué. Il doit être simple d'utilisation. Toutefois les fonctionnalités offertes par le logiciel sont relativement nombreuses et complexes. Elles visent à aider le linguiste non seulement à extraire les informations nécessaires, mais aussi à les traiter qualitativement et quantitativement afin de créer des ontologies sur la base d'occurrences attestées. De plus, cet outil est libre et distribué gratuitement.

L'outil que nous présentons dans cet article a ceci de spécifique par rapport à ceux qui existent aujourd'hui, de permettre de poser des requêtes récursives. Par exemple étant donné une construction verbe + ANY + nom_i (en supposant que les noms soit le COD du verbe), on veut pouvoir interroger les propriétés du nom_i, en demandant quels sont les adjectifs avec lesquels il peut se combiner dans n'importe quelle construction. Une deuxième requête doit alors être possible: nom_i + adj. Notre objectif initial était de développer un logiciel permettant de faire de la création et de la gestion de corpus, de l'extraction d'informations et de présenter et pouvoir exploiter les résultats dans une seule et même interface. Tout cela de manière dynamique : l'utilisateur a la possibilité de réutiliser les résultats des requêtes précédentes.

Cet article s'articule de la manière suivante : la première partie présente les objectifs et les motivations théoriques de l'utilisation du logiciel. La deuxième partie est consacrée à la description du premier module qui compose le logiciel CorpuSearch, à savoir la gestion de corpus, la troisième au module gérant l'extraction d'informations et la quatrième partie est consacrée à la présentation et l'exploitation des résultats. Dans une cinquième partie, nous proposons une série d'améliorations futures à apporter au logiciel. Tout au long de cet article, nous illustrerons notre propos par des captures d'écrans.

Objectif et motivation théorique de l'outil : la notion de classe évolutive et de sélecteur

Dans nos travaux sur l'exploitation de corpus pour l'analyse d'objets grammaticaux abstraits (*avec, par, à cause de* …) (Grivel 2000) nous mettons au point une méthode nous permettant d'extraire les restrictions de sélection de ces objets sans établir *a priori* de treillis ou d'ontologies.

Considérons la classe des noms de sentiment comprenant entre autres *crainte* et *peur*. D'autre part, considérons le sens *manière* de *avec*, que l'on analyse généralement comme « avec + $N_{(de\ sentiment)}$ ». Comment expliquer l'alternance *avec* $^{??}$ *peur / crainte* ?

Pour arriver à dégager des classes fines et consistantes nous procédons de la manière suivante :

1. Désambiguïsation syntaxique du corpus
2. Extraction via des règles de classes sémantiques selon la méthode par sélection
3. Dégagement des propriétés des classes et généralisation
4. Test

La méthode par sélection repose sur l'hypothèse que les classes sémantiques ne sont pas définies *a priori*, ni ne correspondent à de distributions syntaxiques particulières. Elles émergent en revanche à partir des comptabilités sémantiques selon la formule simple qui résume ainsi cette méthode : « il est possible affirmer que *X* est une classe ayant une réalité sémantique si tous ou partie de ses membres sont compatibles avec au moins deux autres classes *Y* et *Z*, déjà identifiées selon cette même méthode, par récursivité ». *Y* et *Z* sont dits les *sélecteurs* de *X*. L'identification des premières classes est issue d'une méthode inductive posant des hypothèses elles-mêmes vérifiées selon le procédé schématisé ci-dessus.

L'intérêt théorique majeur est que le logiciel permet la création d'ontologies sur base d'occurrences attestées. Grâce aux requêtes récursives que permet le logiciel, on peut dégager des « restrictions » de sélection plus fine, obtenues à partir du corpus, et non plus mappées à partir d'un travail manuel sur des ontologies. L'idée sur laquelle repose cet intérêt théorique est que les distributions dégagent des classes partageant certaines mais pas toutes les caractéristiques et que les requêtes enchaînées permettent de dégager des propriétés transversales à l'ensemble de la classe et des propriétés spécifiques à un sous-ensemble des éléments. Pour clarifier notre propos, nous donnons ici une illustration d'une requête récursive :

Exemple :
Requête *n*
DET:ART + « chat » + 'manger' + DET:ART + ~Nom
Requête *n+1*
~Nom + ADJ

L'intérêt des requêtes récursives dans cet exemple est que le sélecteur ADJ dans la requête *n+1* ne sélectionnera qu'une partie seulement des noms obtenus lors de la requête *n*.

