

Classifier pour extraire : représentations et méthodes

PATRICK MARTY*

FABIEN TORRE†

GRAppA - Université Charles de Gaulle - Lille 3

Résumé

Dans le cadre de la recherche d'information, cet article s'intéresse à l'extraction d'information (EI) et plus spécifiquement à l'utilisation des techniques de classification supervisée (CS) pour cette tâche. Nous y présentons les différentes problématiques soulevées par ce passage de l'EI à la CS et nous explicitons ce passage en supervisé pour trois systèmes d'extraction existants : BWI [FK00], IBWI [KB00] et ME [CN02]. Nous proposons ensuite deux nouveaux codages pour un problème d'EI: l'un basé sur les motifs, l'autre sur une description purement attribut-valeur. Les premières expérimentations menées avec ces codages sur des données classiques en EI, nous permettent de démontrer la pertinence de l'utilisation de la CS et l'intérêt de nos codages, mais aussi de mettre en évidence de nouvelles difficultés. Nous concluons finalement en mettant en balance les avantages et les inconvénients amenés par l'utilisation de la CS pour une tâche d'EI et en dressant la liste des pistes ouvertes.

Abstract

Classer pour extraire : représentations et méthodes

PATRICK MARTY[‡]

FABIEN TORRE^{**}

Introduction

Ces dernières années, Internet et le World Wide Web ont connu un développement notable. Le WEB est devenu la plus grande source de données et d'informations de la planète. Devant le nombre croissant de documents électroniques disponibles en ligne, le besoin d'outils performants pour rechercher les documents correspondant à un critère d'intérêt particulier est de plus en plus prégnant. C'est le propos de la *recherche d'information* (RI).

Une sous-problématique de la RI est l'*extraction d'information* (EI). Elle consiste à extraire automatiquement des données d'un document ou d'un corpus de documents.

Par exemple, avec le corpus *Seminar Announcements* *, composé de 485 annonces de séminaires, il s'agit d'extraire, pour chaque document, l'heure de début (*stime*) et l'heure de fin (*etime*) de la conférence, l'endroit où elle a lieu (*location*) et le nom de l'intervenant (*speaker*). Sur cet exemple, il y a quatre données différentes à extraire : dans la terminologie de l'EI, on parle de *champ* ou de *slot*. Par ailleurs, un champ donné peut avoir une ou plusieurs instances dans un même document (pour une conférence, il peut y avoir plusieurs intervenants).

Diverses techniques d'EI ont été développées, chacune dédiée à un format précis de documents, du texte libre en langage naturel au fichier XML à la structure rigide. Certains systèmes réalisent l'extraction à l'aide d'une analyse du sens du texte du document et emploient des techniques de traitement du langage naturel [SFAL95, Huf95]. D'autres au contraire se fondent essentiellement sur l'aspect syntaxique ou structurel du document. Dans ce dernier cas, un document peut être vu soit comme une séquence « plate » [Kus97, HD98], soit comme un arbre [SMAA01, KdBBB02] (notamment dans le cas de documents HTML ou XML). Malgré ces distinctions, tous ces systèmes partagent le but de produire un *wrapper* : un programme d'EI qui produit des données structurées à partir de documents. L'écriture manuelle d'un tel programme étant fastidieuse et source d'erreurs, les documents eux-mêmes étant changeants, les recherches se sont orientées vers la génération automatique, autrement dit l'induction de wrappers.

Une manière de réaliser cette induction consiste à la ramener à un problème de *classification supervisée* (CS), puis à utiliser l'un des nombreux algorithmes dédiés à ce type de tâches [CM02]. En CS, on suppose fournis des exemples étiquetés par leurs classes ; ainsi, dans le cas le plus simple (à deux classes), chaque exemple pourra être soit *positif*, soit *néгатif*. À partir de ces données, il s'agit alors d'apprendre un

*disponible à <http://www.isi.edu/info-agents/RISE/>.

classifieur capable de prédire la classe d'un nouvel exemple. Les premiers travaux utilisant la CS pour l'EI montrent que la piste est prometteuse [SMR99, FM99, FK00, FM00, KB00, CN02].

Dans cet article, nous nous plaçons dans le cadre des systèmes n'utilisant que l'aspect *séquence de tokens* des documents et poursuivons l'exploration des méthodes permettant de passer d'un problème d'EI à un problème de CS. Le plan est le suivant :

- nous discutons à la section 1 des différentes problématiques soulevées par ce passage de l'EI à la CS et des réponses apportées par les principaux systèmes existants ;
- à la section 2, nous proposons deux nouveaux codages d'un problème d'EI : l'un basé sur les motifs, l'autre sur une description purement attribut-valeur ;
- nous décrivons ensuite, section 3, les premières expérimentations menées avec ces codages sur des données classiques en EI et présentons les résultats obtenus ;
- enfin, nous concluons à la section 4 en tirant un premier bilan de ce travail et en indiquant les perspectives qu'il ouvre.

1 Passages en CS

En EI, les wrappers sont généralement induits à partir d'exemples dans lesquels l'information à extraire est marquée. Il est donc naturel d'envisager l'utilisation de la CS pour résoudre ce problème ; d'autant plus que de nombreuses méthodes sont à disposition, souvent accompagnées de garanties théoriques ou empiriques [CM02].

La définition d'un problème d'apprentissage passe par le choix de deux langages, le plus souvent le premier étant inclus dans le deuxième [CF82] :

- le langage permettant de représenter les exemples ;
- le langage permettant de représenter les hypothèses (celui-ci correspondra à l'espace de recherche de l'algorithme d'apprentissage).

Nous discutons maintenant de la définition de ces différents langages.

