Université Paris III - La Sorbonne Nouvelle

Master Ingénierie Linguistique et Traitement Automatique du Langage
Parcours Recherche&Développement

Mémoire de Master 2

Sciences du Langage

Présenté et soutenu par

Ilaria TIDDI

Découverte de patrons de dépendances pour la construction d'ontologies

Mémoire dirigé par Mme Isabelle Tellier

préparé à l'Ecole Centrale Paris, Projet parlance

7 septembre 2012

Jury :

Encadrante de Mémoire :	Isabelle Tellier	-	Paris III
Encadrante de Stage :	Marie-Aude Aufaure	-	ECP
Examinateur :	Sylvain Kahane	-	Paris X

Remerciements

Je voudrais remercier toutes les personnes, proches ou pas, sans lesquelles ce projet n'aurait jamais vu sa fin.

Tout d'abord, je tiens à remercier Mme Tellier pour m'avoir suivie dans l'écriture du mémoire, pour la patience et l'attention qu'elle a employées pour corriger toutes mes erreurs, pour avoir répondu à mes questions et avoir éclairci mes doutes, mais surtout pour ses enseignements précieux qui m'ont aidée dans mes choix futurs.

Un remerciement particulier va à Mme Aufaure pour m'avoir donné l'incroyable possibilité d'effectuer mon stage à l'Ecole Centrale Paris, sans lequel je n'aurais jamais compris que, lorsqu'on veut quelque chose, on peut l'obtenir. J'en profite pour remercier toute l'équipe de *Business Intelligence* pour tous les moments et pauses café/baby-foot passés ensemble. Merci à Nesrine et Yves pour avoir suivi soigneusement mes travaux pendant six mois.

Merci aussi à M. Kahane pour tout ce que j'ai appris pendant ces deux ans grâce à lui : ses cours, que j'ai suivis avec enthousiasme, ont conditionné mon futur et je ne peux qu'en être heureuse. Je le remercie aussi pour avoir cru en moi et mes capacités, alors que je ne voyais pas mon potentiel.

Je remercie tous les profs du Master PluriTAL pour m'avoir transmis leur passion pour le TAL. En particulier je tiens à remercier M. Serge Fleury qui a mis en place ce Master et qui a cru dans mes capacités rien qu'en regardant mon dossier "atypique". Je m'excuse surtout pour le grand nombre de mails que je lui ai envoyées, auxquelles il a toujours répondu avec une telle rapidité et attention au point de me faire croire qu'il soit un héros qui ne dort jamais.

Le plus grand remerciement va à ma famille qui, même si loin, n'a jamais arrêté de me soutenir : à mes parents, sans lesquels je ne serais arrivée jusqu'ici, pour avoir toujours appuyé mes choix même s'ils ne les comprenaient pas, et à mes frères, pour le lien qui ne nous sépare dans aucune situation. Peu importe la distance, je sais que vous seriez toujours à côté de moi.

Une pensée particulière va à deux personnes qui ont suivi de près mon travail, directement ou pas : Emanuele et Mario. Merci des conseils, des corrections, des moyens brusques mais fraternels, vous êtes ce que je veux devenir et vous m'avez aidée à trouver la bonne voie pour le faire.

Last but not least, Arnaud, mon épaule, mon "prof", mon exemple à suivre, je ne serais pas ici sans toi. Merci pour le soutien, les moments ensemble et ceux où on ne l'était pas, les soirées passées à se prendre la tête sur mes programmes, pour croire toujours en moi et me suivre dans toutes mes folies. Qui sait ce que le futur nous réserve, l'important est qu'on le fasse ensemble.

Ringraziamenti

Vorrei fare un ringraziamento a tutte le persone che hanno fatto parte di questo lavoro, direttamente e non.

Ringrazio Isabelle Tellier per avermi seguita durante la stesura della tesi, per aver risposto alle mie infinite domande, per avermi corretto pazientemente e per tutti i suoi insegnamenti che mi hanno aiutato a prendere delle decisioni importanti per il futuro.

Un ringraziamento particolare a Marie-Aude Aufaure, per avermi dato la possibilità di fare il tirocinio al laboratorio MAS dell'École Centrale Paris, con il quale ho potuto capire che, quando si desidera fortemente qualcosa, é davvero possibile ottenerlo. Ne approfitto per ringraziare tutto il gruppo *Business Intelligence* per i momenti e pause caffè/biliardino passati insieme. Grazie a Yves e Nesrine che hanno seguito e diretto il mio lavoro per sei mesi.

Grazie anche a Sylvain Kahane per quello che mi ha insegnato in questi due anni : i suoi corsi, che ho seguito con entusiasmo e attenzione, mi hanno aiutata a trovare la mia strada. Grazie anche per aver creduto in me e nelle mie capacità anche quando io non mi rendevo conto del mio potenziale.

Ringrazio i prof della specialistica PluriTAL, per avermi trasmesso la loro passione per il TAL. In particolare, grazie mille a Serge Fleury per aver organizzato il corso di laurea e per aver accettato un curriculum così "atipico" come lo era il mio. Mi scuso per tutte le e-mail che gli ho mandato, a cui ha sempre risposto con una tale rapidità al punto che ora sono convinta che Lui sia un supereroe che non dorme mai.

Il grazie più grande va alla mia famiglia, che non ha mai smesso di appoggiarmi anche se a distanza : ai miei genitori, senza i quali non sarei mai arrivata qui, per credere sempre nelle mie scelte anche senza capirle, e ai miei fratelli, il mio esempio da seguire in ogni situazione. Nonostante le distanze so che sarete sempre con me.

Un pensiero particolare va a due persone che mi hanno seguita, direttamente e non, in questo lavoro : Emanuele e Mario. Per i consigli, per le correzioni, per i modi bruschi ma fraterni, siete quello che "vorrei fare da grande" e mi avete aiutato a trovare la strada giusta.

Last but not least, Arnaud, la mia spalla, il mio "prof", il mio esempio da seguire, non sarei mai arrivata fin qui senza il tuo aiuto. Grazie di tutto, per i momenti insieme e quelli in cui siamo stati lontani, tutte le serate passate ad arrovellarci il cervello sui miei programmi, per credere sempre in me e seguirmi in tutte le mie pazzie. Non so cosa ci riservi il futuro, ma l'importante é che saremo insieme.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Sommaire

Ce mémoire s'inscrit dans le cadre du travail effectué pendant mon stage de recherche au sein de l'Ecole Centrale Paris. En particulier, ce travail fait partie du Projet Parlance, qui sera introduit en détail par la suite.

Le sujet concerne des domaines très variés : si d'un côté cela a permis d'avoir une vue d'ensemble sur les grandes thématiques du *Traitement Automatique des Langues* (TAL), de l'autre j'ai pu mettre à profit toutes les connaissances que j'ai acquises pendant ces deux années de Master.

1.1 Contexte

Le *World Wide Web* représente aujourd'hui une source d'information en constante évolution. L'accès au réseau Internet étant de plus en plus disponible, la quantité d'informations contenues dans le Web, différant par contenu, type, forme ou thématique, a connu une croissance exponentielle.
Face à cela, accéder rapidement et automatiquement à ces informations est devenu fondamental. La dernière décennie a connu l'essor du **Web Sémantique** (ou Web 3.0, ou Web des données) en réponse à un besoin de représenter les contenus textuels du Web de façon sémantiquement structurée, afin de faciliter le traitement automatique et l'interrogation avec des moteurs de recherche sémantiques. Cependant, les contenus non-structurés du Web des documents restent encore peu exploitables par les technologies du Web Sémantique. D'autre part, il est encore difficile d'envisager un passage à l'utilisation à large échelle du Web Sémantique de la part des utilisateurs, à cause de la grande quantité d'informations et domaines à

traiter.
Dans cette optique, il faut envisager des technologies capables de *comprendre* les textes en langage naturel, et les transformer en information structurée. C'est ici que le travail d'un ingénieur et celui du linguiste se rencontrent : les techniques linguistiques sont la clé pour l'analyse des contenus, alors que l'ingénierie est nécessaire pour savoir comment structurer efficacement ces informations. Ainsi faisant, le coût de l'intervention humaine se réduirait considérablement à l'avantage de l'exploitation automatique.

Ce travail de recherche est situé dans cette problématique : en cherchant à combler le vide entre information structurée et non-structurée, il faut exploiter les connaissances du langage naturel afin de rendre efficaces les technologies sémantiques existantes. Le long du travail, nous remarquerons une continuité qui s'inscrit entre le structuré et non-structuré, où la syntaxe et sémantique sont les acteurs principaux pour la réalisation de cette continuité.

1.2 Le Traitement Automatique des Langues

Le **Traitement automatique des Langues**, ou **Linguistique computationnelle** (en anglais, *Natural Language Processing*), est un domaine interdisciplinaire qui a connu un grand essor dans les dernières décennies, parallèlement au développement technologique. Le but principal de cette discipline est de rendre les machines capables d'atteindre le niveau humain dans les tâches telles que la communication, qu'elle soit orale ou écrite [Jurafsky 2000].

1.2.1 L'informatique et la linguistique se rencontrent

Historiquement, le TAL est né comme discipline entre l'informatique et la linguistique, mais d'autres disciplines, comme les sciences cognitives ou l'électronique ont, de leur côté, abordé les mêmes thématiques.

Tout d'abord, l'objet d'étude du TAL est la "langue naturelle". Les langues définies "naturelles", en opposition à celles "artificielles" dont s'occupent l'informatique et la mathématique logique, sont des créations spontanées (auxquelles on ne peut pas attribuer de naissance précise) utilisées par les humains pour communiquer [Tellier 2008]. C'est la linguistique qui s'occupe d'étudier scientifiquement les langues naturelles, pour en découvrir des comportements communs, universels. Les linguistes fournissent au TAL les données à manipuler, et quelles règles les caractérisent. L'informatique, dont les fondements sont le codage de données et des traitements (les **algorithmes**), contribuent en modélisant les informations fournies par les linguistes.

Les bases Pendant les années '20-'30 les théories de Saussure (1857-1913) introduisent des concepts de linguistique générale sur lesquels le TAL fondera ses racines. Ces analyses sont ensuite promues par Roman Jakobson (1896-1982), qui apporte aussi des éléments importants pour l'analyse de la communication (les "six fonctions de la communication").
En 1936 Alan Turing, mathématicien anglais, idéalise pour la première fois un modèle de computation algorithmique. La *Turing's machine* donne pour la première fois une notion précise d'algorithme.

Les années '40-'50 Les premiers travaux en informatique peuvent dater des années '40-'50 : ils se basent sur les modèles de computation algorithmique de Turing (1950). Les travaux s'intensifient sur deux paradigmes en particulier : les automates et les modèles probabilistes / de la théorie de l'information [Jurafsky 2000]. Les théories des automates amènent aux théories des langages formels : Noam Chomsky, en 1957, envisagea pour la première fois les automates à états-finis pour caractériser une grammaire.
Le deuxième paradigme voit le développement des algorithmes probabilistes, pour le traitement du langage écrit et oral : pour la première fois, le concept d'entropie est emprunté de la thermodynamique pour définir le contenu informatif d'un langage.
C'est aussi dans cette période qu'on voit la naissance des premiers appareils de reconnaissance vocale.

Les années '60 (1957-1970) Cette période est articulée entre une voie symbolique et une stochastique.
Les théories chomskiennes sont un point important du paradigme symbolique. Chomsky, qui soutient le primat de la **syntaxe** sur les autres domaines de la linguistiques, définit une hiérarchie (connue comme la "hiérarchie de Chomsky"), où les langages, artificiels et naturels, sont organisés selon leur complexité. Ses "grammaires génératives et transformationnelles" seront par la suite à la base des théories syntaxiques pour la représentation des langages naturels. La deuxième voie de recherche, issue de l'informatique, se développe sous le nom d'**Intelligence Artificielle** (1956) : elle est focalisée sur le raisonnement et la logique. La voie stochastique se développe plutôt des domaines de la statistique et l'ingénierie électronique : les théories bayésiennes et les réseaux de neurones sont utilisés pour la compréhension des textes en basant les observations sur les séquences de mots.
On voit en ce moment naître les premiers systèmes pour l'interprétation du langage naturel et la reconnaissance des textes, ainsi que les premiers corpora en ligne.

Les années '70-'90 Cette période voit l'essor des théories qui dominent le TAL encore aujourd'hui.
La voie stochastique s'affirme dans les algorithmes de reconnaissance et synthèse vocale, en exploitant particulièrement les modèles de Markov cachés (HMM) et les méthodes de décodage des canaux de communication bruités. À cette époque se dif-

fusent aussi les théories de la sémantique formelle pour la représentation des connaissances et la formalisation des raisonnements dessus [Tellier 2008], [Manning 1999]. On introduit alors dans cette modélisation des éléments de logique ainsi que la pragmatique (l'étude de l'utilisation du langage en contexte). Les graphes conceptuels et les réseaux sémantiques datent de cette époque.
Dans les années '80 on voit aussi un "retour à l'empirisme", où les modèles probabilistes et les approches dites *data-driven* (guidées par des données étiquetées) sont utilisés dans touts les champs du TAL, de la reconnaissance vocale, au tagging *part-of-speech* (attribuer une catégorie grammaticale à un mot) à la sémantique.
Les recherches s'intensifient vers la fin de cette période aussi dans la génération automatique des textes.

Affirmation du TAL comme discipline et essor du *Machine Learning* Les années '90 ont connu une intensification de la collaboration des différentes disciplines à l'intérieur du TAL.
La **linguistique de corpus** fait son apparition pour s'occuper des grandes quantités de données écrites et orales qui sont rendues de plus en plus disponibles (un exemple parmi tous : le *Penn Treebank* [Marcus 1993]). Les données annotées dans ces corpora font l'objet de l'étude de l'**apprentissage automatique** (issu de l'intelligence artificielle), qui produit des programmes capables d'apprendre et s'améliorer sur la bases de ces données. La communauté de l'apprentissage automatique contribue avec ses techniques pour l'analyse du langage naturel : les techniques comme les SVM (*Support Vector Machines*, pour la représentation géométrique de l'espace) et les modèles probabilistes (bayésiens et d'entropie maximale) deviennent un standard pour la linguistique computationnelle [Jurafsky 2000]. La capacité de stockage et la puissance de calcul des machines permettent aux systèmes *data-driven* de rejoindre une performance qui n'aurait pas pu être envisagée avant.

Le défi de nos jours Aujourd'hui, la croissance du Web permet d'avoir une énorme quantité de données à disposition pour l'analyse linguistique et augmente le besoin de systèmes de recherche et extraction d'information performants. Le défi d'aujourd'hui vise à réduire significativement l'intervention humaine pour le traitement de ces données en ligne.
Du côté de l'apprentissage automatique, le progrès des approches statistiques en particulier dans la traduction automatique ont démontré l'efficacité de l'**apprentissage non-supervisé** : les machines apprennent elles-mêmes les informations dont elles ont besoin sur la base de données non-annotées. Ces techniques ont l'avantage de réduire considérablement les coûts et la difficulté d'annotation des corpora.

1.2.2 Le processus de communication

La différence entre le TAL et les autres domaines exploitant des données est la connaissance de la langue : le but principal de cette discipline est de comprendre ce qui est exprimé, afin de pouvoir le re-exploiter.

La chaîne de la communication concerne plusieurs niveaux, plus ou moins représentables. Prenons par exemple la phrase

The Olive Garden is based in Orlando

D'abord, il est important de "reconnaître les mots", quelles séquences de sons et comment les analyser acoustiquement : c'est la tâche de la **phonétique** et de la phonologie. Dans notre exemple :

ðə ˈɑlɪv ˈgɑrdən ɪz ˈbeɪst ˈɪn ɔrˈlændoʊ

Ensuite, il faut connaître les variations des mots, donc connaître la **morphologie** de la langue :

$$is_{[3^{rd}pers,present]} \rightarrow be$$
$$based_{[pastParticiple]} \rightarrow base$$

Il est aussi important de tenir compte de la **syntaxe**, pour connaître les relations existant entre les mots et leur donner un ordre d'importance. Une première étape détecte le rôle grammatical de chaque mot (cf. 1.1) :

The	*Olive*	*Garden*	*is*	*based*	*in*	*Orlando*
DT	NNP	NNP	VBZ	VBN	IN	NNP
dét	nom prop	nom prop	verbe (3^apers, sing)	verbe (part. prés)	prép	nom prop

Table 1.1 – Attribution des rôles grammaticaux

Ensuite, les mots peuvent être regroupés en suites de mots qui "vont ensemble" (les syntagmes), jusqu'à représenter la phrase de façon arborescente (cf. 1.2).

The	*Olive*	*Garden*	*is*	*based*	*in*	*Orlando*
DT	NNP	NNP	VBZ	VBN	IN	NNP
DT	NP		VBZ	VBN	IN	NP
NP			VBZ	VBN	PP	
NP			VBZ	VP		
NP			VP			
S						

Table 1.2 – Regroupement en syntagmes (analyse syntaxique)

Finalement, la **sémantique** aide à apprendre le sens intrinsèque de la phrase, sur la base des *traits distinctifs* de chaque mot qui, en se combinant, peuvent donner

			Orlando est...
The Olive Garden is	**based in** **based by** **based on**	*Orlando*	un **lieu** l' **agent** de l'action la **source** (ça peut être un roman)

TABLE 1.3 – Exemple d'interprétations sémantiques

un nouveau sens. Le tableau 1.3 en est un exemple (bien que les phrases semblent absurdes, elle sont valables !).

La pragmatique est la discipline qui lie la phrase à son contexte. Par exemple, elle permet de déterminer, dans un contexte bien défini, que :

$$\begin{array}{c} \textit{Olive Garden} \xrightarrow{is-A} \textit{Restaurant} \\ \textit{Orlando} \xrightarrow{is-A} \textit{City} \end{array}$$

et non pas d'autres entités (*Orlando* peut bien être une personne !).

Suite à cette analyse, nous obtenons une représentation interne de l'information qui est élaborée grâce aux connaissances que nous avons acquises. De même, le trai-

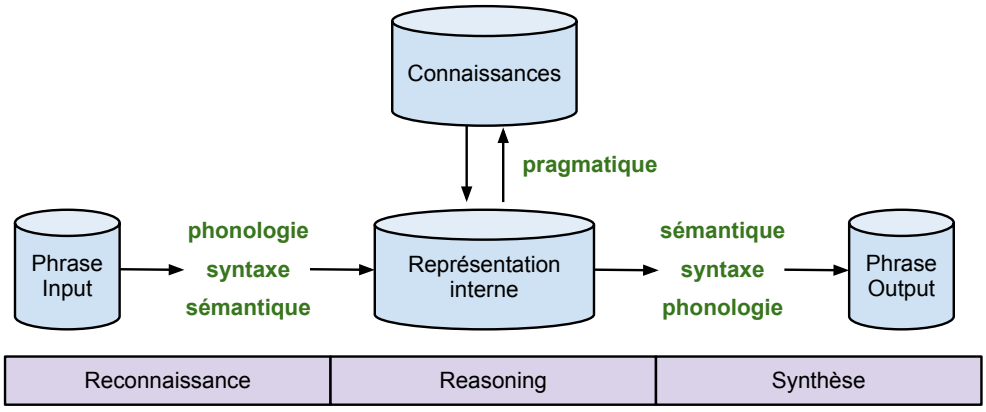


FIGURE 1.1 – Chaîne de traitement interne de l'information

tement du langage vise à créer des systèmes capables d'interagir en langue naturelle, potentiellement basés sur l'architecture représentée dans la figure 1.1. Nous verrons ensuite comment la technologie d'aujourd'hui s'est rapprochée de cet idéal là.

1.2.3 De la Syntaxe à la Sémantique

Avant de continuer, il faut présenter plus en détail deux niveaux d'analyse linguistique qui seront le fil conducteur pendant ce mémoire : la syntaxe et la sémantique.

La syntaxe, du grec *syntaxis* "mettre ensemble dans l'ordre", consiste à lier les unités lexicales de la phrase afin d'obtenir un énoncé. Celles-ci peuvent en effet s'organiser en structures hiérarchiques regroupant plusieurs mots en syntagmes (verbal, nominal etc), comme montré dans 1.2. Depuis les théories de Chomsky, l'analyse syntaxique a été largement étudiée afin de découvrir toutes les structures possibles existant dans une langue, c'est-à-dire la *grammaticité* de celle-ci. Nous avons vu comment la modélisation formelle de la syntaxe a été au centre des études pendant les dernières 50 années, en donnant lieu à des formalismes tels que les grammaires de constituants ou de dépendances. L'analyse syntaxique est donc à la base de tout processus de Traitement Automatique des Langues, en reconnaissance ou en génération.

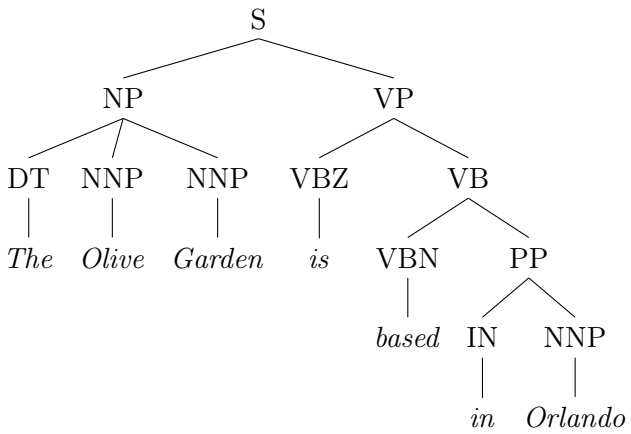


FIGURE 1.2 – Exemple d'arbre de constituants

Un autre aspect de la linguistique est autant important pour le TAL : la sémantique. C'est une discipline qui va au-delà de la linguistique, fortement liée aussi à la psychologie et, historiquement, à la philosophie. La sémantique étudie le "sens" des mots, la représentation interne des "choses" du monde externe. Définir ce que c'est le "sens" n'est cependant, pas une tâche facile. L'idée que les machines puissent comprendre les liens entre les unités lexicales a motivé les études en sémantique formelle et a fait que les technologies d'aujourd'hui reposent sur un interfaçage entre sémantique et syntaxe.

1.2.4 Le Web du XXI siècle

En 1998, Tim Berners Lee, co-inventeur du langage HTML, propose une nouvelle architecture pour le Web [Berners-Lee 1998]. L'idée principale consiste à représenter les informations dans les contenus textuels des pages (la "connaissance du Web"), dans un format plus structuré.
Les données dans les pages (ou contenus textuels), regroupées sous forme d'ontologie, sont l'une des bases de ce nouveau Web, dit *Web Sémantique* [Amardeilh 2007].

Une ontologie est définie communément comme une conceptualisation (ou structuration) d'un domaine [Zhou 2007]. Ces données sont ensuite annotées par des métadonnées sémantiques, pour faciliter leur exploitation.

Mais qu'est-ce qu'une métadonnée ?
En général, une **métadonnée**, littéralement "donnée à propos de donnée", est un marqueur qu'on introduit les fichiers (mais aussi dans les langages de programmation) qui fait référence à une information dans le texte. Dans le cas du Web Sémantique, les fichiers que l'on traite sont les pages Web. Ces pages sont constituées d'une partie textuelle (les "données") et d'une partie de métadonnées (en HTML, ce sont principalement des informations concernant la structuration de la page (1), comme `<div/>`, `<p/>` ou `<br/>`). L'objectif du Web Sémantique est de donner à ces métadonnées un côté plus sémantique (2), qui soit plutôt proche du "sens" de la donnée représentée que de la simple structure de la page :

(1)`<p>`*The Olive Garden chain is based in Orlando, Florida* `</p>`
(2) *The* `<restaurant>`*OliveGarden*`</restaurant>` *is based in* `<location>`*Orlando, Florida*`</location>`

À partir de cette organisation, plusieurs améliorations peuvent être envisagées :

- les moteurs de recherches bénéficient de la structure des ontologies et des métadonnées pour améliorer leurs résultats. La recherche est rendue plus efficace par rapport à la simple recherche *plain text* (consistant à examiner tous les mots des documents et trouver ceux qui correspondent aux mots saisis par l'utilisateur). De plus, les moteurs sont capables de dériver une connaissance qui est contenue dans le Web, mas n'est pas directement exprimée (on appelle ce processus "inférrer") ;
- les applications accèdent à une connaissance formalisée par les utilisateurs, ce qui encourage le partage, la visualisation et l'exploitation de mêmes données ;
- les systèmes d'information peuvent collaborer plus aisément, en intégrant les données provenant de sources hétérogènes dans les ontologies.

Depuis, différentes communautés, de l'ingénierie des connaissances au TAL, des systèmes collaboratifs à la recherche d'information, ont focalisé leurs recherches sur la création, l'exploitation et la réutilisation d'ontologies.

Même si les ontologies trouvent les applications dans un grand nombre de technologies, leur coût en terme de temps et effort de création reste encore haut. La difficulté principale réside dans le fait que les ontologies doivent avoir une couverture suffisante du domaine, mais aussi être capables de fournir des généralisations du modèle si besoin. De plus, la structure de l'ontologie doit être standardisée, afin que différentes communautés puissent les exploiter. Pour ce faire, le défi principal à l'heure actuelle reste l'acquisition automatique d'ontologies, c'est-à-dire leur construction en analysant des données de différentes sources de façon automatique ou semi-automatique. Cette discipline, appelée Apprentissage

automatique d'ontologies (*Ontology Learning*) a pris pied dans les dernières années et consiste à créer automatiquement de la connaissance ontologique en utilisant des techniques d'apprentissage automatique. Cela réduirait drastiquement le coût de l'intervention humaine et, par conséquent, augmenterait les possibilités d'exploitation de la part des utilisateurs. Le but ultime de cette discipline est de combler le vide entre les symboles, c'est-à-dire le langage naturel et les concepts, représentant les pensées humaines [Cimiano 2006b].

L'apprentissage d'ontologies s'inscrit exactement dans la continuité entre les données du Web non-structurées et le Web Sémantique. Cette discipline est au centre de ce travail de recherche, car elle intègre les techniques linguistiques à celle de l'apprentissage automatique.

1.2.5 Applications du TAL aujourd'hui

Le Traitement Automatique des Langues se retrouve dans certaines applications de tous le jours, sans que nous y fassions attention.

- Les **agents conversationnels** voient leur développement dans les technologies d'aujourd'hui. L'objectif étant d'assister un utilisateur dans ses besoins dans n'importe quel contexte. L'exemple le plus connu peut être le système SIRI© d'Apple [1], où l'assistant personnel, qui répond aux questions, fait des recommandations et des actions en utilisant les applications du système selon les besoins de l'utilisateur. À petite échelle, les agents conversationnels se retrouvent de plus en plus souvent dans les sites afin de faciliter, voire remplacer, la recherche par mots-clés. La reconnaissance vocale et la génération automatique de textes (ou synthèse vocale, à l'oral) ne sont que les tâches les plus "visibles" de la chaîne ; ces systèmes impliquent bien sûr des ontologies pour la représentation de la connaissance et le raisonnement, l'apprentissage automatique et l'analyse syntaxique automatique pour la compréhension du langage naturel.
- Sur la même voie, un autre type de systèmes devient de plus en plus fréquent : les systèmes question-réponse basés sur la recherche sémantique. L'idée principale est de créer des systèmes capables de répondre à des questions (fournir des documents Web ainsi qu'une réponse trouvée dans ceux-ci) telles que *Quand est-ce que Albert Einstein est mort ?* Pour ce faire, l'idée du Web Sémantique d'étiqueter les données avec des métadonnées sémantiques est donc fondamentale. L'inférence faite sur les connaissances des ontologies est la clé pour répondre à des questions, plus ou moins complexes. La classification trie le type de question demandée, alors que la recherche et l'extraction d'information cherchent l'information pertinente dans les textes. Certaines techniques linguistiques telle que l'analyse syntaxique, la désambigüisation des mots ou la correction grammaticale peuvent aussi aider la tâche d'extraction. Il est

1. http ://www.apple.com/iphone/features/siri.html

encore impossible d'effectuer une analyse syntaxico-sémantique complète, car en terme de temps de calcul cela coute encore beaucoup, mais les moteurs de recherche tels que Google, Yahoo ! et Bing sont en train d'intensifier leurs efforts afin d'améliorer la recherche sémantique. Des systèmes questions-réponse à citer sont START [2], AQUA [Enrico 2003] ou Brainboost [3].

- Considérant le grand nombre d'informations présentes dans des pages Web produites dans différentes langues, un autre grand défi du traitement du langage consiste dans la **traduction automatique**. Le but de cette discipline est de pouvoir traduire un document d'une langue dans une autre. Ceci est possible grâce à des analyses syntaxiques combinées avec des techniques statistiques. À présent, la traduction automatique se révèle utile si elle reste spécifique à un domaine, mais il est encore ardu de l'appliquer à grande échelle. De plus, les techniques statistiques impliquent des grands corpora, qui ne sont pas toujours disponibles. Une aide importante vient aussi des *corpora parallèles*, visant à aligner les données en différentes langues. Le système du *Google Translator* est le cas le plus connu de traduction automatique, mais d'autres outils existent, tels que les outils de traduction automatique fournis par *Systran* [4] ou l'outil *Trados* [5], plutôt pour l'assistance à la traduction.

1.3 Parlance, un projet pour le TAL

La présentation du Traitement Automatique des Langues nous amène à la contextualisation du travail ici présenté.

Le projet Parlance [6] est un projet de recherche européen (2011-2014) entre les Universités de Cambridge, Edinburgh, Genève, l'Ecole Centrale Paris, le groupe Yahoo !Research de Barcelone et la société Intelligent Software Components de Madrid. L'objectif du projet est la construction d'une application mobile se rapprochant de la performance humaine pour la communication homme-machine.

La problématique envisagée est que les systèmes actuels limitent leur recherches à une "recherche unique", sans explorer les interactions multiples avec l'utilisateur. Si l'utilisateur change ses critères de recherche, une nouvelle recherche est directement lancée : le système ne compte pas sur la possibilité d'améliorer les résultats précédents suite à une conversation incrémentale avec l'utilisateur.
La nouveauté du projet consiste donc à créer un système ayant plus d'aptitudes, notamment à reconnaître et générer de l'information de manière incrémentale en temps réel (c'est-à-dire, sans attendre la fin de la question de la part de l'utilisateur), en apprenant des nouvelles connaissances sans intervention humaine, et en prenant

2. http ://start.csail.mit.edu/
3. http//www.answers.com/topic/brainboost-1
4. http ://www.systran.fr/
5. http ://www.trados.com/en/
6. https ://sites.google.com/site/parlanceprojectofficial/home

en compte la personnalisation de l'utilisation. Les données sur lesquelles le système s'appuierait sont les données réelles du Web, qui seront simultanément utilisées en trois langes différentes (anglais, espagnol, mandarin), deux domaines d'application (les informations touristiques et la recherche d'objets) et en recherche interactive hyper-locale.

Idéalement, une conversation entre un utilisateur et le système se déroulerait comme suit :

U : Right, I'm looking for a house with three bedrooms, ...2 bathrooms with a garden.
S :.....uhuh.....[*La recherche commence : **incrementalité***].....There are 4 houses and 2 flats in a 3 mile radius from your current location that match your search [***recherche hyperlocale***]. 2 are by the river and 4 are in the centre of town. One near the river has been reduced in price.
U : Is it near a school ?
S : Yes it's near St Thomas School. [*"school" n'est pas dans le vocabulaire, il est ajouté : **dynamicité***]
S : Oh by the way there's an open house this afternoon.
U : Right I'll try that one. What's the address ?
S : 26 Waldorf drive.
U : Oh I don't like that street. [***personnalisation** : mise à jour du profil, la rue est marquée avec une "score de préférence" bas.*]
U : Are there any other properties near St Thomas School.
S : Yes, there is the other one I mentioned near the river. It is on Cambridge Road.
U : Oh that's a lovely street.[*Marquée avec un score haut*]
S : Great, it has an open viewing this afternoon or I can give you the estate agent's phone number.
U : Okay, I'll go to the viewing

Le projet touche donc plusieurs problématiques du traitement du langage que nous avons vues dans cette partie introductive. L'architecture du système est représentée dans la fig 1.3. Deux niveaux sont présents, un de connaissances et un de gestion de la parole. Une composante centrale s'occupe de l'échange des données entre ces deux parties. La composante de *speech recognition* s'occupe de comprendre l'information de l'utilisateur et la passe au composant de gestion (*interaction manager*). Celui-ci interroge la base de connaissance et le profil de l'utilisateur pour récupérer les informations nécessaires, qui sont ensuite passées au composant de *Speech Generation* qui répond à l'utilisateur en langage naturel.

Alors que d'autres participants s'occupent de la partie reconnaissance et syn-

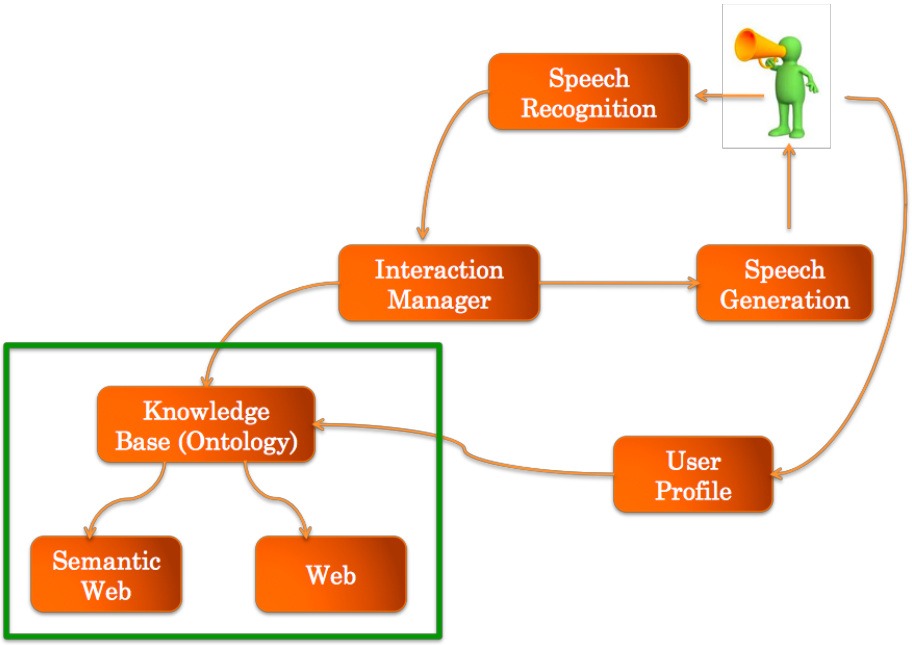


FIGURE 1.3 – Architecture du système PARLANCE

thèse vocale, notre contribution (le carré vert de la fig 1.3) se situe dans la gestion de la base de connaissance sur laquelle le système s'appuie. Comme l'on voit dans la figure, il s'articule entre le Web sémantique et le Web non-structuré. Deux grands objectifs sont donc au coeur de ce travail : d'un côté, il faut fournir une ontologie pour le système, non pas en créer une nouvelle mais exploiter les données déjà existantes dans la communauté ; de l'autre, il faut rendre l'ontologie capable d'apprendre la structuration d'un nouveau concept, si demandé, directement à partir des connaissances non-structurées. Ce travail concerne donc le Web Sémantique, l'apprentissage d'ontologies mais aussi l'extraction d'information à partir des textes et donc les aspects plus formels du langage.

Chapitre 2

Le Web Sémantique

Sommaire

2.1 Introduction

Le World Wide Web connaît depuis des années un développement incessant en terme de quantité d'informations disponibles et d'utilisateurs ainsi que de contributeurs. Son succès est dû principalement à la simplicité de sa structure, qui a permis de développer, fournir, atteindre et utiliser aisément du nouveau contenu [Lausen 2005].
En 2010, on compte que le Web contient plus que 2 milliards de pages [Patel-Schneider 2002]. L'accès aux informations qui y sont contenues devient enfin de plus en plus compliqué. Ces informations sont en effet exprimées en langage naturel, et ne sont pas accessibles par une machine (*machine-readable*). D'autre part, l'hétérogénéité des formats et des sources produit un éparpillement des informations : les mêmes données proviennent et sont encodées de façons différentes sans qu'il n'y ait aucun lien les rapprochant.

Face à cela, Tim Berners-Lee propose en 2000 une nouvelle architecture, pour "amener le Web à exploiter son vrai potentiel" [Studer 2003]. Le Web Sémantique a pour objectif de donner une structure aux contenus sémantiques du Web, grâce à laquelle les machines peuvent accéder aux données plus facilement. Plus précisément : rendre les machines capables de comprendre la sémantique des données et documents fournis dans le Web. Le Web Sémantique n'est en effet vu que comme une extension du Web actuel, dans lequel humains et machines peuvent

collaborer, au-delà des capacités actuelles, en utilisant une information dont le contenu sémantique est bien défini [Berners-Lee 2001], [Patel-Schneider 2002]. Ce processus permet aux machines de "comprendre" le sens des données utilisées.

2.1.1 Caractéristiques du Web Sémantique

Comme indiqué par [Studer 2003], les caractéristiques du Web Sémantique peuvent se résumer comme suit :

- fournir une syntaxe commune pour les déclarations conçues pour les machines ;
- établir des vocabulaires communs ;
- définir un langage logique de référence ;
- utiliser celui-ci pour l'échange de vérités.

Le but du Web Sémantique est, tout d'abord, de rendre la connaissance "universelle". C'est en effet cette caractéristique du Web qui permet d'envisager la possibilité que l'information actuellement de nature aussi variée soit réunie en une connaissance universellement disponible, connectée au-delà des disciplines, des sources ou des langages. Autrement dit : *anything can link to anything*. De plus, grâce à ces connections entre les contenus des différentes sources, cette structure permettrait de décentraliser l'information.
La structuration de l'information rendrait les machines capables de raisonner automatiquement, ou bien de réaliser des inférences sur les données dont elles disposent. À l'heure actuelle, les systèmes sont limités dans leurs raisonnements car aucune sémantique n'est prise en compte dans la description des données exploitées. Au contraire, une organisation structurée avec des métadonnées sémantiques permet une interrogation plus efficiente et rapide. L'idée de base est, donc, de changer la façon de représenter la connaissance.
Cela peut être possible grâce à la définition de langages plus "sémantiquement expressifs", permettant d'établir et exprimer d'un côté les connaissances (en forme de données) et de l'autre, les règles avec lesquelles raisonner sur les premières. Un pas de plus vers le Web Sémantique est donc fait, grâce à l'entrée de la logique dans les systèmes.
Les informations sont collectées dans des ontologies. Ce concept a été repris de la philosophie, où il désigne une théorie cherchant à comprendre l'existence et la nature des choses. En informatique, les ontologies sont des documents définissant formellement des relations entre termes. Dans une ontologie, on retrouve d'une part des classes et des relations entre elles, qui définissent "la connaissance" du point de vue formel ; de l'autre, des instances des classes, c'est-à-dire comment cette connaissance est représentée dans le monde réel. L'universalité du Web Sémantique est donc rendue possible par une connexion que l'on établit entre les données qui expriment la même connaissance mais appartiennent à différentes ontologies.
Dans cette optique, l'idéal ultime du Web Sémantique est d'assister l'évolution de la connaissance humaine dans sa totalité, en la rendant disponible et échangeable entre

plusieurs communautés. À l'heure actuelle, en effet, les communautés se concentrent indépendamment sur les mêmes connaissances, rendent leurs données disponibles mais aucun lien n'est fait entre elles, ce qui rend la recherche de cette information plus longue et coûteuse. Le Web Sémantique minimaliserait ces efforts, en unifiant petit à petit ces informations et rendant le Web effectivement universel, comme il a été conçu.

