

Optimizando FOIL para la Extracción de Información de la Web*

P. Jiménez, J.L. Arjona, J.L. Álvarez

Universidad de Huelva, Dep. de Tecnologías de la Información,
Escuela Politécnica Superior. Crta. Huelva - La Rábida. Palos de la Frontera 21071
{patricia.jimenez, jose.arjona, alvarez}@dti.uhu.es

Abstract. Para abaratar los costes de las soluciones de Integración de aplicaciones web amigables necesitamos sistemas automáticos que permitan navegar hasta la información de interés, extraerla, estructurarla y verificarla. Los extractores de información son los elementos que permiten extraer la información de la web, y existe mucho trabajo en investigación que tiene como objetivo automatizar su construcción a partir de técnicas de aprendizaje automático. El algoritmo FOIL, ha demostrado ser una buena solución al problema anterior, sin embargo, su ineficiencia en este contexto, imposibilita su uso desde un punto de vista ingenieril. En este artículo presentamos una serie de ideas originales que tienen como objetivo la optimización del algoritmo FOIL, lo que permitirá su uso, abaratando la construcción de extractores de información usados en soluciones de integración.

Key words: extracción de información, EAI, FOIL

1 Introducción

Las soluciones de Integración de Aplicaciones Empresariales permiten la sincronización y/o colaboración de las distintas aplicaciones que gestionan los procesos de negocio de una empresa. El objetivo de estas soluciones es incrementar los activos de la empresa proporcionando valor sobre la información y procesos ya definidos. Por otra parte, los servicios Web nos proporcionan la tecnología necesaria para llevar a cabo soluciones de integración, siempre y cuando dispongamos de APIs que nos permita acceder a las distintas aplicaciones a integrar. El problema se plantea cuando los costes asociados al proceso de reingeniería inversa para construir dichas APIs no son permisibles (aplicaciones no desmantelables).

En el contexto de la Web, las aplicaciones son concebidas para ser utilizadas por seres humanos, y es habitual que no proporcionen una interfaz programática para acceder a una vista estructurada de la información que gestionan, ni lo que es peor, desmantelarlas. Para integrar estas aplicaciones con otras aplicaciones empresariales, es necesario disponer de un conjunto de APIs que ofrezcan la

* Este trabajo esta parcialmente financiado por el proyecto IntegraWeb - Wrapping (P08-TIC-4100)

posibilidad de interactuar con la aplicación como si de un ser humano se tratara. Estas APIs deberían de aceptar una serie de consultas definidas en un lenguaje estructurado de alto nivel, como SQL o SPARQL¹, rellenar formularios a partir de ellas [1, 2], ejecutarlos y navegar a través de los resultados [2], devolver las páginas que contengan la información de interés, extraer dicha información [4] y dotarla de estructura de acuerdo con la ontología que estemos utilizando [5] y, por último, comprobar que la información obtenida es correcta [6, 7].

Los extractores de información juegan un rol de vital importancia en las soluciones anteriores y este es el motivo, por el que cada vez más, están apareciendo herramientas basadas en algoritmos de aprendizaje automático que tienen como objetivo ayudar a los ingenieros en el desarrollo de soluciones de integración de aplicaciones web amigables. No obstante, no todos los algoritmos de aprendizaje son de utilidad para ser incorporados en herramientas de apoyo al proceso de ingeniería. Algunos, como CLS [9], ID3 [10], C4.5 [11] y CART [12], utilizan lenguajes de especificación poco expresivos a la hora de describir el problema. Otros, como FOIL [13], FFOIL [13] y PROGOL [14], utilizan lenguajes de una gran capacidad expresiva pero presentan problemas de eficiencia que desaconsejan su uso desde un punto de vista ingenieril.

En este artículo, presentamos una propuesta en la que apostamos por la expresividad que nos ofrece FOIL y mejoramos su eficiencia de cara a ser utilizado en soluciones de integración de aplicaciones empresariales. En concreto, presentamos 12 optimizaciones, de las que 6 pueden ser vistas como optimizaciones independientes de su dominio de aplicación, las restantes 6 están centradas en el dominio concreto de la extracción de información en la Web.

El resto de este artículo se organiza de la siguiente manera: la siguiente sección analiza el trabajo relacionado, tanto en lo referente a métodos de aprendizaje inductivo, como a los sistemas de extracción de información de la Web. La sección 3 describe el algoritmo FOIL, la sección 4 presenta las optimizaciones propuestas. Finalmente, la sección 5 presenta las conclusiones y trabajos futuros.

2 Trabajo relacionado

2.1 Métodos de aprendizaje inductivo

En el aprendizaje inductivo se parte de un conjunto de entidades cuyas clases son previamente conocidas y se trata de obtener, a partir de las mismas, una caracterización de cada clase. El aprendizaje en estos sistemas es, a grandes rasgos, el siguiente: dado un conjunto de entrenamiento formado por entidades cuyas clases son conocidas, encontrar una serie de reglas que permitan predecir la clase de una entidad desconocida, en función de los valores de sus atributos. Existen dos diferentes aproximaciones a este proceso.

Los sistemas como CLS, ID3, C4.5 y CART representan estas reglas con un árbol de decisión. Este árbol se construye a partir de un conjunto de entrenamiento mediante un proceso reiterativo. Cada nodo representa un test sobre

¹ <http://www.w3.org/TR/rdf-sparql-query/>

un atributo, las aristas se determinan en función de los diferentes valores del atributo. El proceso finaliza cuando en un nodo todas las entidades son de la misma clase; en este caso, se convierte en un nodo hoja etiquetado con esa clase. La decisión del atributo a seleccionar en cada test es diferente para cada algoritmo. En CLS un experto determina el atributo que debe usarse en el test. En ID3 se utiliza el concepto de ganancia de información. En C4.5 se usa el concepto de ganancia de información normaliza y en CART se elige al atributo que es capaz de diferenciar en mayor medida a las diferentes entidades. Otros algoritmos de inducción, como los de la familia de algoritmos AQ [16], representan cada clase mediante una expresión lógica disyuntiva. Esta expresión se construye mediante nuevas conjunciones que tratan de caracterizar sólo a las entidades de la clase en cuestión. El proceso termina cuando todas las entidades de un clase se pueden obtener a partir de la expresión lógica disyuntiva.