1.1 Des textes aux exemples

La première étape a pour but d'obtenir des données structurées à partir des textes fournis en langage naturel. La structuration la plus élémentaire consiste à transformer chaque texte en une suite d'unités syntaxiques ou *tokens* de longueur fixe. Par exemple, on peut découper le texte caractère par caractère. Des découpages plus fins sont possibles si l'on caractérise les endroits où couper (par exemple, dès qu'il y a suite d'espaces, de tabulations, ou de passages à la ligne) ou les tokens eux-mêmes (séquence de caractères alphabétiques, séquence de caractères numériques, caractère de ponctuation, séquence d'espaces ou de tabulations, etc.). Ces types de tokens peuvent être généralisés ou raffinés suivant les connaissances disponibles : par exemple, **Token** dénote n'importe quel token de n'importe quel type ; **PasMotAnglais** désigne tout token de type alphabétique qui ne se trouve pas dans le dictionnaire anglais disponible sur **Unix/Linux**, etc.

Finalement, chaque token sera accompagné de son type celui-ci pouvant aller du plus général (token quelconque) au plus spécifique (le contenu du token lui-même)

en passant par des types intermédiaires (*nom propre* par exemple). Ainsi, les espaces étant ignorés, le texte *'Time : 9 PM'* peut être représenté par la séquence de tokens

Alpha('Time') Ponct(':') Num('9') Alpha('PM')

ou encore par

Alpha('Time') Ponct(':') Num('9') PasAnglais('PM')

Du processus de segmentation d'un document en tokens, émerge la notion de *séparateur*. Un séparateur est une interposition entre deux tokens adjacents.

En vue de l'apprentissage, il faut maintenant utiliser la séquence des tokens et des séparateurs représentant le texte pour constituer une base d'exemples. Un préalable est de décider de ce que va représenter un exemple. Pour cela, deux choix sont possibles :

- un exemple code un séparateur (de début ou de fin d'une donnée à extraire ou tout autre séparateur) ; dans ce cas, les apprentissages des débuts et des fins se feront indépendamment ;
- un exemple code une sous-séquence ; par rapport au premier, ce choix revient à apprendre en même temps les caractérisations des débuts et des fins.

La description des exemples étant fixée, il restera à définir un langage d'hypothèses : l'apprentissage y cherchera la meilleure définition séparant les exemples positifs des exemples négatifs.

Enfin, une hypothèse étant apprise, nous aurons à préciser son mode d'utilisation pour extraire des données de nouveaux textes. On retrouvera systématiquement le codage de ces textes sous forme d'exemples et notre hypothèse désignera parmi eux les exemples positifs. Si les exemples codent les sous-séquences, le processus s'arrête là : les chaînes de caractères associées aux exemples identifiés comme positifs sont extraites. Par contre, si les exemples codent des séparateurs, il restera encore à associer chaque séparateur reconnu comme un début à un séparateur reconnu comme une fin. Cette chaîne de transformation est illustrée à la figure 1.

Dans la suite de cette section, nous présentons trois systèmes existants (BWI [FK00], IBWI [KB00] et ME [CN02]) en indiquant pour chacun d'eux :

- les tokens considérés ;
- la nature des exemples choisie ;
- la famille d'hypothèses utilisée et le mode d'extraction associé ;
- l'algorithme de CS mis en œuvre.

1.2 bwi [FK00]

Représentation des exemples dans BWI.

Dans BWI, un token est soit une séquence de caractères alphanumériques, soit un caractère de ponctuation, soit le caractère *'retour chariot'*. Un exemple est un séparateur dans un texte donné. Implicitement, un exemple dans BWI est donc une partition du texte initial en deux séquences de tokens (la séquence avant le séparateur considéré et la séquence qui vient après ce séparateur). Cela amène à trois types d'exemples selon qu'ils représentent des séparateurs de début, des séparateurs de fin ou des séparateurs quelconques.

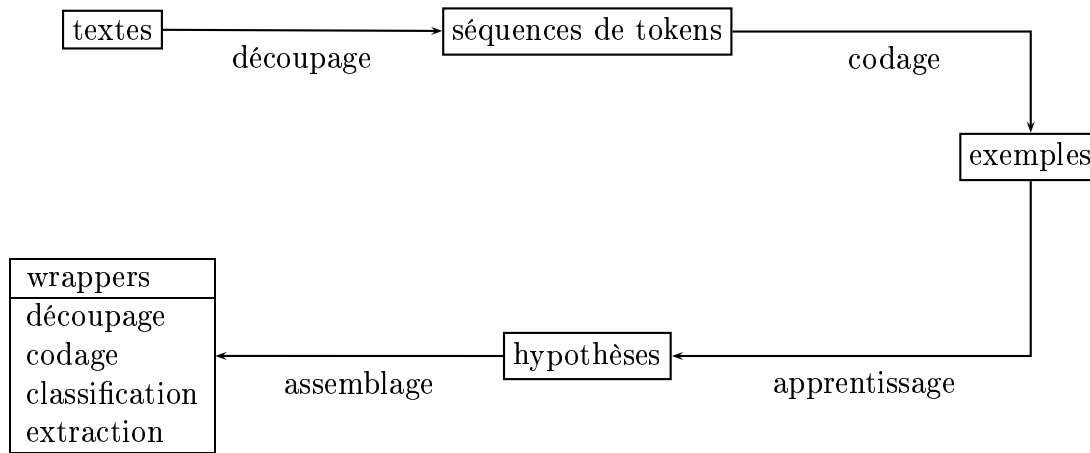


FIG. 1 – Chaîne de transformation

Représentation des hypothèses dans BWI.