2.1.2 Structure du Web Sémantique

Plusieurs étapes sont donc requises pour la réalisation du Web Sémantique. Cesci sont généralement représentées par une structure pyramidale[1], montrée en 2.1 et proposée pour la première fois par Tim Berners-Lee lors de la présentation du Web [Berners-Lee 1998]. Chaque niveau, plus complexe et spécialisé que le précédent, repose sur les résultats du niveau au-dessous. Cependant, les niveaux sont indépendants les uns des autres de façon à ce qu'ils puissent être opérationnels et développés de façon autonome [Amardeilh 2007]. Le Web Sémantique repose, pre-

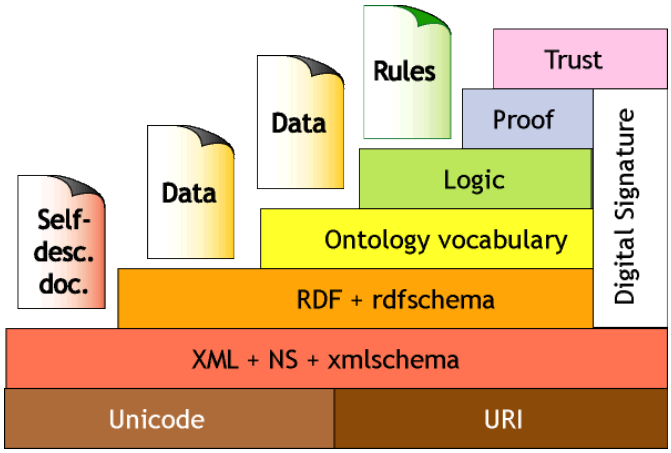


FIGURE 2.1 – La Pyramide du Web Sémantique

mièrement, sur les URIs et l'Unicode. Alors que ce dernier est le standard pour l'échange de symboles, c'est-à-dire comment les entités sont encodées, les *Uniform Resource Identifiers* fournissent des identifiants pour tout type de donnée à laquelle on fait référence [Patel-Schneider 2002]. Les URLs (*Uniform Resource Locator*), dont le nom est familier à tout utilisateur habituel du Web, n'est qu'une sous-classe des URIs, servant à identifier de manière unique des pages Web. Toutes les ressources auxquelles on fait référence dans le Web Sémantique est caractérisée donc par un URI.
Le langage XML et l'*XML Schema* (la grammaire validant un document XML) du

1. http ://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

deuxième niveau représentent la forme arborescente de représentation des informations. Les données créées au premier niveau doivent être représentées en format XML et validées par le schéma XML (ou DTD). Les *Namespaces* (NS), finalement, permettent de combiner plusieurs documents, représentant les mêmes entités mais avec un contenu différent.
Ces deux niveaux représentent la base syntaxique du Web Sémantique.
Les données sont ensuite représentées sous forme de triplets, en langage RDF (*Resource Description Framework*). Le RDF est la vraie première nouveauté du Web Sémantique, et a été défini par le W3C [2] comme "le modèle standard pour l'échange des données" [3]. Le langage RDF facilite la collecte et l'échange des données hétérogènes en forme ou source, facilitant la collaboration entre applications qui échangent une information compréhensible par les machines [Lassila 1999]. Plus simplement, le RDF définit un modèle de données qui rend compte de la sémantique de celles-ci. Le langage RDF représente la connaissance dans un format de graphe orienté (cf. 2.2).
Les descriptions RDF consistent en trois types d'objets : les ressources, les propriétés, les triplets. Les **ressources** sont tous les objets décrivant la connaissance (pages Web ou parties d'une page, concepts ou autre), identifiés par les URIs ; les **propriétés**, des attributs, caractéristiques spécifiques ou relations décrivant les ressources. Une ressource R_1 (dite Sujet) comme *Olive Garden*, et liée à une ressource R_2 comme *United States* (dite Objet), par une relation r (dite Prédicat) *located in*, forme un **triplet**, ou *statement*. Comme le XML *Schema*, le schéma RDF (RDFS) exprime les contraintes sur les entités représentées, en établissant des relations *instanceOf* (instance-de) ou *isA* (est-un) entre les classes et relations (*Olive Garden is a Restaurant*).
Comme introduit dans la section précédente, ces données sont ensuite collectées dans des ontologies. Le langage RDF permet de représenter un premier niveau d'ontologie

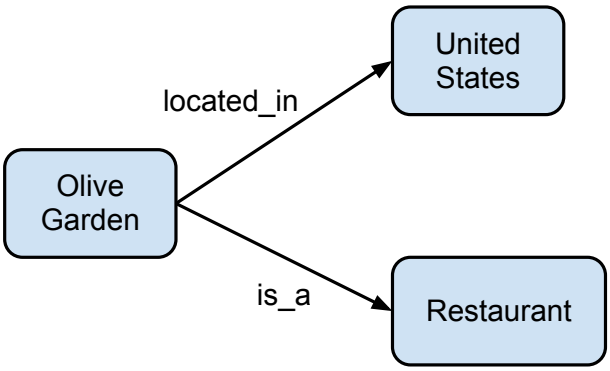


FIGURE 2.2 – Exemple de graph orienté

2. World WIde Web Consortium, abrégé W3C, fondé par Tim Berners-Lee pour encourager la compatibilité entre les différentes technologies du Web
3. http ://www.w3.org/RDF/

relativement simple. Mais afin de pouvoir représenter une connaissance plus complexe, il est nécessaire de définir un langage plus complexe, permettant un raisonnement de la part des machines. Le langage des ontologies est un ensemble restreint de contraintes logiques qui définissent une terminologie. Ceci est rendu possible grâce au langage OWL (*Ontology Web Language*) qui, plus expressif par rapport au RDF, définit des classes, des attributs, des relations et des axiomes et les combine aux opérateurs logiques pour faire des raisonnements. Un vocabulaire n'est, finalement, qu'un ensemble de classes et propriétés. Comme le langage OWL et les ontologies sont basés sur la logique, ce niveau a tendance à être assimilé avec le niveau supérieur [Studer 2003]. Dans cette étape "ontologico-logique" finalement, il est possible de faire des déductions logiques, c'est-à-dire d'inférer des conclusions non-explicites, si *OliveGarden-isA-Restaurant*, *OliveGarden-locatedIn-UnitedStates* et *Restaurant-locatedIn-Location* alors *UnitedStates-isA-Location* (cf. 2.3).

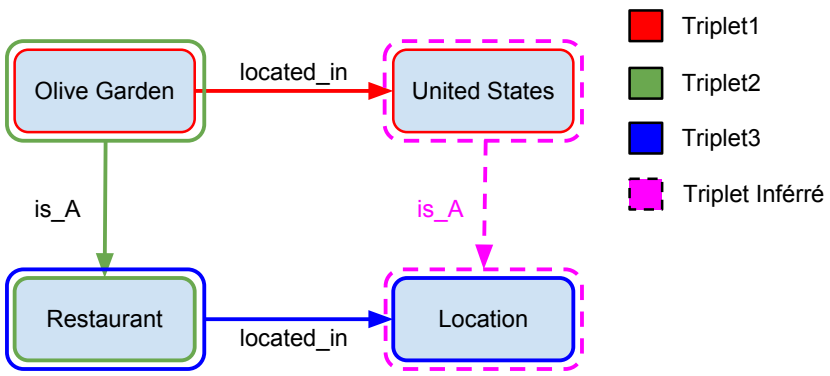


FIGURE 2.3 – Exemple d'inférence

Les derniers niveaux concernent la vérification de la validité des raisonnements. Les machines doivent avoir donc les moyens d'expliciter les étapes du raisonnement qui ont amené à un certain résultat. Cela consiste donc à authentifier les données fournies en entrée.

2.1.3 Applications du Web Sémantique

Mais quelle discipline peut bénéficier du Web Sémantique ?
Le Web Sémantique est tout d'abord un stockage massif de données : autrement dit, cette discipline est fortement liée à celle des bases de données. Le RDF permet le stockage de données et métadonnées, mais aussi la transformation des bases de données relationnelles en triplets. Les modèles ontologiques, qui ne sont que des grandes bases des données, ont cependant l'avantage de permettre la déduction et l'inférence sur les données non explicitement déclarées. Cependant, elles nécessitent

encore des efforts pour améliorer la consistance et le passage à l'échelle des méthodes pour leur traitement et interrogation.

De l'autre côté, le *Web Mining* ("fouille du Web") intensifie actuellement ses efforts afin d'exploiter les techniques de fouille de données pour extraire l'information dans le Web. On parle alors de *Web of Data* ("Web de données") plutôt que de Web Sémantique, en opposition au Web 2.0 dit "documentaire" ou "des documents", qui n'est qu'un énorme ensemble de documents. Alors que les métadonnées sémantiques améliorent la recherche d'information dans les pages Web, les *Web Mining* profite des techniques de TAL pour traduire directement les contenus de ces pages pour le peuplement d'ontologies et le liage des données.

2.2 Ontologies du Web Sémantique

Il est impossible de présenter la totalité des ontologies existantes à l'heure actuelle, car la récente évolution du Web Sémantique a fait que le nombre de données disponibles a augmenté considérablement. Nous nous limitons ici à donner une vue globale des ontologies existantes les plus connues.

2.2.1 DBpedia

Poussée par la volonté d'intégrer les données existantes et le fait que les ontologies ont tendance à se spécifier dans un domaine trop restreint [Auer 2008], la communauté du Web Sémantique a ressenti le besoin d'une information plus organisée et de plus large couverture que précédemment. Promu par les universités de Berlin, de Leipzig et de Philadelphie, ce projet est actuellement un des plus gros efforts communautaires pour l'extraction et la structuration de l'information pour le Web Sémantique [Bizer 2009b].
Le projet vise à exploiter l'énorme potentiel de Wikipedia [4] : une connaissance encyclopédique qui va au-delà des domaines et des langues en mettant en relation les plus possible d'entités y existant. Cependant, cette information se révèle être parfois inconsistante, ambigüe, et de provenance inconnue ou fausse [Bizer 2009b]. De plus, la recherche d'information dans Wikipedia est purement textuelle, ce qui peut se révéler extrêmement limitant. L'uniformité et la structuration deviennent donc fondamentales.
Le projet DBpedia a comme but principal de structurer les contenus textuels de Wikipedia afin de rendre cette encyclopédie exploitable par les technologies du Web Sémantique. Contrairement aux techniques *top-down*, où le design du schéma conceptuel vient avant les données réelles (population de l'ontologie), la construction de DBpedia dérive directement des informations textuelles de Wikipedia [Auer 2008]. Encore plus important, l'ontologie DBpedia évolue suivant l'évolution de Wikipedia, ce qui lui permet de rester une base dynamique et à jour. Les données, transformées en RDF, forment donc une base de données multi-domaines et multi-langue, ex-

4. http ://www.wikipedia.org/

ploitable par d'autres technologies, ainsi que librement accessible dans le Web. La couverture multi-domaines a fait que les bases ontologiques existantes ont référencé leurs entités vers celles de DBpedia, ce qui l'a rendue une des bases de références pour le Web Sémantique. De plus, en cohérence avec les principes du Web Sémantique, ces données sont aussi connectées à d'autres datasets. Le résultat consiste en plus de 2 milliards de triplets RDF, décrivant plus de 2,5 milliards de données, incluant personnes, compagnies, films et lieux [Bizer 2009b].

Processus d'Extraction Les données Wikipedia pour chaque langue sont fournies en dumps régulièrement mis à jour [5]. Ces dumps sont l'input pour l'extraction des informations. En particulier, les contenus Wikipedia les plus exploitables par DBpedia sont les *infobox*. Les infobox (cf. 2.4) représentent les informations pertinentes pour une entité dans des tableaux organisés en attributs-valeurs, situés en haut à droite de la page Wikipedia. Les triplets sont créés à partir de ces informations : l'URI du Sujet, c'est-à-dire de l'entité de référence, est défini en l'extrayant à partir de l'URL de la page

`http://en.wikipedia.org/wiki/Olive_Garden`
→ `http://dbpedia.org/resource/Olive_Garden`

Pour chaque attribut de l'infobox, une propriété est créée avec le *namespace* `http://dbpedia.org/property/` (→ `http://dbpedia.org/property/KeyPeople`). Les attributs peuvent être génériques, comme le nom (*label*) ou les informations liées à la structure de la page Wikipedia (*abstract, wikipediaPageName, image*) ou bien spécifiques à l'entité (*products*).

Pour l'objet, il faut un processus en plus pour distinguer s'il s'agit d'une ressource ou d'un *datatype*. Pour ce faire, 55 types de valeurs littérales ont été établies manuellement. Les entités sont aussi classifiées suivant 4 schémas ontologiques différents : (i) les catégories Wikipedia (plus de 415.000), (ii) les classes YAGO (plus de 286.000 catégories), (iii) les classes UMBEL (plus que 20.000 dérivées de l'ontologie OpenCyc) et (iv) les classes de l'ontologie DBpedia (une hiérarchie manuellement créée de 170 classes).

Applications Existantes Les données DBpedia sont accessibles de différentes façons.

- *Linked Data* (cf. ci-dessous). Les données DBpedia sont rendues accessibles en format RDF et connectées à d'autres datasets, pour qu'elles soient accessibles par les moteurs de recherche et les navigateurs sémantiques.
- Des *endpoint*s SPARQL. Ces services utilisent le langage SPARQL et ses extensions pour interroger la base de données DBpedia. Un exemple est l'*endpoint* hébergé par le Virtuoso Universal Server [6]

5. http ://dumps.wikimedia.org/
6. http ://virtuoso.openlinksw.com

FIGURE 2.4 – Exemple d'infobox pour l'entité Olive Garden

- Les dumps RDF. La base de connaissance DBpedia a été partagée en plusieurs sous-bases, rendues disponibles en plusieurs formats et langues sur le Web [7]
- *Lookup Index*. Un service Web [8] propose, pour un label donné, un URI DBpedia. Il utilise Lucene [9] pour indexer les informations et un algorithme de type PageRank pour trouver l'URI le plus pertinent.
- *DBpedia Mobile* [Becker 2008]. Il s'agit d'une application mobile issue du Web Sémantique qui utilise les locations DBpedia pour la navigation de l'utilisateur. En exploitant les informations du GPS, cette application fournit une carte contenant les points intéressants présents dans DBpedia. Il permet à l'utilisateur de publier, chercher ou découvrir des points d'intérêts en tant que *Linked Data*.
- *DBpedia Spotlight* [10]. C'est un outil de reconnaissance d'entités existant dans DBpedia dans un texte en langage naturel. Il permet d'effectuer des tâches d'extraction d'entités nommées et de reconnaissance de noms propres.

2.2.2 YAGO2

Suivant la tendance consistant à dériver des faits automatiquement à partir de Wikipedia, YAGO [11] (*Yet Another Great Ontology !*), évolué actuellement en YAGO2, est une ontologie développée et alimentée par le Max-Planck Institute de Saarbrücken présentant une large couverture de domaines et une grande precision [Suchanek 2007].
YAGO2 apporte une réponse à un manque de contrôle sur la qualité des informations fournies dans les technologies existantes : en prenant le cas de DBpedia, on remarque l'impossibilité de vérifier l'exactitude et la consistance de ses informations ainsi qu'un manque de structuration des relations (qui apparaissent plusieurs fois sous différents labels : *lenght, lenght-in-km, lenght-km...*). Le but prépondérant de DBpedia, en effet, reste de fournir une interconnexion entre différentes bases de connaissances et non pas de vérifier l'information fournie. Pour avoir une information fiable, l'intervention humaine est encore nécessaire. D'autre tentatives de "sémantisation de Wikipedia" ([Weber 2006], [Wu 2007], [Ponzetto 2007]) se sont révélées aussi efficaces, mais pas encore assez performantes [Suchanek 2007].
Yago utilise de même les infobox de Wikipedia, mais dérive les faits (un fait étant une relation n-aire telle que "qui a gagné quel prix et quand") des catégories Wikipedia au lieu d'utiliser des techniques de TAL sur les contenus textuels des pages. Les catégories Wikipedia (les catégories pour *Olive Garden* sont "*Restaurant chains based in the United States*" et "*Italian restaurants*"), en effet, fournissent non seulement des informations sur l'appartenance d'une instance à une classe (*OliveGarden-isA-RestaurantChain*) mais aussi sur les attributs (*OliveGarden-type-Italian*) et sur les

7. http ://wiki.dbpedia.org/Downloads37
8. http ://lookup.dbpedia.org/api/search.asmx
9. http ://lucene.apache.org/core/
10. https ://github.com/dbpedia-spotlight/dbpedia-spotlight
11. http ://www.mpi-inf.mpg.de/yago-naga/yago/index.html

relations (*OliveGarden-location-UnitedStates*). Cependant, la hiérarchie de catégories Wikipedia n'est pas suffisante pour construire une taxonomie de concepts et relations. Wordnet, dont on parlera par la suite, est donc utilisée pour organiser les données en taxonomie grâce à la hiérarchie de concepts que cette base linguistique fournit. Cette méthode, exploitant WordNet pour sa taxonomie fiable et Wikipedia pour la grande quantité d'informations contenues, a montré une précision de plus que 95% faisant de YAGO l'une des bases de connaissances les plus utilisées [Suchanek 2008]. YAGO a ensuite évolué en YAGO2, sous la volonté de tenir compte des données spatiales et temporelles [Hoffart 2011], [Hoffart 2010].

Format des données Le modèle YAGO, très similaire au RDFS d'un point de vue conceptuel, se différencie des autres comme étant un format purement textuel, tabulaire, organisé en triplets SPO (subject-predicate-object, plus Time&Space pour les triplets de YAGO2) :

`#1: Olive_Garden` wasCreatedOnDate `1982`

Chaque triplet a un identifiant (`#1`) qui est réutilisé pour exprimer des relations n-aires :

`#2: #1` createdIn `Orlando,Florida`
`#3: #2` isA `restaurantChain`

Ainsi, il est possible d'établir des relations n-aires entre triplets et de dériver "Olive garden is a restaurant chain created in Florida in 1982". Ce type de représentation permet aussi d'effectuer plus aisément des requêtes de type question-réponse, en formalisant une question comme "When was created the OliveGarden ?" en :

`OliveGarden` wasCreated `$i`

pour aller matcher le triplet correspondant dans l'ontologie.
De plus, tout littéral est reconnu comme étant une classe, sans distinction : les Datatypes sont aussi des classes (organisées hiérarchiquement) en relation avec une classe (qui peut être une instance ou un concept).

Outils existants YAGO peut être utilisé de différentes façons.
Il peut être chargé dans des bases de données (Postgres, MySQL ou Oracle) ou converti en RDF, N3 ou XML, à l'aide d'outils disponibles [12]. Alternativement, il est possible de l'interroger en SPARQL soit localement, soit sur un endpoint [13]. Deux outils visuels sont aussi disponibles : pour la navigation [14] et l'interrogation basée sur des patrons SPOTLX [15].

12. http ://www.mpi-inf.mpg.de/yago-naga/yago/downloads.html
13. http ://lod.openlinksw.com/sparql
14. https ://d5gate.ag5.mpi-sb.mpg.de/webyagospotlx/Browser
15. https ://d5gate.ag5.mpi-sb.mpg.de/webyagospotlx/WebInterface

2.2.3 Schema.org

En juin 2011 les trois premiers moteurs de recherche Bing, Google et Yahoo ! ont proposé *Schema.org* [16], un ensemble de termes d'annotation à destination des développeurs de sites Web. L'objectif principal est de faciliter la recherche dans les pages Web à l'aide de tags sémantiques. Ces efforts comprennent des outils libres et de la documentation pour encourager les utilisateurs du Web à publier des données sémantiques, avec les termes de Schema.org. Ces données sont publiées en HTML5, mais les tags sont utilisés pour créer des groupes d'entité-valeur, appelés microdata, basés généralement sur les contenus de la page.

Suivant le principe du Web Sémantique, Schema.org fournit aussi des mapping entre ses termes et ceux des différentes ontologies ou vocabulaires. Ces termes sont organisés en une ontologie disponible en différents formats selon les besoins, RDF (Turtle, XML, NTriples), JSON et CSV.
Il existe différents outils pour publier, traiter, transformer ou interroger les données, en différents langages. Nous nous limitons à en citer quelques uns :
- **Processing des données** RDFlib [17] (Python), A parser for HTML microdata [18] (Perl), Any23 [19] (Java)
- **Publication** Schema Creator [20] (online), Web.instata [21] (génération de données à partir d'un CSV)
- **Plateformes** Schema for Wordpress [22], Virtuoso Sponger [23]

2.3 *Linked Data*

La notion de *Linked Data* fournie en [Bizer 2009a] représente un ensemble de connexions entre données de différentes sources. Les "sources" dont nous parlons peuvent être, par exemple, des bases de données appartenant à différentes communautés, ou des données indiquant le même concept mais hétérogènes car fournies par des systèmes différents. Les *Linked Data* indiquent plus simplement les données publiées sur le Web, suffisamment définies pour être compréhensibles par des machines, connectées entre elles pour que l'accès à l'information soit global.

Les *Linked Data* sont encodées en RDF sous forme de triplets Sujet-Predicat-Objet. Les ressources sont décrites avec les namespaces référençant un dataset spécifique. Dans l'exemple ci-dessous, le *namespace* `dbpedia` n'est qu'un pointeur vers le URI " http ://dbpedia.org". Les langages RDF et OWL fournissent les vocabu-

16. http ://schema.rdfs.org/
17. https ://github.com/edsu/rdflib-microdata
18. https ://metacpan.org/module/HTML : :HTML5 : :Microdata : :Parser
19. http ://developers.any23.org/
20. http ://schema-creator.org/
21. https ://github.com/mhausenblas/web.instata
22. http ://schemaforwordpress.com/
23. http ://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VirtSponger

```
R1: <dbpedia:Restaurant>
R2: <http://sw.opencyc.org/concept/Mx4rvVjyg5wpEbGdrcN5Y29ycA>

S:<dbpedia:Restaurant>
P:<owl:sameAs>
O:<http://sw.opencyc.org/concept/Mx4rvVjyg5wpEbGdrcN5Y29ycA>
```

FIGURE 2.5 – Exemple de "linkage" des données

laires pour la description et le "linkage" des données. La création des vocabulaires est libre mais, selon les principes du Web Sémantique, il est important de ne pas oublier d'établir la connexion des données avec d'autres datasets. L'innovation consiste à utiliser ce langage non seulement pour mettre en évidence les connexions sémantiques entre données du même *dataset*, mais aussi à connecter les données entre *datasets* différents en utilisant des triplets d'équivalence. Par exemple, le concept *Restaurant* dans DBpedia (`R1`) et dans l'ontologie OpenCyc (`R2`) n'est pas référencé de la même manière, mais il faut établir une connexion entre ces deux concepts dans un triplet, en utilisant la relation `owl:sameAs`.

Pour faciliter la connexion des données, certains "principes des *Linked Data*" ont été établis en [Berners-Lee 2009] :

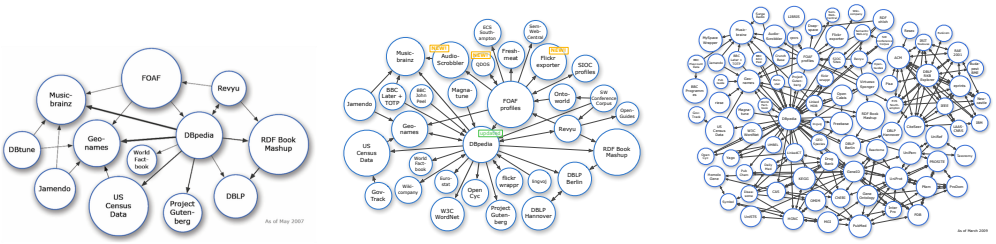
- Utiliser les URI pour nommer les entités
- Utiliser le protocol HTTP pour les URIs pour que les données soient accessibles depuis l'extérieur
- Fournir des informations pertinentes, en utilisant le standard RDF (et SPARQL)
- Inclure les liens vers d'autre URI, afin de laisser la possibilité de découvrir plus d'information

Le résultat de ces interconnexions est appelé Web des données (*Web of Data*) ou bien Web Sémantique. Si d'un côté, ce Web partage des points avec le Web documentaire (comme les données de tout type et librement publiables) de l'autre, les nouveautés apportées (aucune contrainte dans le choix du vocabulaire de description, le RDF pour la connexion de données) le rend plus qu'une simple extension, une évolution de plus vers le Web 3.0.

2.3.1 Le projet *Linking Open Data*

Les principes du Web de données se retrouvent appliqués dans *Linking Data Project*[24].
Le projet, fondé en 2007 et supporté par le W3C, a comme objectif d'exploiter les informations contenues dans le Web, en les récupérant des basés de données existantes, en les convertissant au format RDF et en les rendant disponibles sur le

24. http ://linkeddata.org/



(a) 2007 (b) 2008 (c) 2009

FIGURE 2.6 – Evolution du Web of Data

Web [Bizer 2007]. Depuis quelques années donc, les efforts de cette communauté ne cessent pas de s'intensifier, ce qui a porté une croissance remarquable des contenus disponibles (cf. 2.6, 2.7). La rapide évolution est principalement due à la nature *open source* du projet, permettant à toute communauté d'alimenter et agrandir les données.
La situation en septembre 2011 comptait plus que 31 milliards de triplets RDF [Bizer 2009a], c'est-à-dire 295 bases mises en relation, traitant de domaines différents. Comme introduit dans la section précédente, DBpedia est le point centrale du Web des données. Les médias (musique, télévision, film, radio), les données gouvernementales (statistiques, recensements), géographiques, bibliographiques (publication scientifiques ou livres), sciences et médecine, les personnes, les compagnies, les réseaux sociaux, etc. en font partie. Les "connexions" sont exprimées dans le graphe par les arcs entre datasets : elle peuvent être bidirectionnelles ou unidirectionnelles selon que les liens existent dans les deux bases de données ou pas.

Outils de publications Plusieurs outils ont été développés jusqu'à présent pour l'exploitation des *Linked Data*. Nous en citons ici quelques uns :

- **Plateformes de publications de données**. Ces outils permettent de publier les données au format RDF et de les interroger avec des services Web SPARQL. Parmi les plus connus, on retrouve :
  - D2R Server : http ://www4.wiwiss.fu-berlin.de/bizer/d2r-server/
  - Linked Data API http ://purl.org/linked-data/api/spec
  - Linked Media Framework http ://code.google.com/p/kiwi/
  - Joseki http ://www.joseki.org/
- **Editeurs et Validateurs** des données RDF, comme :
  - Hyena http ://hypergraphs.de/
  - Vapour http ://validator.linkeddata.org/
- **Moteurs de recherche sémantiques** pour la recherche des données RDF, tels que :
  - Swoogle : http ://swoogle.umbc.edu/
  - Watson http ://watson.kmi.open.ac.uk/WatsonWUI/
  - SameAs : http ://sameas.org/

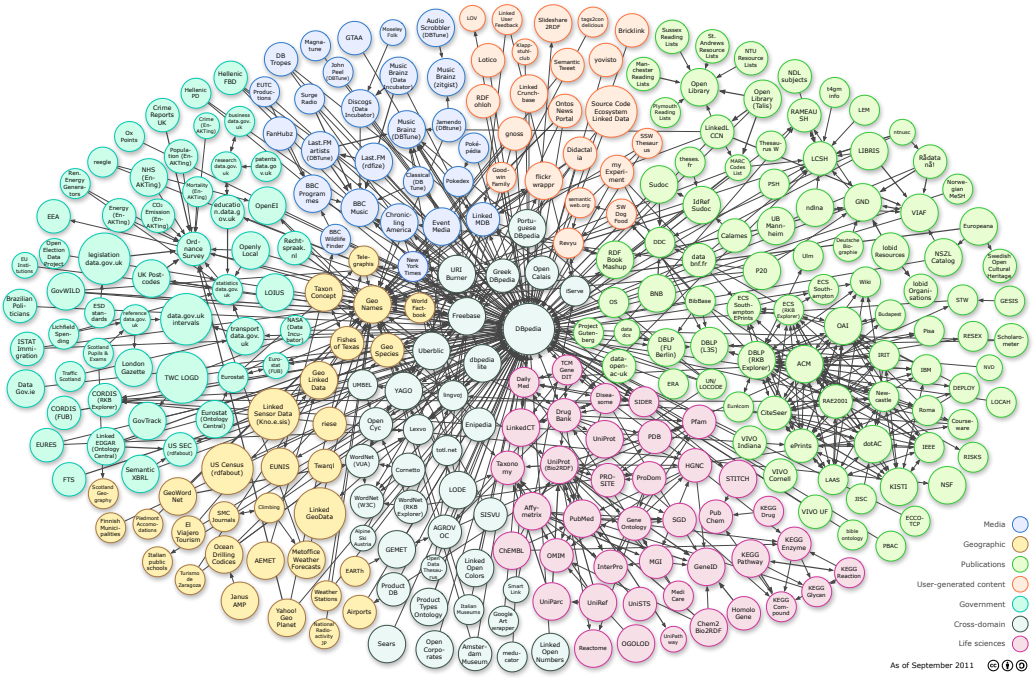


FIGURE 2.7 – Situation du Web of Data en 2011

  - MicroSearch : http ://research.yahoo.com/pub/2602

- **Navigateurs** du Web Sémantique :
  - RazOrBase : http ://www.razorbase.com/
  - OpenLink Data Explorer http ://www.w3.org/wiki/OpenLinkDataExplorer
  - DBpedia Mobile : http ://wiki.dbpedia.org/DBpediaMobile

Chapitre 3

L'Apprentissage d'ontologies

Sommaire

3.1 *Ontology Learning* : entre structuré et non structuré

Le terme *Ontology Learning* a été proposé en 2001 pour la première fois dans [Maedche 2001] et peut être défini comme l'acquisition automatique d'un modèle ontologique à partir de données [Cimiano 2006b]. Cette discipline se trouve à l'intersection entre le machine learning, la fouille des données, l'ingénierie linguistique et le Traitement Automatique des Langues.
Compte tenu du grand nombre de données disponibles, les communautés de recherche ont focalisé leur attention sur les méthodes de fouille de données dites *data-driven*, fondées sur l'ingénierie ontologique et la modélisation des connaissances. L'ensemble de ces techniques, appelées *Ontology Learning*, est une clé importante pour réduire les coûts de temps et d'effort humain que demandent la construction et, surtout, la maintenance des ontologies [Staab 2009]. L'apprentissage d'ontologies a suscité un grand intérêt suite au développement du Web Sémantique, auquel il est strictement lié, menant les recherches à développer des techniques capables d'obtenir aisément des données structurées à partir de la grande quantité de données du Web. L'idée sous-jacente est que si l'on disposait d'une ontologie comprenant toute la connaissance, et si tous les documents étaient annotés sur celle-ci, les machines seraient capables de répondre à des questions extrêmement sophistiquées [Biemann 2005].
L'apprentissage d'ontologies permet de réduire l'intervention d'un expert du domaine : ceci rend les techniques et les ressources disponibles à des nouvelles

communautés, pouvant aussi être intégrées dans des systèmes plus généraux. L'ouverture à d'autres applications faciliterait la communication entre disciplines différentes.

Bien que des techniques pour l'acquisition automatique d'une connaissance globale ne soient pas envisageables à l'heure actuelle, les approches d'apprentissage d'ontologies apportent un soutien remarquable à l'ingénierie ontologique.

3.2 Apprentissage d'ontologies à partir des textes

La construction d'une ontologie consiste à détecter des concepts et les relations qui les lient. Ceci implique que la quantité de données utilisées soit suffisante pour avoir un modèle assez complet. Les ontologies peuvent être apprises à partir de sources différentes, qu'elles soient structurées (schéma des bases de données, UML, tableaux), semi-structurées (HTML, XML, dictionnaires ou taxonomies) ou ressources purement textuelles : dans ce cas, on parle d'apprentissage d'ontologies à partir de textes (*Ontology Learning from texts*). Alors que les données structurées facilitent la tâche d'apprentissage des schémas mais restent peu accessibles, les données semi-structurées et textuelles ont l'avantage d'être nombreuses et accessibles mais nécessitent des traitements en plus, appelant en particulier les techniques linguistiques.
La grande quantité de données textuelles dans le Web a contribué au développement de l'apprentissage d'ontologies à partir des textes. Faute de techniques atteignant la perfection à l'heure actuelle, il a été largement démontré cependant que l'apprentissage d'ontologies peut apporter une aide importante au traitement des ressources textuelles et à la compréhension des textes.
Les techniques d'apprentissage utilisées dans ce cadre se divisent en deux parties, correspondant approximativement à la distinction entre approches supervisées et approches non-supervisées :

1. techniques de clustering. On construit une ontologie de zéro, et les approches le plus utilisées sont :
    - similarité distributionnelle (analyses syntaxiques ou des fenêtres de mots)
    - extraction de patrons (patrons de Hearst)
2. techniques de classification. On enrichit dans ce cas une ontologie déjà existante.
3. techniques hybrides, rassemblant les méthodes linguistiques et statistiques

3.3 Apprentissage d'ontologies à partir du Web

Les dernière recherches se sont ensuite focalisées sur la construction d'ontologies à partir du Web, compte tenu de ce qu'il rend disponibles et accessibles une grande

quantité de documents.
Les techniques utilisées pour cette tâche sont les mêmes que celles pour l'apprentissage à partir de textes. Un pré-traitement est généralement requis pour extraire la partie textuelle des pages HTML/XHTML.

Approches indépendantes du domaine Ce type d'approches consiste à enrichir une petite ontologie dite "minimale" ou "granulaire" avec des nouveaux concepts et relations en utilisant des techniques de fouille de textes. En général, les méthodes utilisées sont (i) patrons lexico-syntaxiques, (ii) techniques linguistiques, (iii), clustering ou classification, (iv) techniques statistiques, (v) règles d'association et (vi) méthodes hybrides [Mustapha 2009].

Approches dépendantes du domaine Ces approches se focalisent sur la construction de taxonomies sans s'appuyer sur une structure déjà existante (ontologies ou *thesauri*). Elle s'appuient sur les moteurs de recherches pour effectuer des requêtes sur le Web et calculer des mesures de recherche d'information [Mustapha 2009]. Ce sont des approches incrémentales.

Web-Mining à partir des structures HTML Certaines approches exploitent les structures des pages Web en faisant l'hypothèse que celles-ci contiennent les informations les plus importantes. Dans [Yamaguchi 1998], on utilise les syntagmes nominaux apparaissant dans les *headings* d'une page pour découvrir des concepts et des relations taxonomiques. D'autres travaux de ce type sont présentés dans [Brunzel 2008] et [Mustapha 2009].

Apprentissage à partir des ontologies en ligne L'objectif de ces techniques est de permettre aux utilisateurs de réutiliser et importer des ontologies ou des modules d'ontologies. Elle évitent donc la prolifération des ressources au bénéfice de leur enrichissement.

Construction d'ontologies à partir des dictionaires Web Une nouvelle voie de recherche appelée *Wikipedia Mining* a été proposée en [Kotaro 2007] pour construire des thésauri à partir de Wikipedia.

3.4 Tâches de l'apprentissage d'ontologies

En faisant référence au triangle sémiotique, l'apprentissage d'ontologies représente le processus d'encodage de l'information, où étant donné un signifiant non-interprété, il faut identifier son référent [Sowa 2000]. Ce processus est exactement l'inverse que celui de la production de textes (cf. 3.1). Les **signifiants**, correspondant aux instances, sont liés à un **signifié** : leur apprentissage est appelé population d'ontologies [Cimiano 2006b].

L'apprentissage d'une ontologie se décompose en plusieurs étapes. La première tâche à accomplir est l'apprentissage des concepts, et des relations qui les lient. Pour ce faire, les techniques linguistiques permettent de lier les signifiants aux référents qu'ils représentent, en obtenant les synonymes, les termes et les définitions d'un concept. Les concepts appris sont ensuite liés avec des relations (hiérarchiques, pour construire une taxonomie, et non-hiérarchiques, pour connecter différents sous-domaines de l'ontologie). Afin de permettre l'interprétation des concepts et des relations, il faut aussi apprendre un schéma d'axiomes logiques pour définir la transitivité, réflexivité, symétrie et disjonction des relations, et permettre aussi l'inférence de faits pas explicitement appris dans les données.
L'apprentissage ontologique peut se résumer donc comme dans la figure 3.2.

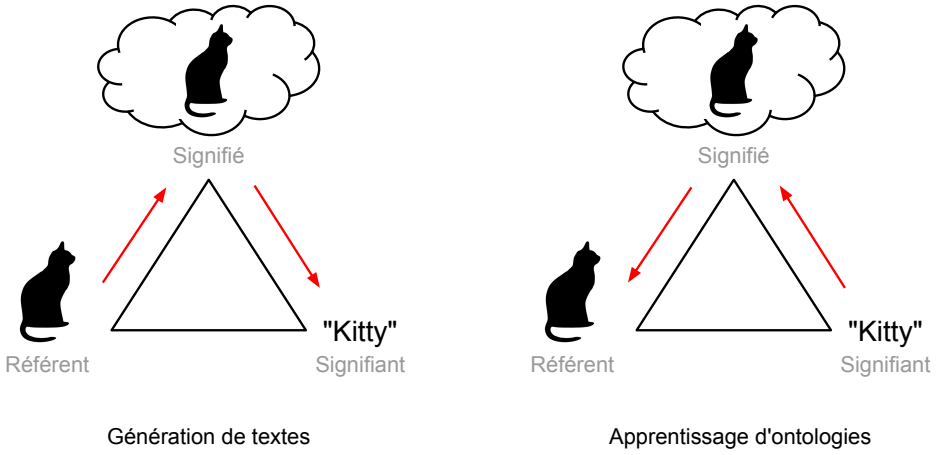


FIGURE 3.1 – Encodage et décodage de l'information

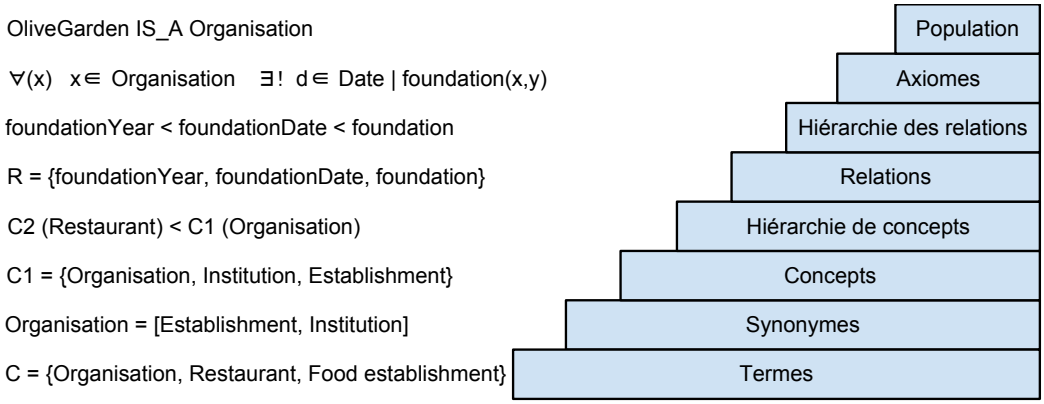


FIGURE 3.2 – Etapes de l'apprentissage d'ontologies

3.4.1 Acquisition des termes et des synonymes

C'est le point de départ car un terme est la désignation linguistique d'un concept. Un terme est défini comme un mot (ou un groupe de mots) pertinent pour un concept donné [Cimiano 2006b]. Généralement, cette tâche est effectuée en calculant les fréquences des mots à l'intérieur d'un corpus de documents. Idéalement, un terme fréquent dans un groupe de documents signifie qu'il est pertinent pour le domaine du corpus. D'autres techniques se sont révelées être plus performantes, comme l'utilisation des mesures TF/IDF [Staab 2009] ou des taxonomies de concepts [Frantzi 1998]. Certaines approches ont aussi reconnu l'importance des ressources du Web et proposé d'exploiter les structures arborescentes des pages Web pour l'extraction de termes et synonymes [Brunzel 2008].
La recherche des synonymes consiste ensuite à détecter un ensemble $Ref_{(C)}$ où tous les éléments se réfèrent au même concept. Pour chaque mot w, on construit un vecteur v_{w1} avec les mots les plus fréquents autour de ce mot, dans un espace défini comme *contexte*. Ensuite on calcule la similarité entre les vecteurs v_{w1} et v_{w2} et, au-delà d'un seuil, on considère $w1$ et $w2$ comme synonymes. Parmi les approches, on compte celles de [Lin 1998], [Grefenstette 1992], des approches exploitant les statistiques et le Web [Baroni 2004], [Turney 2001] des approches plus hybrides telles que [Curran 2002].

