Extraction de relations dans les documents Web

Rémi Gilleron * , Patrick Marty * , Marc Tommasi * , Fabien Torre*

* Projet Mostrare Inria Futurs & Université de Charles de Gaulle - Lille III
59653 Villeneuve d'Ascq CEDEX FRANCE
prenom.nom@univ-lille3.fr

Résumé. Dans cet article, nous étudions différentes manières de représenter des relations n-aire dans des documents arborescents comme XML ou XHTML. Ensuite nous présentons un système d'induction de wrapper pour ces relations . Le processus d'extraction est itératif. À une étape i fixée, le wrapper i extrait la composante i avec la connaissance des $i-1$ premières composantes. Les résultats expérimentaux montrent que les performances de notre système sont meilleures que celles des systèmes existants pour la plupart des représentations arborescentes des relations n-aire, notamment dans le cas des valeurs factorisées et dans celui des valeurs manquantes.

1 Introduction

La prolifération des documents sur Internet a conduit à l'élaboration de programmes nommés *wrappers* pour collecter de l'information sur les sites Web. Un tel programme est difficile à concevoir et ne peut être que spécifique à un site donné. Deux approches sont alors envisageables : la première consiste à assister l'utilisateur averti dans la conception du wrapper (c'est le cas du système Lixto [1]), soir générer automatiquement le wrapper en limitant l'intervention de l'utilisateur à l'annotation des informations à extraire sur quelques documents.

Les premiers systèmes d'induction de wrappers n'utilisaient que l'aspect textuel des documents [5, 8]. Avec l'apparition de XML, ces approches purement séquentielles sont devenues insuffisantes. Les meilleurs systèmes prennent aujourd'hui en compte la structure arborescente des documents du Web [2, 4, 7, 9, 10]. Cette approche est fondée sur le fait que l'essentiel des des documents présents sur Internet est produit automatiquement par programme et présente donc des régularités exploitables par les méthodes d'apprentissage automatique. Nous nous inscrivons dans cette démarche en proposant un système d'induction qui utilise à la fois les vues textuelle et arborescente.

Beaucoup de systèmes d'induction de wrappers sont conçus pour des tâches unaires : un wrapper unaire extrait par exemple un ensemble de noms de produits ou de prix à partir d'un site Web marchand. Un wrapper n-aire extrait les instances d'une relation n-aire, par exemple les couples (nom du produit,prix). Il existe deux approches pour concevoir de tels wrappers : soit combiner n wrappers unaires, soit apprendre directement le wrapper n-aire. Le premier cas nécessite soit l'obtention d'un modèle pour la combinaison [3], soit une intervention de la part de l'utilisateur [6, 9], soit l'utilisation d'heuristiques. La seconde approche est illustrée par les systèmes WIEN [8] et SOFT MEALY [5] qui tous deux recherchent des délimiteurs textuels pour

repérer les composantes des tuples. Citons également LIPX [10] qui lui utilise une représentation en logique du premier ordre mêlant les vues séquentielle et arborescente.

Nous étudions à la section 2 les différentes organisations des pages Web du point de vue de la structure arborescente sous-jacente. Cela nous amène à distinguer cinq cas dont certains sont délicats pour les systèmes existants. Notre objectif est de proposer un sytème capable de traiter tous ces cas avec comme principe fondateur d'extraire la $i$ème composante en exploitant la connaissance des $i-1$ premières. Ces idées constituent la base d'un algorithme présenté à la Section 3. et évalué en Section 4 sur des jeux de données standards (RISE), sur des données artificielles et sur des données réelles issues de sites Web. Dans chaque cas, les résultats montrent la supériorité de notre système sur les méthodes existantes.

2 Représentations arborescentes d'une relation n-aire

Nous considérons que les données sont stockées dans un document arborescent comme XML ou XHTML. Plusieurs points de vue sont possibles. La *vue DOM* consiste à voir le document comme un arbre. La *vue séquentielle* est comparable à un flux de caractères. La *vue feuillage* correspond à la séquence des feuilles textes de la vue DOM. Ces différentes vues sur les données sont illustrées par la Figure 1.