Les hypothèses utilisent ici des motifs comparables à des expressions régulières et portant sur les tokens et leurs types. Précisément, une hypothèse est un couple (g, d) de tels motifs : cette hypothèse couvre un séparateur donné si g matche la séquence de tokens avant le séparateur (à sa gauche) et si d matche la séquence de tokens après le séparateur (à sa droite). Par exemple, l'hypothèse ('Time' Token , Num Ponct Num PasAnglais) couvre le séparateur noté par | dans le texte 'Time : | 9 :30 PM'. La forme d'un motif, et donc sa longueur, ne sont pas fixées *a priori* mais découvertes par apprentissage.

Apprentissage par BWI.

L'induction du wrapper, pour un champ, consiste tout d'abord à utiliser deux fois la CS :

- une première fois avec les exemples de début qui jouent le rôle des *positifs* et tous les autres exemples utilisés comme *négatifs* ; cela fournit un classifieur caractérisant les exemples de type *début* ;
- une seconde fois pour caractériser les exemples de type *fin* (qui cette fois jouent le rôle des *positifs*, tous les autres étant considérés comme *négatifs*).

Pour effectuer ces apprentissages, BWI utilise l'algorithme de boosting ADABOOST [FS97, SS98]. Cette méthode nécessite la donnée d'un *apprenant faible*, c'est-à-dire d'un algorithme d'apprentissage faisant un peu mieux que l'aléatoire. Dans BWI, cet apprenant se nomme LEARNDETECTOR. Il construit itérativement une hypothèse par extension de ses motifs (initialement vides). À chaque étape, un motif peut être étendu par l'ajout d'au plus L tokens ou types. Tous les motifs possibles, à gauche et à droite, sont énumérés et celui contribuant le plus à améliorer les performances de classification de l'hypothèse est conservé. Ce processus est répété tant qu'il est possible d'améliorer l'hypothèse courante.

Extraction par BWI.

Les deux classifieurs produits permettent d'identifier dans un nouveau texte les séparateurs de début et de fin : il reste à les ré-associer. Pour cela, BWI enregistre les longueurs des séquences de tokens d'un champ observées sur les données d'apprentissage. Deux séparateurs, le premier identifié comme un début et le second comme une fin, doivent être associés si la longueur de la séquence de tokens qu'ils délimitent a déjà été observée lors de l'apprentissage.

Critique de BWI.

L'algorithme LEARNDETECTOR génère les règles par extension en essayant d'ajouter à chaque fois au motif une fenêtre de taille L . Au final, un motif peut couvrir l'ensemble du texte et la complexité de LEARNDETECTOR est exponentielle en L . Un autre défaut majeur de BWI est qu'il considère les séparateurs de début et de fin indépendamment. D'une part, la ré-association des débuts et des fins est délicate : elle utilise une liste mémorisant les longueurs observées lors de l'apprentissage mais aucun lissage n'est appliqué et cette liste peut donc être *trouée*. Par exemple, si BWI n'a rencontré que des séquences de longueur 1, 3, 4, et 5, aucune séquence de taille 2 ne sera extraite même si les séparateurs la délimitant sont identifiés comme étant respectivement un début et une fin. D'autre part, il y a une perte d'information à dissocier les débuts et les fins pendant l'apprentissage : la description d'un séparateur n'est pas informée du séparateur qui lui est associé ce qui empêche de découvrir des règles comme, par exemple, *le séparateur de début est précédé du caractère « : » et le séparateur de fin suivi du caractère « . »*.

1.3 ibwi [KB00]

Ce système est né du constat que beaucoup de documents ont une structure régulière et qu'il est par conséquent probablement pertinent d'exploiter les similarités entre les documents. IBWI utilise pour cela une approche de type *k-plus proches voisins* : dans ce cas les nouveaux exemples à classer sont simplement comparés aux exemples connus.

Représentation des exemples dans IBWI.

Contrairement à BWI, un exemple dans IBWI n'utilise pas la notion de séparateurs mais code directement une séquence de tokens. IBWI se limite aux séquences de taille inférieure à la taille maximale des séquences à extraire, observée sur les données d'apprentissage. Par conséquent, le nombre d'exemples peut devenir très grand ; à ce propos, les auteurs indiquent que lors des expérimentations, IBWI n'utilise qu'un quart à la moitié des exemples par manque de place en mémoire. Dans IBWI, chaque exemple est décrit par un ensemble d'attributs donnés à la table 1 (notons que certains de ces attributs peuvent être difficiles à renseigner sur les textes à traiter mais cette question n'est pas discutée par les auteurs).

TAB. 1 – Les attributs décrivant un exemple dans IBWI

- la valeur littérale de la séquence de tokens de l'exemple ;
- la longueur de cette séquence ;
- sa position depuis le début du document ;
- le type de la séquence (*alphanumérique*, *alphabétique*, ou *numérique*) ;
- une information sur le premier caractère du premier token : *majuscule*, *minuscule*, ou *autre* s'il ne s'agit pas d'une lettre ;
- les deux séquences à extraire voisines dans le document ;
- les trois symboles spéciaux (comme ; et :) rencontrés à gauche et à droite de la séquence.

Apprentissage par IBWI.

L'EI est ramenée à la classification m -aire d'exemples : il y a une classe pour chacun des m champs à extraire. Chaque attribut est doté d'un poids et pour chaque type d'attribut une distance spécifique est définie. Pour comparer deux objets, la distance entre chaque attribut est calculée à l'aide de la distance appropriée et la similarité entre les deux objets est définie comme la moyenne pondérée des distances calculées par attribut. L'apprentissage consiste à régler les poids des attributs. La procédure employée par les auteurs consiste à fixer les poids empiriquement et à évaluer les performances de IBWI par validation croisée sur l'ensemble d'apprentissage. Les valeurs conduisant aux meilleures performances sont conservées.

Extraction par IBWI.