La définition de synonyme est généralement reprise de Wordnet [1] : un groupe de mots partageant une signification commune permettant de former un concept pertinent dans un domaine donné. Wordnet est une base de données lexicale fournissant des informations linguistiques sur les mots : les synonymes, les relations d'hypo- et hyperonymie, méronimie, ainsi que les relations entre groupes de mots similaires, les *synset*s.

3.4.2 Formation des concepts et hiérarchisation

Cette étape consiste à donner une définition formelle des concepts et de les enrichir avec leurs ensembles $Ref_{(C)}$. À noter que les éléments dans ces derniers peuvent être des formes lexicales, mais aussi d'autres types de données, comme des arbres syntaxiques ou des cadres de sous-catégorisation. Le processus de hiérarchisation des concepts consiste ensuite à dériver des relations *subClassOf* pour un ensemble de concepts C.

Différentes méthodes existent pour cette tâche : (i) clustering de concepts, (ii) analyse linguistique et (iii) méthodes inductives [Cimiano 2006b]. Dans le premier cas, on retrouve la *Formal Concept Analysis* (FCA) [Staab 2009], qui permet de former les concepts et de les organiser en hiérarchie en une seule étape, sur la base des attributs distinguant chaque concept. Les techniques d'analyse linguistique sont utilisées pour détecter une description d'un concept en langage naturel, et Wordnet est souvent utilisé pour la désambiguïsation des concepts. L'*inductive learning*, enfin,

1. http ://wordnet.princeton.edu/

est appliqué pour détecter des règles communes à un groupe de concepts.

Plusieurs techniques sont utilisées pour la hiérarchisation des concepts : notamment, les patrons lexico-syntaxiques, la classification, l'analyse syntagmatique et le clustering.

Les **patrons lexico-syntaxiques** ont été étudiés déjà dans les années '80 pour l'extraction d'information et la détection de relations lexicales [Staab 2009], mais c'est en 1992 que [Hearst 1992] propose pour la première fois leur utilisation pour la détection automatique d'hyponymes et hyperonymes à partir des corpora. La nouveauté réside dans la définition d'une série de patrons pour la détection des hyponymes. Ces patrons sont :

1. NP_{hyper} *such as* $\{\text{NP}_{hypo}\}^*$ $\{(and \mid or)\}$ NP_{hypo}
Food establishments such as restaurants and pubs...
2. *Such* NP_{hyper} *as* $\{\text{NP}_{hypo}\}^*$ $\{(and \mid or)\}$ NP_{hypo}
Such food establishments as restaurants or pubs...
3. $\{\text{NP}_{hypo}\}$ $\{,\text{NP}_{hypo}\}$ * $\{,\}$ $\{(and \mid or)\}$ *other* NP_{hyper}
Restaurants, pubs, or other food establishments
4. NP_{hyper} $\{including|especially\}$ $\{\text{NP}_{hypo},\}^*$ $\{(and \mid or)\}$ NP_{hypo}
Food establishments, including pubs and restaurants..

Depuis, ces patrons ont été largement utilisés dans la construction automatique d'ontologies et l'extraction d'information [Cimiano 2006b], [Maedche 2001].

Le **clustering** permet de regrouper les concepts, représentés comme des vecteurs, sur la base de caractéristiques communes, qui constituent l'espace de représentation de ces vecteurs. Trois types d'approches sont exploitées généralement : (i) clustering agglomeratif, où chaque cluster est constitué dans la phase initiale par un seul terme, et ensuite on constitue des clusters de plus en plus larges jusqu'à rejoindre un critère défini ; (ii) clustering divisif qui, inversement, a une phase initiale où il existe un seul groupe composé par tous les termes, et puis on constitue des clusters plus spécifiques en étudiant les différences entre vecteurs et (iii) clustering conceptuel, où un treillis de mots est constitué et on détecte quels attributs sont les plus représentatifs entre deux mots.

Les **structures syntagmatiques** ont été étudiées en faisant l'hypothèse que certaines d'entre-elles peuvent révéler une relation taxonomique. En particulier, les suites ADJ-NN montrent souvent que le syntagme nominal est une sous-classe d'un syntagme composé juste du mot taggé NN : effectivement, $Italian\ restaurant_{[hypo]}$ est une sous-classe de $Restaurant_{[hyper]}$. Ces méthodes ont été explorées dans [Cimiano 2006b], [Buitelaar 2004] et [Sanchez 2005]. L'approche proposée dans [Poesio 2008] exploite l'extraction de relations à partir des ressources du Web, en se focalisant sur l'importance des **valeurs** des attributs pour la récupération d'un concept. Des patrons lexicaux sont donc établis :

VALUE is the ATTRIBUTE of CONCEPT : $[Brown]_{val}$ *is the* $[color]_{att}$ *of* $[dogs]_c$
the VALUE_1 or VALUE_2 ATTRIBUTE : *the* $[brown]_{val}$ *or* $[blue]_{val}$ $[color]_{att}$
the VALUE ATTRIBUTE is : *the* $[brown]_{val}$ $[color]_{att}$ *is ...*

Finalement, la **classification** peut être utilisée lorsqu'on dispose d'une hiérarchie déjà définie (dans Wordnet ou d'autres sources) et qu'on entraîne des classifieurs pour ranger dans la hiérarchie des nouveaux termes.

3.4.3 Apprentissage et hiérarchisation des relations

Cette tâche, appelée aussi *relation learning* consiste à extraire les relations, propriétés et attributs avec leur domaine (dom_r) et co-domaine ($range_r$), et les organiser en hiérarchie. Les techniques de machine learning et traitement du langage trouvent leur principale application dans cette étape, afin de pourvoir découvrir des relations inconnues entre deux concepts. Les approches les plus communes sont

- les **collocations**, c'est-à-dire l'analyse statistique d'une "fenêtre" de mots apparaissant entre deux concepts ;
- les **dépendances syntaxiques**, comme les dépendances prédicat-argument (en se focalisant sur les verbes), qui ont été particulièrement explorées pour le domaine biomédical dans [Ciaramita 2005], [Cimiano 2005], [Akbik 2009]
- les **patrons lexico-syntaxiques**, de type patrons de Hearst, pour les relations *part-of*, *cause*, *purpose* [Berland 1999]
- **apprentissage des *qualia structures***, qui sont des *templates* (formulaires) décrivant les attributs qu'une entité possède (but, composants, propriétés formelles...). Les relations apprises dans DBpedia, déjà cité, en sont un exemple.

Une fois détectés les concepts c_1 et c_2, il faut trouver le label de la relation r_k et son niveau d'abstraction, pour pouvoir ensuite déterminer un hiérarchie $r_k < r_m$.

3.4.4 Définition des axiomes

Les schémas d'axiomes permettent de définir les propriétés des relations (transitivité, disjonction, réflexivité) d'un point de vue logique. La tâche consiste à identifier quelles sont les caractéristiques des concepts, couples de concepts ou relations, pour en faire des axiomes. Alors que, jusqu'à présent, les techniques pour les couches inférieures de la pyramide de l'apprentissage d'ontologies ont largement été explorées [Buitelaar 2008], le raisonnement reste relégué à des domaines spécifiques comme la bioinformatique et la médecine, qui nécessitent des axiomisations complexes. L'extension de ces techniques à d'autres domaines a été proposée en [Völker 2009] : les constructions linguistiques sont utilisées pour transformer des définitions en langage naturel en axiomes logiques. Par exemple, la coordination *men and women* indique probablement une disjonction entre la classe *man* et la classe *woman* :

```
<owl:Class rdf:ID="Man">
  <rdfs:subClassOf rdf:resource="#Human"/>
  <owl:disjointWith rdf:resource="#Woman"/>
</owl:Class>
```

De même, dans [Shamsfard 2004], les auteurs proposent d'exploiter les constructions quantifiées telles que *every man loves a woman* pour extraire des contraintes universelles :

$$\forall \text{ x} \in \text{MAN } \exists \text{ y} \in \text{WOMAN tels que } loves(x, y)$$

Même les arbres de dépendances peuvent être utilisés pour apprendre des axiomes entre les relations : en étudiant la structure passive/active, par exemple, on peut dériver que *writes* est la relation inverse de *written_ by* [Lin 2001].

```
<owl:ObjectProperty rdf:ID="writes">
  <owl:inverseOf rdf:resource="#written_by" />
</owl:ObjectProperty>
```

Nous rappelons que, si une relation R_1 est l'inverse d'une relation R_2, alors $\forall$ x,y : $R_1(x, y)$ ssi $R_2(y, x)$.

3.4.5 Population

Cette tâche consiste à apprendre les aspects additionnels de ce qui constitue un domaine, c'est-à-dire les instances et les valeurs des relations. Nous rappelons qu'une instance est une entité i liée à un concept c par une relation *instanceOf*.
La population de l'ontologie est strictement liée à la reconnaissance d'entités nommées (NER) et à l'extraction d'information (IE). L'extraction d'information consiste à remplir des formulaires (les *templates*) à partir d'une information non structurée (les textes). Les techniques plus récentes pour l'IE se basent sur l'induction automatique de règles d'extraction [Cimiano 2006b]. Ces techniques ne permettent pas l'extraction de plus d'un formulaire à la fois, rendant le processus long. La reconnaissance d'entités nommées permet, au contraire, l'extraction des instances d'un concept donné, qui est généralement *Person*, *Organisation* ou *Location*. D'autres classes ont été ajoutées à l'ensemble des entités nommées, notamment à l'intérieur de l'*ACE Evaluation*[2]. La population des ontologies essaie de regarder plus loin et de récupérer les instances pour chaque classe de l'ontologie.

Des techniques de TAL associées au *Machine Learning* sont souvent utilisées pour cette tâche [Maynard 2008]. Les approches d'apprentissage faiblement supervisé, issues de l'IE, sont explorées dans [Jean-Louis 2011], [Tanev 2006]. Une méthode appelée *Learning by Googling* a été proposée dans [Cimiano 2006b], avec l'idée d'exploiter la connaissance collective, c'est-à-dire les données du Web, et de les filtrer dans une deuxième étape. Les systèmes PANKOW et C-PANKOW [Buitelaar 2004], [Cimiano 2005] comprennent une approche semi-supervisée exploitant un mécanisme de génération de patrons relatifs à une relation sémantique ou à un concept d'une ontologie, et visent à récupérer les instances correspondantes dans le Web. La recherche dans le Web est menée par le moteur de recherche Google™. C'est une approche supervisée car les patrons sont définis à la

2. http ://www.itl.nist.gov/iad/894.01/tests/ace/

main, mais comme ils ne varient pas entre les domaines, ils permettent de rendre l'approche non-supervisée, car il n'y a pas besoin d'un ensemble d'apprentissage pour un nouveau domaine.

3.5 Outils Existants

Comme nous l'avons vu, l'apprentissage d'ontologies est une discipline qui a suscité énormément d'intérêt dans les dernières années, ce qui a suscité un accroît remarquable des technologies et outils disponibles.

OntoLearn Ce système [3] est basé sur une composante de désambigüisation de mots, pour dériver un ensemble de termes dénotant un concept sur la base des gloses Wordnet.

OntoLT Cet outil [4], disponible aussi comme plugin Protégé [5], permet l'extraction de termes en utilisant des mesures comme TD/IDF et les analyses syntagmatiques pour la détection des hiérarchies.

Terminae C'est un ensemble d'outils pour l'ingénierie ontologique à partir de textes développé en réponse au besoin de réduire les coûts de traitement de textes, et d'améliorer les performances de l'ingénierie ontologique avec l'apport des récentes approches de Traitement Automatique des Langues [Aussenac-Gilles 2008]. L'approche Terminae exploite des analyses linguistiques pour l'extraction de termes et documents, permet la modélisation et l'axiomisation ainsi que la réutilisation des ontologies existantes. La représentation de la connaissance permet de passer aisément des formes linguistiques au modèle conceptuel et inversément, tout en respectant le formalisation dans la logique formelle. Les formes linguistiques, dites terminologiques, sont des données structurées où on peut stocker toute information relative à un terme qui est extraite à l'aide des techniques de TAL (structures grammaticales, métadonnées documentaires etc.)

Text2Onto Il s'agit d'un ensemble d'outils [6] pour l'apprentissage d'ontologies à partir du Web. Il a été développé dans le but d'être un support au Web Sémantique et à ses applications [Maedche 2001]. Derrière une composante de gestion des documents (corpora, mais aussi schémas et ontologies existantes), les ressources sont traitées avec des techniques de TAL : l'extraction des termes est basée sur des mesures de similarité, et l'extraction des relations sur la FCA.

3. http ://code.google.com/p/ontolearn/
4. http ://olp.dfki.de/OntoLT/OntoLT.htm
5. http ://protege.stanford.edu/
6. http ://code.google.com/p/text2onto/

XTreeM Cette approche se base sur la structure des documents Web, au lieu des textes [Brunzel 2008]. Ceci lui permet donc d'être applicable indépendamment de la langue ou du domaine. Le corpus, ainsi que le domaine, est défini sur la base d'une requête lancée au Web et sur la collection de tous les documents retournés par le moteur de recherche. Cette approche est focalisée sur les trois premières couches de la pyramide de l'Ontology Learning. Pour l'extraction des termes, l'approche s'appuie sur des vocabulaires spécifiques aux domaines et des statistiques sur les fréquences des mots. Les synonymes et les relations sémantiques sont ensuite appris avec des méthodes de fouille de données, et particulièrement avec une approche appelée *Group-by-Path* qui exploite les structures arborescentes des pages XHTML.

Chapitre 4

Les Grammaires de Dépendances

Sommaire

Nous avons largement introduit dans les chapitres précédents les grammaires de dépendances. Dans cette section, nous allons présenter en détail ce qu'est ce formalisme et comme on peut l'exploiter.

4.1 Introduction

4.1.1 Notions de dépendances

Les grammaires de dépendances sont un formalisme pour la représentation syntaxique d'une phrase. Celle-ci est représentée par un ensemble de relations de dépendances, qui sont des relations binaires et asymétriques entre un mot tête et un mot dépendant.

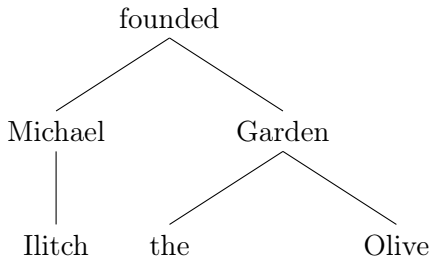


Figure 4.1 – Exemple d'arbre de dépendances

Un peu d'histoire À bien chercher dans l'histoire, Pāṇini s'était déjà rapproché de ce type de conception lors de son étude sur la grammaire du Sanscrit, bien avant l'ère moderne [Kübler 2009]. La tradition plus récente de ce formalisme est

cependant attribuée au travail de Lucien Tesnière, publié en 1959 [Nivre 2005a]. Ces grammaires ont gagné un nouvel intérêt depuis les années '90, après la démonstration de leur haute performance dans les langues slaves et celtiques [Mel'čuk 1988].

Dans son oeuvre, Tesnière soutient que :

- La phrase est un ensemble organisé, dont les éléments sont les **mots**
- Les mots de la phrase ne sont pas isolés, mais liés par des *connexions* entre eux : ce sont les **dépendances**
- Dans une dépendance, il existe un mot supérieur, dit **régisseur** et un mot inférieur, le **subordonné**

Contrairement à la grammaire traditionnelle, où les fonctions syntaxiques sont le prédication et la subordination, Tesnière refuse la première, en soutenant que le sujet dépend d'un verbe autant que les autres compléments [Tesnière 1959]. Il ne reste donc que la subordination, c'est-à-dire la dépendance. Cette connexion peut autrement être graphiquement représentée (cf. 4.2) comme un trait vertical où le régisseur (que nous appellerons dorénavant **tête**) est lié à son subordonné (**dépendant**).

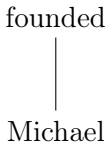


FIGURE 4.2 – Représentation graphique d'une connexion, dite *Stemma*

Tesnière distingue entre mots "vides" et "pleins" d'un point de vue sémantique : alors que les premiers (comprenant prépositions, conjonctions, articles et pronoms conjoints) ne sont pas porteurs de sens, ces derniers (noms, verbes, adjectifs et adverbes) sont les éléments qui permettent la compréhension de la phrase. Seuls le mots pleins peuvent entrer en relation de dépendance (autrement dit, on ne retrouvera pas de prépositions ou d'articles en tant que nœuds), et pour cette raison, avec ce formalisme la structure syntaxique de la phrase reste très proche de la structure sémantique.
Le fait d'être plein ou vide n'est pas le seul critère pour la détection d'une tête et d'un dépendant. Des critères syntaxiques ou sémantiques ont été établis, par exemple liés à la position des mots dans la phrase, ou au fait que leur présence soit obligatoire ou optionnelle [Nivre 2005a].

Les têtes et modifieurs sont liés par des dépendances, c'est-à-dire une fonction grammaticale. Mel'čuk [Mel'čuk 1988] distingue trois types de relation : morphologique, syntaxique et sémantique, qui lient deux mots selon des critères bien définis [Polguère 2009]. Les relations syntaxiques sont aussi divisées en endoncentriques et esocentriques [Nikula 1986]. Dans une dépendance endocentrique, la tête peut

remplacer tout le syntagme sans qu'il y ait de changement structural de la phrase ; on parle alors de relations tête-modifieur, catégorie à laquelle appartiennent, par exemple, les relations de type adjectival (1a). Dans les dépendances esocentriques ceci n'est pas possible : on parle alors de relation tête-complément, et c'est le cas des relations comme sujet ou objet (1b).

(1a) *Mike Ilitch was* [the first$_{mod}$ founder$_{tete}$]$_{GN}$ *of the Olive Garden*
→ *Mike Ilitch was* [the founder]$_{GN}$ *of the Olive Garden*
(1b) *Mike Ilitch was the first founder$_{tete}$ of* [the Olive Garden]$_{comp}$
→ **Mike Ilitch was the founder$_{tete}$ of*

La dépendance n'est pas la seule relation qui peut exister entre deux mots. Tesnière parle aussi de jonction et translation. La jonction est une relation entre mots dépendants de la même tête (2a) ou entre têtes du même dépendant (2b). La translation se retrouve lorsqu'un mot fonctionnel change la fonction syntaxique d'un mot lexical (3).

(2a) [*Michael*]$_h$ *and* [*Marian*]$_h$ *founded* [*the Olive Garden*]$_d$.
(2b) [*Michael*]$_h$ *founded* [*the Olive Garden*]$_d$ *and* [*Champion Foods*]$_d$.
(3) [*Michael*]$_h$*'s* [*Olive Garden*]$_d$ ⇒ MODIFIEUR.

Ce formalisme permet donc de garder une représentation plus proche de la sémantique d'une phrase, grâce à l'utilisation des dépendances qui lient directement des mots qui ne le seraient pas d'un point de vue syntaxique. Les grammaires de dépendances se révèlent extrêmement faciles à exploiter et efficaces. Cette caractéristique et d'autres, comme l'utilisation des mots comme nœuds au lieux des constituants a contribué au développement d'outils d'analyse syntaxique en dépendances très performants, et à leur large utilisation dans plusieurs tâches du TAL, comme l'extraction d'information, la traduction automatique et l'augmentation des ressources lexicales [McDonald 2011].

4.1.2 Définitions formelles

Une phrase est un ensemble de mots tels que :

$$S = \{w_0, w_1..w_n\}$$

où w_0 correspond à un nœud artificiel ROOT, racine de la phrase.

Chaque mot est lié à un autre par une dépendance $d \in R$, où $R = \{d_1...d_m\}$ est l'ensemble des dépendances qui peuvent lier deux mots.

Une structure de dépendance est un graphe orienté étiqueté $G = (V, A, R)$, où $V \subseteq \{w_0...w_n\}$ est l'ensemble des éléments lexicaux, A l'ensemble des arcs et R l'ensemble des relations grammaticales. Un arc étiqueté $(w_j, r, w_k) \in A$ est une dépendance reliant la tête w_j et le dépendant w_k étiqueté par la relation r.

Les graphes satisfont certaines propriétés :

1. Une seule racine : il n'y a pas de graphe où $w_i \in V$ et $w_i \rightarrow w_0$ ($\rightarrow$ répresentant "est tête de")
2. Une seule tête : chaque nœud a au plus une tête, pour $w_i \rightarrow w_j$ il n'y a pas de $w_{i'} \rightarrow w_j$ et $i' \neq i$
3. Acyclicité : le graphe ne doit pas contenir de cycles tels que $w_i \rightarrow w_j$ et $w_i \xrightarrow{*} w_j$ ($\xrightarrow{*}$ étant la "fermeture reflexive transitive de la relation")
4. Projectivité : étant donnée une représentation linéaire de la phrase, les arcs sont ordonnés de façon à ce qu'il n'y ait pas de croisement : si $w_i \rightarrow w_j$, alors $w_i \xrightarrow{*} w_k$ pour tout w_k où $i < k < j$ ou $j < k < i$

4.1.3 Types de représentation

Les arbres de dépendances peuvent se représenter de différentes façons. Nous en présentons quelques unes, que nous utiliserons aussi pour la partie d'expérimentation.
On peut distinguer :

- Une représentation tabulaire. Le format CONLL(*Conference on cOmputational Natural Language Learning*) est actuellement le plus répandu et utilisé dans les différences communautés de recherche [Nivre 2007] pour la représentation des données .

ID	FORM	LEMMA	CPOSTAG	POSTAG	FEATS	HEAD	DEPREL	PHEAD	PDEPREL
1	The	the	DT	DT	_	3	det	3	det
2	Olive	olive	NNP	NNP	_	3	nn	3	nn
3	Garden	garden	NNP	NNP	_	5	nsubj	5	nsubj
4	was	be	VBD	VBD	_	5	auxpass	5	auxpass
5	founded	found	VBN	VBN	_	0	root	0	root
6	by	by	IN	IN	_	5	prep	5	prep
7	Michael	michael	NNP	NNP	_	8	nn	8	nn
8	Ilitch	ilitch	NNP	NNP	_	6	nn	6	nn

TABLE 4.1 – Exemple d'arbre de dépendances au format CONLL

Dans ce format, les phrases sont séparées par une ligne vide, et chaque phrase consiste en une suite de *tokens* (les mots), un par linge. Les *tokens* sont décrits par 10 champs, séparés par une tabulation, qui sont : identifiant numérique du mot, commençant à 1 pour chaque nouvelle phrase (ID) ; forme du mot/signe de ponctuation (FORM) ; lemma du mot (LEMMA) ; deux POS-tags, dépendants de la langue, un générique et un spécifique (CPOSTAG, POSTAG) ; une série d'attributs morphologique ou syntaxiques (FEATS) ; la position (repérée par son identifiant) de la tête régissant le mot (HEAD) ; le type de dépendance qui le lie à sa tête (DEPREL), la tête et la relation de dépendance projective, dans le cas où l'arbre ne le serait pas (PHEAD, PDEPREL)

- Une représentation textuelle, où les dépendances sont de la forme GRAMMREL(HEAD-INDEX, DEPENDENT-INDEX). C'est un des formats utilisés par le Stanford Parser [de Marneffe 2008].

```
det(Garden-3, The-1)
nn(Garden-3, Olive-2)
nsubjpass(founded-5, Garden-3)
auxpass(founded-5, was-4)
root(ROOT-0, founded-5)
prep(founded-5, by-6)
nn(Ilitch-8, Michael-7)
pobj(by-6, Ilitch-8)
```

- Une représentation graphique linéaire. Les mots de la phrase sont écrits dans l'ordre, et chaque dépendance est une flèche orientée de la tête au dépendant, étiquetée avec une relation grammaticale.

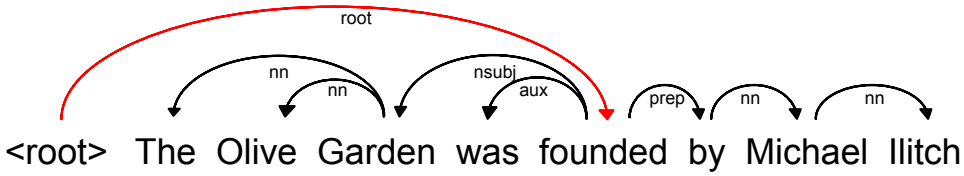


FIGURE 4.3 – Représentation linéaire

- Une représentation en forme arborescente. L'ordre des mots peut dans ce cas aussi ne pas être respecté.

4.1.4 Théories des dépendances : similarités et différences

À partir des grammaires de dépendances plusieurs théories ont été développées à travers les années. Parmi les plus connues [McDonald 2011] on retrouve :

- *Word Grammar* (Hudson 1984)
- *Meaning-Text Theory* (Mel'cuk 1988)
- *Functional Generative Description* (Sgall 1986)
- *Constraint Dependency Grammar* (Maruyama 1990)

Celles-ci ont des points en commun mais aussi des divergences.

Mono- *vs* Multi-stratal Différents niveaux linguistiques sont traités dans les dépendances, ce qui crée une division parmi les différentes conceptions. Certaines théories séparent les niveaux de représentation. On parle par exemple de syntaxe profonde et de surface [Mel'čuk 1988], [Sgall 1986]. La représentation mono-stratale, au contraire, prévoit une seule représentation qui essaie de concilier tous les niveaux linguistiques.

Les nœuds Les éléments pouvant constituer un nœud dans l'arbre syntaxique sont souvent un point assez discuté. Alors que dans la plupart des théories ce sont les mots de la phrase qui constituent les nœuds dans l'arbre, certaines conception admettent d'autres éléments, comme un groupe de mots (les nucleus dissociés [Tesnière 1959]) ou des unités plus petites qu'un mot (grammèmes) dans les cas des dépendances morphologiques [Mel'čuk 1988], [Mel'cuk 2011].

Les dépendances Un autre point de divergence est l'ensemble de relations grammaticales à utiliser pour la construction des arbres : si certaines théories se focalisent sur l'aspect syntaxe de surface (sujet, objet, compléments), d'autres préfèrent exploiter le niveau sémantique (agent, patient, but), ou encore utiliser une combinaison des deux.

Les gouverneurs Les problématiques naissent aussi autour du choix des têtes. Comme déjà dit, les critères pour les choisir sont à la fois sémantiques et syntaxiques : or, il peut arriver que les deux niveaux ne coïncident pas (*I believe in the system*). Dans ce cas là, Tesnière propose d'adopter les nucléus dissociés, composés par un mot fonctionnel et un mot plein, et, en utilisant l'opération de transfert, on met en relation de dépendance la tête avec le mot plein (cf. 4.5).

La coordination Le traitement de la coordination est aussi un point difficile qui divise les théories de dépendances.

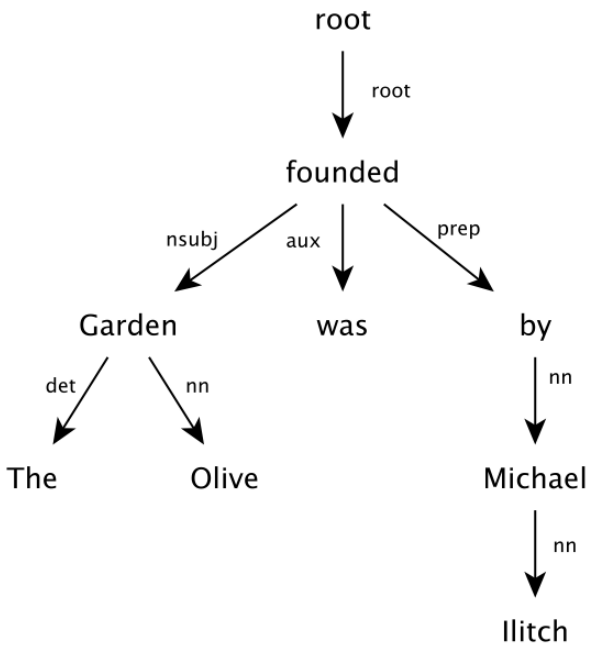


FIGURE 4.4 – Représentation arborescente

$$[Michael\ Ilitch]_{subj}\ owns\ [Little\ Ceasar]_{dobj}\ and\ [Olive\ Garden]_{dobj}.$$

Little Ceasar et *Olive Garden* sont également dépendants de *owns* : une solution adoptée est de prendre la conjonction *and* comme tête, et les deux coordonnés comme dépendants [Sgall 1986]. Une deuxième solution est de prendre un seul coordonné comme dépendant et de le lier à son coordonné avec une relation de conjonction [Mel'čuk 1988]. Une troisième solution, qui semble accorder les deux précédentes, est de prendre les deux comme dépendants du verbe et de les lier entre eux par la relation de conjonction [de Marneffe 2008](cf. 4.6).

En définitive, tous ces formalismes et représentations retrouvent leur accord dans la notion de dépendances entre les mots, sur lesquelles se base la structure syntaxique de la phrase.

4.2 Analyse syntaxique automatique

Mais comment faire comprendre tout ça à un ordinateur ?

Le *parsing* à dépendances est une implémentation pour l'analyse syntaxique automatique inspirée par les grammaires de dépendances [Kübler 2009]. Le succès de la structure en dépendances a permis une exploitation facile dans les technologies linguistiques modernes (comme la traduction automatique ou l'extraction d'information). Cette structure permet de mieux gérer les syntaxes des langues sans ordre de mots, ainsi que de gérer des langues différentes mais avec des syntaxes similaires (par exemple, provenant de la même famille).
Que ce soit avec les techniques de *Machine Learning* ou de l'induction grammaticale, à l'aide des machines de plus en plus puissantes, l'analyse en dépendances se révèle à l'heure actuelle très performante et ne cesse pas de s'améliorer.

D'un point de vue d'implémentation, une structure en dépendance est une représentation incluant des nœuds lexicaux (dont une racine, définie comme un "nœud artificiel"), annotés éventuellement d'autres informations (lemmes, POS-tag, entité),

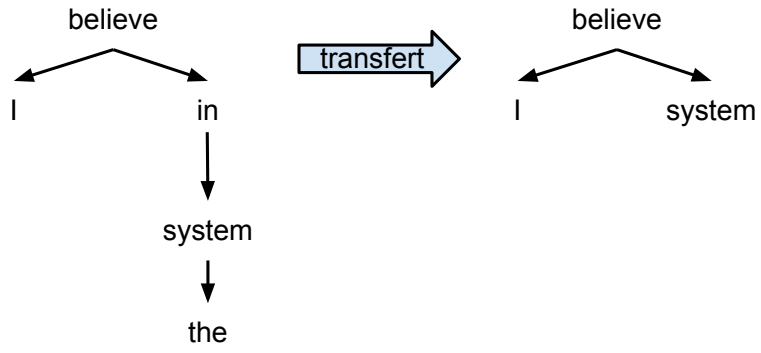


FIGURE 4.5 – Exemple de transfert

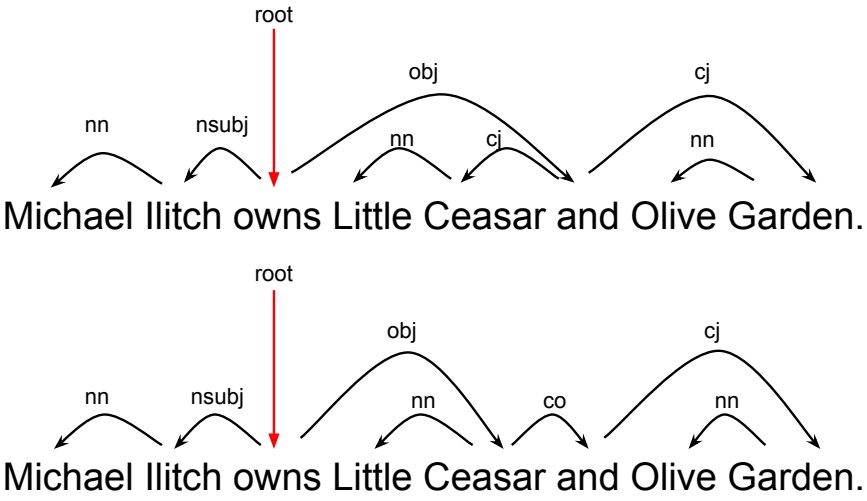


FIGURE 4.6 – Types de coordination

connectés par des arcs, éventuellement étiquetés avec une relation de dépendance. Il s'agit plus simplement de fournir un graphe de dépendances G à une phrase $S = w_0, w_1...w_n$, où w_0 est la racine.
L'apprentissage automatique d'un analyseur syntaxique est divisé en deux étapes :

- Apprentissage : dériver un modèle d'analyse syntaxique M ;
- Analyse syntaxique : dériver l'arbre optimal concevable par l'analyseur M pour une nouvelle phrase S

Les méthodes d'analyse syntaxique peuvent être divisées en deux catégories, les *data-driven* et les *grammar-based*. Dans le premier cas, on utilise l'apprentissage supervisé et un corpus linguistiquement annoté pour analyser les nouvelles phrases. Dans le deuxième cas, on établit une grammaire formelle et on vérifie si la phrase donnée en entrée est reconnue par celle-ci. Cependant, des méthodes hybrides ont déjà été explorées dans le passé, en particulier en exploitant les PCFG [Nivre 2006].

La plupart des approches guidées par les données utilisent un corpus annoté sur lequel les algorithmes d'apprentissage supervisé apprennent les règles pour pouvoir analyser une nouvelle phrase. Ces approches sont utilisables indépendamment du domaine et de la langue, pourvu qu'il existe un corpus annoté. Effectivement, il est nécessaire d'avoir une grande quantité de données pour garantir une bonne robustesse (la capacité d'analyser une phrase). Jusqu'à présent, les communautés linguistiques ont produit des corpus dans un grand nombre de langues, permettant le développement de ce type d'approches.
Les premiers travaux dans ce type d'approche sont attribués à Eisner [Nivre 2005a], qui a défini plusieurs modèles probabilistes pour l'analyse en dépendances [Eisner 1997], en montrant pour la première fois que l'utilisation d'un modèle probabiliste génératif associé au *supervised learning* atteignait une très

grande performance. Eisner utilise les POS-tags à côté des tokens et des relations grammaticales (non étiquetées), en définissant la probabilité conjointe des mots, tags et dépendances. La probabilité P qu'un mot w_i ait de générer des fils est conditionnée par la tête et les autres dépendants.

Il existe différentes classes d'approches guidées par les données, mais les plus connues sont les *transition-based* et les *graph-based*.
Les méthodes *transition-based* utilisent un automate pour déterminer l'arbre de dépendances correspondant à la phrase en entrée. Le modèle à apprendre dans la phase d'apprentissage détecte la nouvelle transition sur l'historique des transitions effectuées ; ensuite, dans la phase d'analyse syntaxique, on construit la séquence de transitions optimale selon le modèle appris. On parle, dans ce cas, de *Shift-Reduce Dependency Parsing*. L'inférence du modèle est locale, car le graphe est construit graduellement en choisissant le score le plus haut d'une transition. Dans [Kudo 2000], une approche probabiliste d'analyse syntaxique discriminative est proposée, où les SVM sont exploitées pour entraîner des classifieurs capables de prédire la prochaine action d'un parseur probabiliste qui construit des dépendances non étiquetées. Inspiré par celui-ci, le travail de [Nivre 2003], [Nivre 2005b] propose par contre des dépendances étiquetées, afin que l'information fasse partie des attributs pour la prédiction de la prochaine action. Parmi les références de ce type de parseur on peut citer le MaltParser [1].
Les méthodes *graph-based*, proposées par [Mcdonald 2006], [Eisner 2000] définissent un ensemble de graphes de dépendances candidats pour une phrase. Dans la phase d'apprentissage le modèle dérive la totalité des graphes candidats avec leurs scores et dans la phase d'analyse syntaxique le graphe avec le meilleur score est choisi pour étiqueter la phrase en entrée. L'inférence du modèle est donc dans ce cas globale. Cette méthode est appelée *Maximum-Spanning Tree Parsing*. Nous citerons le MSTParser [2] comme exemple de parseur *graph-based*.

Les modèles guidées par les données acceptent n'importe quelle phrase, qu'elle soit grammaticale ou pas, et donnent en sortie l'arbre avec le meilleur score. Les méthodes guidées par la grammaire, au contraire, n'acceptent qu'une phrase qui soit définissable par la grammaire formelle du modèle. Cependant, souvent cette grammaire est apprise directement sur un corpus annoté, qui fait que les approches *grammar-based* peuvent inclure aussi des éléments des *data-driven*. Des exemples d'analyseurs *grammar-driven*, sont : Charniak-Johnson Parser [3], le Stanford Parser [4] et le Berkeley Parser [5]. Ces approches se divisent en *Context-Free Parsing* et *Constraint-based Parsing*.
Dans le premier cas, on formalise la grammaire de dépendances sur la base d'une

1. http ://www.maltparser.org/
2. http ://www.seas.upenn.edu/ strctlrn/MSTParser/MSTParser.html
3. http ://bllip.cs.brown.edu/resources.shtml
4. http ://nlp.stanford.edu/software/lex-parser.shtml
5. http ://code.google.com/p/berkeleyparser/

grammaire de constituants. L'arbre de dépendances est dérivé de celui syntagmatique et utilise les mêmes algorithmes d'analyse syntaxique des *Context-Free Grammars* (*Shift-reduce/chart-parsing*) pour obtenir une bonne performance face à l'ambiguïté. Gaifman [Gaifman 1965] et Hays [Hays 1964] ont été promoteurs de cette technique dans les années '60, mais cette approche a fait que pendant longtemps les grammaires de dépendances n'étaient considérées que comme une extension des CFG [Nivre 2005a].
Les deuxièmes méthodes, dites aussi *Eliminative Parsing*, conçoivent l'analyse syntaxique comme un problème de satisfaction de contraintes : la grammaire formelle est définie comme un ensemble de contraintes sur des graphes correctement formés, et l'analyse syntaxique consiste à trouver le graphe qui satisfasse toutes les contraintes. Cette approche, proposée pour la première fois par [Maruyama 1990], peut par contre se révéler problématique car (i) il pourrait ne pas y avoir d'arbre satisfaisant toutes les contraintes (problème de robustesse) et (ii) il pourrait y avoir plusieurs arbres (problème d'ambigüité).
Il existe aussi une troisième tradition appelée stratégie de *Deterministic Parsing* (Covington), conçue pour être une analyse *left-to-right* où chaque nouveau mot est lié en tant que tête ou modifieur avec un mot déjà apparu par une dépendance.

4.3 Conclusion

En général, les grammaires de dépendances sont un bon compromis entre l'expressivité des représentations syntaxiques et la difficulté de l'analyse automatique. Bien qu'elles fournissent moins d'informations que les représentations en constituants, les structures prédicat-argument des grammaires de dépendances se rapprochent beaucoup plus de l'interprétation sémantique d'une phrase. Les relations grammaticales "bilexicalisées" sont une aide importante à la désambiguïsation. Pour cette raison, les structures en dépendances sont la solution idéale pour la création de systèmes de traitement du langage hautement performants.

Chapitre 5

Architecture proposée et contributions

Sommaire

Nous avons vu dans les sections précédentes quels sont les principaux domaines intéressés par ce mémoire. Nous allons maintenant découvrir quelle approche nous proposons, et les contributions que nous souhaitons donner.