Module de gestion et de création de corpus

L'objectif poursuivi par l'intégration de ce module de gestion de corpus était de permettre à l'utilisateur de créer ses propres corpus à partir de différentes sources d'informations (courriels, forums de discussions, article de presse, textes journalistiques, …) dans des formats divers. L'utilisateur est libre d'utiliser soit un texte déjà balisé (en partie du discours), soit un texte brut. Toutefois, afin d'exploiter toutes les possibilités offertes par le logiciel en terme de formulation de requête, le corpus à partir duquel ces requêtes vont être effectuées doit être balisé. C'est la raison pour laquelle, nous avons intégré un logiciel de balisage en partie du discours à notre logiciel. Donc, si l'utilisateur choisit l'option d'utiliser un texte brut, il a la possibilité de baliser son texte dans le logiciel (pour la version Linux/UNIX) ou en ligne (pour la version MS Windows). Le taggueur que nous avons choisi d'intégrer à notre logiciel est TreeTagger (http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html). La version Windows du logiciel ne peut intégrer TreeTagger pour des raisons de licence. Une fois le corpus taggué, il est stocké sous la forme d'une base de donnée dans le logiciel. Toutes ces actions se font de la manière la plus conviviale qui soit vu que les fonctions de création et de gestion de corpus ont été implémentées à la manière de l'ouverture d'un nouveau document dans un logiciel permettant de faire du traitement de texte.

L'interface qui gère la partie création et gestion de corpus possède les fonctionnalités suivantes :

1. une fonction permettant d'insérer des corpus,
2. une fonction pour appeler le taggueur qui permet de tagguer un texte brut avant de l'insérer dans le logiciel,
3. tous les corpus indexés sont accessibles à tout instant,
4. le corpus doit pouvoir être visualisé à n'importe quel moment du traitement par appel d'un éditeur.

Module d'extraction et de formulation de requêtes

Afin de faciliter la tâche de l'utilisateur, nous avons élaboré une syntaxe de requête simple mais puissante. Cette syntaxe offre la possibilité à l'utilisateur d'utiliser différentes informations dans une même requête. Ces informations peuvent être morphologiques, syntaxiques ou sémantiques. Lorsque l'utilisateur formule une requête, il a la possibilité d'introduire dans sa requête des occurrences, des lemmes, des balises... Il peut également mélanger ces différents types d'informations. Pour permettre à l'utilisateur d'utiliser cette fonctionnalité, nous avons défini une grammaire de requêtes.

Afin de satisfaire les besoins des utilisateurs, nous avons définis un grammaire de requête propre à notre logiciel. Nous avons d'abord définis la liste des terminaux que la grammaire doit pouvoir gérer. Ces terminaux sont les suivants :

1. des balises (c'est-à-dire les parties du discours) : en majuscule : DET:ART,
2. des occurrences (des mots simples tels qu'ils sont attestés - y compris les pluriels, les verbes déclinés etc. …) : mots placés entre apostrophes : 'chiens', 'chat'…
3. des lemmes (les entrée dictionnaire des occurrences) : mots placés entre des guillemets : « dire », « chien »…
4. des classes (des listes d'occurrences sémantiquement proches) : définies par l'utilisateur, précédées du symbole ~ : ~Animaux
5. des rôles (des listes d'occurrences jouant le même rôle sémantique) : définies par l'utilisateur, précédées du symbole \ : \Mouvement
6. une fonction de "saut". Il s'agit d'une fonction qui permet d'ignorer tout ce qui est contenu entre des autres types de terminaux. Les sauts peuvent être bornés ou non suivant la syntaxe [*m*-*n*] indiquant qu'il faut ignorer entre *m* et *n* mots. Par exemple, la requête "TAG + [4-6] + ~Animaux" signifie qu'un intervalle de minimum quatre et maximum six mots doit être compris entre la balise « TAG » et un des lemmes de la classe ~Animaux. Autre exemple, à la requête "DET:ART [] chat" va correspondre le texte "le beau chat" car "beau" aura été matché par [].

Comme nous l'avons, mentionnés précédemment, toutes les suites possibles de terminaux peuvent être l'objet d'une requête. Voici quelques exemples de requêtes à titre d'illustration.

1. TAG + TAG — exemple : DET:ART + NOM
2. TAG + occurrence — exemple : DET:ART + 'chat'
3. TAG + classe — exemple : DET:ART + ~Animaux
4. Classe + [] + occurrence
5. …

De plus, la grammaire de CorpuSearch comporte les opérateurs suivants:

1. | : correspondant au « ou »
2. - : correspondant à l'exclusion d'un élément dans les listes (ex. ~Animaux-« chat » ...)