Pour extraire des informations d'un document, toutes les séquences de taille inférieure à la taille limite sont transformées en exemples attribut-valeur. Puis, pour chaque exemple, et pour chaque classe, on récupère les k exemples les plus proches. On observe alors la distance moyenne de l'exemple à chaque classe et on s'intéresse à la plus petite de ces distances : si elle est supérieure à un seuil d'extraction fixé, l'exemple n'est associé à aucune des classes définies ; si elle est inférieure à ce seuil, l'exemple est étiqueté par la classe correspondante et la séquence de tokens dudit exemple est extraite.

Critique de IBWI.

Comme tous les systèmes à base de k -plus proches voisins, le point sensible est la définition de la distance entre exemples et le réglage du k . Cette difficulté est ici aggravée par le fait que IBWI requiert en plus la valeur du seuil d'extraction. Les auteurs indiquent que ces paramètres sont fixés empiriquement, selon le problème d'extraction considéré : la valeur de k entre 1 et 3 et le seuil entre 0.25 et 0.6. Enfin, il nous semble que le réglage empirique des paramètres de IBWI est susceptible de produire un wrapper trop spécifique, uniquement dédié au corpus à traiter.

1.4 me [CN02]

Représentation des exemples dans ME.

Un exemple représente un mot (dans son sens usuel) et est codé par un vecteur attribut/valeur. ME utilise des attributs (table 2) qui portent sur la syntaxe et la sémantique des mots. ME définit quatre classes par champ à extraire plus une classe supplémentaire. Clairement, la classe d'un exemple représentant un mot m sera :

- *c-debut* si le m est le premier mot d'un champ c ;
- *c-fin* si le m est le dernier mot d'un champ c ;
- *c-milieu* si m se trouve dans un champ c sans être ni le premier ni le dernier mot ;
- *c-unique* si m est le seul mot d'un champ c ;
- *pas-un-champ* si m n'est pas dans une partie du texte à extraire.

Apprentissage par ME.

ME se fonde sur le modèle probabiliste du maximum d'entropie qui permet d'estimer la probabilité d'appartenance d'un exemple à chacune des classes prises en compte. [BDPDP96, Rat]. Chacun des attributs décrivant les exemples définit une contrainte sur le modèle. L'apprentissage d'un classifieur consiste à déterminer les poids des attributs d'après l'ensemble d'apprentissage, afin d'obtenir le modèle de plus grande entropie satisfaisant les contraintes. Ainsi le langage d'hypothèses n'est pas explicite.

Extraction par ME.

Le modèle du maximum d'entropie associé à un mot, une fois transformé en exemple, la probabilité d'appartenir à chacune des classes. Ainsi à la séquence de mots d'un document correspond plusieurs séquences de classes possibles et, souvent, choisir la classe la plus probable pour chaque mot conduit à une aberration. Par exemple, avec deux champs c_1 et c_2 à extraire, on pourrait obtenir la séquence de classes suivante : ... *c₂-debut c₁-fin c₂-milieu c₁-debut* ... Pour obtenir une séquence de classes cohérente, des contraintes sur la succession des classes dans une séquence sont définies par les auteurs (par exemple on ne peut avoir *c-milieu* avant *c-debut*). Un algorithme, semblable à celui de VITERBI pour les HMM [Jr.73], permet de trouver la séquence de classes cohérente la plus probable. Finalement, pour chaque champ c , l'information extraite est la séquence de mots associée à la sous-séquence de classes de la forme *c-debut c-milieu ... c-milieu c-fin* ou bien *c-unique*.

Critique de ME.

Certains attributs sont adaptés au corpus Seminar Annoncements, comme par exemple l'attribut *le mot avant le dernier ' : ' à gauche de m*. En effet, dans une annonce de séminaire, l'heure de début est souvent précédée du texte « *Time* : », l'intervenant de la conférence peut être annoncé par « *Speaker* : » et le lieu par « *Location* : ». C'est pourquoi cet attribut n'est pas assez général et trop dépendant du contexte des données à extraire.

TAB. 2 – Attributs utilisés par ME pour représenter un mot m (m_{-i} et m_{+i} dénotent respectivement les mots précédant et suivant m).

Chaînes de caractères
<ul style="list-style-type: none"> – m, m_{-1}, m_{+1} – (m_{-2}, m_{-1}) et (m_{+1}, m_{+2}) – le mot avant le dernier ‘:’ à gauche de m
Booléens
<ul style="list-style-type: none"> – m commence par une majuscule et se trouve dans une phrase ? – idem pour $(m_{-1}$ et $m_{+1})$ – m est le premier mot d’une phrase ? – m correspond à l’expression régulière $[0-9]^+ : [0-9]^+ ?$ – m se trouve dans la liste des noms de famille les plus répandus ? – m se trouve dans la liste des prénoms les plus courants ? – m est présent dans le dictionnaire anglais de Unix/Linux ?

2 Nouveaux codages

Dans cette section, nous repartons des constatations négatives émises sur les systèmes présentés à la section précédente, à savoir :

- les motifs de BWI peuvent couvrir une portion très grande du texte ;
- on perd à apprendre les séparateurs de début et de fin indépendamment ;
- certains codages comme celui de ME utilise des descripteurs *ad hoc*.

Dans la suite, nous proposons de nouveaux motifs pour repérer des séparateurs, puis une représentation attribut-valeur riche et générale portant sur la chaîne à extraire. Les résultats des expérimentations menées sur ces nouveaux codages sont présentés section 3.

2.1 Description par motifs

Notre but est ici d’observer les performances d’un système proche de BWI, en considérant toujours les séparateurs indépendamment, mais avec un langage d’hypothèses plus simple.

Représentation des exemples.

Les exemples considérés sont les séparateurs indiquant le début ou la fin d’un champ et sont représentés à la manière de BWI.

Représentation des hypothèses.