5.1 Entre Web Sémantique et non-structuré

L'essor des techniques d'apprentissage ontologique a poussé à s'interroger sur la possibilité d'exploiter les données non-structurées du Web et d'intégrer les nombreuses connaissances y compris dans la nouvelle génération du Web 3.0. Certaines de ces données sont désormais déjà présentes en forme accessible, *open-source* et structurée. Elles peuvent donc constituer l'ossature d'une connaissance qui peut être enrichie avec les données du Web 2.0. De plus, beaucoup d'efforts ont été faits par la communauté du Web Sémantique pour lier les données des différentes sources et domaines avec des relations d'équivalence (les *sameAs* que nous avons vus précédemment) et cela a permis d'obtenir des données fiables et désambigüisées. Ces données sont à la base des technologies actuelles, qui se focalisent sur la recherche sémantique.
Bien que ces efforts aient été intenses, la différence de disponibilité et accessibilité des données textuelles et celles du Web Sémantique est encore remarquable : l'ontology learning peut essayer de combler ce vide en fournissant des techniques de structuration d'une grande quantité de données non-structurées sans aucune intervention humaine.

5.1.1 Découverte d'entités du Web à partir de patrons linguistiques

Ce mémoire explore donc cette voie entre les deux mondes des données et suit donc deux directions : d'un côté, il y a la volonté d'utiliser les données structurées du *Linked Data* comme la base pour la construction d'ontologies. Les *Linked Data*, dont la taille et l'interconnexion s'accroissent de jour en jour, sont un point important à exploiter pour leur richesse et extensibilité. Ces données peuvent non seulement être la base pour ne pas avoir à construire une ontologie de zéro (*from scratch*), mais elles sont aussi exploitables pour l'apprentissage des relations à partir des textes. L'idée est donc principalement d' **apprendre les patrons entre entités du Web des données à partir du Web des documents, en particulier des snippets**. Ces patrons, spécifiques d'une relation sémantique (ou attribut ontologique), peuvent ensuite être utilisés pour peupler l'ontologie avec des nouvelles entités et valeurs d'attributs. La figure 5.1 montre comment, le patron *serves up*, extrait à partir de l'entité *Olive Garden* (existante parmi les *Linked Data*), permet de récupérer les nouvelles entités *Chez Gladines* et *Chez Prospère* qui, elles, sont présentes dans le Web comme données non-structurées.

[**Olive Garden**]$_{LOD}$ serves up authentic [**Italian cuisine**]$_{LOD}$

$\Downarrow$

[**Chez Gladines**]$_{Web}$ serves up [**Basquian cuisine**]$_{Web}$

[**Chez Prospère**]$_{Web}$ serves up [**French cuisine**]$_{Web}$

FIGURE 5.1 – Exemple de découverte d'entité

La méthode hybride ici proposée touche tous les domaines analysés jusqu'à présent : des techniques linguistiques, statistiques et d'apprentissage automatique sont appliquées pour la **découverte d'entités à partir de patrons linguistiques pour la construction d'ontologies**.

Nous avons vu comment les patrons linguistiques ont été largement exploités dans l'extraction d'information non-supervisée et l'apprentissage d'ontologies pour la découverte des entités. Les patrons lexico-syntaxiques de type Hearst ont montré l'efficacité de ces techniques, mais la nouvelles recherches, issues de la reconnaissance d'entités nommées et de l'extraction de relations, cherchent à utiliser les grammaires de dépendances pour l'identification des relations.
La direction prise dans ce mémoire est donc celle d'utiliser les **grammaires de dépendances** pour l'apprentissage de patrons spécifiques aux relations sémantiques existant dans le *Linked Data*. Cette voie, largement explorée dans les corpora de domaine, a été moins explorée pour les données textuelles du Web.

5.2 Approche Proposée

L'architecture que nous présentons ici pour l'apprentissage d'ontologies en utilisant le Web des *Linked Data* et les grammaires de dépendances consiste en 5 étapes

(cf. 5.3)

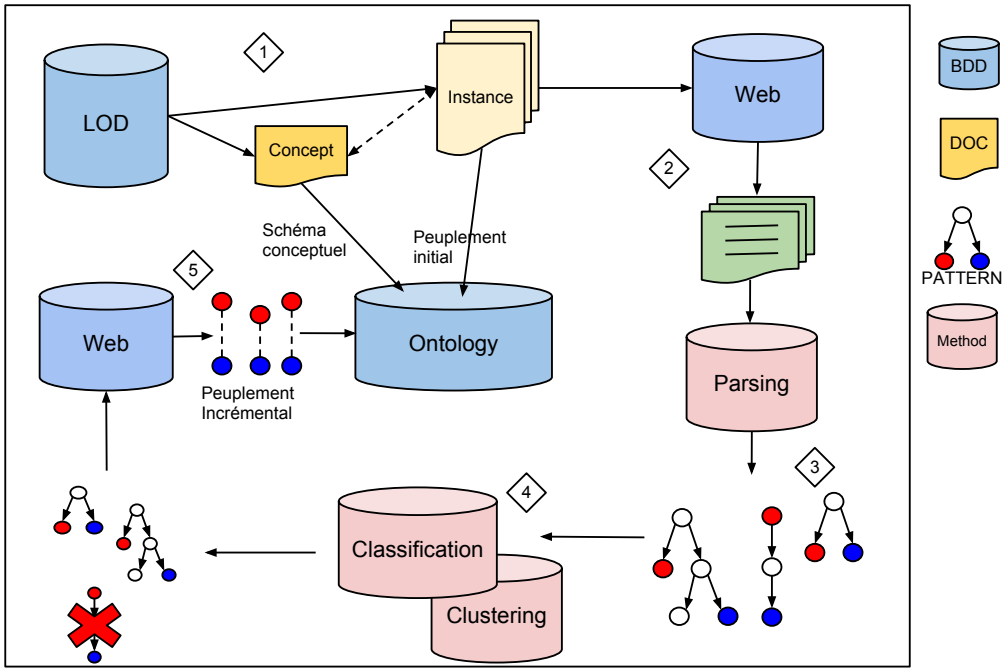


FIGURE 5.2 – Architecture d'extraction de dépendances

1. Extraction des *Linked Data* Les données du Web Sémantique sont le point de départ pour le système. Dans cette section, on utilise les ontologies déjà existantes dans un double processus : d'un côté, on extrait le schéma d'un concept, c'est-à-dire le concept et l'ensemble des attributs qui le caractérisent, par exemple :

$$Restaurant = \{founder,\ foundingDate,\ owner,\ product...\}$$

Ensuite, on extrait les instances de ce concept, avec leurs attributs respectifs (cf. 5.1).

Instance	founder	foundingDate	owner	product
Olive Garden	Mike Ilitch	1996	Darden Rest.	Italian cuisine
Teany Café	_	2002	Kelly Tisdale	vegan food
CKE Rest.	Carl Karcher	_	Apollo Mgmt	_

TABLE 5.1 – Instances et valeurs pour *Restaurant*

Notre contribution dans cette section est l'utilisation de l'ontologie de **Schema.org pour l'ossature des concepts** intégrée avec **les instances et les valeurs de DBpedia** grâce aux équivalences existant dans le Web des données. Nous montrons donc comment tirer parti de leurs interconnexions (en utilisant les relations `same-as`), qui évitent la redondance des données.

2. Recherche Web Le but de cette étape est la construction du corpus de contenus du Web. À partir de cette étape, nous travaillons sur les couples instance-valeur d'attribut (donc un sujet et un objet dans le schéma des instances). Pour chaque couple, on construit un corpus D de documents incluant les deux entités du couple. Pour la relation *owner* de l'exemple précédent, nous chercherons dans le Web les requêtes suivantes : r_1= *"Olive Garden"+"Darden Restaurants"*, r_2= *"Teany Café"+"Kelly Tisdale"*, r_3= *"CKE Restaurant"+"Apollo Management"*.
Au lieu de traiter des pages Web, où les contenus textuels sont difficilement accessibles (car nécessitant un parsing de la page), le corpus est constitué par des **snippets retournés par le moteur de recherche**.

3. Analyse syntaxique du corpus L'objectif à cette étape est de récupérer un patron qui lie deux entités. Une structure syntaxique en dépendances est dérivée de chaque document. De chaque arbre comprenant les deux valeurs, est dérivé un patron candidat spécifique à la relation sémantique correspondante. Un patron est plus simplement, le chemin le plus court entre deux entités à l'intérieur de l'arbre. Le format en dépendance est l'innovation qui permet de récupérer **juste l'information sémantique, sans éléments superflus**. Par exemple :

The first owner of $[Teaney\ Café]_I$ *was the famous* $[Kelly\ Tisdale]_{VAL}$
~~*The first*~~ ***owner of*** *Teaney Café* ~~*was the famous*~~ *Kelly Tisdale*
$\Rightarrow$ owner_of(Kelly Tisdale, Teaney Café)

4. Filtrage des patrons Cette phase permet de redéfinir les patrons à l'aide des techniques d'apprentissage automatique. Chaque patron, c'est-à-dire un sous-arbre de dépendances, est analysé d'un point de vue linguistique par les **techniques de clustering et classification** pour créer, ensuite, des groupes de patrons pour chaque relation, et pour identifier quels patrons sont spécifiques à une classe.

owner_of(Kelly Tisdale, Teaney Café)
own(Darden Restaurants, Olive Garden)
$\Rightarrow p_1 = \text{own}_{lemma}(\text{Y,X})$
$p_1 \in R, R = [ownership]$

5. Découverte de nouvelles entités Les patrons ainsi validés sont utilisés pour récupérer de nouveaux couples d'entités, à retrouver dans de nouvelles phrases, extraites à l'aide **d'un moteur de recherche**. Ces couples sont utilisés pour l'enrichissement de l'ontologie construite pendant la première phase (cf. 5.3).

$[Mission\ Ranch\ Restaurant]_{NI}$ *is owned by* $[Clint\ Eastwood]_{NV}$
$[Robert\ DeNiro]_{NV}$ *owns the* $[Ago\ Restaurant]_{NI}$
$[The\ Matsuhisa\ Restaurants]_{NV}$ *are privately owned by the* $[Matsuhisa\ family]_{NV}$

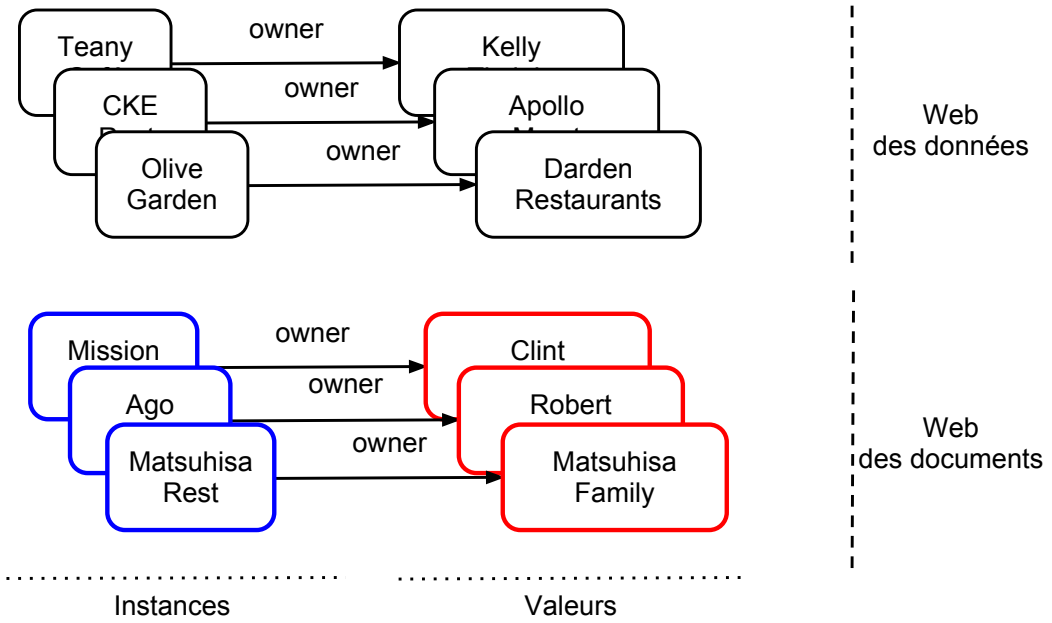


FIGURE 5.3 – Enrichissement du concept *restaurant*

Chapitre 6

Expérimentation

Sommaire

La découverte de patrons se décompose en deux processus : la création de patrons avec les grammaires de dépendances, et un affinement de ces patrons en utilisant l'apprentissage automatique.
Nous allons présenter nos expérimentations étape par étape, en montrant nos contributions, nos résultats mais aussi les défaillances et les solutions trouvées à chaque fois.

6.1 Extraction

La phase d'extraction consiste en deux parties :

1. Extraction du schéma d'un concept : par schéma, nous entendons ici un concept choisi et l'ensemble des attributs en relation avec ce concept.
2. Extraction des instances du concept et leurs valeurs d'attributs

6.1.1 Extraction des concepts

L'ontologie choisie pour l'extraction des schémas conceptuels est celle fournie par *Schema.org*.
L'utilisateur sélectionne un concept qui est connu dans l'ontologie et une requête SPARQL est automatiquement créée afin d'extraire tous les attributs qui sont en relation avec lui. Considérant la structure hiérarchique de l'ontologie, certains attributs seront spécifiques à ce concept alors que d'autres seront hérités de ses classes-mères. Par exemple, pour *Restaurant* nous aurons un ensemble d'attributs tels que :

Attribut	Type d'Attribut	Hérité de
name, type, description, url	DataType	Thing
telephone, faxNumber	DataType	Place
event, review, geoCoordinates, aggregateRating, address	ObjectType	
contactPoint, employee, founder location, member	ObjectType	Organisation
email, foundingDate	Datatype	
branchOf	ObjectType	LocalBusiness
currenciesAccepted, openingHours paymentAccepted, priceRange	DataType	
acceptsReservations, menu, servesCuisine	DataType	Restaurant

TABLE 6.1 – Ensemble d' attributs pour le concept *Restaurant*

Pour notre expérimentation, nous avons utilisé 10 concepts, avec certains attributs en communs et certains spécifiques. Ces concepts (*Actor*, *Hospital*, *Hotel*, *Movie*, *Musician*, *Painting*, *President*, *Restaurant*, *Sitcom*, *University*), sont organisés hiérarchiquement (Fig. 6.1) :
Chaque concept avec ses attributs est sauvegardé dans un module ontologique,

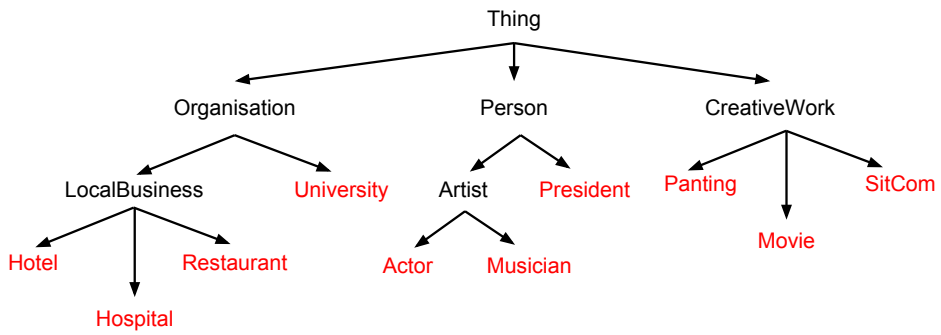


FIGURE 6.1 – Hiérarchie des concepts choisis

c'est-à-dire une ontologie indépendante (au format OWL). Ces modules sont ensuite rassemblés dans une seule ontologie modulaire. La modularisation d'ontologies est une technique de l'ingénierie des connaissances, visant à faciliter l'exploitation des ressources. Issue du génie logiciel, elle signifie une organisation d'une application assurant l'indépendance entre fonctions distinctes, afin qu'elle soit facilement modifiable, et utilisable par d'autres ressources. L'ingénierie ontologique reprend ce principe pour concevoir des ontologies comme une combinaison de composants de connaissance (modules) indépendants, auto-conteneurs et réutilisables [d'Aquin 2007]. Chaque module concerne un domaine (plus ou moins spécifique) différent et est mis en relation avec d'autres modules indépendants.

6.1.2 Extraction des Instances

Une fois que les concepts et les attributs sont extraits, il faut récupérer des instances. Nous avons déjà vu que dans une ontologie il existe un niveau conceptuel et un niveau réel : une instance est l'entité du monde réel qui a une relation `instanceOf` avec un concept C (Fig. 6.2).

L'ontologie *Schema.org* ne contenant que les schémas des concepts, nous avons

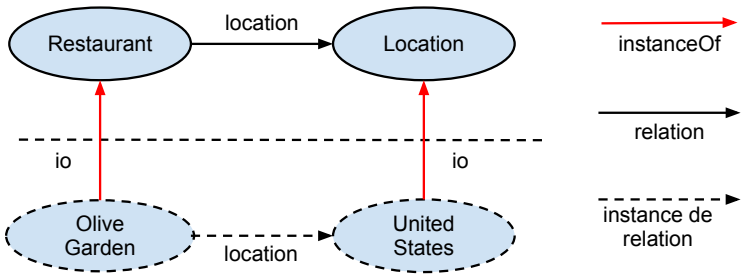


FIGURE 6.2 – Exemple d'instances

donc utilisé *DBpedia*. Pour chaque concept, 100 instances ont été extraites.

Comme précedemment expliqué, nous nous intéressons ici à l'extraction de patrons relatifs à un attribut ontologique. Pour faire cela, on récupère toutes les instances ainsi que leurs valeurs pour chaque attribut . Idéalement, nous obtenons 100 instances et 100 valeurs. Quelques extraits sont montrés dans le tableau 6.2.

6.1.3 Problèmes, et solutions choisies

Incongruité de DBpedia Bien que DBpedia soit une source fiable et riche en informations, des problèmes apparaissent car ces informations s'avèrent être désordonnées, ambigües, voire erronées. Comme nous l'avons déjà évoqué dans les parties précédentes, DBpedia est connectée à plusieurs ontologies, ce qui se traduit dans une

SemRel	Instance	Valeur
Location	Olive Garden	Orlando
	Hungry Jack's	Osborne Park Western
	Little Caesar	Detroit
BranchOf	Olive Garden	Darden Restaurants
	Little Caesar Enterprises	Little Caesars
	Hungry Jack's	Competitive Foods Australia
Founder	Olive Garden	Mike Ilitch
	Little Caesar Enterprises	Jim Nicholson
	Hungry Jack's	David Edgerton
...		

TABLE 6.2 – Exemple d'instances pour *Restaurant*

prolifération de relations, ainsi que de valeurs. Plusieurs cas peuvent se produire.
La même information peut être retrouvée dans différentes ontologies, produisant à chaque fois un nouveau triplet. Un premier cas est dû à la diversité de *namespaces*. La localisation, relation entre *Olive Garden* et *Orlando*, est exprimée comme `<dbprop:location>` mais aussi `<dbpedia-owl:location>`. Une différence de *namespaces* peut créer des erreurs. De plus, la valeur peut aussi ne pas être la même : selon la granularité, l'objet de ce triplet est *Orlando*, *Florida* ou *United States*. Ce qui produit plus de triplets et plus de difficulté pour la formulation des requêtes Web. Le problème dû à la granularité de l'information se retrouve aussi dans les noms des attributs : les relations liées à un lieu (*location* ou *foundation* pour *Organisation*, *birth* ou *death* pour *Person*), sont divisées en *place* (plus générique) ou *city* et *country* (plus spécifiques). Ceci se vérifie aussi pour les relations temporelles : *date* et *year* indiquent le même événement, mais avec un niveau de spécification différent.
Le schéma de DBPedia n'étant pas fixe ni contrôlé, un autre cas peut se produire : les erreurs de frappe ou, plus simplement, d'attributs en double dans le même *namespace*. Pour la relation de nombre d'employés, par exemple, nous retrouvons *numberOfEmployee*, *numEmployee*, *numberEmployees*. Pour le fondateur d'un restaurant, on constate le problème de qualité des noms d'attributs, on trouve en effet l'attribut *founder*, *Founder*, mais aussi *foubder*.
Ces problématiques font grandir rapidement le nombre d'attributs à traiter. De plus, voir proliférer les attributs fait aussi que les valeurs s'éparpillent : souvent, pour un attribut il n'y a que peu d'instances concernées (entre 0 et 5).
Avec une extraction automatique, nous retrouvons, donc, 721 relations pour la dizaine de concepts considérés. Cela rend le traitement difficile et nécessite donc, avant la prochaine étape, un filtrage.

Solutions Une solution peut être de regrouper les valeurs des attributs proches manuellement. L'intervention humaine est sûre et permet d'éviter les erreurs dues à l'automatisation de la tâche, mais coûte, d'un autre côté, beaucoup de temps et

ne permet plus de profiter de la granularité des attributs. Il s'agit d'un côté de perdre une partie de l'information disponible et de l'autre de gagner sur la fiabilité de cette information même.
Une solution automatique à cette étape peut être aussi envisagée pour filtrer les attributs avec trop peu de valeurs d'une part, et pour rapprocher les attributs similaires de l'autre, à l'aide des mesures de similarité entre *labels*. Dans le premier cas il est évident qu'il y aurait une perte de couples candidats, mais la quantité de données perdue dans le filtrage resterait, de toute façon, très faible par rapport à celle encore disponible. La similarité des attributs, de son côté, permettrait d'éviter les problèmes dus aux fautes de frappe, mais pas de clusteriser les attributs plus ou moins spécifiques.

Afin de réduire les attributs pour le traitement suivant, nous avons utilisé un filtrage automatique pour éliminer les attributs avec moins de 5 valeurs. Le résultat a donné 294 attributs exploitables. Ensuite, un clustering manuel a permis de réduire les attributs utilisables à 50 : 17 pour les *CreativeWork*, 18 pour *Person*, 15 pour *Organisation*.

Attribute Matching L'autre grand problème qui se présente à cette étape est l'utilisation de deux ontologies différentes, une pour le schéma et une pour les instances. Comme nous venons de voir, il est fondamental d'avoir un schéma fixe afin que les informations ne soient pas dispersées. Cependant, les attributs peuvent ne pas se correspondre et, surtout, il peut être difficile de rapprocher le niveau conceptuel et celui du monde réel (autrement dit, "rattacher" les instances et leurs valeurs au schéma conceptuel).
Dans ce cas, on peut recourir à des mesures de similarité pour calculer la distance entres attributs (*founder* et *foundedBy* seraient très proches) mais cela ne permettrait pas de rapprocher *year* et *date*.

La solution mise en place vient des initiatives liées au Web des *Linked Data* : un mapping [1] entre classes et relations des deux ontologies est fourni pour faciliter les rapprochements. Dans un *mapping*, les classes et les propriétés sont mises en relation grâce à des triplets d'équivalence (cf. 6.3) :
Ce triplet nous permet de savoir que l'attribut *parentOrganisation* de DBpedia

```
– <owl:ObjectProperty rdf:about="http://dbpedia.org/ontology/parentOrganisation">
    <rdfs:label xml:lang="en">parent organisation</rdfs:label>
    <rdfs:domain rdf:resource="http://dbpedia.org/ontology/Organisation"/>
    <rdfs:range rdf:resource="http://dbpedia.org/ontology/Organisation"/>
    <owl:equivalentProperty rdf:resource="http://schema.org/branchOf"/>
  </owl:ObjectProperty>
```

FIGURE 6.3 – Exemple de attribute matching

1. http ://mappings.dbpedia.org/server/ontology/export

peut être remplacé, suivant le schéma de Schéma.org, par *branchOf*. Ce mapping, cependant, ne tient pas compte du niveau de spécification des propriétés et ne permet pas de réduire automatiquement le nombre d'attributs à traiter.

6.1.4 Outils

Voici la liste des outils que nous avons utilisés dans cette étape.

Virtuoso SPARQL endpoint Le service Virtuoso implémente le protocole SPARQL[2] pour l'interrogation et l'utilisation des documents RDF. Ce service fournit des résultats dans le format standard XML, mais offre aussi d'autres possibilités, comme JSON, HTML ou CSV. DBpedia repose sur le service Web Virtuoso pour l'interrogation de ses données. Pour la récupération des données, nous formulons trois types de requêtes :

1. Sélection des instances. La requête utilisée est la suivante :

   ```
   SELECT DISTINCT ?instance WHERE {?instance isA #Concept}
   ```

   Cette requête permet de sélectionner toutes les instances relatives à un concept (`#Concept`) que l'utilisateur choisit librement. `isA` est la relation d'appartenance à une classe donnée, comme *OliveGarden-isA-Restaurant*

2. Selection du schéma de concept

   ```
   SELECT DISTINCT ?property ?class WHERE {
   {?property rdfs:range #Concept}
   UNION
   {?property rdfs:range ?class. #Concept rdfs:subClassOf ?class}
   UNION
   {?property rdfs:isDefinedBy ?class. #Concept rdfs:subClassOf ?class}
   UNION
   {?property rdfs:isDefinedBy #Concept}
   UNION
   {?property rdfs:domain #Concept}
   UNION
   {?property rdfs:domain ?class. #Concept rdfs:subClassOf ?class}
   }
   ```

   Cette longue requête permet de récupérer tous les attributs dont le `#Concept` est domaine ou co-domaine (ou `type` ou *isDefinedBy*). En particulier, cela permet de récupérer aussi les attributs que le `#Concept` hérite de ses classes-mères.

3. Sélection des valeurs d'instances.

   ```
   SELECT DISTINCT ?property ?value WHERE {#Instance ?property ?value}
   ```

2. http ://www.w3.org/TR/rdf-sparql-protocol/

Permet de récupérer les attributs et valeurs d'attributs qui sont associés à `#Instance`. Cette instance n'est pas sélectionnée par l'utilisateur, mais il s'agit du résultat de la requête 1. Pour chaque résultat envoyé par 1 (*Olive Garden*, *Little Caesar*, etc), une requête 3 est envoyée pour récupérer toutes les valeurs associées au résultat.

Algorithme 1 Récupération des valeurs d'instance

$concept \leftarrow$ "$restaurant$"
$listeInstances \leftarrow selectionnerInstancesDe(concept)$
pour $i = 0$ à $taille(listeInstances)$ **faire**
  $valeurs \leftarrow selectionnerValuersDe(listeInstances[i])$;

Le framework Jena Le Apache Jena Project [3] est un ensemble d'outils et librairies Java pour la construction de technologies sémantiques et pour le *Linked Data*. Ces outils comprennent, entre autres, des librairies pour lire, traiter et écrire des données en RDF (XML, N-triples et Turtle) et en OWL, ainsi qu'un moteur de requêtes SPARQL pour l'interrogation des données.
Dans cette étape, nous avons utilisé ce *framework* pour lire les ontologies existantes, effectuer les requêtes SPARQL qui extraient les informations que nous recherchons, et écrire les modules ontologiques au format OWL.

6.1.5 Évaluation

L'évaluation à cette étape consiste dans l'analyse des données que nous allons traiter dans les étapes futures.
Comme nous l'avons déjà montré auparavant, nous avons extrait 100 instances par concept. En réalité, nous constatons que certains concepts, (*Hotel* et *University*) n'en contiennent pas autant. Le nombre d'instances par concept est présenté figure 6.3. Ceci ne constitue pas en soi un gros problème, ces chiffres nous permettent tout de même de récupérer assez de snippets pour l'apprentissage des patrons.

Concept	Num. Instances
Actor, Artist, Film, Hospital, Painting, President, Restaurant, TVSerie	100
Hotel	85
University	71

TABLE 6.3 – Nombre d'instances pour chaque concept

L'évaluation des informations fournies pour chaque instance (c'est-à-dire, les valeurs d'attributs) présente un plus grand intérêt. Comme nous l'avons déjà constaté,

3. http ://jena.apache.org/index.html

plusieurs bruits et incongruences existent dans DBpedia, ce qui conduit à l'extraction de mauvais couples instance-valeur. Pour en citer quelques-uns :

- les dates et lieux des événements tels que naissance, fondation, mort sont souvent mélangés. Par exemple :

  (1a) "Olive_Garden" *foundationPlace* "1982"
  (1b) "Sveinn_Bjornsson" *deathDate* "Reykjavik"

- Valeur d'attributs avec trop d'informations (donc difficiles à gérer pour la création des snippets), ou trop peu précises.

  (2a) "Dante_Alighieri" *deathDate* "–09"
  (alors que "Dante_Alighieri" *deathDate* "1321" existe aussi)
  (2b) "The_Pruitts_of_Southampton/The_Phyllis_Diller_Show"
  productionCompany "Film-
  ways_TV_Productions_in_association_with_PhilDil_Productions_Limited"

- Erreurs diverses (encodage, mauvaise extraction, pas de contrainte sur la valeur de l'attribut)

  (3a) "Taylor_Elizabeth" *award*
  "List_of_awards_and_nominations_received_by_Elizabeth_Taylor"
  (3b) "Carter_Jimmy" *award* "20"
  (3c) "Jack_in_the_Box" *areaServed* "19.0"
  (3d) "Aerated_Bread_Company" *areaServed* " "

La figure 6.4 donne une idée de la quantité de valeurs erronées pour chaque groupe.

FIGURE 6.4 – Valeurs erronées pour chaque groupe

6.2 Web corpus

La deuxième étape du système est la constitution du corpus à partir du Web. Cela consiste tout simplement à prendre, pour chaque attribut que nous traitons, l'instance et la valeur d'attribut correspondant (par exemple, pour *owner*, nous prenons *Olive garden* et *Darden Restaurants*) et à lancer une requête à un moteur de recherche, en le "forçant" à nous rendre tous les documents contenant les deux valeurs dans la même phrase. La recherche Web permet d'avoir un accès rapide et précis à des documents contenant une information requise. Dans le cas de ce travail, la recherche Web nous assure de récupérer le maximum de phrases où apparaissent ces deux valeurs, afin d'étudier le lien entre elles d'un point de vue linguistique.
Notre proposition dans cette étape est l'utilisation des *snippets*, c'est-à-dire le groupe de phrases retourné par le moteur de recherche pour chaque document (cf.6.5)

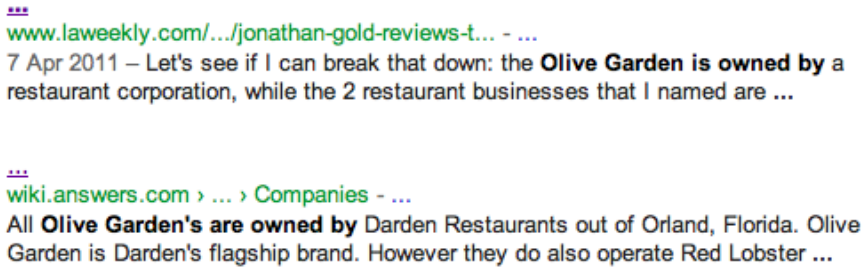


FIGURE 6.5 – Exemple de Snippets

Le snippet, étant plus petit, permet d'éviter le long processus d'extraction des contenus des pages Web : les phrases intéressantes, les documents étant déjà indexés par le moteur de recherche, sont retournées directement et facilement exploitables.
Pour obtenir des phrases où sont présentes les deux entités en même temps, il est nécessaire de "forcer la requête". Ceci est fait en mettant les deux entités entre guillemets ("*Olive Garden*"), pour obtenir les documents où les tokens de chaque entité apparaissent ensemble (et non *Olive* séparément de *Garden*) ; ensuite, les deux entités sont liées par la conjonction *AND* (ou &) pour forcer des résultats où l'on retrouve chaque couple.

```
"Olive Garden"AND"Darden Restaurants"
```

6.2.1 Problèmes et Solutions

Le Web est plein d'erreurs ! La recherche Web est l'une des étapes les plus importantes. En effet, de la qualité du corpus que nous analysons dépendra directement la qualité des patrons de dépendances que nous allons extraire.
La constatation évidente à cette étape est que les résultats fournis par les moteurs de recherche sont loin d'être parfaits : le corpus extrait est très bruité, et sa qualité

est très variable en fonction de la relation qui est traitée.
Nous avons détecté différents types d'erreurs :

1. Phrases ne contenant pas les deux entités en même temps. Le snippet entier contient effectivement les deux entités, mais elles sont dans des phrases séparées par des points (a) ou trois points (b). Exemple :

   (a) ***Olive Garden*** *is an American casual dining restaurant chain specializing in Italian-American cuisine. It is a subsidiary of* ***Darden Restaurants****, Inc., which is...*
   (b) *The* ***Darden*** *family of restaurants features some of the most recognizable and successful ...Red Lobster,* ***Olive Garden****, LongHorn Steakhouse ...*

2. Phrases contenant une seule des deux entités. Si peu de résultats sont trouvés, ou bien la visibilité de ces documents est faible, nous obtenons des résultats inexploitables. Exemple :

   Visit the ***Olive Garden*** *Web site to purchase gift cards, locate Italian restaurants near you and explore our fine Italian dining menu*

3. Phrases hors sujet. Des sites ayant plus de visibilité sont affichés car, pour des raisons principalement commerciales, ils sont reliés à une entité. Le cas le plus exemplaire, celui de Facebook. Exemple :

   Holiday Inn*. Holiday Inn on Facebook; Holiday Inn UK & Ireland on Facebook; Holiday ...* ***InterContinental Hotels Group*** *2012*

4. Phrases dans une autre langue. Nous avons montré comment la recherche Web est forcée par notre requête à chercher les pages sur un marché particulier (anglophone). Cependant, tous les résultats ne respectent pas cette demande :

   Eurostars Hotels *cuenta con màs de 70 exclusivos hoteles de lujo. Reserve ahora su hotel ...* ***Grupo Hotusa***

5. Phrases agrammaticales. Ce dernier cas est un sujet plus sensible : en effet, rien n'empêche les moteurs de recherche de renvoyer des résultats tels que

   2012 ***Wyndham Hotel*** *Group, LLC. All rights reserved.|* ***Days Inns*** *Terms of Use | Privacy Policy | Press & Media*

   c'est-à-dire des extraits des contenus structuraux des pages, morceaux sans verbes, où apparaissent seulement des noms propres (parmi lesquels, nos entités).

La coréférence textuelle Reprenons le cas de la phrase (a) :

Olive Garden *is a [...] restaurant chain [...].* ***It*** *is a subsidiary of* ***Darden Restaurants*** *[...]*

Si nous prenons la deuxième phrase, c'est *it* qui est lié à *Darden Restaurants*, et non *Olive Garden*. Pourtant, nous savons que les deux entités *Olive Garden* et *Darden*

Restaurants sont liées. Pourquoi ? C'est le phénomène de la coréférence. Halliday [Halliday 1976] explique comment dans une langue il est impossible d'interpréter sémantiquement certaines constructions, car elles font référence à quelque chose en dehors de la phrase. La représentation sémantique et celle grammaticale de ces phrases ne coïncident pas. C'est bien le cas de notre exemple. La relation de référence est dite *cataphorique* si le référent vient après et *anaphorique* dans le cas contraire. Différents types de référence existent (personnelle, démonstrative ou comparative) [Halliday 1976]. Dans notre corpus, les coréférences personnelles (où l'on utilise les pronoms, ainsi que les pronoms possessifs) sont les plus fréquentes.

La résolution de la coréférence est un sujet de grand intérêt et constitue une discipline à part entière pour le TAL. En général, pour la résolution de coréférences on utilise des techniques d'apprentissage automatique. Dans la tache d'extraction d'information, en particulier dans l'extraction de relations entre entités nommées, la résolution de la coréférence peut être très importante car elle permet de lier les entités au-delà du niveau de la phrase.

Sentence Boundary Disambiguation La détection des phrases séparées par un point (phrase (b)) n'est pas aussi facile qu'on peut imaginer. Il s'agit d'une problématique bien connue dans le traitement des textes : le *Sentence Boundary Disambiguation*, ou détection de phrases, est à la base de beaucoup de systèmes de TAL [Gillick 2009]. Bien qu'implémentée de façon très efficace, la détection n'est pas toujours parfaite, et implique donc une possible perte d'informations.

Solutions La raison de ces erreurs est attribuée à plusieurs facteurs, mais le problème principal est la pollution des résultats par des informations publicitaires. Malheureusement cela ne peut être évité si on utilise des outils dont le principal revenu est la publicité. Nous avons par conséquent dû mettre en place des solutions pour réduire le bruit dans notre corpus.

Nous avons utilisé l'outil GATE [4] pour effectuer un filtrage préliminaire. GATE est un programme pour le *text processing* qui nous a permis, à l'aide de son plugin ANNIE (*A Nearly New Information Extraction system*), d'éliminer la plupart des phrases qui ne nous intéressaient pas. ANNIE nous a permis de :

- Eliminer les phrases qui n'étaient pas en anglais (problème 4). Il existe en effet des outils d'identification automatique de la langue (on peut citer aussi LexTex [5], le LanguageIdentifier [6] de Apache, ou jExSLI [7]), qui utilisent dans la plupart des cas des techniques de classification pour calculer la similarité entre textes.

4. http ://gate.ac.uk/
5. http ://www.lextek.com/langid/
6. http ://nutch.apache.org/apidocs-1.1/org/apache/nutch/analysis/lang/LanguageIdentifier.html
7. http ://hlt.fbk.eu/en/technology/jExSLI

- Détecter et séparer les phrases (problème 1). En moyenne, un snippet contient deux ou trois phrases, qu'il faut détecter. L'analyseur syntaxique n'est pas capable de séparer deux phrases, si elle lui sont données en entrée en même temps. ANNIE s'occupe de détecter les phrases à la fois séparées par des trois points "..." (elles sont les "morceaux" de texte où le moteur de recherche a identifié les deux entités) ou par un seul point.
- Eliminer les phrases ne contenant pas les deux entités en même temps (problème 2 et 3). Ainsi, les phrases hors sujet sont donc écartées, celles avec une seule entité également.

Ce traitement implique l'élimination de toutes les phrases potentiellement intéressantes mais contenant une coréférence. Inclure une étape supplémentaire pour le traitement de la coréférence textuelle aurait été une tâche longue, et dont il était impossible de savoir si elle améliorerait significativement les résultats. Ce faisant, des phrases *a priori* correctes, c'est-à-dire contenant un patron potentiellement bon, sont éliminées (c'est le cas de la phrase (a)), mais leur nombre reste toujours très bas par rapport à la grandeur du corpus. Cette caractéristique du Web joue en notre faveur : bien que nous écartions des informations potentiellement bonnes, la quantité de données disponibles est telle que nous pouvons nous permettre d'en perdre une partie. Ce qui ne peut pas arriver avec un corpus plus petit : dans ce cas, étant moins bruité, il serait très intéressant, voire nécessaire, d'inclure une tâche d'extraction de coréférences dans le processus.

Il nous reste à traiter le problème 5, que nous traiterons dans l'étape d'analyse syntaxique.

6.2.2 Outils

Bing Web Search Les grands moteurs de recherche (Google, Yahoo ! et Bing) fournissent chacun des APIs (*Application Programming Interface*) permettant l'utilisation de leurs outils de recherche d'information par des applications externes. Dans notre cas, nous avons utilisé la *Search Engine* de Microsoft, Bing [8], permettant d'effectuer autant de recherches que l'on souhaite, au moins pour des fins non-lucratives, [9].La récupération des snippets est, en soi, très facile.