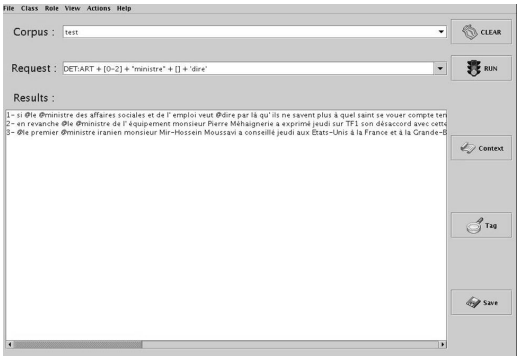
Enfin, l'utilisateur à la possibilité de voir le résultat de sa requête soit sous la forme de phrases telles qu'elles existent dans le corpus, soit sous la forme de liste en entourant d'accolades la partie de la requête dont il souhaite récupérer la liste.

Afin de venir en aide à l'utilisateur, plusieurs fenêtres d'aide contextuelle ont été ajoutées au logiciel. Ces fenêtres d'aide rappellent à l'utilisateur quelle est la syntaxe de formulation de requêtes, quelles sont les balises disponibles, …

L'aspect dynamique qui fait la spécificité et l'originalité de ce module permettant la formulation des requêtes est que l'utilisateur a la possibilité de réutiliser les résultats (par exemple les classes) dans une nouvelle requête sans devoir changer d'environnement de travail.

Afin d'illustrer notre propos, nous présentons ici différentes requêtes telles qu'elles ont été formulées dans CorpuSearch.

Extraction d'occurrences :
Exemple : "président"
Extraction à partir des lemmes :
Exemple : 'dire' => dit, disons, dire, disent…
Extraction à partir des classes syntaxiques :
Exemple : DET:ART
Fonction de saut:
Exemple : DET:ART + [0-2] + "ministre" + [] + 'dire'
Récupération de listes :
Exemple : {DET:ART} + [0-2] + 'ministre' + [] + 'dire' => permet de récupérer la liste des articles précédant le lemme ministre.

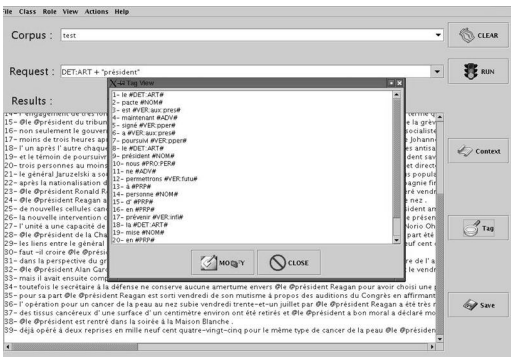


Afin de rester le plus proche possible de la terminologie linguistique, le logiciel propose les classes et les rôles comme deux entités distinctes. Toutefois, du point de vue du logiciel, ces deux concepts sont représentés sous la forme de listes d'occurrences et sont gérés de manière identique par le moteur. L'utilisateur peut soit créer ces listes manuellement, soit utiliser les résultats de requêtes pour créer ces listes. Supposons que l'utilisateur s'intéresse à la classe sémantique des métiers et qu'il souhaite étudier les verbes d'actions qui sont utilisés avec les membres de cette classe. Il crée alors un fichier dans lequel il met une occurrence représentant un métier par ligne. Il enregistre ce fichier en lui donnant le nom de la classe (i.e. ici Métier). Il peut alors commencer des requêtes en utilisant cette classe *~Métier*. Chaque fois que le logiciel rencontre un phrase contenant un des mots de cette classe, il l'affiche.

Module de visualisation et d'exploitation des résultats des requêtes

La partie du logiciel concernant les résultats des requêtes permet deux choses : d'une part, elle permet à l'utilisateur de visualiser les résultats de la requête qu'il a formulée ; d'autre part, l'utilisateur a la possibilité de ré-utiliser les résultats de la requête de manière récursive et également d'éditer les résultats afin de modifier les éventuelles erreurs de balisages.

Concernant la visualisation des résultats, le logiciel est doté de fonctionnalités permettant à l'utilisateur d'une part de déterminer la taille du contexte (c'est-à-dire le nombre de phrases qui se trouvent avant et après dans le corpus); entourant la/les phrase(s) qui correspondent à la requête formulée. D'autre part, le logiciel offre également la possibilité de voir la phrase telle qu'elle a été balisée par TreeTagger. En cas d'erreur de balisage, l'utilisateur peut éditer le fichier contenant la phrase et modifier le balisage manuellement. Enfin, il a également la possibilité de créer un nouveau corpus en prenant l'ensemble des phrases résultats.