Par contre, pour nous, une hypothèse est un couple de motifs dont la forme est fixée *a priori*, et donc la taille maximale d’un motif est connue. Une forme de motif

gauche possible est $token_{g_1} * token_{g_2}$. Un tel motif matche les tokens à gauche d'un séparateur donné si $token_{g_2}$ est le token immédiatement à gauche du séparateur, et si $token_{g_1}$ se trouve au plus F tokens à gauche dudit séparateur. Le symbole $*$ désigne n'importe quelle séquence de tokens de longueur au plus $F - 2$ et peut être vu comme une généralisation de plusieurs tokens. Les formes de motifs essayées conduisent à différentes classes d'hypothèses présentées à la table 3. Par exemple, les hypothèses de la classe $H_{1,4}$ (*i.e.* $F = 4$) couvrant le séparateur `|` dans le texte `'Time : | 9 :30 PM'` sont `('Time' ' : ' , '9' ' :')`, `('Time' ' : ' , '9' '30')` et `('Time' ' : ' , '9' 'PM')`. Pour la classe $H_{2,4}$, on a `('Time' Ponct , Num ' :')`, `('Time' Ponct , Num '30')` et `('Time' Ponct , Num 'PM')`.

Apprentissage.

Comme dans BWI, les hypothèses sont apprises pour les exemples positifs seulement. Par contre, la classe d'hypothèses étant fixée *a priori*, toutes les hypothèses caractérisant les exemples positifs peuvent être décrites par énumération avant le boosting. Au cours du boosting, le rôle de l'apprenant faible se limite à la recherche de la meilleure règle en fonction des poids des exemples.

2.2 Description par attribut-valeur

L'objectif est ici de revenir à un langage d'hypothèses plus simple (l'attribut-valeur) et de proposer un codage qui se voudrait général, valable pour tout problème d'extraction. Dans cette démarche, nous avons décidé de multiplier les attributs avec l'idée d'utiliser ensuite un apprenant capable de sélectionner parmi eux les plus intéressants.

Représentation des exemples.

Un exemple est ici un vecteur d'attributs destiné à représenter une séquence extraite d'un texte et délimitée par ses séparateurs de début et de fin. Les attributs utilisés sont décrits à la table 4. Les différents types de tokens font qu'un exemple est décrit par une cinquantaine d'attributs. Certains attributs portent sur la séquence elle-même et son entourage proche (on peut apprendre sur ce qu'il y a à gauche et à droite ce qui est un avantage sur les codages par séparateurs indépendants) et d'autres sur l'environnement plus lointain (distances au début du document ou dernier token de type *numérique* rencontré, etc.). Comme les exemples dénotent des sous-séquences, il n'y a que deux classes possibles : *positif* si la séquence dont est issu l'exemple est à extraire, *négatif* sinon.

TAB. 3 – Différentes classes d'hypothèses utilisées

$H_{1,F}$	$(token_{g_1} * token_{g_2}, token_{d_1} * token_{d_2})$
$H_{2,F}$	$(token_{g_1} * type_{g_2}, type_{d_1} * token_{d_2})$
$H_{3,F}$	$(type_{g_1} * type_{g_2}, type_{d_1} * type_{d_2})$

TAB. 4 – Attributs décrivant une sous-séquence repérée par les délimiteurs s_d et s_f .

Séquence de tokens entre s_d et s_f
<ul style="list-style-type: none"> – le texte entre s_d et s_f ; – la longueur de cette chaîne ; – la chaîne de types correspondante ; – pour chaque type, son nombre d’occurrences entre s_d et s_f ; – la distance relative de s_d au début du document ; – la distance relative de s_f à la fin du document ; – le numéro de ligne à laquelle se trouve s_d ;
Séquence des F tokens à gauche de s_d (respectivement à droite de s_f)
<ul style="list-style-type: none"> – la chaîne littérale des valeurs de ces tokens ; – la chaîne de types correspondante ;
Types à gauche de s_d (respectivement à droite de s_f)
<ul style="list-style-type: none"> – pour chaque type, la distance au premier token de ce type à gauche de s_d (respectivement à droite de s_f) ; – les valeurs de ces tokens.

Représentation des hypothèses.

Une hypothèse est une généralisation des exemples décrits ci-dessus : elle est construite sur les mêmes attributs mais peut présenter des valeurs supplémentaires :

- le symbole ? pour signifier que l’attribut est sans importance dans la règle, cette valeur est la plus générale et couvre n’importe quelle autre valeur ;
- les attributs de type numérique peuvent avoir comme valeur un intervalle $[min, max]$; dans ce cas, une valeur sera couverte si elle est comprise entre min et max .

Apprentissage.

Nous nous sommes livrés à deux types d’apprentissage sur les données ainsi reformulées, tous deux basés sur une opération de moindre-généralisé [Tor99]. Cette opération consiste à calculer l’hypothèse la plus spécifique qui couvre un ensemble d’exemples donnés (dans le langage d’hypothèses que nous avons défini, celle-ci est unique). Pour l’apprentissage, l’idée est de construire pour chaque classe des moindres-généralisés corrects, c’est-à-dire qui ne couvrent pas d’exemples d’autres classes. Ceci nous permet :

- d’apprendre une théorie compréhensible avec GloBo [Tor99], un système stochastique ayant pour brique de base le calcul de moindres-généralisés corrects ;
- d’apprendre un classifieur performant grâce au boosting en utilisant le calcul de moindre-généralisé correct comme apprenant faible.

3 Expérimentations

3.1 Codage par motifs

Les expériences ont été réalisées par cinq étapes de validation croisée sur le corpus Seminar Annoncements. Après codage des textes en exemples, il apparaît immédiatement qu’un déséquilibre fort existe entre les classes : 800 positifs contre 220 000 négatifs.