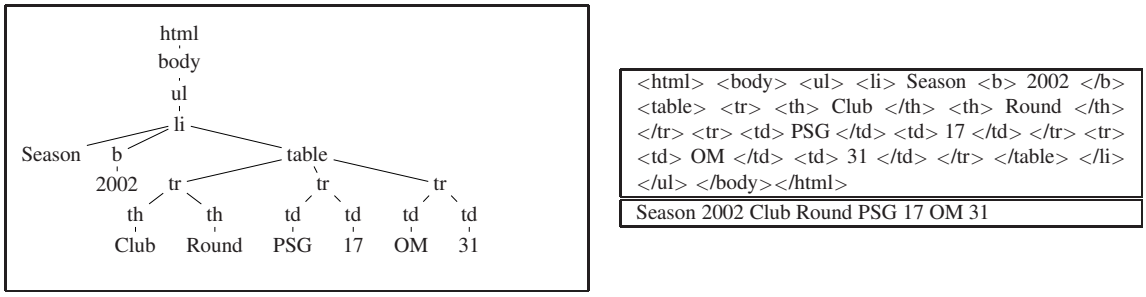


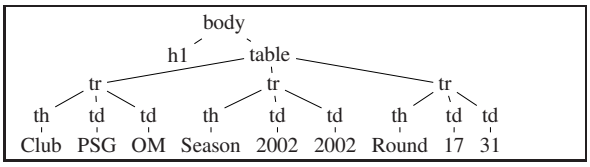
FIG. 1 – *Trois vues d'un document : vue DOM (gauche), vue textuelle (en haut à droite), vue séquentielle des feuilles (en bas à droite).*

Une étude des documents du Web nous a amené à distinguer les cas suivants :

Cas 1. Une première représentation prend la forme d'une table avec une première ligne contenant les noms des composantes et les lignes suivantes contenant les données. Dans ce cas, les sous arbres correspondants aux tuples ne sont pas entrelacés, mais chacun d'entre eux possède une racine distincte, comme on peut le constater sur la Figure 1.

Cas 2. Une autre représentation est celle d'une liste où les tuples sont stockés séquentiellement. Dans la vue DOM, il existe un plus petit sous arbre contenant chaque tuple et seulement lui. Mais ici tous les tuples partagent la même racine (le constructeur de la liste).

Cas 3. Une table peut être tournée : la première colonne contient les noms des composantes et les données sont contenues dans les colonnes suivantes. Dans ce cas les sous arbres correspondants aux tuples sont entrelacés. Par exemple, en considérant la relation ternaire (Season, Club name, points) de la figure 2, les tuples à extraire sont `(2002,PSG,17)` et `(2002,OM,31)`. Il faut remarquer que les pères des différentes composantes d'un même tuple doivent porter le même numéro de fils.

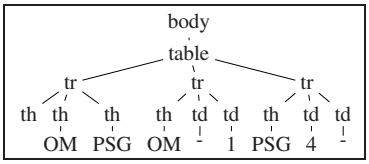


Club	PSG	OM
Season	2002	2002
Round	17	31

FIG. 2 – *Table tournée*

Cas 4. Pour éviter les répétitions, on peut avoir recours à des tables à pivots et de manière générale à des valeurs factorisées. Par exemple dans la Figure 1, la valeur `2002` apparaît seulement une fois mais est partagée par plusieurs tuples : `(2002,PSG,17)` et `(2002,OM,31)`.

Cas 5. Les tables croisées permettent de représenter des relations où les composantes sont presque toutes dépendantes entre elles. Cela correspond aux tables de distances ville à ville, aux tables de conversions de devises, etc (voir figure 3).



	OM	**PSG**
OM	-	1
PSG	4	-

FIG. 3 – *Table croisée*

3 Apprendre à extraire une relation n-aire

Dans le cas unaire, la tâche d'extraction considérée consiste à repérer certaines feuilles textes d'un document arborescent (nous ne considérons pas les cas où les données à extraire sont situées dans une feuille texte ou sur plusieurs feuilles textes). Cette tâche est transformée en un problème de classification binaire qui consiste à décider si une feuille est à extraire ou pas. Une fois codées en vecteurs d'attributs, les feuilles sont fournies à un classifieur qui classe chacune d'entre elles. Une feuille est extraite si elle est classée comme à extraire. L'induction d'un wrapper s'effectue à partir de documents dont les feuilles à extraire ont été annotées. Ces dernières constituent les exemples positifs et les autres les exemples négatifs.

Considérons maintenant une relation cible n-aire : soit $t = (v_1, \ldots, v_n)$ un tuple d'arité n, et t_i le tuple d'arité i $(v_1, \ldots, v_i)$. Un élément important de notre système est le codage d'une telle relation en attribut-valeur. Ce codage est effectué par une fonction notée rep_i et dont le détail est présenté maintenant.

Le **codage de base d'un noeud** p est fourni par les attributs suivants : label, position dans la séquence des fils du père de p, profondeur et hauteur, nombre de fils, taille du sous arbre enraciné en p, label du frère gauche et du frère droit de p, valeurs éventuelles des attributs `XHTML` `class` et `id`.

La **représentation d'une feuille** l est donnée par l'application du codage de base à l, à ses 5 ancêtres dans la vue `DOM`, puis par le contenu brut des feuilles textes précédente et suivante dans la vue séquentielle des feuilles. Ainsi une feuille est représentée par 57 attributs.