Une requête n'est qu'une URL qui, envoyée à un serveur HTTP, renvoie une réponse dans un format désiré. La requête est composée comme suit (cf. 6.6) :

- URL, pour se connecter au moteur de recherche Bing ;
- ID, pour avoir accès aux informations, chaque utilisateur est muni d'un identifiant ;
- Marché, qui indique vers quelles pages se retourner (dans l'exemple, celles en anglais, créées aux Etats-Unis) ;
- Requête, c'est-à-dire les mots-clés que nous cherchons ;

8. http ://msdn.microsoft.com/en-us/library/dd251056

9. L'API est passée sur le MarketPlace Azure et devenue payante. Pour une utilisation académique, il sera tout de même possible d'effectuer jusqu'à 5000 requêtes par jour

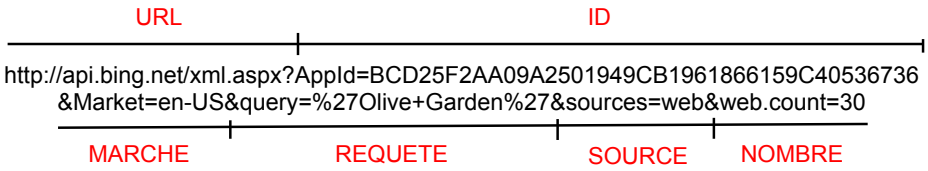


FIGURE 6.6 – Une requête

- Source, qui indique quel type de documents chercher (pages Web, mais aussi images, news, blogs ou même vidéo) ;
- Count, pour limiter le nombre de résultats (nous avons limité à 30)

D'autres options existent, mais ne sont pas obligatoires. Notamment, le format des résultats peut être XML (par défaut), JSON ou SOAP.

Nous avons choisi le format XML (cf. 6.7), afin de pouvoir récupérer avec une

```
<web:WebResult>
  <web:Title>
    Darden Restaurants - A Leader in the Full-Service Restaurant
        Industry
  </web:Title>
  <web:Description>
    The Darden family of restaurants features some of the most
        recognizable and successful brands in full-service dining: Red
        Lobster, Olive Garden, LongHorn ...
  </web:Description>
  <web:Url>
    www.darden.com/
  </web:Url>
  <web:CacheUrl>
    http://cc.bingj.com/cache.aspx?q=%22Olive+Garden%22%2b%22Darden+
        Restaurants%22&d=4697268992674396&mkt=en-US&setlang=en-US&w=
        d22d8b3b,bb7d08a7
  </web:CacheUrl>
  <web:DisplayUrl>
    www.darden.com/
  </web:DisplayUrl>
  <web:DateTime>
    2012-07-04T04:34:00Z
  </web:DateTime>
</web:WebResult>
```

FIGURE 6.7 – Résultat renvoyé par la requête

simple requête XPath tous les snippets (correspondant à *web :Description*). En annexe un script Python (cf. annexe A) effectuant cette tâche, l'envoi des requêtes et la récupération des résultats.

6.2.3 Évaluation

L'évaluation de cette étape n'est sûrement pas facile, mais c'est l'une des plus significatives car elle nous permet de donner un jugement sur le corpus que nous utilisons.
À la fin de l'étape d'extraction du corpus du Web, nous disposons de plus de 165.000 snippets, chacun correspondant à un couple instance-valeur. La répartition exacte est montrée figure 6.8.

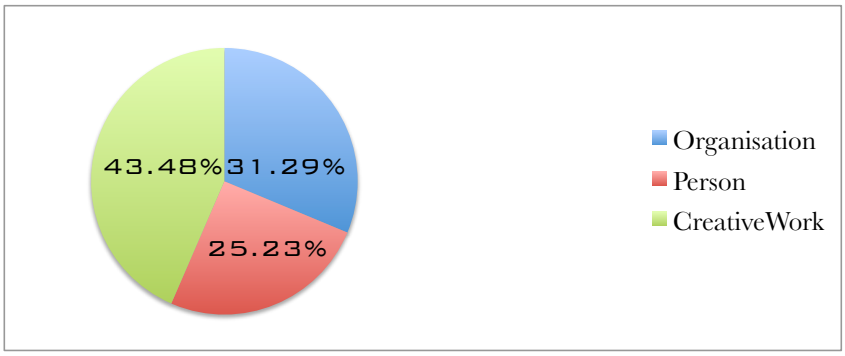


FIGURE 6.8 – Répartition des snippets dans le corpus total

Ces snippets, nous l'avons vu, ne sont pas exploitables dans leur intégralité : la plupart, au contraire, ont été écartés suite à la phase de filtrage avant l'analyse syntaxique.

Analyse automatique Etant donné le grand nombre de données, il est impossible, d'envisager une évaluation par précision et rappel sur la totalité du corpus : nous avons donc séparé l'évaluation en deux parties, une évaluation automatique et une évaluation manuelle. Dans cette première évaluation nous avons compté combien de snippets nous avons obtenu depuis le Web et combien ont effectivement "passé" le filtrage via ANNIE. Une vision spécifique à la classe *Organisation* (où nous avons classé le concept *Restaurant*) est présentée en 6.9, où l'on voit un détail du nombre de snippets (en rouge) qui ont été considérés analysables.

En moyenne, 1 snippet sur 3 (32,02%) est utilisable. La plupart des relations restent sur cette moyenne, il n'y a que la relation *location* où 50% des snippets sont potentiellement bons (51,16%). Ceci est principalement dû au fait que *location* est une relation très "générique", qui concerne plusieurs concepts, donc nous avons plus de chances de récupérer de bonnes phrases sur le Web. Au contraire, une relation comme *officialSchoolColor*, "spécifique" à un domaine (*University*), a beaucoup moins de documents s'y référant et le pourcentage de bons snippets est seulement de 18%. Même phénomène avec *foundingDate* et *keyPleople*, qui sont des relations communes à plusieurs domaines (*Restaurant*,*University*,*Hotel*), et se retrouvent donc dans plus de documents, ce qui engendre plus de snippets corrects : leurs pourcentages d'exactitude sont respectivement de 39% et 42%.

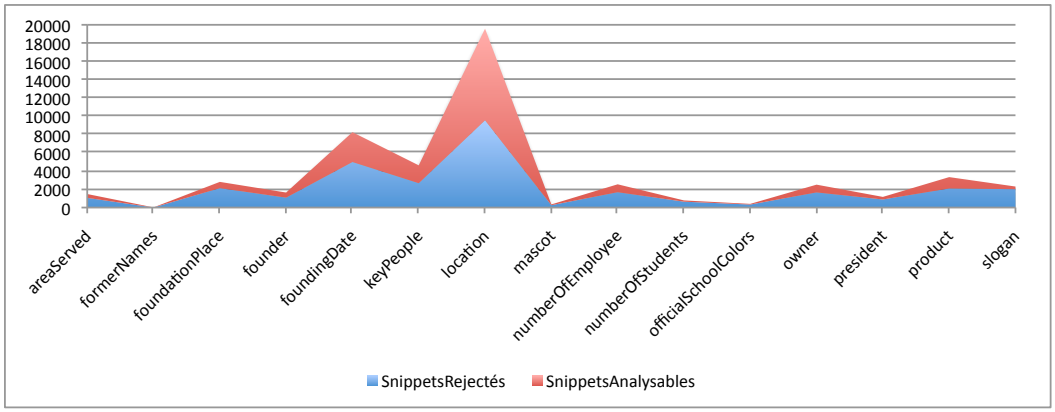


FIGURE 6.9 – Répartition des snippets sur la classe *Organisation*

Ci-dessous (cf. 6.4) les statistiques pour les deux autres classes (moyenne d'exactitude et les relations avec le plus et le moins de snippets corrects) :

Classe	M(correct)	relationMax	relationMin
Person	24%	birthDate(51,36%) partner(50,92%)	award(10%) militaryBranch(9,33%)
CreativeWork	32,03%	location(51,17%) releaseDate(43,7%)	cinematography(19,03%) openingTheme(15,15%)

TABLE 6.4 – Résumé d'évaluation pour Person et CreativeWork

Dans le cas de *Person*, la moyenne d'exactitude est plus basse car il y a une majorité de relations granulaires (par exemple : *movement*, pour *Artist*, ou *party* pour *President*) par rapport au nombre de relations génériques (seulement lieu et date de mort et naissance). Par contre, nous avons constaté que dans la plupart des relations de la classe *CreativeWork* (mais aussi d'*Organisation*) l'objet (c'est-à-dire la valeur d'attribut) est une personne : nous pouvons donc envisager, dans un deuxième temps, de travailler sur la transitivité des patrons que nous allons découvrir.

Analyse manuelle L'analyse manuelle nous sert à voir si le filtrage des snippets est effectivement utile et combien de perte d'information correcte nous avons. Cette analyse consiste à évaluer la précision et le rappel des entités correctes. Nous rappelons brièvement que :

$$P = \frac{ValeursSelectionneesCorrectes}{ValeursExtraitesTotales} \tag{6.1}$$

$$R = \frac{ValeursSelectionneesCorrectes}{ValeursCorrectesTotales} \tag{6.2}$$

$$F = \frac{2(P \times R)}{(P + R)} \tag{6.3}$$

où $ValeursSelectionneesCorrectes$ sont les snippets extraits qui sont effectivement corrects, $ValeursExtraitesTotales$ les snippets extraits au total et $ValeursCorrectesTotales$ les snippets totaux qui auraient dû être extraits.

Cette analyse est conduite sur un échantillon de 50 snippets pour 5 relations de chaque classe. Un tableau récapitulatif se trouve en 6.5. Bien que les résultats,

	Precision	Rappel	F-Mesure
Organisation	81,72%	66,09%	73,08%
Person	85,29%	54,72%	66,67%
CreativeWork	71,59%	67,74%	69,61%

Table 6.5 – Précision et Rappel du filtrage

en particulier le rappel, soient assez décevants, nous nous appuyons sur les bons taux de précision que le filtrage nous donne (la plupart des résultats donnent une précision au-delà de 90%). Cela signifie que nous écartons beaucoup de snippets dans cette étape, mais que parmi ceux que nous gardons, très peu sont mauvais. Nous renonçons donc à la quantité en faveur de la qualité. Ce choix est directement lié à la nature des données que l'on trouve sur le Web. Les données sont en effet en très grande quantité, mais d'une qualité très variable, on peut donc se permettre d'être très sélectif dans le choix des corpus. La mise en place d'outils complexes pour l'exploitation des données de mauvaise qualité n'est pas nécessaire, car le Web nous fournit de toute façon des données en quantité suffisante.

6.3 Analyse syntaxique

Une fois que nous avons collecté le corpus, nous rentrons dans le coeur du programme et abordons enfin l'utilisation des grammaires de dépendances.

L'objectif de cette étape est d'extraire une série de patrons de dépendances qui soient représentatifs d'une relation donnée.

La phase d'analyse syntaxique est divisée en deux parties : l'analyse des phrases et la création des patrons.

6.3.1 Analyse syntaxique

L'analyse syntaxique consiste à donner en entrée une phrase, et à obtenir en sortie sa structure syntaxique, sous forme d'un arbre de dépendances.

Une étape préliminaire avant l'analyse syntaxique est d'effectuer un nettoyage grossier. Celui-ci aide à éliminer les éléments qui rendent le texte du Web très bruité : l'objectif étant de fournir une phrase le plus propre possible, afin d'éviter

```
root(ROOT-0, ORLANDO-1)
nn(Inc.-7, The-4)
nn(Inc.-7, Olive-5)
nn(Inc.-7, Garden-6)
nsubj(company-25, Inc.-7)
auxpass(located-14, is-9)
cop(located-14, is-9)
det(chain-11, a-10)
attr(located-14, chain-11)
prep_of(chain-11, restaurants-13)
dep(Inc.-7, located-14)
det(States-18, the-16)
nn(States-18, United-17)
prep_in(located-14, States-18)
cop(company-25, is-20)
det(company-25, a-21)
advmod(held-23, privately-22)
amod(company-25, held-23)
amod(company-25, holding-24)
dep(ORLANDO-1, company-25)
partmod(company-25, headquartered-26)
appos(Orlando-28, Florida-29)
prep_in(headquartered-26, Orlando-28)
```

(a) Phrase "naturelle"

```
(1)
nn(G arden-3, The-1)
nn(Garden-3, Olive-2)
nsubj(chain-6, Garden-3)
cop(chain-6, is-4)
det(chain-6, a-5)
root(ROOT-0, chain-6)
prep_of(chain-6, restaurants-8)
partmod(restaurants-8, located-9)
det(States-13, the-11)
nn(States-13, United-12)
prep_in(located-9, States-13)
(2)
nn(G arden-3, The-1)
nn(Garden-3, Olive-2)
nsubj(company-9, Garden-3)
cop(company-9, is-4)
det(company-9, a-5)
advmod(held-7, privately-6)
amod(company-9, held-7)
amod(company-9, holding-8)
root(ROOT-0, company-9)
partmod(company-9, headquartered-10)
prep_in(headquartered-10, Orlando-12)
```

(b) Phrase processée par ANNIE et nettoyée

FIGURE 6.10 – Différences d'analyse syntaxique

une mauvaise analyse syntaxique. Nous avons identifié une série de caractères qui rendaient l'analyse moins efficace, et nous les avons éliminés : ils comprennent, entre autres, certaines abréviations comme *"Jr.", "Inc.", "Corp.", "Ltd", "®", "©"* mais aussi des ponctuations problématiques comme *"–", „", "*", "|"*.
La différence peut être vue dans cet exemple :

ORLANDO – The Olive Garden Inc., is a chain of restaurants located in the United States...The Olive Garden Inc. is a privately held holding company headquartered in Orlando, Florida...

Ainsi analysée, la phrase ne serait pas exploitable : *ORLANDO* – est donné comme racine, *Inc* est la tête résultante de l'entité que nous cherchons. Avec la détection de phrases de ANNIE et le nettoyage de cette étape, nous avons deux phrases, proprement analysées (cf. 6.10).

Chaque nouvelle phrase, est enfin analysée et associée à un arbre de dépendances. C'est ici qu'un nouveau filtrage a lieu : l'élimination des phrases non verbales. Si, dans l'ensemble des nœuds que nous avons, nous ne retrouvons pas de verbe, la phrase ne sera pas prise en compte. La détection du verbe se fait sur l'ensemble des nœuds (les unités lexicales, qui sont indexées de 0-la racine, à n-où n est la longueur de la phrase) : chaque unité lexicale est associée à son POS-tag et, si dans cette liste il n'y a pas de verbe, la phrase est éliminée. Les POS-tags de type verbe sont

montrés en 6.6 [Marcus 1993].

POS-tag	Categorie
VB	forme base
VBD	passé simple
VBG	gérondif ou participe présent
VBN	participé passé
VBP	présent, 3e personne non-singulier
VBZ	présent, 3e personne singulier

TABLE 6.6 – POS-tag permettant l'acceptation de la phrase

Le filtrage de cette partie nous permet d'avoir une plus grande certitude concernant la qualité de nos phrases.
La technique pour cette étape est montrée en 2.

Algorithme 2 Filtrage des Phrases

```
corpusSnippets
pour i = 0 à taille(corpusSnippets) faire
  si "..."|"." in corpusSnippets[i] alors
    snippetCourant ← corpusSnippets[i]
    snippetSplitte ← Annie.splitterSnippet(snippetCourant) ;
    pour k = 0 à taille(snippetSplitte) faire
      phraseCourante ← snippetSplitte[k]
      parsee ← Parseur.parserPhrase(phraseCourante) ;
      si "VB" in parsee.listeDeTag() alors
        extrairePatron(parsee) ;
```

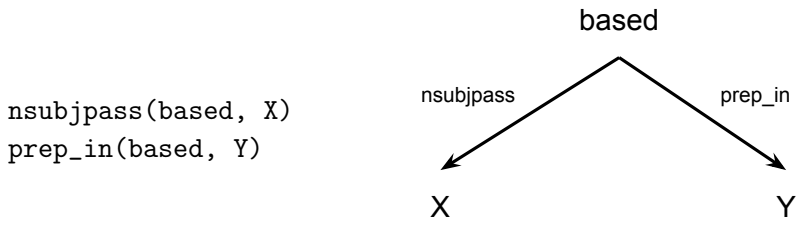
6.3.2 Extraction des Patrons

La deuxième étape consiste à extraire les patrons de dépendances.
L'algorithme d'extraction des patrons est montré dans l'algorithme 3. Chaque snip-

Algorithme 3 Extraction des patrons

```
phrase ← entite1, entite2, arbreDeDependances
tete1 ← recupererTete(entite1) ;
tete2 ← recupererTete(entite2) ;
si tete1 ET tete2 in arbreDeDependances alors
  patron ← arbreDeDependances.recupererPlusCourtChemin(tete1, tete2) ;
```

pet, ainsi que chaque phrase qui y est détectée, est associé, le long du processus, à deux entités, l'instance et la valeur de la relation que nous cherchons à analyser. Notre phrase étant déjà transformée en arbre de dépendances, il faut maintenant



(a) version textuelle (b) version arborescente

FIGURE 6.11 – Exemple de patron pour la relation *location*

trouver quel est le chemin le plus court dans l'arbre reliant les deux entités.
Comme nous avons vu, les entités (des syntagmes nominaux) sont très souvent composées par plusieurs *tokens* (une séquence de symboles) : la plupart sont des noms propres (NNP) comme *"Olive Garden"*, *"United States"*, mais pas seulement : *"University of Maryland"*. Plus l'entité nommée est complexe, plus il est difficile de manipuler et faire des opérations sur l'arbre (*recupererTete(entité)*) en utilisant l'entité nommée complète. Pour allons donc détecter la tête de l'entité dans l'arbre : nous cherchons le nœud le plus proche de la racine, pour ne travailler qu'avec ce *token*-là. Dans la phrase (2) la racine est `company-9` : même à l'œil nu, nous remarquons que `Garden-3` (distance=1) est plus proche de la racine que `Olive-2` (distance=2) et `The-1` (distance=2). Ce processus permet en particulier de retrouver les entités dans une phrase même si celle-ci ne contient pas l'entité dans son intégralité, chose qui arrive souvent avec des noms de personne :

Entity1= Natalie Portman
Entity2= Harvard University
Natalie Portman had "super depressed" moments while attending Harvard Portman graduated from the prestigious Harvard University with a degree in Psychology ...

Le "plus court chemin" entre l'entité1 et l'entité2 est calculé comme l'ensemble minimal de nœuds entre les deux têtes d'entités : à chaque nœud, on ajoute une dépendance (un triplet) au chemin. La dernière étape consiste à effacer les index des nœuds (car ils ne sont pas utiles pour nous) et à remplacer la tête de l'entité (*Garden, Orlando*) par un X et un Y, pour faciliter l'extraction des nouvelles entités dans l'étape suivante. Nous avons ainsi constitué un patron. Si aucun chemin n'est trouvé, le patron est "nul".

Pour la phrase *Olive Garden is based in Orlando*, il en résulte finalement le patron de dépendances de la Fig. 6.11.

Pour chaque relation, nous collectons à la fin de cette étape un ensemble de règles (c'est-à-dire, de patrons) qui sont représentatives de cette relation : plus simplement nous collectons tous les ensembles de dépendances qui lient une entité1 à une entité2

à l'intérieur d'une phrase. Idéalement, ces patrons permettent d'effectuer des nouvelles recherches sur le Web et, appliquées à des nouveaux arbres de dépendances, permettent d'extraire de nouvelles entités. Cela se traduit, en réalité, en une tâche bien plus compliquée, qui nous a amenés à proposer des méthodes par apprentissage automatique par la suite.

6.3.3 Problèmes et Solutions

Les erreurs de l'analyseur syntaxique Nous l'avons vu dans l'état de l'art, l'analyse syntaxique est basée sur des méthodes probabilistes et statistiques. Cela signifie que, plus le corpus appris par le modèle est grand, meilleurs seront ses résultats. Cependant, la nature des textes du corpus d'apprentissage peut ne pas être la même que celle du corpus que nous soumettons à l'analyseur : nous ne pouvons donc pas compter sur une précision d'analyse de 100%. Le corpus Web, étant extrêmement bruité, n'est pas un candidat idéal à soumettre à l'analyseur : seulement les phrases avec des structures simples seront analysées correctement. De plus, le traitement des entités nommées, celles dont nous nous occupons le plus, n'est pas simple pour un analyseur probabiliste (moins l'entité est connue, moins il y aura de chances que l'entité soit analysée correctement).

Même si l'analyseur est performant, nous devons donc compter aussi une marge d'erreur de la part du système.
En général, les erreurs que nous avons détectées sont de plusieurs types :

- impossibilité de détecter la dépendance : une dépendance générique *dep* est attribuée.

  Founder Arlene Klasky founded Csupo while taking care of her 1-month...

  ```
  ccomp(Y/VB, founded/CSD)
  dep(founded/VBD,X/NNP)
  ```

- mauvais POS-tagging : pour le même exemple cité ci-dessus, la relation grammaticale `ccomp` (complément clausal) est générée entre l'entité2 (=Y) et *founded*, car *Klasky* a été reconnu en tant que verbe.

- mauvais découpage des phrases

  Csupo has location in Beverly Hills, CA. Active officers include Arlene Klasky and Gabor...

  ```
  dep(X/NNP, has/VBZ)
  dobj(has/VBZ,location/NN)
  rcmod(location/NN, include/VBP)
  dobj(include/VBP,X/NNP)
  ```

  Les deux phrases ont été reconnues comme une seule, probablement à cause de *CA.*, reconnu comme une abréviation (et donc, à ne pas couper).

- difficulté à gérer la coordination

Mike Ilitch is the founder, Chairman, President, and Chief Executive officer of the Olive Garden

```
nsubj(founder/NN, Y/NNP)
appos(founder/NN, Officer/NNP)
prep_of(Officer/NNP, X/NN)
```

- Difficulté à gérer les syntagmes nominaux complexes

  [Founder and CEO of the Olive Garden and CIA alumnus Mike Ilitch]$_{GN}$ has the secret to a successful career...

  ```
  conj_and(X/NNP, Y/NNP)
  ```

  En particulier, une différence remarquable au niveau de l'efficacité de l'analyse syntaxique se voit si le syntagme nominal coïncide avec la dislocation à gauche. Nous notons la difference d'analyse entre :

  [Founder of the Olive Garden Michael Ilitch]$_{GN}$ has the secret to a successful career...

  ```
  prep_of(Founder/NNP, X/NNP)
  appos(X/NNP, Y/NNP)
  ```

  contre

  [Founder of Olive Garden]$_{GN}$, Michael Ilitch has the secret to a successful career...

  ```
  appos(Y/NNP, founder/NNP)
  prep_of(founder/NNP, X/NNP)
  ```

- Problèmes de reconnaissances d'entités nommées.

  The restaurant was founded by Michael Ilitch in 1971 ...

  ```
  det(restaurant/NN, X/DT)
  nsubjpass(founded/VBZ, restaurant/NN)
  agent(founded/VBN,Y/NNP)
  ```

  À première vue, il n'y a pas d'entité1 à chercher. Cependant, le label de l'entité1 est, dans ce cas, *The Olive Garden*, qui inclut aussi *The*. Comme la recherche du nœud le plus proche de la racine est basée sur les *tokens* d'une entité, le système reconnaît *The* comme le seul élément de l'entité1 présent dans la phrase, en créant ce patron. C'est un problème bien connu dans la reconnaissance des entités nommées, même en français : comment distinguer, en effet, *Le [Café de France]$_{EN}$ reçoit tous les jours plus de 500 personnes* de *[Le Café de France]$_{EN}$ reçoit tous les jours plus de 500 personnes* ?

Solution Nous avons donc mis en place une phase de pré-traitement, afin de nettoyer les phrases au mieux. Idéalement, les phrases à analyser doivent être de la forme $S = GN + GV$. Dans les phrases où ce n'est pas le cas, nous constatons un taux d'erreur supérieur.
Pour ce faire, nous avons effectué un nettoyage grossier avant d'analyser les phrases. Ensuite, nous avons filtré les phrases non verbales (une méthode a été développée exprès). Si la phrase ne contient que des noms propres, celle-ci est écartée. Ce processus ne permet pas de récupérer des syntagmes nominaux comme *Darden Restaurants' Olive Garden's*, ou *Darden Restaurants, owner of Olive Garden*, à moins qu'ils se trouvent dans une phrase avec un verbe. Si d'un côté nous pouvons imaginer une possible perte de patrons candidats, nous devons nous rappeler que (i) l'analyse en dépendances est beaucoup plus efficace lors de la présence d'un verbe fini (une phrase complète) et (ii) nous étudions les dépendances pour analyser les relations, qui sont, dans la plupart des cas, exprimées par des verbes. La perfection étant impossible, il est nécessaire de mettre une limite à l'extraction : la nôtre est de nous concentrer sur des patrons contenant des verbes.
Une autre possible voie d'exploration, pour effectuer un filtrage en plus et réduire les erreurs, pourrait être d'exploiter la structure syntaxique en constituants de la phrase pour ne retenir que celles avec des structures bien définies (comme $S = GN(NNP*) + GV(V + (GP|GN))$).

6.3.4 Outils

Stanford Parser L'outil choisi pour la construction des arbres de dépendances a été le Stanford Parser [10]. Le Stanford Parser est implémenté en Java, et fournit plusieurs analyseurs, un CFG probabiliste (PCFG) lexicalisé, un analyseur en dependances et un simple analyseur en constituants (non-lexicalisé). L'analyseur utilise les contenus structuraux non-lexicalisés et les annotations linguistiques pour l'optimisation de l'analyse syntaxique. Il est disponible pour l'anglais (entraîné sur la section *Wall Street Journal* du Penn TreeBank), l'allemand (le NEGRA corpus), l'arabe (Penn TreeBank arabe) et le Chinois (Chinese TreeBank). Etant *open source*, il est possibile d'entraîner les modèles sur d'autres langues.
Nous utilisons les sorties en dépendances pour notre travail, mais il est possible aussi d'exploiter celles en constituants.
Le choix de cet outil repose principalement sur sa conception des dépendances et des relations grammaticales entre les mots, qui s'avère être particulièrement adaptée pour l'extraction d'information à partir des textes. Pour comprendre cela, il faut voir comment cet outil fonctionne. Les modèles de POS-tag sont entraînés pour calculer la probabilité $P(tag|w_i)$ (c'est-à-dire, la probabilité qu'un mot ait un certain tag) en utilisant les méthodes statistiques de maximum de vraisemblance [Aldrich 1997]. L'algorithme CKY est ensuite utilisé pour la construction des arbres syntaxiques. La construction des arbres de dépendances vient après cette étape : on utilise des

10. http ://nlp.stanford.edu :8080/parser/index.jsp

patrons en constituants, écrits à la main, appliqués sur les arbres de constituants. Dans un premier temps (extraction de dépendances), il faut chercher les têtes à l'intérieur d'un même syntagme. Ensuite (détection de dépendances), chaque tête est liée avec son dépendant et ce "lien" est étiqueté avec une relation grammaticale. Ces relations grammaticales sont au nombre de 56, organisées en hiérarchie de la relation la plus générique (*dep*) à la plus spécifique (cf. 6.12). Pour détecter automatiquement une relation dans une nouvelle phrase, chaque patron est appliqué à l'arbre syntaxique et le patron correspondant à la relation grammaticale la plus spécifique est choisi. Chaque dépendance est finalement représentée comme un triplet, GRAMMREL(HEAD,DEPENDENT).

Les dépendances de Stanford ont un deuxième avantage : la possibilité de "collap-

```
dep
  subj - subject
    nsubj - nominal subject
       nsubjpass - passive nominal subject
    csubj - clausal subject
       csubjpass - passive clausal subject
```

FIGURE 6.12 – Exemple de hiérarchie pour la relation sujet

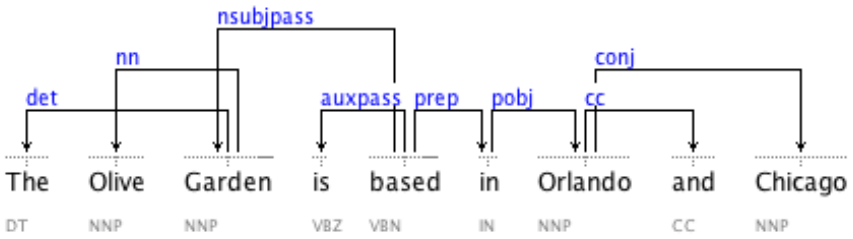
ser" les arbres, afin que la représentation soit proche de la sémantique de la phrase (selon la tradition de Tesnière, donc). Le "collapsing" est effectué au niveau des prépositions et des conjonctions, qui deviennent des relations grammaticales, en privilégiant donc les mots "sémantiquement pleins"(cf. 6.13b). Une troisième possibilité de représentation (cf. 6.13c) permet de gérer aussi les cas de coordination, en liant les têtes aux deux dépendants.

C'est donc ce troisième type de représentation qui a été considéré finalement, car ils est le plus adapté pour la définition des patrons pour chaque relation ontologique. À noter que, dans les cas de coordination, nous ne traitons pas les arbres, mais plutôt des graphes orientés. En particulier, l'acyclicité n'est pas respectée.

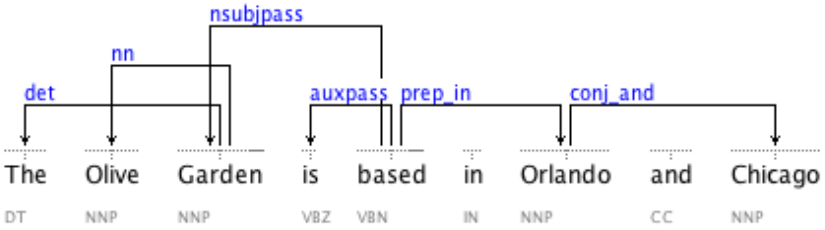
Implémentation d'une grammaire de dépendances L'implémentation du Stanford Parser contient une série de classes [11] permettant de manipuler un arbre de dépendances comme un objet. Les nœuds, les relations grammaticales et les arbres sont construits comme des classes Java, et contiennent des méthodes pour explorer et effectuer une série d'opérations sur l'arbre. L'extraction des patrons, ainsi que la détection des nouvelles entités, se base sur la représentation objet de l'arbre de dépendances (*SemanticGraph*) et des deux mots que nous cherchons (*IndexedWord*, composée par une étiquette, un index et un POS-tag). Trois types d'opération sont nécessaires pour cette étape :

- détecter les *tokens* d'une entité dans le graphe : `getNodesByWord(Word)` ;

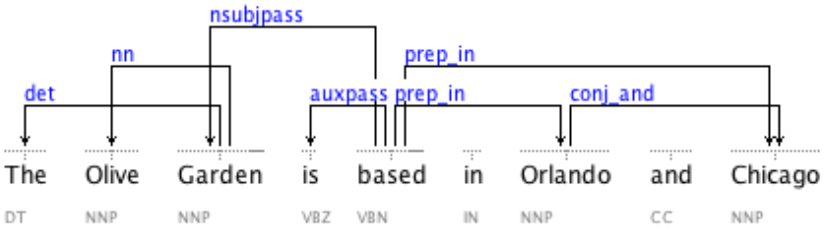
11. http ://tides.umiacs.umd.edu/webtrec/stanfordParser/javadoc/



(a) Représentation Basique



(b) Arbre collapsé



(c) Conjonctions procéssées

FIGURE 6.13 – Représentations possibles de la même phrase avec le Stanford Parser

- détecter le chemin entre deux entités dans un graphe : `getShortestPath(Noeud1, Noeud2)` ;
- trouver les fils d'un nœud, et reconstruire l'entité : `getDescendents(Word)`.

6.3.5 Évaluation

Cette analyse nous permet d'avoir une vision sur le filtrage des patrons. En regardant les graphiques nous avons tout de suite une idée sur la quantité de bruit que nous avons accumulé jusqu'à cette étape. Sur la totalité des phrases analysées, une partie seulement a permis l'extraction d'un patron. Lors de chaque extraction,

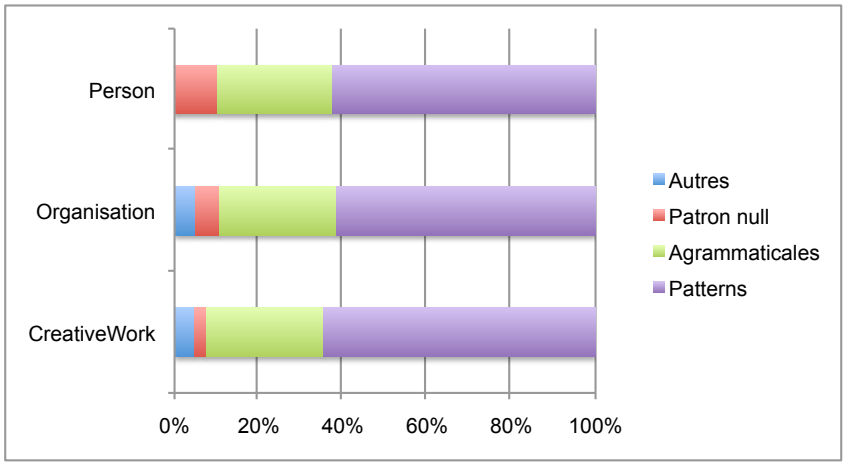


FIGURE 6.14 – Résultats de l'analyse syntaxique pour les trois classes

nous avons compté le nombre de phrases écartées car les patrons étaient nuls, le nombre de phrases agrammaticales et de phrases considérées nulles pour d'autres raisons (trop courtes, par exemple). La figure 6.14 montre les pourcentages pour les trois classes.

Contrairement à ce que nous pouvions imaginer, plus de 60% des phrases permettent d'extraire un patron. Une grande quantité de phrases sont agrammaticales (en vert) : ces chiffres sont assez représentatifs du fait que les pages Web sont bruitées. Une autre portion de phrases est encore perdue lorsque le patron est nul (aucun chemin n'est trouvé entre les deux entités). Ce sont principalement des phrases ayant échappé au premier filtrage. Une remarque que nous pouvons faire est que les phrases agrammaticales sont beaucoup moins nombreuses pour *CreativeWork* que pour *Person*. Ceci s'explique probablement par le fait que, lorsque nous cherchons une information sur une personne, en particulier une personne célèbre, certaines des informations sortent des sites les plus bruités (nous avons déjà cité Facebook, mais aussi IMDB, LinkedIn...).

Pour avoir une idée de la distribution d'erreurs pour chaque attribut, nous avons effectué une évaluation spécifique sur *Organisation*(cf. 6.15).

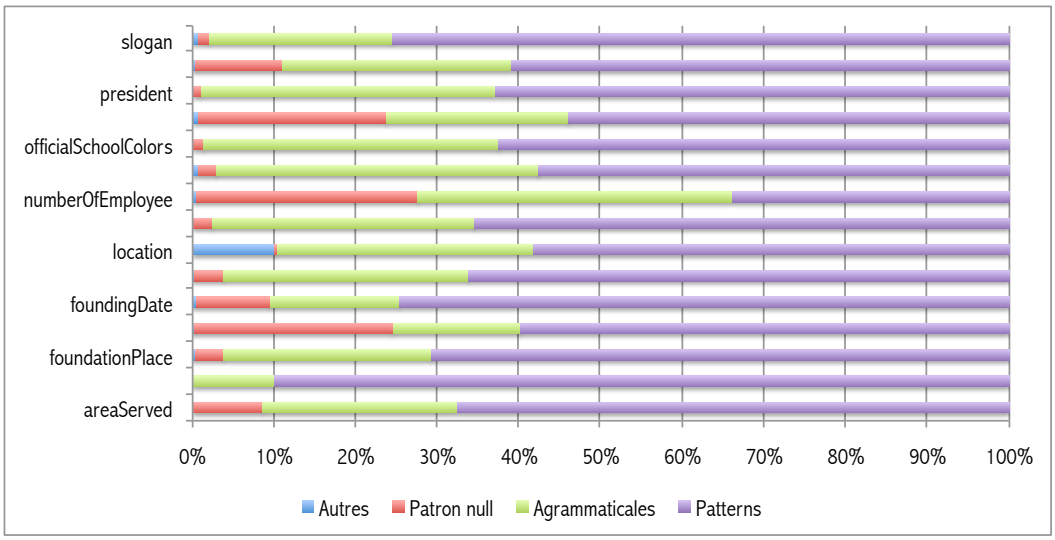


FIGURE 6.15 – Résultats de l'analyse syntaxique pour *Organisation*

Chapitre 7

Analyse linguistique des résultats

Sommaire

Nous allons maintenant voir les résultats que nous avons obtenus, d'un point de vue linguistique.

7.1 Filtrage préliminaire

La première étape a été une analyse globale des résultats : nous avons remarqué que certains patrons, qui étaient les plus fréquents pour une certaine relation r, l'étaient aussi pour beaucoup d'autres. En général, ce sont des patrons à une seule dépendance, et pas suffisamment spécifiques pour permettre l'extraction de nouvelles entités. Par exemple :

$$\text{poss(X,Y)} \rightarrow \text{"}X\text{'s } Y\text{"}$$

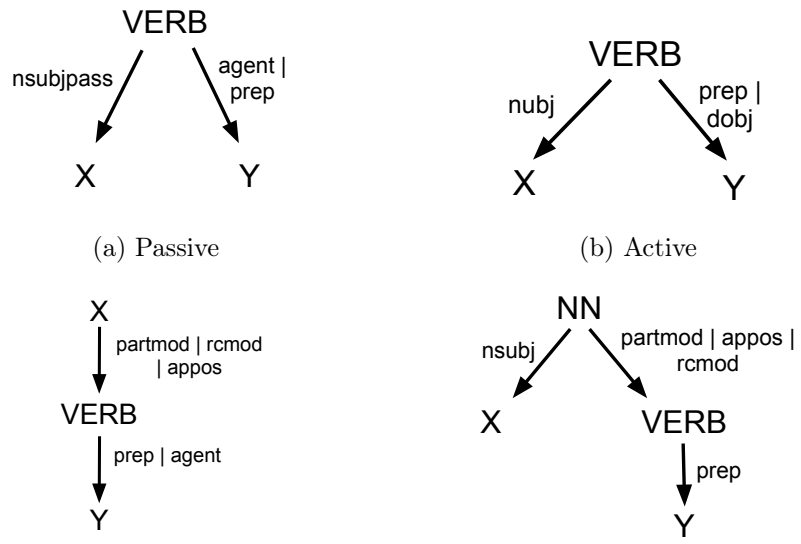
peuvent être un auteur ("*Kubrick's Clockwork Orange*"), une filière d'organisation ("*Darden's Olive Garden*"). Même chose pour les modifieurs de syntagmes nominaux (nn, "*noun compound modifier*" ou prep_of) :

$$\text{nn(X,Y)} \rightarrow \text{"}X\ Y\text{"}$$
("*Darden Olive Garden*", "*Kubrick' Clockwork Orange*")

$$\text{prep\_of(X,Y)} \rightarrow \text{"}Y\ of\ X\text{"}$$
("*Olive Garden of Darden Restaurants*", "*Clockwork Orange of Kubrick*")

Ou encore :

$$\text{prep\_in(X,Y)} \rightarrow \text{"}X\ in\ Y\text{"}$$



(a) Passive

(b) Active

(c) Complément nominal/1

(d) Complément nominal/2

FIGURE 7.1 – Structure passive et structure avec préposition

peut exprimer une relation temporelle, bien que moins fréquente, ("*Olive Garden in 1972*") ou spatiale ("*Olive Garden in Orlando*"). Bien que les POS-tags de X et Y peuvent aider à distinguer les relations (pour une relation temporelle, Y devrait être étiqueté comme "CD", *cardinal number*), ce patron à une seule dépendance ne reste toujours pas suffisant. Nous avons appliqué une mesure *TF-IDF* pour les filtrer : les patrons les plus fréquents dans un document, mais également dans les autres documents, c'est-à-dire non spécifiques, ont été éliminés.

7.2 Les structures de dépendances

Avant de présenter les patrons que nous avons trouvés pour chaque relation, nous allons donner une vue globale de leur structure.

Nous nous sommes en effet rendue compte que les relations grammaticales des patrons étaient globalement toujours les mêmes. Bien évidemment ceci est dû au fait que les structures les plus récurrentes sont aussi les plus faciles à détecter car les plus simples d'un point de vue syntaxique. On remarquera en effet qu'il s'agit de structures à deux ou trois dépendances, donc très simples.
Nous avons détecté 4 groupes en particulier (cf. 7.1).