Pour réduire ce déséquilibre, les exemples négatifs choisis pour l’apprentissage sont ceux proches des positifs. Plus précisément, les séparateurs négatifs conservés pour l’apprentissage sont ceux se trouvant dans une fenêtre de taille F tokens à gauche et à droite d’un séparateur positif. Ce choix est motivé par la conviction que ces séparateurs négatifs constituent des exemples critiques. Sur le corpus Seminar Annoncements, cette procédure isole 3.6% des exemples négatifs : la base d’exemples destinée à l’apprentissage comporte alors 800 positifs et 8000 négatifs. Précisons bien que cette sélection n’est faite que sur les données d’apprentissage : en phase de test du wrapper, celui-ci est évalué sur tous les séparateurs des documents.

Comme le montrent les résultats des expériences (table 5) sur le corpus Seminar Annoncements, on obtient pour le champ *stime* une F -mesure de 89.5%. Ce score honorable montre que la stratégie de sélection ramène des exemples négatifs réellement pertinents. Cependant, la comparaison avec le score obtenu par BWI sur le même champ (99.6%) nous indique que cette sélection a été trop restrictive et qu’elle a écarté certains exemples nécessaires pour l’apprentissage. De plus, les règles employées sont moins précises que celles de BWI : il en faut plusieurs pour décrire les

séquences de tokens que l'on doit trouver autour d'un exemple positif. Ainsi l'information permettant de discriminer un séparateur particulier se trouve répartie dans plusieurs règles. Si seulement quelques-unes d'entre elles sont sélectionnées au cours de l'apprentissage, alors la caractérisation d'un séparateur est incomplète et donc erronée. Enfin, nous supposons que l'espace d'hypothèses n'est pas assez riche (même en jouant sur la valeur F) et que la difficulté de cette approche est de trouver le bon type de motifs.

TAB. 5 – Performances des différentes classes d'hypothèses

Data Set	$H_{1,5}$			$H_{2,5}$			$H_{3,5}$		
	P	R	F1	P	R	F1	P	R	F1
SA-stime	0.898	0.893	0.895	0.887	0.885	0.886	0.730	0.012	0.024
SA-etime	0.693	0.906	0.783	0.760	0.937	0.836	0.055	0.056	0.056
SA-location	0.578	0.419	0.481	0.476	0.412	0.440	0.127	0.038	0.058
SA-speaker	0.537	0.403	0.456	0.424	0.364	0.390	0.191	0.066	0.095

Devant cette constatation, nous nous sommes orientés vers une autre méthode de classification, en apprenant cette fois les séparateurs de début et de fin simultanément, comme décrit à la section 2.2.

3.2 Codage attribut-valeur

Les expérimentations ont été réalisées sur les données *etime* (heure de fin) du corpus Seminar Annoncements selon le protocole classique où 70% des exemples servent en apprentissage et 30% sont laissés pour le test.

Ce protocole nous donne pour l'apprentissage 427 exemples positifs contre 1 763 488 négatifs. Ce fort déséquilibre existe de part la nature même du problème mais aussi à cause de notre codage des exemples qui représentent des séquences. Dans ce cas, le nombre d'exemples négatifs augmentent de manière quadratique avec la taille du texte (linéaire seulement si les exemples représentent des séparateurs).

GloBo.

Une version linéaire de GloBo [Tor99] permet traiter ces données et d'obtenir des règles intuitives comme celles présentées à la figure 2. On observe que ces règles ont du sens et que certaines posent des conditions à la fois sur ce qui se trouve à gauche et à droite de la portion à extraire, justifiant ainsi notre abandon du codage par séparateurs isolés. De plus, on constate que GloBo est parvenu à faire le tri parmi tous les attributs disponibles pour produire des règles qui n'en utilisent que très peu.

Boosting.

Nous essayons maintenant, sur les mêmes données, d'utiliser ADABOOST avec un moindre généralisé en guise d'apprenant faible. Malheureusement, le boosting ne permet pas d'utiliser les 2 millions négatifs, surtout à cause du calcul de moindres

$\left\{ \begin{array}{l} \text{Taille du champ} \\ \text{Séquence des types à gauche} \end{array} \right.$	$= 5$
	$= \text{NUM RC ALPHA PONCT ESPACE}$
$\left\{ \begin{array}{l} \text{Types à gauche} \\ \text{Types à droite} \end{array} \right.$	$= \text{NUM RC ALPHA PONCT ESPACE}$
	$= \text{RC PAS_ANGLAIS PONCT ESPACE LAST_NAME}$
$\left\{ \begin{array}{l} \text{Taille du champ} \\ \text{Nombre de type NUM dans le champ} \\ \text{Distance au type PONCT à droite} \\ \text{Distance au type ALPHA à droite} \end{array} \right.$	$= [4.00..8.00]$
	$= 2.00$
	$= [2.00..199.00]$
	$= [2.00..160.00]$

FIG. 2 – Trois règles découvertes par GloBo et concluant sur la classe *positif*.

généralisés corrects, trop coûteux en présence d'autant de négatifs. Comme pour les motifs, notre solution a consisté à n'utiliser qu'une partie des 70% d'exemples négatifs disponibles en opérant cette fois une sélection aléatoire sur l'ensemble d'apprentissage (en test, tous les négatifs sont naturellement conservés).