Logiciel libre

Le langage de programmation utilisé pour le développement du logiciel est Java, un langage portable sur différents systèmes d'exploitation. Comme nous l'avons mentionné dans l'introduction, le logiciel que nous avons développé est un logiciel libre dont la distribution est gratuite. Par « libre », on entend le fait que le logiciel est fourni avec son code source, c'est-à-dire que tout utilisateur a le droit d'utiliser ce logiciel, mais également de modifier le code source afin d'adapter le logiciel à des besoins plus spécifiques. De plus, ce logiciel est distribué gratuitement. Nous espérons à travers ce logiciel développer une communauté d'utilisateurs et de développeurs afin de partager de manière plus efficace les ressources linguistiques et les outils informatiques.

Améliorations et développement futurs

Lors de la première utilisation du logiciel, le linguiste peut être surpris par la lenteur lors du premier chargement d'un corpus balisé dans la base de donnée. Cette lenteur s'explique par le choix d'utiliser une base de donnée pour stocker les informations ce qui permet en revanche une grande rapidité lors des requêtes. Nous avons également rencontrés d'autres problèmes causés par les lenteurs de Java pour l'affichage des résultats des requêtes lorsque le nombre de phrases est très important.

Concernant les développements futurs, il nous semble indispensable d'ajouter au logiciel des outils statistiques permettant de quantifier les résultats. De plus, comme nous l'avons mentionné dans la partie concernant le balisage du texte brut, seule la version Linux du logiciel intègre un taggueur. Pour les utilisateurs de Windows, nous avons mis une version en ligne de *TreeTagger* (http://perso.enst.fr/~demoulin/treetagger/). A notre connaissance, il n'existe pas de logiciel libre et gratuit permettant de tagguer des corpus. Toute bonne volonté est évidemment la bienvenue pour participer aux développements futurs du logiciel. Nous espérons, en fournissant le code source du programme gratuitement, créer une communauté d'utilisateurs et de développeurs souhaitant contribuer à l'évolution du programme soit en participant aux tests du logiciel, soit en contribuant au développement du logiciel par l'ajout de fonctionnalités que les utilisateurs trouveraient intéressantes.

Téléchargement et disponibilité du logiciel

Le logiciel CorpuSearch est disponible gratuitement en téléchargement à l'adresse suivante http://picolibre.enst-bretagne.fr/projects/csearch. A l'heure actuelle, il existe une version fonctionnant sous Windows et une version fonctionnant sous Linux.

Remerciements

Nous tenons ici à remercier les élèves de l'Ecole Nationale Supérieure de Télécommunications pour l'implémentation logicielle de *CorpuSearch*. Ces quatre élèves sont Caroline Vignoles, Damien Tabusse, Cathy

Nouvet et Edouard Marques. Nous tenons également à remercier M. Samuel Tardieu, Maître de Conférences au Département Informatique et Réseaux de l'ENST pour ces conseils précieux concernant l'implémentation ainsi que M. Talel Abdessalem, Maître de Conférences au Département Informatique et Réseaux de l'ENST, pour son aide concernant l'utilisation des bases de données. Enfin, nous tenons également à remercier le professeur Jacques Jayez qui a eu l'idée de base qui a permis de spécifier le cahier de charge de ce projet.

Références

[1]Cadiot, P. (1997). *Avec*, ou le déploiement de l'éventail. In C. Guimier (éd.), *Co-texte et calcul du sens*. Caen : Presses Universitaires de Caen, pp. 135-155.

[2]Fellbaum, C., Martha P., Hoa ŦD., Lauren D. & Susanne W. (2001). `Manual and automatic semantic annotation with WordNet." In: *Proceedings of the NAACL 2001 Workshop on WordNet and Other Lexical Resources*, Pittsburgh.

[3] Grivel, L., Guillemin-lanne, S., Lautier, C. & Mari A. (2000). La construction de composants de connaissance pour l'extraction et le filtrage de l'information sur les réseaux. *ISKO* : Paris.

[4]Habert, B., Nazarenko, A. & Salem, A. (1997). *Les linguistiques de corpus*. Armand Colin : Paris.

[5]McHenry, T., Wilson A. (1996). *Corpus Linguistics*. Edinburgh University Press: Edinburgh.