La **représentation d'un tuple** d'arité i se fait par le codage de base de la i-ième composante notée l et de la description des dépendances entre l et la graine ainsi qu'entre l et la composante

Algorithm 1 Extraction

Input: un document d; n classifieurs $c_1, \ldots, c_n$.

Notation: L est l'ensemble des feuilles de d, S_i est l'ensemble des tuples candidats à l'étape i, T_i est l'ensemble des tuples sélectionnés dans S_i.

1: $S_1 = \{(l) \mid l \in L\}$;$T_1 = \{t_1 \in S_1 \mid c_1(rep_1(t_1)) = true\}$
2: **for** $i = 2$ **to** n **do**
3: $S_i = \{t_i \mid t_i = (t_{i-1}, l),\ t_{i-1} \in T_{i-1},\ l \in L,\ l$ does not occur in any $t_{i-1} \in T_{i-1}\}$
4: $\forall t_i \in S_i$, if $c_i(rep_i(t_i)) = true$, add t_i to T_i
5: if all t_i based on the same t_{i-1} are classified false by c_i, add $(t_{i-1}, \texttt{null})$ to T_i
6: **end for**

Output: T_n l'ensemble des tuples n-aire extraits

Algorithm 2 Apprentissage

Input: un échantillon S de documents annotés.

1: $S^+ = \{t = (l_1, \ldots, l_n)\}$ l'ensemble des n-tuples de feuilles à extraire dans S.
2: $R = C = \emptyset$
3: **for** $i = 1$ **to** n **do**
4: $S_i^+ = \{t_{|I} \mid t \in S^+\}$
5: S_i^- = tout tuple dont les $i-1$-ières composantes sont correctes mais pas la i-ième
6: $T_i^+ = rep_i(S_i^+)$ et $T_i^- = rep_i(S_i^-)$
7: $c_i = C5(T_i^+, T_i^-)$ et ajouter c_i à la séquence C de classifieurs
8: **end for**

Output: C

$i-1$. Le codage de la dépendance entre deux feuilles p et m est constituée du codage de leur plus petit ancêtre commun a, des longueurs des plus courts chemins dans la vue `DOM` entre p et a, m et a et entre p et m, du nombre de feuilles textes se trouvant entre p et m dans la vue séquentielle des feuilles, et pour $1 \leq k \leq 5$ de la différence entre la position (relative à leur père) du k-ième ancêtre de p et du k-ième ancêtre de m. Ainsi un tuple est représenté par 220 attributs quel que soit i différent de 1.

Supposant que l'ordre des composantes est fixé (la première étant nommée la *graine*), l'extraction est réalisée par l'algorithme 1.

Du point de vue de l'apprentissage, une tâche d'extraction n-aire est transformée en n problèmes de classification supervisée, chaque problème consistant à classer des tuples d'arité i comme étant à extraire ou pas. L'algorithme d'apprentissage est donné par l'algorithme 2. Pour apprendre le classifieur c_i, il faut fixer l'ensemble d'apprentissage (ligne 4). Les exemples positifs (ensemble S_i^+) sont la projection des tuples annotés (ensemble S^+) sur les i premières composantes. Lors de la première itération, est considéré comme exemple négatif (ensemble S_i^+) toute feuille qui n'est pas une graine. Pour les étapes suivantes, les exemples négatifs sont déterminés par la fonction *negativeExamples*.

4 Expériences

La qualité de notre système est classiquement évaluée à travers la f-mesure (F) des wrappers produits. Chaque composante obtenue doit correspondre exactement à la valeur à extraire et

chaque tuple construit doit être exact sur chacune de ses composantes. Un critère important est celui du nombre de documents nécessaires pour l'apprentissage : l'annotion étant une tâche coûteuse et fastidieuse, ce nombre doit être aussi petit que possible. Nous menons toutes nos évaluations avec uniquement deux documents en apprentissage.Cette procédure est itérée 50 fois sur les corpus les plus importants, tandis que sur les corpus de 10 documents, nous effectuons une validation croisée 5 fois.

Les documents présents dans RISE[1] correspondent aux deux premiers cas de la Section 2. Deux tâches sont difficiles : S1 qui ne présente pas de structure régulière et IAF pour lequel trois des six composantes à extraire ont des valeurs manquantes. Nous obtenons $F = 100$ sur BIGBOOK, OKRA et S20 ; sur S1, nous souffrons de ne pouvoir utiliser la partie textuelle des feuilles ($F = 88.61$) ; enfin, sur IAF, nous faisons mieux que LIPX [10] ($F = 64.22$ contre $F = 46$) et que WIEN qui n'est pas en mesure de traiter les valeurs manquantes.