Nous avons effectué les premiers essais avec 0.05% des négatifs et un nombre d'étapes de boosting fixé à 100. Avec ce paramétrage, les classes sont équilibrées, l'apprentissage est rapide et les résultats satisfaisants du point de vue de la CS : 99.22% des exemples positifs sont bien reconnus comme tels et 98.81% des négatifs sont également bien classés. Cela semble indiquer que le problème n'est pas très compliqué ; comme avec GloBo, notre représentation permet bien l'existence de règles pertinentes qui distinguent les positifs des négatifs. Malheureusement, l'interprétation de ces résultats dans le cadre de l'EI est plus décevante :

- les 99.22% de rappel obtenus sur les positifs correspondent à 128 exemples de l'ensemble de test justement reconnus comme positifs (un seul positif n'a pas été identifié) ;
- parmi les négatifs, 1.2% sont également reconnus comme positifs ; or, ces 1.2% représentent 6 350 exemples ; autrement dit, nous avons ramené environ 6 500 exemples dont seulement 128 sont réellement attendus et cela conduit grossièrement à une précision de 2% et à une F -mesure de 4%.

Naturellement, nous avons essayé d'améliorer ces résultats : en prenant en compte plus de négatifs lors de l'apprentissage et en fixant un nombre d'étapes de boosting plus élevé. L'augmentation de ces deux paramètres permettent d'améliorer la précision avec une efficacité plus grande pour le premier : les gains obtenus grâce à l'augmentation du nombre d'exemples négatifs sont présentés à la table 6. On observe que le rappel se dégrade lentement ; cependant nos essais montrent que cette dégradation peut être amortie en augmentant les étapes de boosting (passer de 100 à 200 semble suffisant). La table 6 présente également les temps de calcul. Ceux-ci incluent le temps consacré à l'évaluation du classifieur, c'est-à-dire le test des règles issues du boosting sur les 500 000 exemples de l'ensemble de test.

De nouvelles expérimentations sont en cours à plus grande échelle et avec pour objectif d'affiner nos chiffres par validation croisée. Cependant, les temps de calcul

TAB. 6 – Résultats du boosting appliqué au codage attribut-valeur des données *etime* du corpus Seminar Annoncements.

Négatifs	Rappel	Précision	F -mesure	Temps d'apprentissage
0.05 %	99.22 %	2.13 %	4.16 %	3 heures et 30 minutes
0.10 %	100.00 %	3.48 %	6.72 %	4 heures et 45 minutes
0.30 %	98.45 %	7.03 %	13.13 %	12 heures
1.00 %	94.57 %	14.65 %	25.36 %	2 jours et 12 heures
1.50 %	95.35 %	17.30 %	29.29 %	4 jours et 18 heures
3.00 %	95.35 %	18.72 %	31.30 %	18 jours

nécessaires ne nous ont pas permis d'obtenir les résultats avant la soumission de cet article.

4 Discussion et Perspectives

Dans cet article, nous avons utilisé la méthodologie classique de définition d'un problème de CS (définitions du langage des exemples, du langage des hypothèses et choix d'un algorithme d'apprentissage) pour observer sous le même angle des systèmes existants qui, pour résoudre leur problème d'EI, ont fait le choix de le transformer en un problème de CS. Nous avons constaté que les transcriptions possibles étaient multiples et en avons nous mêmes proposées deux nouvelles.

Nous passons maintenant en revue les questions qui se posent au cours de cette transcription, les éléments de réponse que nous pouvons apporter et les difficultés restées ouvertes.

Codage des exemples.

Le premier choix important consiste à déterminer ce que représente les exemples, l'alternative est clairement entre un codage des séparateurs et un codage direct des séquences. L'information portée par un exemple de type séquence est beaucoup plus riche et l'on peut donc espérer découvrir avec ce codage des régularités qui seront perdues dans le codage par séparateurs. De plus, l'extraction est triviale après la classification si l'on a codé sous forme de séquences, tandis que des séparateurs identifiés isolément doivent encore être rassemblés en couples. Malheureusement, ces avantages se payent par des exemples négatifs plus nombreux, des classes plus déséquilibrées (ce qui peut mettre en difficulté beaucoup de méthodes d'apprentissage) et finalement, par un temps de calcul plus long. L'inconvénient du temps d'apprentissage peut être relativisé : celui-ci peut être important sans que cela ne soit un inconvénient majeur, seule compte en définitive la rapidité d'extraction du wrapper obtenu.

Apprentissage supervisé.

Une fois le codage effectué, toutes les techniques de la CS peuvent être utilisées. Cependant, on ne peut pas oublier que l'on est en train de traiter un problème

d'extraction, en particulier si l'on réfléchit à la mesure à optimiser au cours de l'apprentissage. Le taux d'erreur habituel en CS n'a pas de sens ici. Les deux classes ne sont pas de même taille : la règle majoritaire qui identifie tous les exemples comme négatifs fait une erreur inférieure à 0.1%. Observer cette mesure sur les deux classes n'a pas de sens non plus car les différents types d'erreur ne sont pas équivalents : ne pas identifier un exemple négatif est sans importance mais il ne faut surtout pas le classer comme positif. C'est bien sûr ce que mesure la précision. On peut donc légitimement transformer un problème d'EI en un problème de CS mais au moment d'évaluer le résultat, il faut impérativement prendre en compte l'objectif d'origine, celui de l'EI. Il nous apparaît même probable que l'on ne puisse se passer de prendre en compte cette spécificité en cours d'apprentissage. Cette problématique a déjà été identifiée dans le domaine médical où l'on veut identifier un maximum de patients à risque en récupérant un minimum de patients sains. Dans cette branche, précision et rappel sont nommés respectivement *sensibilité* et *spécificité*, et les méthodes d'apprentissage cherchent à optimiser les deux critères en même temps. Au final, les meilleurs classificateurs obtenus, incomparables entre eux (ils constituent le front de Pareto) peuvent être visualisés sur une courbe ROC [PF97].

Perspectives.