La structure 7.1a fait référence aux phrases passives : beaucoup de patrons sont en effet de la forme "X patient de Y". Une variante se retrouve avec des prépositions, lorsqu'il s'agit de patrons liés à une entité de temps ou d'espace (qui ne peut donc pas être l'agent de la situation mais est lié au verbe par préposition). Par exemple :

(1a) nsubjpass(founded,X) agent(founded,Y)
→ "*X is founded by Y*"
(1b) nsubjpass(founded,X) prep_in(founded,Y)
→ "*X is founded in Y*"

La structure 7.1b est, au contraire, la structure d'une phrase active simple, où X est l'agent. Dans ce cas aussi nous avons la variante avec préposition :

(2a) nsubj(attended,X) dobj(attended,Y)
→ "*X attended Y*"
(2b) nsubj(studied,X) prep_at(studied,Y)
→ "*X studied at Y*"

Les structures 7.1c et 7.1d sont très similaires, et concernent les dépendances étiquetées comme "appos" (apposition), "partmod" (modifieur participial), "rcmod" (modifieur de la phrase relative) et, pour les dates, "tmod" (modifieur temporel). Ces syntagmes sont dépendants d'un nom mais comprennent les mêmes composants que les autres structures, un verbe, une préposition, un complément direct. Des exemples sont :

(3a) nsubj(university,X) partmod(university,based) prep_in(based,Y)
"*X is a university based in Y*"
(3b) partmod(X,based) prep_in(based,Y)
"*X, which is based in Y*"

Ces structures sont interchangeables, dans le sens où nous pourrions utiliser deux patrons différents par structure syntaxique (mais contenant les mêmes lemmes !) et récupérer les mêmes entités (cf. 3a et 3b). Dans les sections suivantes, il faudra donc imaginer que plusieurs structures, correspondant aux 4 groupes montrés ci-dessus, existent pour chaque patron donné en exemple.

À noter finalement que ces deux dernières constructions sont extrêmement importantes car elles nous permettent d'extraire des patrons également depuis des phrases plus hétérogènes, mais dont la signification reste la même, qu'il serait impossible d'extraire avec des patrons lexico-syntaxiques. Par exemple, nous pourrons extraire le patron 3b de :

"*X, based in Y*"
"*Based in Y, X...*"
"*X, that is based in Y...,*"

7.2.1 La relation *Location*

Dans les relations que nous traitons il y en a des génériques, qui se réfèrent à plus d'un concept, et des plus spécifiques à un seul concept. Le cas d'une relation générique est celui de *location* (c'est une relation qui concerne tout établissement, quel qu'il soit, mais aussi le lieu d'un film ou d'une série). Nous avons retrouvé 230

patrons (sur 20.000 phrases, il n'y en avait que 230 qui étaient réellement exploitables). La répartition de la fréquence des patrons pour *location* est la suivante : Il y a bien une répartition respectant la loi de Zipf, où peu de patrons sont très

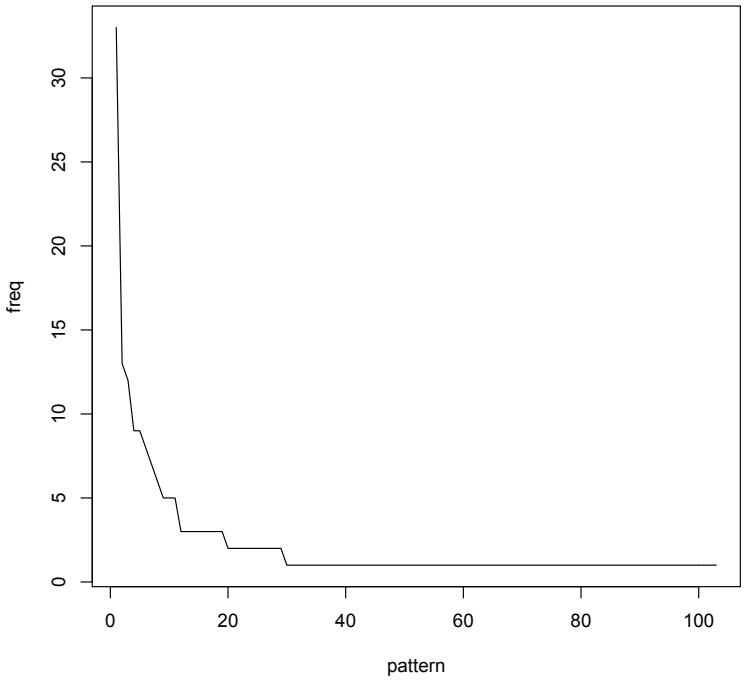


FIGURE 7.2 – Répartition des fréquences des patrons

fréquents, et beaucoup de patrons ont une fréquence de 1. La distribution n'est pas très uniforme car nous notons des sauts très nets. Alors que les patrons les plus fréquents sont ceux que nous nous attendions :

(1a) nsubjpass(located,X) prep_in(located,Y)
(1b) nsubjpass(based,X) prep_in(based,Y)
(1c) nsubjpass(situated,X) prep_in(situated,Y)
→ "*X is situated/located/based in Y*"

nous trouvons des patrons bien plus intéressants, qui nous permettent de spécifier une classe plutôt qu'une autre :

(2a) nsubj(operates,X) prep_in(operates,Y)
(2b) nsubjpass(headquartered,X) prep_in(headquartered,Y)
(2c) nsubj(organisation,X) partmod(organisation,located) prep_in(located,Y)
⇒ concept : Company

Ce sont des patrons qui se réfèrent plus à une société : "*Olive Garden operates in Florida*", "*Olive Garden is headquartered in Florida*" ou "*Olive Garden is an organisation located in Florida*".
D'autres concepts sont récupérables en utilisant des sous-patrons 1 :

(3a) partmod(clinic,located) prep_in(located,Y) nsubj(clinic,X)
→ "*X is a clinic located in Y*"
⇒ concept : Hospital

(3b) partmod(institution,Located) prep_in(Located,Y) nsubj(institution,X)
(3c) nsubj(university,X) partmod(university,based) prep_in(based,Y)
(3d) nsubj(college,X) partmod(college,located) prep_in(located,Y)
→ "*X is a $w (located/based) in Y*"
$w = { college, university, institution}
⇒ concept : University

Des patrons plus complexes, et par conséquent plus spécifiques, sont aussi présents :

(4a) nsubj(provider,X) prep_of(provider,care) prep_in(care,Y)
⇒ concept : Hospital

(4b) nsubj(institution,X) partmod(institution,offering) dobj(offering,education)
partmod(education,located) prep_in(located,Y)
(4c) nsubj(institution,X) prep_of(institution,learning) partmod(learning,located)
prep_in(located,Y)
⇒ concept : University
→ "*X is an institution (of learning/offering education) located in Y*".

La localisation géographique n'est pas seulement exprimée par la préposition "in" :

(5a) nsubj(restaurant,X) prep_with(restaurant,locations)
prep_across(locations,Y)
(5b) nsubj(chain,X) prep_of(chain,pizzerias) partmod(pizzerias,serving)
prep_throughout(serving,Y)
→ "*X is a restaurant with locations across Y*"
→ "*X is a chain of pizzerias serving throughout Y*"
⇒ concept : Restaurant

(5c) nsubj(company,X) partmod(company,headquartered)
prep_in(headquartered,suburb) prep_of(suburb,Y) (5d) poss(headquarters,X)
prep_near(headquarters,Y)
(5e) nn(Situated,X) prep_in(Situated,heart) prep_of(heart,countryside)
appos(countryside,Y)
→ "*X is a company headquartered in the suburbs of Y*"
→ "*X's headquarters near Y*"
→ "*X is situated in the heart of the Y countryside*"

Finalement, cette relation n'est pas spécifique qu'aux organisations :

6a. nsubj(film,X) partmod(film,released) prep_in(released,Y)
6b. nsubjpass(released,X) prep_in(released,Y)
→ "*X is (a film)* released in Y*" ⇒ concept : Movie, Sitcom
où Y n'est pas une date mais un lieu : *"A Clockwork Orange" is a film released in the UK.*

7.2.2 Des relations aux patrons classiques : *foundingDate, birthDate* et *branchOf* ...

Certaines relations, en particulier celles qui sont liées à un concept de temporalité (càd liées à une date), n'ont pas trop de diversité de patrons : les lemmes utilisés restent très standard. C'est le cas de la relation *birthDate* (concernant les personnes), où tous les patrons (290) contiennent le mot "*born*". La plupart ont la forme :

(1a) nsubjpass(born,X) prep_in(born,Y)
→ "*X is born in Y*"

Certaines variantes comprenant le mois ou le jour de la semaine sont aussi présentes :

(1b) nsubjpass(born,X) prep_on(born,$month) num($month,Y)
(1c) nsubjpass(born,X) prep_on(born,$day) appos($day,$month) num($month,Y)
→ "*X is born on ($day+) $month Y*"
$1 \leq$ $day ≤ 31 | $day={Monday, Tuesday...}
$month = { January, February, March...}

Il y a une remarque intéressante à faire ici concernant le type d'information que nous extrayons. Jusqu'à présent, nous ne nous sommes basée que sur la syntaxe de la phrase : autrement dit, nous avons essayé de démontrer qu'il existent des sous-structures syntaxiques communes entre phrases qui expriment la même (relation) sémantique. Il nous arrive, ce faisant, d'obtenir des patrons qui s'avèrent être des variantes "sémantiquement" plus spécifiques d'un patron plus général. Par exemple :

(2a) partmod(X,born) tmod(born,June)num(June,Y)
(2b) partmod(X,born) tmod(born,March)num(March,Y)
→ "*X, born on March Y...*", "*X, born on June Y...*"

(où *tmod* est un modifieur temporel entre *born* et *Y*). Ce qui nous intéresse, au contraire, est que ces deux patrons soient rapprochés :

(2c) partmod(X,born) tmod(born,$month) num($month,Y)
→ "*X, born on $month Y...*"

Pour généraliser 2a et 2b il faudrait tenir compte d'une composante sémantique : *$month* est un ensemble qui contient exactement 12 éléments, les mois.

Une solution pour ce faire serait d'utiliser la programmation logique inductive (PLI) : celle-ci permettrait de généraliser le patron pour n'importe quel mois, mais

en imposant que le même mois apparaisse dans les deux emplacements, comme nous avons fait manuellement dans 1c ou 2c.

Les patrons de *foundingDate* (pour *Organisation*) sont légèrement plus variés en termes de structures et de formes lexicales. Sur 772 patrons, les groupes les plus fréquents sont :

(3a :3h) nsubjpass($w,X) prep_in($w,Y)
$w={founded, launched, established, opened, purchased, pioneered, made, created}
→ "*X was $w in Y*"

De même que dans le cas de *location*, ce groupe permet de repérer des sous-concepts de *Organisation* :

(4a) nn(restaurant,X) nsubj(opened,restaurant) prep_in(opened,Y)
→ "*the X restaurant opened in Y*"
⇒ *Restaurant*
(4b) nn(school,X) nsubj(opened,school) prep_in(opened,Y)
→ "*X is school a school opened in Y*"
⇒ $University$

Un dernier ensemble de patrons présente une variation de termes :

(5a) prep_in(announced,Y) dobj(announced,creation)prep_of(creation,X)
→ "*... announced in Y the creation of X*"
(5b) prep_since($w,Y) nsubj($w,X)
$w={operates, serves, opened}
→ "*X $w since Y*"

Une autre relation commune à toutes les classes dans *Organisation* est *branchOf*. Dans ce cas aussi nous avons retrouvé des patrons "classiques" :

(6a :6f) "nsubj($w,X) prep_of($w,Y) "
$={division, branch, part, subsidiary, trademarks, affiliate}
→ "*X is $w of Y*"
(6g :6i) "nsubjpass($w,X) agent($w,Y) "
$={founded, owned, held}
→ "*X is $w by Y*"

D'autres patrons possèdent une sémantique plus compliquée, comprenant un sens de "temporalité" (un événement passé) :

(7a) rcmod(X,acquired) agent(acquired,Y)
→ "*X, acquired by Y*"
(7b) nsubj(integrating,Y) dobj(integrating,acquisition) prep_of(acquisition,X)
→ "*Y is integrating the acquisition of X*"
(7c) nsubj(acquired,Y) dobj(acquired,interest) prep_in(interest,X)
→ "*Y acquired interests in X*"
(7d) nsubj(bought,Y) dobj(bought,X)
→ "*Y bought X*"

7.2.3 ...aux relations aux patrons surprenants

Alors que nous étions sûr de récupérer certaines relations, puisqu'elles étaient extrêmement fréquentes et déjà connues dans la littérature, nous avons obtenu des résultats satisfaisants aussi pour des relations très spécifiques et pour lesquelles nous n'avions pas beaucoup de données. C'est ici que nous voyons un autre potentiel des grammaires de dépendances : même pour des patrons plus compliqués à extraire, nous avons eu des bons résultats.

Un de ces cas par exemple est la relation *numberOfStudents*. Les résultats de la recherche Web pour une entité numérique ne sont pas très précis : la plupart des résultats sont en effet, des phrases à écarter. Malgré un grand nombre de patrons erronés pour cette relation, nous avons obtenu 50 patrons valides, parmi lesquels :

(1a) nsubj(saw,X) dobj(saw,enrollment) num(students,Y)
prep_of(enrollment,students)
→ "*X saw an enrollment of Y students*"
(1b) nsubj(has,X)dobj(has,population)prep_of(population,Y)
→ "*X has a population of Y*"

les plus fréquents avec respectivement 6 et 5 occurrences, plus une série d'autres également très satisfaisants :

(1c) num(students,Y) partmod(students,enrolled) prep_in(enrolled,X)
→ "*Y students enrolled in X*"
(1d) num(students,Y) nsubj(attending,students) dobj(attending,X)
→ "*Y students are attending X*"

ou plus classiques comme

(1e) nsubj(has,X) dobj(has,students) num(students,Y)
→ "*X has Y students*"
(1f) num(students,Y) nsubj(study,students) prep_at(study,X)
→ "*Y students study at X*"

Nous pouvons donc constater que les termes de cette relation sont "*enroll, attend, study*".

De même, la relation *numberOfEpisodes* (21 patrons)

(2a) rcmod(X,aired) num(episodes,Y) prep_for(aired,episodes)
→ "*X, aired for Y episodes...*"
(2b) num(episodes,Y) dobj(had,episodes) nsubj(had,X)
→ "*X had Y episodes*"
(2c) nsubj(ran,X) num(episodes,Y) dobj(ran,episodes)
→ "*X ran Y episodes*"

Ces patrons peuvent être suffisamment spécifiques pour nous permettre de retrouver des entités et valeurs dont nous pouvons être sûrs qu'ils appartiennent à la bonne catégorie (université, série).

Nous avons quelques exemples de patrons corrects pour *Person* aussi :

(3a) (graduated,X) prep_from(graduated,Y)
(3b) nsubj($w,X) prep_at($w,Y)
→ "*X $w (from | at) Y*"
$w = {studied, enroll}
⇒ almaMater

(4a) rcmod(X,died) prep_of(died,Y)
→ "*X, who died of Y, ...*"
(4b) nsubj(passed,X) prep_from(passed,Y)
→ "*X passed from Y*"
(4c) nsubjpass(killed,X) from_from(killed,X)
→ "*X was killed from Y*"
⇒ deathCause

et aussi, comme variante de 4b :

(4b1) nsubj(passed,X), prep_after(passed,suffering) prep_from(suffering,Y)
→ "*X passed after suffering from Y*"

et encore

(5a) partmod(X,joined) dobj(joined,Y)
→ "*X, who joined Y, ...*"
(5b) nsubjpass(elected,X) nn(Member,Y) prep_as(elected,Member)
→ "*X was elected as Y Member*"
(5c) nn(party,Y) dobj(led,party) nsubj(led,X)
→ "*X led the Y party*"
⇒ Party

7.2.4 et pour les *CreativeWorks* ?

Les relations concernant les artefacts se sont révélées très similaires les unes aux autres. Les structures syntaxiques extraites étaient uniformes, et chaque relation était représentée par quelques termes (peu nombreux). Ces termes ne sont, en général, que des verbes. Un tableau de résumé est présenté ci-dessous 7.1. Le seul cas qui diffère légèrement est celui de *author* qui, au moins, permet de différencier entre les concepts *Movie* et *Sitcom* :

(1a) nsubj(film,X) partmod(film,written) agent(written,Y)
→ "*X is a film written by Y*"
⇒ Movie
(1b) nsubj(sitcom,X) partmod(sitcom,written) agent(written,Y)

Relation	Termes
director	directed
creator	created
author	written co-written
distributor	published released distributed
editor	edited
company	aired produced

TABLE 7.1 – Résumés des patrons pour *CreativeWorks*

(1c) nsubj(series,X) partmod(series,created) agent(created,Y)
"*X is a sitcom written by*", "*X is a series created by Y*"
⇒ Movie

ou un peu plus spécifique :

(1d) nsubj(writes,Y) dobj(writes,third-season) prep_of(third-season,X)
→ "*Y writes the third-season of X*"

C'est une distinction assez fictive, que nous avons établie : souvent les termes se retrouvent dans une relation au lieu d'une autre ("*directed by*" dans la relation *author* au lieu de *director*, *distributed by* dans la relation *company* au lieu de *distributor* etc.) Beaucoup de confusion est créée en effet dans ces relations, probablement car aucune différence n'est faite au niveau des ontologies entre ces attributs (*creator, author* et *director* peuvent faire partie d'une seule super-relation, ainsi que *distributor, producer*, et *company*), et on retrouve les mêmes informations sous plusieurs relations.

7.2.5 Un cas d'échec

Une relation pour laquelle nous n'avons pas récupéré de patron du tout a été *musicComposer*, pour les artefacts. Nous avons cherché la raison à cela. La relation ne se réfère pas au concept *Song* (que nous n'avons pas traité), mais bien à *Sitcom*. En général, les phrases pour cette relation ressemblent plutôt à des extraits de pages, en format structuré, plutôt qu'à des contenus textuels.

(INSTANCE) *Anywhere But Here* (VALEUR)*Danny Elfman*
(SNIPPET) *Danny Elfman - Anywhere But Here Score Suite mp3 Just who's the Mom here ?*

L'analyse syntaxique n'est donc pas réussie, ou très mal réussie.

Chapitre 8

Affinement des patrons

Sommaire

Dans le chapitre précédent nous avons montré les patrons que nous avions extraits. Inconsciemment, pour les montrer et les expliquer, nous les avons regroupés par similarité. En effet, ce processus permet d'avoir plus d'ordre et de regrouper des ensembles de patrons similaires. De plus, le filtrage des "bons" patrons a été fait à la main, chose qui peut prendre du temps. Nous nous sommes donc interrogée sur la possibilité d'effectuer une série d'étapes automatiques supplémentaires pour l'affinement des patrons de dépendances, avant d'effectuer l'extraction des nouvelles entités.

Notre proposition consiste à utiliser l'apprentissage automatique. La raison qui nous a poussée à explorer les voies de la fouille de textes et de données est principalement liée à notre volonté d'améliorer un système idéalement très performant mais qui, étant donné le grand nombre de problèmes survenus jusqu'à présent, rend les résultats attendus accompagnés d'une grande quantité de bruit.

Nous présentons donc ici deux procédures fondamentales de l'apprentissage automatique, le clustering et la classification, appliquées à nos résultats.

Nous rappelons que l'"apprentissage automatique" (dit aussi *Machine Learning*) est le domaine de la fouille des textes et des données qui étudie comment produire des programmes qui s'améliorent par l'expérience, en se confrontant à des données[Tellier 2008]. Pour *clustering*, nous indiquons l'*apprentissage non-supervisé*, c'est-à-dire les approches visant à extraire de l'information à partir des données fournies sans indication supplémentaire (non-étiquetées). Le terme *classification* indique plus simplement les méthodes d'*apprentissage supervisé*, où les résultat visé

par l'apprentissage est déjà disponible car on dispose déjà des données étiquetées au départ.

L'objet de nos analyses et de nos études reste, aussi dans ces traitements d'apprentissage automatique, les grammaires de dépendances.

8.1 Clustering

Le clustering est le processus qui permet la division de données similaires en groupes. Il s'applique à l'exploration des données, à la recherche d'information, à la fouille de données et à beaucoup d'autres disciplines. C'est un processus d'"apprentissage non-supervisé" : les algorithmes de clustering sont capables, sans intervention humaine (et sans corpus annoté) de créer des regroupements sur la base de la description des données (sous la forme d'attributs). Pour ce faire, ces algorithmes se basent sur une notion de distance entre les données calculée grâce à ces attributs. Ces regroupements peuvent donc être considérés comme des "configurations d'attributs".

Nous avons donc besoin du clustering pour analyser nos patrons, dans l'espoir de créer des clusters de patrons similaires partageant les même caractéristiques. En effet, nous avons constaté que beaucoup de patrons se ressemblaient, ayant certains aspects en commun.

Pour effectuer les analyses nous avons utilisé une approche de type K-Means. Inversement aux algorithmes de clustering hiérarchique (des attributs à la constitution des clusters), les approches de ce type créent d'abord les clusters et ensuite cherchent les données appartenant à chaque cluster. Les clusters possèdent un *centroïde*, et chaque objet (instance) est analysé en calculant la distance du centroïde du cluster.

L'inconvénient du clustering est qu'il faut choisir un nombre de clusters désirés avant de commencer le processus. De plus, les K-Means sont assez sensibles aux nombres de clusters, donc il faut faire plusieurs tests et analyser les différents résultats pour trouver un résultat convenable. Il faut donc analyser manuellement les patrons de dépendances pour avoir une idée sommaire de combien de groupes nous désirons. Finalement, notre but est d'affiner les patrons qui, à cause des nombreux processus par lesquels le corpus est passé, ne sont pas parfaits et présentent des incongruités, ainsi que des erreurs.
Nous avons remarqué trois types de similarité.

1. Similarité des patrons sur la base des **lemmes**.

(1a) nsubj(opened,Y) dobj(opened,X)
(1b) nsubjpass(opened,X) agent(opened,Y)
⇒ relation : *founder*

Les dépendances sont différentes (littéralement, (1a) correspond à "*Y opened X*" et (1b) "*Y was opened by X*"), mais le concept sémantique est le même : *open*. On pourrait donc envisager de clusteriser les patrons en utilisant les lemmes : *open(X, Y)*.

2. Similarité sur la base des **relations grammaticales** :

(2a) appos(Y,CEO) prep_of(CEO,X)
(2b) appos(Y,president) prep_of(president,X)
⇒ relation : *keyPeople*

Ils sont composés des mêmes relations grammaticales (et expriment les même types de relation sémantique : (2a) "*Y, president of X*", (2b) "*Owner of X, Y...*") et peuvent donc être regroupés, comme instances d'un sur-patron : *appos(Y, CEO/president) prep_ of(CEO/president, X)*

3. Patrons différant juste par quelques dépendances :

(3a) nn(restaurant,X) nsubjpass(opened,restaurant) agent(opened,Y)
(3b) nsubjpass(opened,X) agent(opened,Y)
⇒ relation : *founder*

Où (3a) correspond à "*the X restaurant was opened by Y*" et (3b) à "*X was opened by Y*" , et qu'il faut donc réduire à un seul patron.

C'est en particulier ce troisième point qui nous a poussée à explorer le clustering. En effet, plusieurs patrons sont identiques, mais certaines dépendances, dues en particulier à des erreurs d'analyse syntaxique, font que des patrons diffèrent.

8.1.1 Outils

Clustering avec Weka Nous avons utilisé Weka pour effectuer nos analyses. Weka est un outil rassemblant des algorithmes d'apprentissage automatique permettant d'effectuer des tâches de fouille des données [1]. Il s'agit d'un *masterpiece* du *Data-Mining* qui est utilisé depuis longtemps (la première version date d'il y a 11 ans) dans la communauté de l'apprentissage supervisé [Sharma(sachdeva) 2012]. Il contient plusieurs onglets, pour le pré-traitement des données, la classification, le clustering, l'extraction de règles d'association, et la visualisation. Nous utilisons ici l'onglet "clustering".

La partie la plus délicate dans la fouille de données est la conception de la matrice à donner en entrée à l'algorithme. En effet, il faut réfléchir sur quels sont les facteurs vraiment importants, ceux qui sont discriminants pour la séparation des données en catégories.

La constitution d'un fichier de fouille de textes sont :

1. http ://www.cs.waikato.ac.nz/ml/weka/

- la constitution de l'espace de représentation (donc, l'ensemble des attributs qui sont présents dans au moins une donnée)
- la représentation de chaque donnée dans l'espace

Dans l'entête du fichier, montrée en 8.1, se trouvent les attributs (les colonnes) avec leurs noms et les valeurs qu'ils peuvent accepter (dans notre cas, un entier correspondant aux occurrences de l'élément dans le patron). L'ensemble des attributs correspond à l'espace de représentation. Suivant les considérations que nous avons faites en début de la section, nous avons utilisé trois types d'attributs : (i) les lemmes, (ii) les relations grammaticales et (iii) les POS-tags présents dans chaque patron. Nous avons donc effectué une double lecture du corpus des patrons afin de :

- récupérer tous les lemmes, les POS-tags et les relations grammaticales, pour créer l'espace de représentation
- lire chaque patron, pour détecter quels éléments il contient, c'est-à-dire construire son vecteur (comment il est représenté dans l'espace)

Un script Python (cf. annexe B) nous a permis de créer aisément ce fichier (au format .arff demandé par Weka).

```
@attribute "nsubjpass" real
@attribute "agent" real
....
@attribute "located" real
...
@data
```

FIGURE 8.1 – Entête du fichier .arff pour Weka

Lemmatisation Pour extraire les lemmes de tout le corpus, nous avons récupéré tous les mots (têtes ou dépendants) présents dans les patrons, et nous les avons lemmatisés avec TreeTagger [2]. Cet outil permet d'annoter des mots avec leurs tags *part-of-speech* et leur lemme. La sortie est fournie en trois colonnes (mot - POS - lemme) et nous avons donc utilisé la commande `awk` pour ne récupérer que les lemmes.

```
TreeTagger/bin/tree-tagger -token -lemma TreeTagger/lib/english.par
          listeMot | awk '{print $3}' > listeLemmes.txt
```

Les POS-tags sont déjà compris dans le patron, il n'y a pas besoin d'utiliser le TreeTagger.

2. http ://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/

8.1.2 Expérimentation et résultats

Nous avons effectué des tests pour deux relations de *Organisation* : *numberOf-Students*, où nous avions peu de patrons à traiter, comme essai préliminaire, et *location* comme exemple de relation où nous avions beaucoup d'imperfections et de patrons à regrouper. La modalité de test que nous avons choisie est l'utilisation du set d'entraînement.

NumberOfStudents Dans ce cas, nous avons choisi d'utiliser comme attributs les relations grammaticales et les lemmes des mots (sans POS-tag). De notre côté, nous avons estimé que les patrons étaient à classifier en 5 classes, nombre que nous avons saisi en option de la commande de lancement des SimpleKMeans. Nous avons choisi d'utiliser la distance euclidienne (mais d'autres distances sont disponibles : distance de Manhattan, distance de Minkowski, distance de Chebyshev).
La totalité des attributs (24) est visible dans l'entête de la sortie (cf. 8.2). On

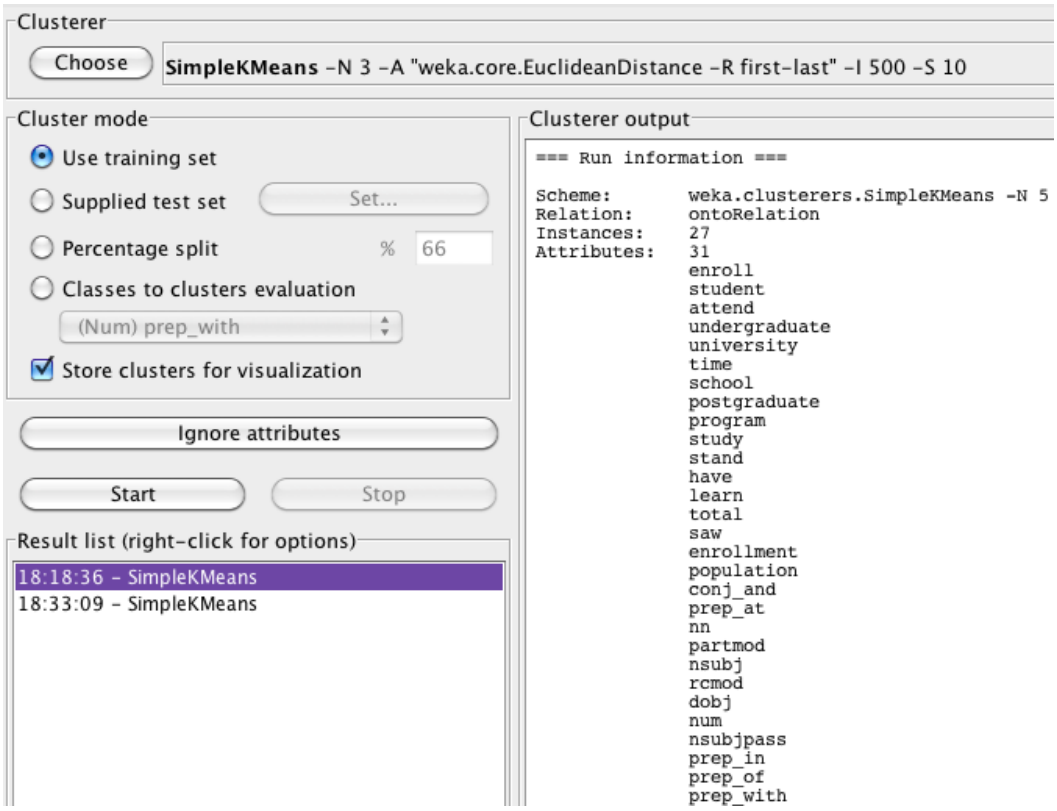


FIGURE 8.2 – Entête de la sortie des SimpleKmeans

voit aussi le nombre d'instances que nous avions (33). Nous avons volontairement choisi d'ignorer les deux attributs "X" et "Y", car nous les considérons comme non discriminants (ils sont présents dans chaque patron), ainsi que "NJ" (probablement une erreur d'extraction).

Dans la même fenêtre des résultats nous pouvons voir ensuite les clusters en format tabulaires (les colonnes sont les clusters et les lignes les attributs), ainsi que les statistiques et pourcentages d'instances appartenant à chaque cluster. Les valeurs affichées pour un cluster donné sont les coordonnées du vecteur moyen qui le caractérise, calculées sur la base des valeurs des instances appartenant au cluster : ce vecteur moyen est appelé "centroïde". En regardant les centroïdes (cf. 8.3) nous pouvons déterminer les caractéristiques du cluster. Le premier cluster

Cluster centroids:

Attribute	Full Data (27)	Cluster# 0 (7)	1 (11)	2 (2)	3 (3)	4 (4)
enroll	0.963	1.1429	0	2	2	2
student	1.6296	0.8571	2	2	2	1.5
attend	0.1481	0	0.3636	0	0	0
undergraduate	0.0741	0.2857	0	0	0	0
university	0.2222	0	0.3636	1	0	0
time	0.0741	0.2857	0	0	0	0
school	0.0741	0	0.1818	0	0	0
postgraduate	0.0741	0.2857	0	0	0	0
program	0.0741	0	0	0	0.6667	0
study	0.1481	0.2857	0.1818	0	0	0
stand	0	0	0	0	0	0
have	0	0	0	0	0	0
learn	0.0741	0	0.1818	0	0	0
total	0.0741	0	0	1	0	0
saw	0.1481	0	0	0	0.6667	0.5
enrollment	0.4444	0	0	1	0.6667	2
population	0.1481	0.5714	0	0	0	0
conj_and	0.037	0	0.0909	0	0	0
prep_at	0.3333	0.5714	0.3636	0	0	0.25
nn	0.037	0	0	0	0.3333	0
partmod	0.1481	0	0	1	0.6667	0
nsubj	0.8148	0.8571	0.9091	0.5	0.3333	1
rcmod	0.0741	0.1429	0.0909	0	0	0
dobj	0.5185	0.5714	0.4545	1	0	0.75
num	0.9259	0.7143	1	1	1	1
nsubjpass	0.0741	0.2857	0	0	0	0
prep_in	0.1852	0	0.1818	0	1	0
prep_of	0.2222	0.2857	0.0909	0	0.3333	0.5
prep_with	0.0741	0	0.0909	0.5	0	0

FIGURE 8.3 – Clustering à 5 pour *NumberOfStudents*

contient 7 instances, autour des lemmes "*enroll*" et "*student*" (ce sont ceux qui possèdent les plus grandes moyennes de fréquences). Les relations grammaticales sont principalement subject (*nsubj*), object direct (*dobj*) et les prépositions *of* et *at*. Comme verbes nous avons aussi "*study*", et on retrouve aussi les hyponymes de *student* "*undergraduate*" et "*postgraduate*" avec les mêmes fréquences moyennes. Cela signifie que les patrons "*X enrolls Y postgraduates*" et "*Y undergraduated study at X*" sont reconnus similaires par les KMeans.
Le cluster 1 (le deuxième donc) contient la plupart des instances (11), où "*student*" est la valeur la plus fréquente, suivi de "*attend*" et "*university*". Ici aussi les patrons sont des phrases actives (vue la fréquence du *nsubj* et *dobj*) et les KMeans ont rapproché les patrons que nous pourrions regrouper en un seul "*Y students (are attending/attend) the Y school/university*".
Le troisième cluster contient seul deux instances où le lemme "*total*" est distinctif "*X with an enrollment totaling Y students*". Les quatrième et cinquième clusters contiennent des patrons de type "*X saw an enrollment of Y students*", se différen-

ciant par les relations grammaticales.

Nous allons donc essayer avec 3 clusters, pour voir s'il y a une amélioration (cf. 8.4). Dans ce cas, nous voyons que le cluster 0 contient bien les patrons (11 instances) "*Y students are enrolled in Y*" (passive), "*Y students enroll X*" (active), mais aussi tous les patrons autour de "*enrollment*". Dans le cluster 1 (12 instances), nous avons *attend* comme verbe, et comme noms "*student*" et ses hyponymes, ainsi que le groupe "*university*", "*program*" et "*school*" qui sont sémantiquement proches. Le cluster 3 contient *grosso modo* les caractéristiques des deux derniers clusters de la clusterisation précédente.

		Cluster#		
Attribute	Full Data (27)	0 (11)	1 (12)	2 (4)
enroll	0.963	1.6364	0	2
student	1.6296	1.2727	1.8333	2
attend	0.1481	0	0.3333	0
undergraduate	0.0741	0	0.1667	0
university	0.2222	0	0.3333	0.5
time	0.0741	0.1818	0	0
school	0.0741	0	0.1667	0
postgraduate	0.0741	0.1818	0	0
program	0.0741	0	0	0.5
study	0.1481	0	0.3333	0
stand	0	0	0	0
have	0	0	0	0
learn	0.0741	0	0.1667	0
total	0.0741	0	0	0.5
saw	0.1481	0.3636	0	0
enrollment	0.4444	0.9091	0	0.5
population	0.1481	0.3636	0	0
conj_and	0.037	0	0.0833	0
prep_at	0.3333	0.3636	0.4167	0
nn	0.037	0	0	0.25
partmod	0.1481	0	0	1
nsubj	0.8148	0.9091	0.9167	0.25
rcmod	0.0741	0.0909	0.0833	0
dobj	0.5185	0.6364	0.4167	0.5
num	0.9259	0.8182	1	1
nsubjpass	0.0741	0.1818	0	0
prep_in	0.1852	0.0909	0.1667	0.5
prep_of	0.2222	0.4545	0.0833	0
prep_with	0.0741	0	0.0833	0.25

FIGURE 8.4 – Clustering à 3 pour *NumberOfStudents*

Nous jugeons donc cette clusterisation acceptable.

Location Nous essayons cette fois avec un attribut plus complexe, *location*. Pour cette relation, après l'élimination manuelle des patrons trop génériques, nous avons obtenu 114 patrons à clusteriser. Parmi ces patrons, il y en a qui sont génériques, il y en a qui sont plus spécifiques, mais il y en a aussi qui sont issus d'une mauvaise analyse syntaxique. Par exemple :

$$[X \textit{ is located in } Y]_{GEN}$$
$$[X \textit{ is a restaurant chain located in } Y]_{SPE}$$
$$[X \textit{ is located in } Y \textit{ City}]_{ERR}$$

Notre volonté reste surtout de rapprocher les patrons qui diffèrent seulement par quelques attributs, comme dans l'exemple cité ci-dessus. Comme dans l'exemple précédent la première étape est de décider combien de clusters nous souhaitons obtenir. En regardant nos données, et en se basant particulièrement sur les verbes, nous décidons le nombre de 11 groupes. Nous choisissons aussi dans ce cas la distance euclidienne pour les calculs.

Une fois le clustering lancé, nous allons analyser les groupes. Cette fois, nous essayons d'interpréter linguistiquement les centroïdes et de dériver un patron pour chaque groupe.

1. Le premier cluster contient des lemmes comme *company, chain, firm, organisation, franchise* (POS-tag : NN), le verbe (VBZ) *headquarter* et les relations grammaticales *nsubjpass, prep_in, prep_of, prep_near, prep_around*. 18 données sont comprises dans ce cluster. Nous pouvons recréer un patron comme :

   `the X $w is headquartered $prep Y`
   `$w`= *organisation, chain, company, franchise, firm*
   `$prep`= *in, near, around*

   À noter que dans le même cluster il y a aussi *suburb* et *prep_of*. Le patron sera alors :

   `the X $w is headquartered $prep (the suburbs of)* Y`

   Ce patron est valable pour plusieurs concepts, il est donc générique.

2. Ce cluster, contenant 22 données, tourne autour des verbes *base*, *situate*, et *operate* :

   `X is a $w ($verb`$_{pass}$`| $verb`$_{act}$`) $prep Y`
   `$w`= *branch, company, hotel, store, restaurant, university*
   `$verb`$_{pass}$= *situated, based*
   `$verb`$_{act}$= *operates*
   `$prep`= *throughout, out_of, around*

   Les relations grammaticales concernées sont *rcmod, partmod* et *appos*. Un patron de ce type nous permet de récupérer des phrases comme : *X is a company situated around Y, X is a store that operates in Y* etc. Il est donc plus générique.

3. Ce cluster comprend 7 données.

   `X is a (college|store) located in Y`

   Le même centroïde comprend aussi le verbe *released* qui, *a priori*, est un verbe plus spécifique (*Sitcom, Movie*) que *located*. La raison est que les patrons *X is located in Y* et *X was released in Y* ont les mêmes relations grammaticales (*nsubjpass* et *prep_in*). Ils ont donc été classés dans le même cluster.

4. Le quatrième cluster comprend 9 données :

   `X (provides care | operates) $prep Y`
   `$prep`= *near, around*

C'est un patron plus spécifique (*provides care* se réfère sans doute à *Hospital*).

5. 16 données font partie de ce cluster, spécifique au concept *University*.

   `the X $w ($verb`$_{pass}$`| $verb`$_{act}$`) $in (the center of)* Y`
   `$w`= *institution, college, university*
   `$verb`$_{pass}$= *located*
   `$verb`$_{act}$= *offers an education*

   Bien évidemment, pour un verbe passif la relation grammaticale correspondante sera *nsubjpass*, alors que pour un verbe actif ce sera *dobj*.

6. Le sixième cluster comprend une seule donnée. Le patron correspondant est :

   `X is situated in Y`

   c'est un patron générique.

7. Ce cluster regroupe 5 données. Nous remarquons ici deux sous-groupes :

   (1)`X is a $w established in (the heart of)* Y`
   `$w`= *hotel, school, chain*
   (2)`X is a $w film released in Y`

   Ce cluster nécessite donc un affinement ultérieur.

8. Le huitième cluster comprend 8 données. Ce patron diffère légèrement des autres, car il n'y a pas de relation sujet-verbe :

   `X, [operator in (the countryside of)* Y)]`$_{appos}$

   mais seulement une relation d'apposition entre les deux entités.