Nous complétons notre évaluation avec de nouveaux corpus[2] construits à partir des résultats du championnat français de football. Chaque jeu de données contient 20 documents XHTML, chacun présentant entre 5 et 15 enregistrements. Les jeux de données $L_0, \ldots, L_9$ décrivent les quatre premiers cas de la Section 2 et la relation à extraire est systématiquement (*club*,*season*,*round*,*date*). Pour chaque jeu, nous disposons d'une version exhaustive et d'une version où 37.5% des *dates* sont manquantes. Le problème CT correpond aux tables croisées (cas 5) et la relation cible est (*win*,*club1*,*club2*). En l'absence de valeurs manquantes, nous réussissons sur tous ces problèmes ($F = 100$) ; les valeurs manquantes ne sont pénalisantes que sur deux problèmes où le nombre de documents présentés en apprentissage n'est pas suffisant (nous obtenons alors $F = 78.01$ et $F = 76.42$, $F = 100$ dans les autres cas).

Enfin, nous nous évaluons sur des sites présents sur le web en visant les organisations les plus difficiles : les tables tournées (cas 3), les tables pivot avec factorisation (cas 4) et les tables croisées (cas 5). Nous avons récupéré et annoté 10 documents XHTML des sites EXCITE WEATHER et *Bureau of Labor Statistics*. Dans le premier cas, la relation à extraire est (*town*,*day*,*weather*,*high*,*low*), chaque document contenant 5 tuples, la composante *town* étant factorisée et les autres composantes étant présentées dans une table tournée. Dans le second cas, il s'agit de la relation ternaire (*value*,*year*,*quantile*) et d'une organisation en table croisée. Nous obtenons ici respectivement $F = 100$ et $F = 98.52$.

5 Conclusion

Nous avons présenté un système capable d'induire automatiquement des wrappers dédiés à l'extraction n-aire à partir de documents du Web. Ce système présente les avantages suivants : l'extraction des composantes et la construction des tuples sont réalisées simultanément ; apprentissage et extraction sont rapides et peu de documents annotés sont nécessaires pour générer un wrapper efficace ; le système est facilement paramétrable à travers différentes stratégies ; l'utilisation de stratégies basiques suffit à notre système pour surclasser les systèmes existants sur bon nombre de tâches. Nous travaillons maintenant à l'amélioration de ce système en privilégiant les directions suivantes : exploiter la vue textuelle des documents ; pouvoir traiter des documents partiellement annotés ; concevoir des stratégies plus sophistiquées.

[1] http://www.isi.edu/info-agents/RISE/index.html
[2] http://www.grappa.univ-lille3.fr/~marty/corpus.html

Références

[1] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with lixto. In *28th International Conference on Very Large Data Bases*, pages 119–128, 2001.

[2] J. Carme, R. Gilleron, A. Lemay, and J. Niehren. Interactive learning of node selecting tree transducer. In *IJCAI Workshop on Grammatical Inference*, 2005.

[3] H. L. Chieu and H. T. Ng. A maximum entropy approach to information extraction from semi-structured and free text. In *Proceedings of Eighteenth national conference on Artificial intelligence*, pages 786–791, 2002.

[4] W. Cohen, M. Hurst, and L. Jensen. *Web Document Analysis : Challenges and Opportunities*, chapter A Flexible Learning System for Wrapping Tables and Lists in HTML Documents. World Scientific, 2003.

[5] 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.

[6] L. S. Jensen and W. Cohen. Grouping extracted fields. In *Proceedings of IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, 2001.

[7] R. Kosala, J. V. D. Bussche, M. Bruynooghe, and H. Blockeel. Information extraction in structured documents using tree automata induction. In *6th International Conference Principles of Data Mining and Knowledge Discovery*, pages 299 – 310, 2002.

[8] N. Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, University of Washington, 1997.

[9] I. Muslea, S. Minton, and C. Knoblock. Active learning with strong and weak views : a case study on wrapper induction. In *IJCAI 2003*, pages 415–420, 2003.

[10] B. Thomas. Bottom-up learning of logic programs for information extraction from hypertext documents. In Springer-Verlag, editor, *In proceedings of European Conference on Machine Learning / Principles and Practice of Knowledge Discovery in Databases ECML/PKDD 2003*, September 2003.

Summary

We study how n-ary relations are encoded in tree structured documents like `XML` or `XHTML`. Then we present a wrapper induction system for n-ary relations. The extraction process is iterative. At a given step i, the $i-th$ wrapper extracts the $i-th$ component with the knowledge about the $i-1$ first components. Experimental results show that our system outperforms existing ones for many tree representations schemes in the case of factorized values and in the case of missing values.