Plusieurs voies nous semblent ouvertes par ce travail. Tout d'abord, il nous semble intéressant de poursuivre la réflexion sur la définition des langages d'hypothèses :

- trouver des patrons de motifs efficaces décrivant des séquences et non plus des séparateurs isolés (l'utilisation de techniques de la programmation logique inductive [Mug91] pourrait alors être envisagée pour l'apprentissage) ;
- enrichir encore la description attribut-valeur proposée puisque les méthodes à base de moindre-généralisé semblent le supporter ;
- trouver des codages, de type attribut-valeur ou de type motif, adapté aux documents vus comme des arbres en vue d'extraire à partir de documents HTML ou XML.

Dans nos expérimentations, nous avons été confrontés au trop grand nombre d'exemples négatifs qui rend la plupart des méthodes impraticables. Il s'agit donc d'opérer une sélection d'instances sur les exemples négatifs disponibles pour l'apprentissage. Nous avons expérimenté une sélection aléatoire mais il est clair que plus cette sélection est intelligente, plus l'apprentissage sera performant. C'est dans cet esprit que nous avons essayé d'apprendre en se restreignant aux négatifs les plus *proches* des positifs. Signalons qu'une autre piste pour ne pas souffrir du trop grand nombre de négatifs serait de pouvoir se restreindre à des textes plus courts. Enfin, la dernière perspective est d'adapter des algorithmes d'apprentissage pour qu'ils cherchent une solution en se guidant explicitement à l'aide des mesures de *rappel* et de *précision* et non plus seulement en se fiant au taux d'erreur global. Dans ce domaine, notre intérêt se portera sur les techniques de CS qui prennent en compte le coût de chaque erreur [Dom99], en particulier dans le cadre du boosting [FSZC99]. Comme nous le signalions, des propositions sur cette question ont été faites sur des données de type médical, par exemple d'optimiser l'aire sous la courbe ROC [SAL03].

Remerciements

Ces travaux de recherche ont été soutenus par :

- « CPER 2000-2006, Contrat de Plan état - région Nord/Pas-de-Calais : axe TACT, projet TIC » ;
- « ACI masse de données – ACI-MDD – Accès au Contenu Informationnel pour les Masses de Données et Documents ».

Références

- [BDPDP96] A. L. Berger, S. A. Della Pietra, and V. J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1) :39–71, 1996.
- [CF82] P. R. Cohen and E. A. Feigenbaum. *The Handbook of Artificial Intelligence*, volume 3. HeurisTech Press and William Kaufmann, 1982.
- [CM02] A. Cornuéjols and L. Miclet. *Apprentissage artificiel*. Eyrolles, 2002.
- [CN02] H.-L. Chieu and H.-T. Ng. A maximum entropy approach to information extraction from semi-structured and free text. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI 2002)*, pages 786–791, 2002.
- [Dom99] P. Domingos. Metacost : A general method for making classifiers cost-sensitive. In *Knowledge Discovery and Data Mining*, pages 155–164, 1999.
- [FK00] D. Freitag and N. Kushmerick. Boosted wrapper induction. In *AAAI/IAAI*, pages 577–583, 2000.
- [FM99] D. Freitag and A. McCallum. Information extraction with HMMs and shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.
- [FM00] D. Freitag and A. McCallum. Information extraction with HMM structures learned by stochastic optimization. In *AAAI/IAAI*, pages 584–589, 2000.
- [FS97] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1) :119–139, August 1997.
- [FSZC99] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. AdaCost : misclassification cost-sensitive boosting. In *Proc. 16th International Conf. on Machine Learning*, pages 97–105. Morgan Kaufmann, San Francisco, CA, 1999.
- [HD98] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(8) :521–538, 1998.
- [Huf95] S. B. Huffman. Learning information extraction patterns from examples. In *Learning for Natural Language Processing*, pages 246–260, 1995.

- [Jr.73] G. D. Forney Jr. The viterbi algorithm. In *Proc. IEEE*, volume 61, pages 268–278, 1973.
- [KB00] R. Kosala and H. Blockeel. Instance-based wrapper induction, 2000.
- [KdBBB02] R. Kosala, J. den Bussche, M. Bruynooghe, and H. Blockeel. Information extraction in structured documents using tree automata induction, 2002.
- [Kus97] N. Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, University of Washington, 1997.
- [Mug91] S. Muggleton. Inductive logic programming. *NGCL*, 8(4) :295–317, 1991.
- [PF97] F. Provost and T. Fawcett. Analysis and visualization of classifier performance : Comparison under imprecise class and cost distributions. In D. Heckerman, H. Mannila, D. Pregibon, and Uthurusamy R., editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 43–48. AAAI Press, 1997.
- [Rat] A. Ratnaparkhi. A simple introduction to maximum entropy models for natural language processing.
- [SAL03] M. Sebag, J. Azé, and N. Lucas. Apprendre et optimiser la courbe roc. In *Conférence d’Apprentissage (CAp 2003)*, pages 315–330, 2003.
- [SFAL95] S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert. CRYSTAL : Inducing a conceptual dictionary. In Chris Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1314–1319, San Francisco, 1995. Morgan Kaufmann.
- [SMAA01] H. Sakamoto, Y. Murakami, H. Arimura, and S. Arikawa. Extracting partial structures from HTML documents. In H. Sakamoto, H. Arimura, and S. Arikawa, editors, *Proc. the 14th International FLAIRS Conference*, pages 264–268. AAAI Press, 2001.
- [SMR99] K. Seymore, A. McCallum, and R. Rosenfeld. Learning hidden Markov model structure for information extraction. In *AAAI 99 Workshop on Machine Learning for Information Extraction*, 1999.
- [SS98] R. F. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT-98)*, pages 80–91, New York, 1998. ACM Press.
- [Tor99] F. Torre. GloBo : un algorithme stochastique pour l’apprentissage supervisé et non-supervisé. In M. Sebag, editor, *Actes de la Première Conférence d’Apprentissage*, pages 161–168, 1999.