9. Ce cluster a regroupé ses données autour des noms communs *base* et *location*. Il comprend 7 données.

   `X is a $w1 with $w2 (around | throughout) Y`
   `$w1`= *hospital, pizzeria, restaurant*
   `$w2`= *bases, locations*

10. L'avant dernier cluster comprend 11 données.

    `X is a $w opened (around | in) Y`
    `$w1`= *chain, restaurant, institution, school, store*

    Le verbe *opened* est lié à X par les relations *appos*, *rcmod* et *partmod*.

11. Ce dernier cluster regroupe 10 données. Le patron est :

    `X is a (organisation | institution) established in Y`

    où *established* est lié à X par les relations *appos*, *rcmod* et *partmod*.

Nous voyons comment l'apprentissage non-supervisé nous a effectivement aidée à regrouper les patrons.

Il serait peut être intéressant de voir les résultats du clustering en ajoutant des attributs plus sémantiques, indiquant des relations entre les mots. Notamment, on pourrait exploiter des ressources externes comme Wordnet : si un attribut

permettait de marquer la relation d'hyponymie entre *undergraduate* et *student* ou entre *pizzeria* et *restaurant*, il serait sûrement plus facile de rassembler les patrons contenant ces mots.

Finalement, il ne faut oublier que nous ne pouvons pas évaluer la qualité du clustering, à cause du manque de données étiquetées de référence. C'est bien le défaut de l'apprentissage non-supervisé : ceci nous a poussée à explorer par suite les méthodes de classification.

8.2 Classification

Une dernière section est dédiée à la classification, c'est-à-dire l'apprentissage supervisé, des patrons que nous avons découverts. L'idée sous-jacente est que certains patrons sont spécifiques à une relation ontologique bien définie : par exemple, nous pourrions facilement supposer que "born_in(X,Y)" (Y étant un nombre cardinal) soit un patron spécifique à la relation *birthDate*, alors que "died_in(X,Y)" caractérise la relation *deathDate*.

Il est donc possible d'utiliser un corpus déjà annoté, dans l'espoir de pouvoir analyser les patrons issus d'un nouveau corpus et savoir déjà de quelle relation il s'agit. Nous exploitons le corpus dont nous disposons, c'est-à-dire chaque ensemble de patrons spécifique à une relation : nous connaissons déjà à quelle classe/relation ontologique le patron appartient, donc il est déjà annoté.

Pour ne pas créer de confusion, dans cette tâche la **classe** à définir, c'est-à-dire la colonne ultime de la matrice, est la **relation ontologique** sur les valeurs de laquelle nous avons travaillé précédemment. Les **attributs** de la matrice, dits aussi *features* (les colonnes), sont les dépendances, entendues comme des triplets REL(H,D), et ne doivent pas être confondus avec les attributs ontologiques, qui sont nos classes.

8.2.1 Création de la matrice

Contrairement à ce que l'on pourrait imaginer, nous ne nous sommes pas concentrée sur l'intégralité du corpus des snippets (ce qui est souvent le cas lors de l'extraction des relations entre entités nommées), en définissant des attributs autour du type des entités nommées concernées (*Restaurant* et *Location* pour la relation *location*, *Person* et *Date* pour *deathDate* etc). Nous nous sommes concentrée uniquement sur les dépendances, c'est-à-dire les triplets REL(H,D) que nous avons récupérés pour chaque relation.
Plus précisément, chaque donnée (une ligne du tableau) est un patron, annoté avec la classe/relation ontologique dans laquelle il a été découvert, par exemple :

(1) nsubjpass(founded,X) agent(founded, Y) ⇒ Classe : *founder*
(2) partmod(X,located) prep_in(located, Y) ⇒ Classe : *location*

Les attributs sont tous les triplets qui existent dans la totalité du corpus (pour chaque patron et pour chaque relation ontologique). Les valeurs des attributs pour cette donnée représentent un vecteur de booléens, indiquant la présence ou pas du triplet dans le patron. Autrement dit : si la donnée contient le triplet, la valeur de l'attribut correspondant est `T` ; sinon, `F`. Le tableau 8.1 montre un exemple pour les données (1) et (2).

	nsubjpass (founded, X)	agent (founded,Y)	partmod (X,located)	prep_in(located, Y)	poss (X,Y)	CLASS
(1)	T	T	F	F	F	*founder*
(2)	F	F	T	T	F	*location*

TABLE 8.1 – Exemple de matrice de classification

8.2.2 Outils

Weka Pour la classification, Weka nécessite un corpus déjà annoté pour que les algorithmes apprennent sur celui-ci comment annoter des nouvelles données. Il faut donc ajouter après les attributs une dernière colonne avec la classe à apprendre.

Un nouveau script Python (annexe C) nous a permis d'écrire ce fichier. Alors que dans le clustering nous travaillions sur une seule relation, nous devons retrouver dans ce cas toutes les dépendances dans **toutes** les relations. La double lecture du corpus pour la constitution du fichier .arff est cette fois effectuée sur la totalité des relations : (i) pour l'espace de représentation, nous avons extrait toutes les dépendances existantes et (ii) pour la représentation de chaque patron dans l'espace vectoriel, nous avons détecté quelles dépendances (triplets) y sont contenues. Le vecteur n'est ici construit qu'avec des valeurs booléennes, non plus les occurrences. Dans l'entête du fichier, montrée en 8.5, il faut aussi donner un nom à la relation que nous cherchons (*ontoRelation*) :

8.2.3 Résultats

Une fois le corpus chargé, nous pouvons voir la répartition des données (cf. 8.6). Dans la partie supérieure de l'onglet nous avons un résumé de nos données :

```
Relation: ontoRelation   Attributes: 1206
Instances: 790           Sum of weight: 790
```

```
@relation ontoRelation

@attribute "nsubjpass (founded, X)" {T,F}
@attribute "agent (founded,Y)" {T,F}
@attribute "partmod (X,located)" {T,F}

@data
```

FIGURE 8.5 – Entête du fichier .arff pour la classification

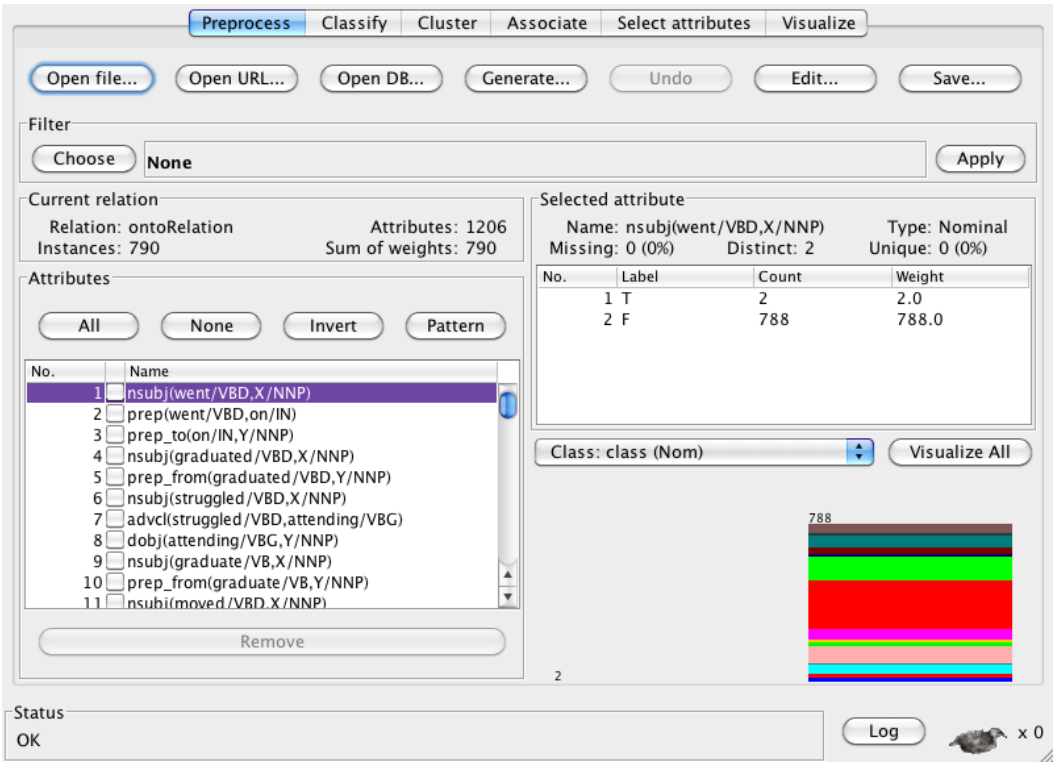


FIGURE 8.6 – Fenêtre Weka après pre-processing

790 patrons comme données (les lignes) et 1206 triplets passées en attributs (les colonnes).

Bien qu'on ne le remarque pas, il y a deux colonnes en bas à droite : une, à droite, pour les valeurs `F` et une à gauche, pour les valeurs `T` (étant une colonne de 2 éléments contre une de 788, il est difficile de la remarquer). Ce que nous voyons dans l'image est la répartition des décisions par rapport à l'attribut sélectionné (dans le cadre en bas à gauche) : 2 fois cette attribut avait la valeur `T`, et 788 fois était `F`. De toutes les 788 valeurs `F` à chaque fois l'annotation a correspondu à une classe, marquée avec une couleur différente. En cochant *class* parmi les attributs, nous pouvons voir les correspondances couleurs-classe ainsi que le nombre de données

étiquetées avec cette classe (cf. 8.7).

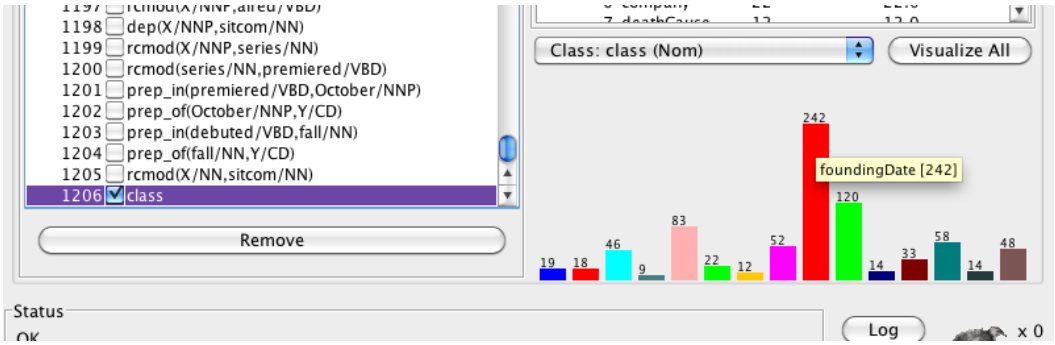


FIGURE 8.7 – Classes et nombre des données

Les algorithmes d'apprentissage étudient ces données pour comprendre quels facteurs sont discriminants pour une relation. Afin d'avoir plus de visibilité sur nos résultats, nous avons utilisé deux classifieurs différents : les SVM, *Support Vector Machines*, et les arbres de décision (*Decision Trees*). Les premiers sont actuellement les méthodes les plus performantes, mais leur résultat n'est pas lisible, bien qu'il soit explicable (grâce à l'équation qui caractérise l'hyperplan séparateur). Par contre, les deuxièmes ont l'avantage de pouvoir fournir un résultat visuel (l'arbre de décision même) pour que voir à l'œil nu quels critères ont guidé la classification.

Les *Decision trees* Les *Decision Trees* sont un algorithme datant des années '80, qui construisent un arbre en définissant des règles sur les attributs pour guider la décision (l'annotation). Les arbres se lisent de haut en bas et chaque nœud correspond à un critère portant sur un des attributs ; chaque feuille est une attribution à une classe et chaque chemin est la classification d'une donnée. Nous avons utilisé l'algorithme J48 de Weka. Le premier test effectué a été une validation croisée à trois plis (cf. 8.8). Les deux tableaux représentent les résultats d'évaluation de l'algorithme : en haut, les valeurs de précision, rappel et F-mesure (après qu'il a essayé de ré-étiqueter automatiquement les données, il les a comparées aux données initiales) ; en bas, la matrice de confusion, qui montre comment les données ont été étiquetées exactement (l'axe x, "étiquetées comme.." et l'axe y, "la vraie classe" : seulement les données de la diagonale sont bien étiquetées).
Des résultats de F-mesure (tableau du haut) nous voyons que les meilleurs résultats sont donnés par les relations *birthDate* et *numberOfStudents* : nous en déduisons que c'est à cause des patrons très spécifiques leur appartenant qu'ils sont plus faciles à étiqueter (nous le voyons aussi dans la matrice de confusion). Probablement, parmi les nœuds de l'arbre moins profonds nous retrouverons des dépendances discriminant ces relations. En général, moins un attribut est profond dans l'arbre (ou bien, plus il est haut), plus la dépendance sera discriminante. Les résultats les plus mauvais sont donnés par *party*, *almaMater* et *award* (aucune ou une seule donnée bien étiquetée). *DeathCause*, que nous croyions facile à détecter, ne donne pas de bons résultats non plus. Pour ces relations, il est sûrement indispensable de fournir plus de données.

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.105     0          1         0.105     0.19        0.713    almaMater
                 0.056     0          1         0.056     0.105       0.638    alternativeNames
                 0.478     0.003      0.917     0.478     0.629       0.84     author
                 0         0          0         0         0           0.678    award
                 0.867     0.001      0.986     0.867     0.923       0.943    birthDate
                 0.591     0.003      0.867     0.591     0.703       0.833    company
                 0         0.003      0         0         0           0.592    deathCause
                 0.654     0.009      0.829     0.654     0.731       0.875    deathDate
                 0.955     0.504      0.456     0.955     0.617       0.72     foundingDate
                 0.35      0.015      0.808     0.35      0.488       0.789    location
                 0.714     0.001      0.909     0.714     0.8         0.85     numberOfEpisodes
                 0.788     0          1         0.788     0.881       0.922    numberOfStudents
                 0.293     0          1         0.293     0.453       0.759    owner
                 0         0          0         0         0           0.684    party
                 0.313     0.005      0.789     0.313     0.448       0.769    releaseDate
Weighted Avg.    0.614     0.158      0.724     0.614     0.583       0.786

=== Confusion Matrix ===

   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o   <-- classified as
   2   0   0   0   0   0   0   0  17   0   0   0   0   0   0 |   a = almaMater
   0   1   0   0   1   0   0   0  16   0   0   0   0   0   0 |   b = alternativeNames
   0   0  22   0   0   0   0   0  23   0   0   0   0   0   1 |   c = author
   0   0   0   0   0   0   0   0   9   0   0   0   0   0   0 |   d = award
   0   0   0   0  72   0   0   0   9   0   0   0   0   0   2 |   e = birthDate
   0   0   0   0   0  13   0   0   8   0   1   0   0   0   0 |   f = company
   0   0   0   0   0   0   0   5   7   0   0   0   0   0   0 |   g = deathCause
   0   0   0   0   0   0   2  34  16   0   0   0   0   0   0 |   h = deathDate
   0   0   0   0   0   0   0   2 231   9   0   0   0   0   0 |   i = foundingDate
   0   0   0   0   0   0   0   0  77  42   0   0   0   0   1 |   j = location
   0   0   0   0   0   1   0   0   3   0  10   0   0   0   0 |   k = numberOfEpisodes
   0   0   0   0   0   0   0   0   7   0   0  26   0   0   0 |   l = numberOfStudents
   0   0   1   0   0   0   0   0  40   0   0   0  17   0   0 |   m = owner
   0   0   0   0   0   0   0   0  14   0   0   0   0   0   0 |   n = party
   0   0   1   0   0   1   0   0  30   1   0   0   0   0  15 |   o = releaseDate
```

FIGURE 8.8 – Validation croisée à trois champs avec Decision Trees

Nous remarquerons que la relation *foundingDate* a été attribuée à tort beaucoup plus souvent que les autres (cf. la colonne i de la matrice de confusion). D'après la fig. 8.7, *foundingDate* est aussi la relation la plus fréquente : un tel déséquilibre initial des classes n'a sûrement pas facilité les choix des algorithmes de classification. Nous nous attendons de trouver des attributs correspondant aux patrons pour cette relation parmi les nœuds les plus profonds (les plus loin de la racine) de l'arbre.

Alors que pour un petit ensemble de données il est possible de voir une version graphique de l'arbre construit, nous devons nous contenter de voir la sortie du classifieur car nous avons un arbre trop grand (cf. 8.9). Cette image nous montre comment l'arbre décide : la première dépendance permet de décider s'il faut étiqueter la donnée avec *numberOfEpisodes* (si la donnée contient le triplet) ou pas (si le triplet n'est pas présent). Ensuite, nous avons *numberOfStudents* et, quelques nœuds après, *birthDate*. Les derniers nœuds de l'arbre sont *foundingDate* et *location*. Nos considérations étaient donc correctes.

Une deuxième classification a été effectuée avec une validation croisée à 10 plis, dans l'espoir que plus de données d'entraînement améliorent les résultats. Cepen-

```
J48 pruned tree
------------------

num(episodes/NNS,Y/CD) = T: numberOfEpisodes (10.0)
num(episodes/NNS,Y/CD) = F
|   num(students/NNS,Y/CD) = T: numberOfStudents (25.0)
|   num(students/NNS,Y/CD) = F
|   |   prep_for(aired/VBN,episodes/NNS) = T: numberOfEpisodes (2.0)
|   |   prep_for(aired/VBN,episodes/NNS) = F
|   |   |   dobj(has/VBZ,population/NN) = T: numberOfStudents (3.0)
|   |   |   dobj(has/VBZ,population/NN) = F
|   |   |   |   nsubj(won/VBD,X/NNP) = T: award (2.0)
|   |   |   |   nsubj(won/VBD,X/NNP) = F
|   |   |   |   |   nsubjpass(born/VBN,X/NNP) = T: birthDate (47.0/1.0)
|   |   |   |   |   nsubjpass(born/VBN,X/NNP) = F
|   |   |   |   |   |   partmod(X/NNP,born/VBN) = T: birthDate (24.0)
|   |   |   |   |   |   partmod(X/NNP,born/VBN) = F
|   |   |   |   |   |   |   dep(X/NNP,born/VBN) = T: birthDate (4.0)
|   |   |   |   |   |   |   dep(X/NNP,born/VBN) = F
|   |   |   |   |   |   |   |   nsubj(born/VBN,X/NNP) = T: birthDate (3.0)
|   |   |   |   |   |   |   |   nsubj(born/VBN,X/NNP) = F
|   |   |   |   |   |   |   |   |   rcmod(X/NNP,born/VBN) = T: birthDate (2.0)
|   |   |   |   |   |   |   |   |   rcmod(X/NNP,born/VBN) = F
|   |   |   |   |   |   |   |   |   |   agent(produced/VBN,Y/NNP) = T: company (6.0)
|   |   |   |   |   |   |   |   |   |   agent(produced/VBN,Y/NNP) = F
|   |   |   |   |   |   |   |   |   |   |   agent(produced/VBN,Y/NNPS) = T: company (4.0)
|   |   |   |   |   |   |   |   |   |   |   agent(produced/VBN,Y/NNPS) = F
```

FIGURE 8.9 – Nœuds hauts de l'arbre de décision

dant, la matrice de confusion pour la validation croisée à 10 plis ne change pas beaucoup(cf. 8.10), et nous n'avons aucune amélioration dans les relations qui n'étaient pas bien étiquetées dans le test précédent.

```
  a   b   c   d   e   f   g   h   i   j   k   l   m   n   o   <-- classified as
  1   0   0   0   0   0   0   0  18   0   0   0   0   0   0 |   a = almaMater
  0   4   0   0   1   0   0   0  13   0   0   0   0   0   0 |   b = alternativeNames
  0   0  28   0   0   0   0   0  17   0   0   0   0   0   1 |   c = author
  0   0   0   0   0   0   0   0   9   0   0   0   0   0   0 |   d = award
  0   0   0   0  77   0   0   0   5   0   0   0   0   0   1 |   e = birthDate
  0   0   0   0   0  13   0   0   7   0   1   0   0   0   1 |   f = company
  0   0   0   0   0   0   0   6   6   0   0   0   0   0   0 |   g = deathCause
  0   0   0   0   0   0   2  35  15   0   0   0   0   0   0 |   h = deathDate
  0   0   0   0   0   0   0   0 234   8   0   0   0   0   0 |   i = foundingDate
  0   0   0   0   0   0   0   0  79  40   0   0   0   0   1 |   j = location
  0   0   0   0   0   0   0   0   3   0  10   0   0   0   1 |   k = numberOfEpisodes
  0   0   0   0   0   0   0   0   5   0   0  28   0   0   0 |   l = numberOfStudents
  0   0   1   0   0   0   0   0  38   0   0   0  19   0   0 |   m = owner
  0   0   0   0   0   0   0   0  14   0   0   0   0   0   0 |   n = party
  0   0   2   0   1   1   0   0  27   1   0   0   0   0  16 |   o = releaseDate
```

FIGURE 8.10 – Validation croisée à dix champs avec les Decision Trees

Support Vector Machines Les derniers tests que nous essayons sont effectués avec des *Support Vector Machines* (algorithme SMO de Weka). Les SVM classent les données en cherchant à les séparer par un hyperplan qui soit le meilleur possible suivant un certain critère. Les SVM réduisent la classification à un problème binaire : il y a autant d'apprentissages que de classes à deviner et, à chaque fois, la séparation est cherchée entre cette classe particulière et toutes les autres ("un contre

tous", comme l'on voit dans la fig. 8.11). Les poids affichés sont les coordonnées de

```
17 Classifier for classes: almaMater, alternativeNames
18
19 BinarySMO
20
21 Machine linear: showing attribute weights, not support vectors.
22
23          0.4266 * (normalized) nsubj(went/VBD,X/NNP)
24  +       0.2841 * (normalized) prep(went/VBD,on/IN)
25  +       0.2841 * (normalized) prep_to(on/IN,Y/NNP)
26  +       0.5963 * (normalized) nsubj(graduated/VBD,X/NNP)
27  +       0.3975 * (normalized) prep_from(graduated/VBD,Y/NNP)
28  +       0.3312 * (normalized) nsubj(struggled/VBD,X/NNP)
29  +       0.3312 * (normalized) advcl(struggled/VBD,attending/VBG)
```

FIGURE 8.11 – Exemple de classification binaire

l'hyperplan qui maximise la marge.
Les résultats de la validation croisée à 10 plis (cf. 8.12) nous montrent que, comme nous nous y attendions, les SVM fonctionnent mieux, bien que nous ne puissions pas en détecter les raisons. La F-Mesure est très élevée pour les mêmes relations qui étaient en haut de l'arbre des *Decision Trees* : *birthDate* et *numberOfEpisodes*. Moins de données ont été étiquetées de manière erronée. De plus, cette fois, nous avons des données bien étiquetées pour *party* (5/13), pour *deathCause* (2/10), pour *award*(1/9). Nous notons donc une légère amélioration.

8.3 Extraction de Nouvelles Entités

Une fois que nous avons défini une série de patrons valables pour la relation, nous pouvons extraire de nouvelles entités. Nous rappelons que le but de ce système est la population automatique d'une ontologie.

Les patrons que nous avons constitués comprennent, nous l'avons vu précédemment, un côté X et un côté Y. Ceux-ci ne sont que des identifiants, X permettant de récupérer une nouvelle entité et Y permettant de récupérer la valeur de l'attribut. Ceci est possible grâce à un système d'expressions régulières qui vont interroger l'arbre d'une nouvelle phrase et en extraire les informations requises, c'est-à-dire les dépendances.
Lorsque nous définissons un patron, celui-ci a trois représentations, chacune avec un but différent :

1. représentation en dépendances, pour voir la composition (les dépendances) du patron
2. représentation par expression régulière, pour appliquer le patron à une nouvelle phrase
3. représentation textuelle, c'est-à-dire comment ce patron est exprimé en langage naturel

Alors que la première représentation nous a servi pour les analyses des dépendances dans les phases de classification et de clustering, les deux autres nous servent

F-Measure

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	<-- classified as
0.48	6	0	0	0	0	0	0	0	13	0	0	0	0	0	0	a = almaMater
0.56	0	7	0	0	1	0	0	0	10	0	0	0	0	0	0	b = alternativeNames
0.747	0	0	28	0	0	0	0	0	17	0	0	0	0	0	1	c = author
0.2	0	0	0	1	0	0	0	0	8	0	0	0	0	0	0	d = award
0.969	0	0	0	0	79	0	0	0	4	0	0	0	0	0	0	e = birthDate
0.757	0	0	0	0	0	14	0	0	7	0	0	0	0	0	1	f = company
0.267	0	0	0	0	0	0	2	5	5	0	0	0	0	0	0	g = deathCause
0.774	0	0	0	0	0	0	1	36	15	0	0	0	0	0	0	h = deathDate
0.703	0	0	0	0	0	0	0	0	228	11	0	0	2	0	1	i = foundingDate
0.709	0	0	0	0	0	0	0	0	47	73	0	0	0	0	0	j = location
0.833	0	0	0	0	0	0	0	0	3	0	10	0	0	0	1	k = numberOfEpisodes
0.918	0	0	0	0	0	0	0	0	5	0	0	28	0	0	0	l = numberOfStudents
0.737	0	0	1	0	0	0	0	0	21	1	0	0	35	0	0	m = owner
0.526	0	0	0	0	0	0	0	0	9	0	0	0	0	5	0	n = party
0.747	0	0	0	0	0	1	0	0	15	1	0	0	0	0	31	o = releaseDate
0.733																

(a) F-Measure (b) Matrice de confusion

FIGURE 8.12 – SVM avec validation croisée à 10 champs

maintenant pour la recherche de nouvelles entités.
La première étape consiste à récupérer les phrases candidates. Pour ce faire, la représentation textuelle du patron (2) sert à créer des requêtes qu'un moteur de recherche lance sur un corpus. Ce requêtes sont constituées par la classe d'appartenance de l'instance originale de laquelle le patron a été extrait (c'est-à-dire le sujet du triplet RDF de la relation), et du patron même en version textuelle. Par exemple :

Olive Garden was founded by Michael Ilitch
relation : *founder*
triplet : <Restaurant><founded><Person>
pattern : `nsubjpass(founded,X) agent(founded, Y)`
requête : "*Restaurant (is/was) founded by*"

Mais quel corpus utiliser ? Ayant déjà vu les complications dues au Web, nous avons utilisé un deuxième corpus de test, Wikipedia. Si Wikipedia permet sûrement de ne pas avoir de bruit dans les textes, la quantité de données contenue et idéalement récupérable n'est surement pas comparable à celle dans le Web. Les deux corpus présentent donc avantages et inconvénients.

La première partie du processus est la même qu'avant l'extraction des patrons : récupération des documents/snippets et analyse des phrases. Ensuite, nous appliquons les patrons aux nouvelles phrases : ceci est fait grâce à l'expression régulière (3) correspondant à chaque patron. Chaque patron est "matché" sur l'arbre syntaxique de la phrase, exactement de la même manière lorsque l'on cherche une expression régulière dans une chaîne de caractères. Pour l'exemple précédent, l'expression sera :

$$\{word : /founded/\} > nsubjpass\{tag : /NNP/\} = X\& > agent\{tag : /NNP/\} = Y$$

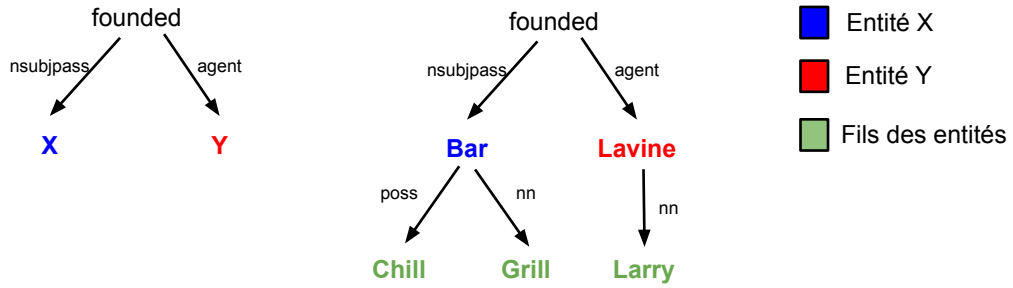


FIGURE 8.13 – Matching du patron sur une nouvelle phrase

le matching permettra de récupérer un nœud X dont le tag est NNP et qui est dépendant dans une dépendance *nsubjpass* et un nœud Y (de tag NNP) dépendant d'une relation *agent*.

Ensuite, on vérifie que le nœud ait un fils et, dans le cas positif, nous reconstituons l'entité dans son intégralité. Un exemple de ce processus est montré en 8.13.

Chaque couple d'entités qui correspond à X et Y sera une nouvelle instance selon le schéma du module que nous sommes en train de traiter :

<Restaurant><founder><Person>
"*Chili's Grill & Bar is a restaurant chain founded by Larry Lavine*"
X = *Chili's Grill & Bar* → <Chili_s_Grill_&_Bar><isA><Restaurant>
Y = *Larry Lavine* → <Larry_Lavine><isA><Person>

alors qu'un nouveau triplet est instancié pour définir la relation découverte :

<Chili_s_Grill_&_Bar><founder><Larry_Lavine>

8.3.1 Experimentation

Pour démontrer la validité de nos patrons nous avons utilisé les patrons les plus fréquents pour chaque relation. Les relations sur lesquelles portent nos expérimentations sont celles du chapitre 7. Nous avons ensuite récupéré 5 snippets pour chaque patron et analysé chaque snippet en dépendances. Ensuite, nous avons compté combien de triplets <X,rel,Y> ont bien été extraits.

Les snippets de Wikipedia diffèrent légèrement de ceux du Web dans leur formation : Wikipedia étant plus structurée que le Web, une des entités intéressées (X ou Y) se retrouve parfois dans l'attribut "title" , alors que le snippet contient seulement des phrases avec un pronom personnel se référant à l'entité, comme :

<p ns="0" title="*Chris Abbott*" snippet="*She* ***is a graduate of University*** *of Oregon , with an MFA from Bennington College in Vermont . She started her career writing for Little ...* "/ >

Dans ce cas, alors, il peut être utile de remplacer le pronom par l'entité du titre de la page Wikipedia. Pour voir la différence, le tableau 8.2 montre combien de triplets exacts sont extraits dans le premier cas (sans utiliser l'attribut *title*) et dans le deuxième (avec remplacement du pronom avec la valeur de *title*).

	Patrons	Snippets	Sans *title*	Avec *title*
almaMater	8	35	18	33
alternativeNames	8	35	15	22
award	8	40	14	19
author	10	38	22	34
birthDate	3	15	4	5
company	4	10	8	10
deathCause	6	21	2	9
deathDate	3	15	1	3
foundingDate	14	48	37	45
location	9	33	13	33
numberOfEpisodes	4	0	0	0
numberOfStudents	6	22	4	21
owner	12	31	10	28
party	6	16	7	15
releaseDate	8	15	6	15

TABLE 8.2 – Résumé d'extraction de nouvelles entités

La récupération des valeurs de *title* améliore nettement l'extraction avec les patrons. Les rares cas où l'extraction n'a pas fonctionné sont les cas où :

- des entités dans title étaient des listes (souvent Wikipedia contient des listes : *List of prolific writers*, *List of winners of the National Book Award* etc.)
- le patron est trop vague : par exemple, le patron *X killed from Y* pour *deathCause* donne une phrase comme *Initial reports indicate that there has been no one killed from the Jordanian side*, alors que ce qui nous intéresse est une phrase comme *Nadav Kudinski was killed from a bomb.*
- le patron ne retourne aucun snippet : c'est le cas, par exemple, de la relation *numberOfEpisodes*, où Wikipedia ne contient pas de réponses pour aucun des 4 patrons (*X made Y episodes*, *X aired for Y episodes*, *X ran for Y episodes*, *X has Y episodes*). Dans ces cas, soit le patron est trop compliqué

  $$X \text{ is situated in the hearth of the } Y \text{ countryside} \rightarrow location$$
  $$X \text{ is a sitcom debuted in the fall of } Y \rightarrow releaseDate$$

  soit la requête pour l'API a été mal formée

  $$sitcom\ ran\ for\ episodes \rightarrow \text{aucun résultat}$$
  $$\neq$$
  $$sitcom\ ran\ for \rightarrow \text{bons résultats}$$

8.3.2 Outils

Wikipedia API Pour la récupération des snippets candidats dans Wikipedia nous nous sommes servie de l'API Wikipedia [3]. De même que le moteur de recherche Bing, elle permet de récupérer les résultats d'une requête en format XML. Le script Python permettant cette récupération se trouve en annexe (D).

Stanford SemGrex Incluse dans le Stanford Parser se trouve la librairie SemGrex. C'est cet outil qui nous a permis d'identifier de nouvelles entités, en utilisant les expressions régulières. Une syntaxe particulière a été définie pour retrouver des dépendances ou des nœuds à l'intérieur d'un graphe, sur la base d'un tag, d'un NER ou bien d'une relation grammaticale. Pour les expressions logiques (AND, OR, NOT), la syntaxe reprend celle des expression régulières.
Brièvement :

- `{}` un nœud
- `{lemma|tag|NER : <string>}` un nœud qui ait comme lemma/tag/NER un <string>. On peut effectuer une regEx sur les <string> en les mettant entre slash (/VB*/, tous les verbes)
- `A > relationGrammaticale B` équivaut à dire relationGrammaticale(A,B)
- `A < relationGrammaticale B` équivaut à dire relationGrammaticale(B,A)
- `A » relationGrammaticale B` les nœuds appartiennent à la même branche, mais il n'y a pas de dépendance directe [relationGrammaticale(A,C) + relationGrammaticale(C,B)]
- inversement, `A « relationGrammaticale B`
- `&` pour rattacher les dépendances :
  `A > relationGrammaticale B & A > relationGrammaticale C` équivaut à dire que le nœud A est gouverneur de B et C
- les parenthèses permettent d'ordonner : `A > relationGrammaticale ( B > relationGrammaticale C)` équivaut à dire que le nœud A est gouverneur de B, qui est gouverneur de C

Pour ce faire nous avons construit une classe RegeExBuilder qui construit de manière récursive les expressions régulières en se basant sur la composition des patrons (cf. annexe E).

3. http ://www.mediawiki.org/wiki/API :Main_page

Chapitre 9

Conclusions

Sommaire

Cette section est dédiée aux conclusions que nous pouvons tirer de nos expérimentations, et aux améliorations à prévoir pour la suite.

9.1 Resumé analytique de l'expérimentation

L'idée principale qui a poussé ce travail a été la combinaison des techniques linguistiques et celles de *Machine Learning* pour l'intégration des données textuelles non-structurées dans les données structurées. Cette proposition se révèle particulièrement intéressante pour la construction et le peuplement d'ontologies.
Nous avons proposé une riche architecture, mais certains processus n'ont pas fourni les résultats souhaités. Nous allons voir rapidement quels ont été les *pro* et *contra* étape par étape, avant d'en tirer une conclusion définitive.

Extraction des *Linked Data* Les *Linked Data* représentent la concrétisation du Web Sémantique, comme son inventeur l'avait conçu. Derrière le choix d'utiliser ce type de données, il y a la volonté d'analyser cette nouvelle conception du Web et d'en découvrir le potentiel.

Pro Ces données sont extrêmement intéressantes car issues directement du Web des documents. Autrement dit, les informations présentes dans les ontologies du Web Sémantique existent déjà, sous une autre forme, dans le Web des documents. Cette dualité est centrale dans notre travail, car elle nous permet d'aller chercher comment les informations structurées sont représentées en langage naturel, dans les pages Web.

De plus, l'interconnexion entre les données de différentes sources évite la prolifération des contenus, et permet un meilleur contrôle de la qualité de l'information.

Les données structurées sont une base fiable pour la construction d'un module d'ontologie.

Contra Le Web des *Linked Data* n'est malheureusement pas encore uniformisé au point de permettre une confiance de 100% sur la qualité des données qui y

sont contenues. Les différentes bases de connaissance existantes ne sont pas toujours connectées, ce qui provoque des erreurs sur les informations extraites. Les bases de connaissance sont en train d'être liées (grâce aux *mapping* que nous avons vus) depuis que les efforts de la communauté du Web Sémantique se sont intensifiés. Cependant il existe encore trop de différences entre chaque source, comme les domaines traités, la granularité de l'information qui y est contenue ou le langage utilisé.

Nous devons prendre en compte une marge d'erreur pour la qualité des informations extraites du Web des données.

Recherche Web L'idée est d'utiliser les moteurs de recherche pour retrouver des informations textuelles sur les données du Web Sémantique. Au lieu d'utiliser les pages Web, nous préférons l'extraction des *snippets*.

Pro Les snippets sont des phrases très courtes contenant les mots saisis lors d'une recherche. L'avantage des snippets est que nous n'avons pas à traiter les structures des pages HTML (tag inutiles à notre but), ni trop de contenus textuels (des phrases qui ne contiennent pas d'information utile). Les pages Web sont en effet déjà indexées par les moteurs de recherche qui, eux, renvoient seulement les morceaux des pages qui peuvent nous intéresser.

Contra Les snippets renvoyés par les moteurs de recherche sont loin d'être parfaits. Au-delà d'un problème lié à la recherche même (un snippet/une page est visible mais ne contient pas la totalité de l'information recherchée), parfois le snippet n'est pas composé d'une seule phrase, mais de plusieurs morceaux regroupés. Ceci a créé une grande confusion et la perte de 2/3 des données que nous traitons.

De plus, le Web étant très bruité et très peu contrôlé, nous avons aussi à faire avec des contenus textuels sans aucune possibilité d'exploitation (annonces, publicités, etc.).

Analyse syntaxique Arriver à cette étape signifie disposer déjà de phrases assez propres pour être analysées. La découverte des patrons est basée sur l'utilisation des grammaires de dépendances.

Pro Les grammaires de dépendances permettent d'obtenir des patrons sémantiquement plus intéressants, car elles prennent en compte directement les relations liant les mots, sans donner d'importance à l'ordre linéaire des mots dans la phrase (c'est le cas des patrons lexico-syntaxiques). De cette façon, seulement les mots importants sont considérés dans les patrons de dépendances, en permettant une généralisation de ces règles qui ne serait pas possible avec d'autres techniques.

Contra L'analyse en dépendances est effectuée par des outils statistiques qui ne sont pas parfaits. Cela veut dire qu'il faut tenir compte aussi d'une marge d'erreur de la part de l'outil que nous utilisons, qui peut causer la production de mauvais

patrons.

Deuxièmement, les dépendances marchent sur des phrases minimales, autour d'un verbe, ce qui peut être limitant pour certaines relations.

Apprentissage Automatique L'apprentissage automatique nous permet de travailler sur les patrons et d'en étudier les caractéristiques afin d'automatiser le plus possible le processus.

Pro Le clustering permet de créer des groupes de patrons proches sur la base des similarités entre éléments. Nous pouvons donc réduire les nombreuses règles produites par l'analyse en peu de groupes, aux caractéristiques linguistiques bien définies (lemmes, NER, POS-tags, dépendances). Le clustering permet, entre autres, de rapprocher aussi un patron plus "mauvais" d'un plus "correct".
La classification nous montre comment certains patrons sont effectivement spécifiques à une classe.

Contra Le clustering et la classification nécessitent encore la supervision humaine. Alors que pour la classification il faut fournir un corpus étiqueté, pour le clustering, bien qu'il s'agisse d'apprentissage non-supervisé, il faut que l'on prenne une décision initiale pour guider les regroupements. Ceci peut se révéler difficile avec un corpus comportant de nombreuses entités.

Extraction de Wikipedia Un test pour le peuplement de l'ontologie a été effectué sur le corpus de Wikipedia.

Pro Les snippets de Wikipedia sont décidément moins bruités que ceux du Web. Il est aussi possible de récupérer les entités du titre du snippet (Wikipedia est un corpus déjà plus structuré que le Web).

Contra Les entités dans Wikipedia sont sans doute moins nombreuses que celles du Web.

9.2 Conclusions et Travaux Futurs

De l'analyse précédente nous comprenons que, au cours de notre travail, il a fallu faire des concessions.

Le Web Sémantique est une bonne base de départ mais est encore loin d'avoir organisé de manière homogène l'information. En revanche, la grande quantité de données du Web des documents est très difficile à exploiter dans son intégralité : pour obtenir des résultats de bonne qualité, il faut parfois renoncer à manipuler de grandes quantités de données.

Nous avons cependant montré que les grammaires de dépendances se révèlent, après beaucoup d'efforts et de traitements, très utiles pour ce type de travail.

Les méthodes d'apprentissage automatique sont la touche finale qui permet de réorganiser les résultats et réduire la marge d'erreurs.

L'extraction d'informations à partir des textes en ligne est un processus qui requiert encore beaucoup de travail et d'amélioration. L'abandon de l'intervention humaine n'est à l'heure actuelle pas encore envisageable. Nous nous limitons ici à montrer comment ce type de travail peut être encore amélioré, en montrant quelques directions futures sur lesquelles travailler.

Utiliser moins de snippets pour la création de corpus peut sans doute réduire le bruit contenu dans le Web. Nous avons en effet remarqué que le taux de bruit dans les phrases augmentait proportionnellement au nombre de snippets extraits pour un couple. Autrement dit : pour le couple *Olive Garden–Mike Ilitch*, les premiers snippets contenaient les deux entités, puis le bruit augmentait en avançant vers le 30ème snippet, jusqu'aux derniers qui ne contenaient plus du tout d'information exploitable. En général, 5 ou 10 snippets sont largement suffisants.

Bien que les snippets se soient révélés pratiques, il serait intéressant de rechercher dans l'intégralité des pages Web pour découvrir les phrases réellement intéressantes, et non pas se contenter d'avoir des morceaux de textes. Ceci implique l'utilisation des ***Web Content Extractors***, les outils qui explorent les structures HTML pour en extraire seulement les contenus textuels. Une analyse des contenus Web demanderait aussi plus de temps de calcul pour l'extraction des phrases et la constitution du corpus (lire une page *vs* lire un snippet).

Une étape à explorer pour avoir une amélioration significative serait la **résolution de la coréférence**, une tâche qui n'est pas simple pour le traitement du langage, mais qui aiderait par contre les machines à comprendre beaucoup plus aisément le langage humain, en particulier sa sémantique. La résolution de coréférences permettrait de récupérer les patrons de ces phrases que nous avons exclues lors du filtrage avant l'analyse syntaxique. Résoudre les anaphores et cataphores pourrait donner des résultats très intéressants au niveau de la découverte des patrons.

Un travail d'affinement de patrons peut être effectué pour les représenter avec plus de caractéristiques linguistiques : lemmes, types d'entités nommées, POS-tags, stemmes etc.

Comme nous avons déjà mentionné dans la section sur le clustering, l'apprentissage symbolique (PLI ou inférence grammaticale) pourrait être une voie à explorer pour la généralisations automatique des patrons.

Pour découvrir les relations entre les données structurées du Web Sémantique, la **reconnaissance d'Entités Nommées** peut être explorée en utilisant les concepts d'une ontologie au lieu des classes standard (Personne, Lieu, Organisation).

Finalement, une voie de recherche à explorer reste l'association des patrons linguistiques aux **patrons ontologiques**[1]. Les *Ontology Design Patterns* sont des structures standardisées pour la construction d'ontologies. L'idée est que certaines

1. http ://ontologydesignpatterns.org/wiki/Main_Page

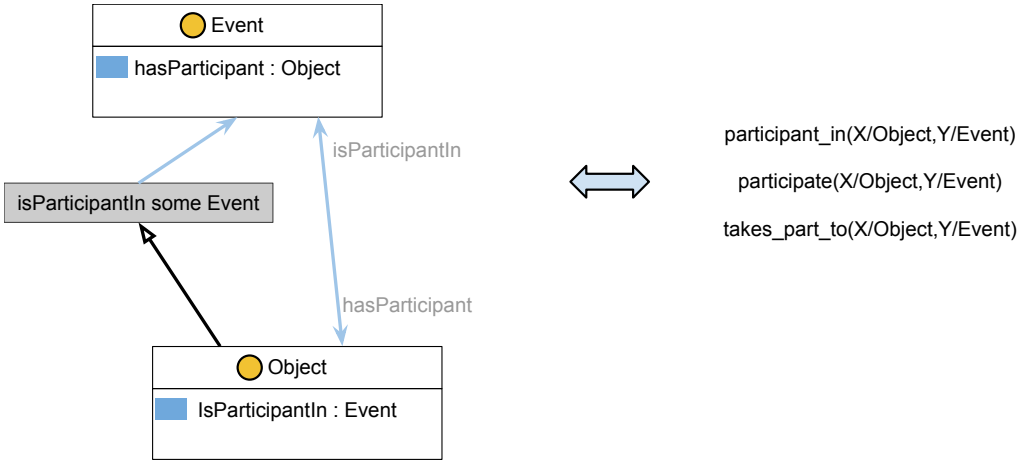


FIGURE 9.1 – Exemple de correspondance d'un *Design Pattern* et un patron linguistique

modélisations conceptuelles sont les mêmes, peu importe la source d'où elles proviennent. Nous souhaitons donc explorer dans le futur les patrons linguistiques qui représentent ces *Desing Patterns* (cf. 9.1).

Annexe A

Extraction des Snippets (Python)

```python
# -*- coding: utf-8 -*-

import urllib
import codecs
import xml.dom.minidom as xml
import os

def readQueries(fileName):
    """
    reads queries in queryFile
    """
    liste = list()
    fd = codecs.open(fileName,'r','utf-8')
    for line in fd:
        line.encode('utf-8')
        line = line.replace('_','+')
        if not line[-1]=='"':
            line = line[:-1]
        if line not in liste:
            liste.append(line)
    fd.close()
    return liste

def getUrl(query):
    """
    url connection (get an xml file)
    called by extractDescription
    """
    try:
        f = urllib.urlopen('http://api.bing.net/xml.aspx?AppId=
            BCD25F2AA09A2501949CB1961866159C40536736&Market=en-US&query
            ='+query+'&sources=web&web.count=30')
        return f.read()
    except UnicodeError:
        print 'unicodeError'
        return ""

def extractDescription(query):
    """
    parses dom and get the descriptions
    calls getUrl()
    """
    print query
    xl = getUrl(query)
```

```
44      if xl != "":
            dom = xml.parseString(xl)
46          liste = dom.getElementsByTagName("web:Description")
            return liste
48      else:
            return None
50
# program
52
fichiers = os.listdir("./docs/queries/")
54 count = 1
for fichier in fichiers:
56
        fd = codecs.open("./docs/resultsPythonPREPROC/corpus"+fichier
            [7:-4],'w',"utf-8")
58      print "reading queries for "+ fichier
        print str(len(fichiers)-count)+" files to go..."
60      for k in readQueries("./docs/queries/"+fichier):
            count+=1
62          descr = extractDescription(k)
            if descr:
64              for i in descr:
                    fd.write(k+"\t"+i.childNodes[0].nodeValue+"\n")
66      fd.close()
        print "done"
68      print "**********"
```

./Annexe/WebExtraction.py

Annexe B

Matrice pour le Clustering

```
# -*- coding: utf-8 -*-

import os
import re
import codecs
import sys


def getLemmes(fd):
  """
  gets the set of lemmes
  uses TreeTagger
  """
  myRe= re.compile("(\w+)/")
  liste=myRe.findall(fd)

  fdw= codecs.open("listeMot","w")
  for mot in set(liste):
    fdw.write(mot+"\n")
  fdw.close()
  os.system("TreeTagger/bin/tree-tagger -token -lemma TreeTagger/lib/
      english.par listeMot | awk '{ print $3 }' > listeLemmes.txt")
  os.system("rm -f listeMot")
  fd2 = codecs.open("listeLemmes.txt", "r", "utf-8")
  listeLemmes=list()
  for lemme in fd2:
    listeLemmes.append(lemme.strip())
  fd2.close()

  return set(listeLemmes)

def getPOS(fd):
  """
  gets the set of posTag
  """
  myR= re.compile("/(\w+)")
  lis = myR.findall(fd)
  return set(lis)


def getGrammRel(fd):
  """
  get the set of posTag
  """
```

```
    myR= re.compile("(\w+)\(")
    lis = myR.findall(fd)
    return set(lis)

def getPatterns(fd):
    """
    extract patterns
    """
    patternListe=[]
    pattern=""
    for line in fd.split("\n"):
        if "***" not in line:
            pattern=pattern+" "+line
        else:
            patternListe.append(pattern)
            # pattern reset
            pattern=""
    return patternListe


def vector(element,liste):
    """
    builds a counter for a pattern
    """
    presenceVector=[]
    for elt in liste:
        myRE=re.compile(elt)
        count=len(myRE.findall(element))
        presenceVector.append(str(count))
    return presenceVector

def buildMatrixALone(liste,listePatterns):
    """
    building classification matrix
    single test
    """
    # the matrix of pattern
    matrix={}
    for pattern in listePatterns:
        # number of occurrences
        k = vector(pattern,liste)

        matrix[pattern]=k

    print matrix
    return matrix

def buildMatrixDouble(liste1,liste2,listePatterns):
    """
    building classification matrix with two lists
    """
    # the matrix of pattern
    matrix={}
    print "Building matrix..."
```

```
    for pattern in listePatterns:
99
      k = vector(pattern,liste1)
101   z = vector(pattern,liste2)

103   # toDo : change p with an id
      matrix[pattern]=k+z
105 #print matrix
    return matrix
107

109 def buildMatrix(listePOS, listeLemmes,listeRel, listePatterns):
    """
111 building classification matrix
    for each pattern:
113   presencesOfPosTags=real
      presencesOfLemmes=real
115   presencesOfGrammaticalRelation=real
    """
117 # the matrix of pattern
    matrix={}
119 for pattern in listePatterns:
      #each [1|0] is the presence of dep in p
121   v = vector(pattern,listePOS)
      k = vector(pattern,listeLemmes)
123   z = vector(pattern,listeRel)

125

      # toDo : change p with an id
127   matrix[pattern]=v+k+z
    print matrix
129 return matrix

131 def writeWekaFile(matrix,features):
    """
133 write matrix in file f
    """
135 print "Writing file..."
    fd=codecs.open("wekaFile.arff","w","utf-8")
137 fd.write("@relation ontoRelation\n\n")

139 for feat in features:
      fd.write("@attribute \""+feat+"\" real\n")
141 fd.write("\n\n")
    fd.write("@data\n")
143 for i in matrix.keys():
    # print i+"("+",".join(k for k in matrix[i])+")"
145   fd.write(",".join(k for k in matrix[i])+","+"CLASS\n")

147 def parse(file):
    """
149 parse dep file
    """
151 fd = codecs.open(file, "r", "utf-8")
```

```
    fr = fd.read()
153
    pos= getPOS(fr)
155 lemmes = getLemmes(fr)
    rel= getGrammRel(fr)
157 patterns = getPatterns(fr)

159 mat = buildMatrixDouble(lemmes,rel,patterns)
    writeWekaFile(mat,list(lemmes)+list(rel))
161



163
    mat = buildMatrix(pos,lemmes,rel,patterns)
165 writeWekaFile(mat,list(pos)+list(lemmes)+list(rel))
    print "***"
167


169 fd.close()
    return
171
file=""
173 if len(sys.argv) < 2:
        sys.stderr.write('Usage: sys.argv[0] ')
175     sys.exit(1)
else:
177     file = sys.argv[-1]

179 parse(file)
```

./Annexe/Cluster.py

Annexe C

Matrice pour la Classification

```
# -*- coding: utf-8 -*-


"""
Matrix must be:
PatternString(B,B,B,B...)
"""

import codecs
import re
import os

def removeIndex(fd):
  myRe= re.compile("(-\d+)")
  fd1=myRe.sub("",fd.read())
  return fd1


def parse():
  """
  parsing dependency file
  gets the set of attributes (coloumn)
  """

  liste=list()

  # treating each file in the directory
  for file in childrenFiles("Files"):
    print file+"********"
    fd = codecs.open(file, "r", "utf-8")

    #removing indexes (should not be necessary)
    #fd=removeIndex(fd)

    for dep in fd.read().split("\n"):

      # adding the dependency to the list
      if not dep in liste and "***" not in dep:
        liste.append(dep)

  return liste

def getPatterns():
  """
```

```
  extracting patterns
  patterns will be the rows of the matrix
  """
  corpusListe=dict()

  # treating each file in the directory
  for file in childrenFiles("Files"):
    fd = codecs.open(file, "r", "utf-8")

    patternListe=list()
    pattern=""
    #fd= removeIndex(fd)
    for line in fd.read().split("\n"):

      if "***" not in line:

        pattern=pattern+" "+line
      else:
        patternListe.append(pattern)
        # pattern reset
        pattern=""
    corpusListe[file]=patternListe
    fd.close()
  return corpusListe

def buildMatrix(listeDep):
  """
  building classification matrix
  """
  # the matrix of pattern:listeOfDep
  matrix={}

  # set of patterns
  setOfPat=getPatterns()

  # for each relation
  print "Building vectors..."
  for relation in setOfPat.keys():

    #for each existing pattern
    for pattern in setOfPat[relation]:

      #each [T|F] is the presence of dep in p
      presenceVector=[]

      # building vectors
      for dep in listeDep:
        if dep in pattern:
          #present
          presenceVector.append("T")
        else:
          #absent
          presenceVector.append("F")

```

```
      # last coloumn is the class
100       presenceVector.append("\""+relation[6:-4]+"\"")


102
          matrix[pattern]=presenceVector
104
    return matrix
106
def childrenFiles(path):
108   """
    get the files in a path
110   """
    files=list()
112   for o in  os.listdir(path):
        files.append(path+"/"+o)
114   files.remove(path+"/.DS_Store")
    return files
116


118 def writeWekaFile(matrix,depList):
    """
120   write matrix in file f
    """
122
    print "Writing .arff file"
124
    fd=codecs.open("wekaFile1.arff","w","utf-8")
126   fd.write("@relation ontoRelation\n\n")

128   for dep in depList:
        fd.write("@attribute \""+dep+"\" {T,F}\n")
130


132   fd.write("@attribute class {"+",".join("\""+f[6:-4]+"\"" for f in
          childrenFiles("Files"))+"}\n")
    fd.write("\n\n")
134   fd.write("@data\n")
    for i in matrix.keys():
136     #print i+"("+",".join(k for k in matrix[i])+")"
        fd.write(",".join(k for k in matrix[i])+"\n")
138
    return
140
setOfDep=parse()
142
m=buildMatrix(setOfDep)
144 writeWekaFile(m,setOfDep)
```

Annexe/Classification.py

Annexe D

Extraction des Snippets Wikipedia

```
# -*- coding: utf-8 -*-

import urllib
import codecs
import xml.dom.minidom as xml
import re


def cleanSnippet(sni):
  c = re.sub("<.*?>","",sni)
  return c

def getUrl(qr):
    """
    url connection (get an xml file)
    called by extractDescription
    """
    string = "https://en.wikipedia.org/w/api.php?action=query&list=
        search&format=xml&srsearch=%22"+qr+"%22&srnamespace=0&srwhat=
        text&srprop=snippet|sectionsnippet&srlimit=5"
    f = urllib.urlopen(string)
    return f.read()


def extractDescription(q):
    """
    parses dom and get the descriptions
    calls getUrl()
    """
    print "extracting snippets..."
    xl = getUrl(q)
    liste= list()
    if xl != "":
        dom = xml.parseString(xl)
        elts = dom.getElementsByTagName("p")
        for tag in elts:
          snippet = tag.getAttribute("snippet")
          title =  tag.getAttribute("title")
          cleaned = cleanSnippet(snippet)

          liste.append(title+"\t"+cleaned)
        return liste
    else:
        return None
```

```
43
def writeAnswers(liste,fd):
45   for i in liste:
       fd.write(i+"\n")
47
def readQ(file):
49   fd = codecs.open(file, "r","utf-8")
     l= fd.readlines()
51   fd.close()
     return l
53


55 fd = codecs.open("listeAnswers.txt", "a", 'utf-8')

57 for i in readQ('queries.txt'):
     i = i.strip()
59   query = i.replace(" ","%20")
     print "writing..."
61   fd.write(query+"\n")
     writeAnswers(extractDescription(query),fd)
63   fd.write("-----\n")
   fd.close()
```

Annexe/wikiExtract.py

Annexe E

Classe Java RegExpBuilder pour la construction d'expressions regulières

```
package ontologyLearning.pattern.Construction;

import java.util.Iterator;
import java.util.List;

import edu.stanford.nlp.ling.IndexedWord;
import edu.stanford.nlp.trees.semgraph.SemanticGraph;
import edu.stanford.nlp.trees.semgraph.SemanticGraphEdge;

public class RegexpBuilder {

  private SemanticGraph graph;
  private IndexedWord origin;
  private IndexedWord target;
  private IndexedWord root;

  public RegexpBuilder(SemanticGraph graph, IndexedWord origin,
      IndexedWord target, IndexedWord root) {
    super();
    this.graph = graph;
    this.origin = origin;
    this.target = target;
    this.root = root;
  }

  public String getRegexp() {
    return buildRegexp(root);
  }

  /**
   * noeud
   *
   * @return
   */
  private String buildRegexp(IndexedWord root) {
    SemanticGraphEdge edge;
    String result = "";
    if (origin.equals(root)) {
      result = "{tag:/" + origin.tag() + "/}=X";
```

```
    } else if (target.equals(root)) {
      result = "{tag:/" + target.tag() + "/}=Y";
    } else {
      result = "{word :/" + root.word() + "/}";
    }
    List<IndexedWord> children = graph.getChildList(root);
    Iterator<IndexedWord> iterator = children.iterator();
    IndexedWord child;
    while (iterator.hasNext()) {
      child = iterator.next();
      edge = graph.getEdge(root, child);
      result += " > " + edge.getRelation() + buildRegexp(child);
      if (children.size() > 1 && iterator.hasNext()) {
        result += " & ";
      }
    }

    if (children.size() >= 1 && !graph.getFirstRoot().equals(root)) {
      return " (" + result + ") ";
    }
    return result;

  }

  public String dependencies() {
    String gr = graph.toPOSList();
    gr = gr.replace(origin.word(), "X");
    gr = gr.replace(target.word(), "Y");
    gr = gr.replaceAll("-(\\d+)", "");

    return gr;
  }

  public String graphToText() {
    String gr = graph.toEnUncollapsedSentenceString();
    gr = gr.replace(origin.toString(), "");
    gr = gr.replace(target.toString(), "");
    return gr;
  }
}
```

Annexe/RegexpBuilder.java

Bibliographie

[Akbik 2009] Alan Akbik et Jürgen Broß. *Wanderlust : Extracting Semantic Relations from Natural Language Text Using Dependency Grammar Patterns.* Juillet 2009. (Cité en page 33.)

[Aldrich 1997] John Aldrich. *R. A. Fisher and the Making of Maximum Likelihood 1912-1922.* Statistical Science, vol. 12, no. 3, pages 162–176, 1997. (Cité en page 74.)

[Amardeilh 2007] Françoise Amardeilh. *Web Sémantique et Information Linguistique : propositions méthodologiques et réalisation d'une plateforme logicielle.* PhD thesis, Paris X University, 2007. (Cité en pages 7 et 15.)

[Auer 2008] Sören Auer, Chris Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak et Zachary Ives. *DBpedia : A Nucleus for a Web of Open Data.* In Proceedings of the 6th International Semantic Web Conference (ISWC), volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, 2008. (Cité en page 18.)

[Aussenac-Gilles 2008] Nathalie Aussenac-Gilles, Sylvie Despres et Sylvie Szulman. *The TERMINAE Method and Platform for Ontology Engineering from Texts.* In Proceedings of the 2008 conference on Ontology Learning and Population : Bridging the Gap between Text and Knowledge, pages 199–223, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press. (Cité en page 35.)

[Baroni 2004] Marco Baroni et Sabrina Bisi. *Using Cooccurrence Statistics and the Web to Discover Synonyms In Technical Language.* In In Proceedings of LREC 2004, pages 1725–1728, 2004. (Cité en page 31.)

[Becker 2008] Christian Becker et Christian Bizer. *DBpedia Mobile : A Location-Enabled Linked Data Browser.* In Linked Data on the Web (LDOW2008), 2008. (Cité en page 21.)

[Berland 1999] Matthew Berland et Eugene Charniak. *Finding Parts in Very Large Corpora*, 1999. (Cité en page 33.)

[Berners-Lee 1998] Tim Berners-Lee et Mark Fischetti. Weaving the web : The original design and ultimate destiny of the world wide web by its inventor. Harper San Francisco, 1st édition, 1998. (Cité en pages 7 et 15.)

[Berners-Lee 2001] Tim Berners-Lee, James Hendler et Ora Lassila. *The Semantic Web.* Scientific American, vol. 284, no. 5, pages 34–43, Mai 2001. (Cité en page 14.)

[Berners-Lee 2009] Tim Berners-Lee. *Linked-data design issues.* W3C design issue document, June 2009. http ://www.w3.org/DesignIssue/LinkedData.html. (Cité en page 24.)

[Biemann 2005] Chris Biemann. *Ontology Learning from Text : A Survey of Methods.* LDV Forum, vol. 20, no. 2, pages 75–93, 2005. (Cité en page 27.)

[Bizer 2007] Chris Bizer, Tom Heath et u.a. *Interlinking Open Data on the Web.* www4.wiwiss.fu-berlin.de/bizer/pub/LinkingOpenData.pdf, 2007. Stand 12.5.2009. (Cité en page 25.)

[Bizer 2009a] C. Bizer, T. Heath et T. Berners-Lee. *Linked data - the story so far.* Int. J. Semantic Web Inf. Syst., vol. 5, no. 3, page 1–22, 2009. (Cité en pages 23 et 25.)

[Bizer 2009b] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak et Sebastian Hellmann. *DBpedia - A crystallization point for the Web of Data.* Web Semant., vol. 7, no. 3, pages 154–165, Septembre 2009. (Cité en pages 18 et 19.)

[Brunzel 2008] Marko Brunzel. *The XTREEM Methods for Ontology Learning from Web Documents.* In Proceedings of the 2008 conference on Ontology Learning and Population : Bridging the Gap between Text and Knowledge, pages 3–26, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press. (Cité en pages 29, 31 et 36.)

[Buitelaar 2004] Paul Buitelaar, Daniel Olejnik et Michael Sintek. *A Protege Plug-in for Ontology Extraction from Text Based on Linguistic Analysis.* In In Proceedings of the 1st European Semantic Web Symposium (ESWS, 2004. (Cité en pages 32 et 34.)

[Buitelaar 2008] P. Buitelaar, P. Buitelaar et P. Cimiano. Ontology learning and population : Bridging the gap between text and knowledge - volume 167 frontiers in artificial intelligence and applications. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2008. (Cité en page 33.)

[Buitelaar 2009] Paul Buitelaar, Philipp Cimiano, Peter Haase et Michael Sintek. *Towards Linguistically Grounded Ontologies.* In ESWC, pages 111–125, 2009. (Non cité.)

[Ciaramita 2005] Massimiliano Ciaramita, Aldo Gangemi, Esther Ratsch, Jasmin Šaric et Isabel Rojas. *Unsupervised learning of semantic relations between concepts of a molecular biology ontology.* In Proceedings of the 19th international joint conference on Artificial intelligence, IJCAI'05, pages 659–664, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc. (Cité en page 33.)

[Cimiano 2005] Philipp Cimiano, Günter Ladwig et Steffen Staab. *Gimme' the context : context-driven automatic semantic annotation with C-PANKOW.* In WWW, pages 332–341, 2005. (Cité en pages 33 et 34.)

[Cimiano 2006a] P. Cimiano, M. Hartung et E. Ratsch. *Finding the Appropriate Generalization Level for Binary Ontological Relations Extracted from the Genia Corpus*, 2006. (Non cité.)

[Cimiano 2006b] Philipp Cimiano. *Ontology Learning and Population from Text : Algorithms, Evaluation and Applications.* PhD thesis, 2006. (Cité en pages 9, 27, 29, 31, 32 et 34.)

[Curran 2002] James R. Curran. *Ensemble methods for automatic thesaurus extraction.* In Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10, EMNLP '02, pages 222–229, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. (Cité en page 31.)

[Dale 2000] Robert Dale, H. L. Somers et Hermann Moisl, editeurs. Handbook of natural language processing. Marcel Dekker, Inc., New York, NY, USA, 2000. (Non cité.)

[de Marneffe 2008] Marie-Catherine de Marneffe et Christopher D. Manning. *The Stanford typed dependencies representation.* In Coling 2008 : Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation, CrossParser '08, pages 1–8, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics. (Cité en pages 41 et 43.)

[d'Aquin 2007] Mathieu d'Aquin, Anne Schlicht, Heiner Stuckenschmidt et Marta Sabou. *Ontology Modularization for Knowledge Selection : Experiments and Evaluations.* Database and Expert Systems Applications, pages 874–883, 2007. (Cité en page 55.)

[Eisner 1997] Jason Eisner. *An Empirical Comparison of Probability Models for Dependency Grammar.* CoRR, vol. cmp-lg/9706004, 1997. (Cité en page 44.)

[Eisner 2000] Jason Eisner. *Bilexical Grammars And Their Cubic-Time Parsing Algorithms*, Mai 22 2000. (Cité en page 45.)

[Enrico 2003] Maria Vargas-Vera Enrico, Enrico Motta et John Domingue. *AQUA : An Ontology-Driven Question Answering System.* In Stanford University, pages 24–26. AAAI Press, 2003. (Cité en page 10.)

[Frantzi 1998] Katerina T. Frantzi, Sophia Ananiadou et Jun-ichi Tsujii. *The C-value/NC-value Method of Automatic Recognition for Multi-Word Terms.* In Proceedings of the Second European Conference on Research and Advanced Technology for Digital Libraries, ECDL '98, pages 585–604, London, UK, UK, 1998. Springer-Verlag. (Cité en page 31.)

[Gaifman 1965] Haim Gaifman. *Dependency Systems and Phrase-Structure Systems.* Information and Control, vol. 8, no. 3, pages 304–337, 1965. (Cité en page 46.)

[Gillick 2009] Dan Gillick. *Sentence boundary detection and the problem with the U.S.* In Proceedings of Human Language Technologies : The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume : Short Papers, NAACL-Short '09, pages 241–244, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. (Cité en page 63.)

[Grefenstette 1992] Gregory Grefenstette. *SEXTANT : exploring unexplored contexts for semantic extraction from syntactic analysis.* In Proceedings of the 30th annual meeting on Association for Computational Linguistics, ACL

'92, pages 324–326, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics. (Cité en page 31.)

[Halliday 1976] Michael A.K. Halliday et Ruqaiya Hasan. Cohesion in english. Longman, London, 1976. (Cité en page 63.)

[Hays 1964] David G. Hays. *Dependency theory : a formalism and some observations*. Language, vol. 40, pages 511–525, 1964. (Cité en page 46.)

[Hearst 1992] Marti A. Hearst. *Automatic acquisition of hyponyms from large text corpora*. In Proceedings of the 14th conference on Computational linguistics - Volume 2, COLING '92, pages 539–545, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics. (Cité en page 32.)

[Hellwig 1986] Peter Hellwig. *Dependency Unification Grammar*. In Proceedings of the 11th coference on Computational linguistics, COLING '86, pages 195–198, Stroudsburg, PA, USA, 1986. Association for Computational Linguistics. (Non cité.)

[Hoffart 2010] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich et Gerhard Weikum. *YAGO2 : a spatially and temporally enhanced knowledge base from Wikipedia*. Research Report MPI-I-2010-5-007, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, November 2010. (Cité en page 22.)

[Hoffart 2011] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, Edwin Lewis-Kelham, Gerard de Melo et Gerhard Weikum. *YAGO2 : exploring and querying world knowledge in time, space, context, and many languages*. In Proceedings of the 20th international conference companion on World wide web, WWW '11, pages 229–232, New York, NY, USA, 2011. ACM. (Cité en page 22.)

[Jean-Louis 2011] Ludovic Jean-Louis. *Approches supervisées et faiblement supervisées pour l'extraction d'événements et le peuplement de bases de connaissances*. These, Université Paris Sud - Paris XI, Dmbre 2011. (Cité en page 34.)

[Jurafsky 2000] Daniel Jurafsky et James H. Martin. Speech and language processing : An introduction to natural language processing, computational linguistics, and speech recognition. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st édition, 2000. (Cité en pages 2, 3 et 4.)

[Kotaro 2007] Kotaro, Takahiro HARA et Shojiro NISHIO. *A Thesaurus Construction Method from Large ScaleWeb Dictionaries*. In Proceedings of the 21st International Conference on Advanced Networking and Applications, AINA '07, pages 932–939, Washington, DC, USA, 2007. IEEE Computer Society. (Cité en page 29.)

[Kübler 2009] Sandra Kübler, Ryan McDonald et Joakim Nivre. Dependency parsing. Morgan and Claypool Publishers, 2009. Synthesis Lectures on Human Language Technologies. (Cité en pages 37 et 43.)

[Kudo 2000] Taku Kudo et Yuji Matsumoto. *Japanese dependency structure analysis based on support vector machines.* In Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora : held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics - Volume 13, EMNLP '00, pages 18–25, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics. (Cité en page 45.)

[Lassila 1999] Ora Lassila et Ralph R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification.* W3c recommendation, W3C, February 1999. (Cité en page 16.)

[Lausen 2005] Holger Lausen, Ying Ding, Michael Stollberg, Dieter Fensel, Ruben L. Hernandez et Sung-Kook Han. *Semantic web portals : state-of-the-art survey.* Journal of Knowledge Management, vol. 9, no. 5, pages 40–49, Mai 2005. (Cité en page 13.)

[Lin 1998] Dekang Lin. *Automatic retrieval and clustering of similar words.* In Proceedings of the 17th international conference on Computational linguistics - Volume 2, COLING '98, pages 768–774, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics. (Cité en page 31.)

[Lin 2001] Dekang Lin et Patrick Pantel. *DIRT - discovery of inference rules from text.* In Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '01, pages 323–328, New York, NY, USA, 2001. ACM. (Cité en page 34.)

[Maedche 2001] Alexander Maedche et Steffen Staab. *Ontology Learning for the Semantic Web.* IEEE Intelligent Systems, vol. 16, no. 2, pages 72–79, Mars 2001. (Cité en pages 27, 32 et 35.)

[Manning 1999] Christopher D. Manning et Hinrich Schütze. Foundations of statistical natural language processing. MIT Press, Cambridge, MA, USA, 1999. (Cité en page 4.)

[Manning 2008] Christopher D. Manning, Prabhakar Raghavan et Hinrich Schtze. Introduction to information retrieval. Cambridge University Press, New York, NY, USA, 2008. (Non cité.)

[Marcus 1993] Mitchell P. Marcus, Mary Ann Marcinkiewicz et Beatrice Santorini. *Building a large annotated corpus of English : the penn treebank.* Comput. Linguist., vol. 19, no. 2, pages 313–330, Juin 1993. (Cité en pages 4 et 70.)

[Maruyama 1990] Hiroshi Maruyama. *Structural Disambiguation with Constraint Propagation.* In Robert C. Berwick, editeur, ACL, pages 31–38. ACL, 1990. (Cité en page 46.)

[Maynard 2008] Diana Maynard, Yaoyong Li et Wim Peters. *NLP Techniques for Term Extraction and Ontology Population.* In Proceedings of the 2008 conference on Ontology Learning and Population : Bridging the Gap between Text and Knowledge, pages 107–127, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press. (Cité en page 34.)

[Mcdonald 2006] Ryan Mcdonald. *Discriminative learning and spanning tree algorithms for dependency parsing.* PhD thesis, Philadelphia, PA, USA, 2006. AAI3225503. (Cité en page 45.)

[McDonald 2011] Ryan McDonald et Joakim Nivre. *Analyzing and integrating dependency parsers.* Comput. Linguist., vol. 37, no. 1, pages 197–230, 2011. (Cité en pages 39 et 41.)

[Mel'čuk 1988] Igor Mel'čuk. Dependency syntax : Theory and practice. State University of New York Press, 1988. (Cité en pages 38, 41, 42 et 43.)

[Mel'cuk 2011] Igor Mel'cuk. *Dependency in Language-2011.* In Wanner Leo Gerdes Kim Hajičová Eva, editeur, International Conference on Dependency Linguistics : Depling 2011, September 5-7 2011, pages 1 –16, 2011. (Cité en page 42.)

[Mustapha 2009] Nesrine Ben Mustapha, Hajer Baazaoui Zghal, Marie-Aude Aufaure et Henda Ben Ghezala. *Survey on ontology learning from Web and open issues.* In Proceedings of the Third international conference on Innovation and Information and Communication Technology, ISIICT'09, pages 1–1, Swinton, UK, UK, 2009. British Computer Society. (Cité en page 29.)

[Nikula 1986] H. Nikula. Dependensgrammatik. Liber, 1986. (Cité en page 38.)

[Nivre 2003] Joakim Nivre. *An efficient algorithm for projective dependency parsing.* In Proceedings of the 8th International Workshop on Parsing Technologies (IWPT), pages 149–160, 2003. (Cité en page 45.)

[Nivre 2005a] Joakim Nivre. *Dependency Grammar and Dependency Parsing.* Rapport technique, Växjö University : School of Mathematics and Systems Engineering, 2005. (Cité en pages 38, 44 et 46.)

[Nivre 2005b] Joakim Nivre, Johan Hall, Sandra Kübler et Erwin Marsi. *Maltparser : A language-independent system for data-driven dependency parsing.* In In Proc. of the Fourth Workshop on Treebanks and Linguistic Theories, pages 13–95, 2005. (Cité en page 45.)

[Nivre 2006] Joakim Nivre. *Two Strategies for Text Parsing.* SKY Journal of Linguistics, vol. 19, pages 440–448, 2006. (Cité en page 44.)

[Nivre 2007] Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel et Deniz Yuret. *The CoNLL 2007 Shared Task on Dependency Parsing.* In Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007, pages 915–932, 2007. (Cité en page 40.)

[Patel-Schneider 2002] Peter F. Patel-Schneider et Dieter Fensel. *Layering the Semantic Web : Problems and Directions.* In Proceedings of the First International Semantic Web Conference on The Semantic Web, ISWC '02, pages 16–29, London, UK, UK, 2002. Springer-Verlag. (Cité en pages 13, 14 et 15.)

[Poesio 2008] Massimo Poesio et Abdulrahman Almuhareb. *Extracting concept descriptions from the Web : the importance of attributes and values.* In Proceedings of the 2008 conference on Ontology Learning and Population : Bridging

the Gap between Text and Knowledge, pages 29–44, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press. (Cité en page 32.)

[Polguère 2009] Alain Polguère et Igor A. Mel'čuk, editeurs. Dependency in linguistic description. Studies in Language Companion Series 111. John Benjamins, Philadelphia, 2009. (Cité en page 38.)

[Ponzetto 2007] Simone Paolo Ponzetto et Michael Strube. *Deriving a large scale taxonomy from Wikipedia.* In Proceedings of the 22nd national conference on Artificial intelligence - Volume 2, AAAI'07, pages 1440–1445. AAAI Press, 2007. (Cité en page 21.)

[Sanchez 2005] David Sanchez et Antonio Moreno. *Web-scale taxonomy learning.* In C. Biemann et G. Pass, editeurs, roceedings of the Workshop on Extending and Learning Lexical Ontologies using Machine Learning Methods, 2005. (Cité en page 32.)

[Sgall 1986] Petr Sgall, Eva Hajičová et Jarmila Panevová. The Meaning of the Sentence and Its Semantic and Pragmatic Aspects. Academia/Reidel Publishing Company, Prague, Czech Republic/Dordrecht, Netherlands, 1986. (Cité en pages 41 et 43.)

[Shamsfard 2004] Mehrnoush Shamsfard et Ahmad Abdollahzadeh Barforoush. *Learning ontologies from natural language texts.* Int. J. Hum.-Comput. Stud., vol. 60, no. 1, pages 17–63, Janvier 2004. (Cité en page 34.)

[Sharma(sachdeva) 2012] Ritu Sharma(sachdeva), M. Afshar Alam et Anita Rani. *Article : K-Means Clustering in Spatial Data Mining using Weka Interface.* IJCA Proceedings on International Conference on Advances in Communication and Computing Technologies 2012, vol. ICACACT, no. 1, pages 26–30, August 2012. Published by Foundation of Computer Science, New York, USA. (Cité en page 91.)

[Sowa 2000] John F. Sowa. *Ontology, Metadata, and Semiotics.* In Proceedings of the Linguistic on Conceptual Structures : Logical Linguistic, and Computational Issues, ICCS '00, pages 55–81, London, UK, UK, 2000. Springer-Verlag. (Cité en page 29.)

[Staab 2009] Steffen Staab et Rudi Studer. Handbook on ontologies. Springer Publishing Company, Incorporated, 2nd édition, 2009. (Cité en pages 27, 31 et 32.)

[Studer 2003] Rudi Studer, Andreas Hotho, Gerd Stumme et Raphael Volz. *Semantic Web - State of the Art and Future Directions.* KI, vol. 17, no. 3, pages 5–, 2003. (Cité en pages 13, 14 et 17.)

[Suchanek 2007] Fabian M. Suchanek, Gjergji Kasneci et Gerhard Weikum. *Yago : A Core of Semantic Knowledge.* In 16th international World Wide Web conference (WWW 2007), New York, NY, USA, 2007. ACM Press. (Cité en page 21.)

[Suchanek 2008] Fabian M. Suchanek, Gjergji Kasneci et Gerhard Weikum. *YAGO : A Large Ontology from Wikipedia and WordNet*. Elsevier Journal of Web Semantics, 2008. (Cité en page 22.)

[Tanev 2006] Hristo Tanev Tanev. *Weakly Supervised Approaches for Ontology Population*. In In Proceedings of EACL-2006, pages 3–7, 2006. (Cité en page 34.)

[Tapanainen 1997] Pasi Tapanainen et Timo Järvinen. *A non-projective dependency parser*. In Proceedings of the fifth conference on Applied natural language processing, ANLC '97, pages 64–71, Stroudsburg, PA, USA, 1997. Association for Computational Linguistics. (Non cité.)

[Tellier 2008] Isabelle Tellier. Introduction au TALN et à l'ingénierie linguistique. D, 2008. (Cité en pages 2, 4 et 89.)

[Tesnière 1959] Lucien Tesnière. Éléments de syntaxe structurale. Éditions Klinksieck, 1959. (Cité en pages 38 et 42.)

[Turney 2001] Peter D. Turney. *Mining the Web for Synonyms : PMI-IR versus LSA on TOEFL*. In Proceedings of the 12th European Conference on Machine Learning, EMCL '01, pages 491–502, London, UK, UK, 2001. Springer-Verlag. (Cité en page 31.)

[Völker 2009] Johanna Völker. *Learning expressive ontologies*. PhD thesis, 2009. (Cité en page 33.)

[Weber 2006] Nicolas Weber et Paul Buitelaar. *Web-based Ontology Learning with ISOLDE*. In Proc. of the Workshop on Web Content Mining with Human Language at the International Semantic Web Conference, 2006. (Cité en page 21.)

[Wu 2007] Fei Wu et Daniel S. Weld. *Autonomously semantifying wikipedia*. In Proceedings of the sixteenth ACM conference on Conference on information and knowledge management, CIKM '07, pages 41–50, New York, NY, USA, 2007. ACM. (Cité en page 21.)

[Yamaguchi 1998] Takahira Yamaguchi, , Takahira Yamaguchi et Masaki Kurematsu. *A Legal Ontology Rapid Development Environment Using a Machine Readable Dictionary*, 1998. (Cité en page 29.)

[Zhou 2007] Lina Zhou. *Ontology learning : state of the art and open issues*. 2007. (Cité en page 8.)