Existen una serie de algoritmos, como FOIL, FFOIL o PROGOL, que combinan aspectos de las dos aproximaciones anteriores, ofreciendo una mayor expresividad. FOIL es un algoritmo de aprendizaje, derivado de AQ e ID3, cuyo objetivo es inferir un conjunto de reglas, a partir de un conjunto de predicados y de ejemplos utilizando una estrategia top-down. Para una relación objetivo R , FOIL encuentra las cláusulas de una en una, de forma similar a AQ, eliminando los ejemplos cubiertos por la nueva cláusula, antes de inducir la siguiente. Como ID3, FOIL usa una heurística basada en la ganancia de información para guiar esta búsqueda. FFOIL es un variante de FOIL pensada para trabajar con relaciones funcionales. Una relación funcional es aquella donde uno o más parámetros se consideran parámetros de salida y quedan inequívocamente determinados por el resto de parámetros. PROLOG hace frente al mismo problema que FFOIL pero con una estrategia down-up.

2.2 Extracción de información de la Web

Podemos definir un extractor de información como un algoritmo genérico que está configurado con un conjunto de reglas que han sido obtenidas aplicando técnicas de aprendizaje automático. Las reglas permiten identificar, de forma unívoca, fragmentos de información de interés de un sitio web. Existen dos estrategias distintas a la hora de plantear el aprendizaje de las reglas: supervisado y no supervisado.

En los sistemas no supervisados, partimos de un conjunto lo suficientemente representativo de páginas webs no etiquetadas. El sistema trata de identificar de forma automática los elementos comunes entre las diferentes páginas webs y construir, a partir de ellos, las reglas de extracción. Su principal ventaja es la rapidez con que son capaces de aprender; sin embargo, no son capaces de trabajar con páginas que presenten la información en diferentes formatos. Algunos sistemas de este tipo son: Roadrunner [17], DEPTA [18] y EXALGA [19].

En los sistemas supervisados, partimos de un conjunto representativo de páginas web, pero, en este caso, la información de interés está etiquetada. Esos metadatos son usados para inducir las reglas que permiten extraer la información. Aunque no cuentan con la limitación de los sistemas no supervisados,

en lo referente a la alternancia de formatos, son sistemas con un proceso de aprendizaje lento y costoso. Algunos sistemas de este tipo son: SRV, STALKER [20] y WIEN [7].

3 FOIL

FOIL[10, 11, 13] es un algoritmo de aprendizaje de reglas de lógica de primer orden para explicar un predicado objetivo P , a partir de un conjunto de ejemplos positivos y predicados de soporte. Para ello, usa una estrategia muy similar a la de los algoritmos de cobertura secuencial y learn-one-rule [23] donde aprende una regla en cada iteración, eliminando las tuplas positivas del conjunto de entrenamiento cubiertas y repitiendo el mismo proceso, hasta que no queden tuplas positivas por cubrir.

Para aprender cada regla, FOIL emplea una estrategia TOP-DOWN, comenzando por la regla más general, la cabecera de la regla, y especializándola mediante la adición de predicados, hasta que la regla no cubra ejemplos negativos. Los posibles predicados se ajustan a una de las siguientes formas: $Q(X_1, X_2, \dots, X_k)$ y $\neg Q(X_1, X_2, \dots, X_k)$, donde al menos, una de las variables X_i debe existir en la regla; $X_i = C$ y $X_i \neq C$, donde X_i es una variable ya existente y C una constante; $X_i = X_j$ y $X_i \neq X_j$, donde X_i y X_j son variables ya existentes; por último, $X_i \leq t$ y $X_i > t$, donde X_i es una variable numérica ya existente y t un umbral.

En la elección del siguiente literal a añadir, FOIL se guía de una estrategia *greddy*. En cada iteración, explora el espacio de los literales candidatos de forma casi-exhaustiva para evaluarlos mediante una heurística basada en la ganancia de información. La ganancia de información mide la mejora que se produce sobre la cobertura al añadir un predicado P a la regla en construcción R .

Aunque a priori FOIL es un algoritmo con un gran potencial, adolece de un problema importante: la generación de nuevos predicados supone un gran costo. Por ejemplo, para un predicado de la forma $P(X_1, X_2, \dots, X_n)$ con n variables, el número de nuevos predicados a explorar viene determinado por $n * (\binom{n}{m} + n! * m)$, donde m es el número de variables existentes en la regla. Para que FOIL sea más eficiente, debemos centrar nuestros esfuerzos en reducir en gran medida este número.

3.1 Caso de Estudio

Si estamos interesados en extraer los coautores de una página web de la base de datos de publicaciones DBLP (<http://dblp.uni-trier.de>), la salida de un extractor de información construido con FOIL sobre la página de la figura 1 sería: $Coauthor(x) :< Krista Lagus, Samuel Kaski, Panu Somervuo >$.

Los predicados seleccionados para el ejemplo son los siguientes: $h(x)$: indica que x se encuentra dentro de una etiqueta $<h>$; $a(x)$: indica que x se encuentra dentro de una etiqueta $<a>$; $td(x)$: indica que x se encuentra dentro de una

2006	
24	Teuvo Kohonen: Self-organizing neural projections. <i>Neural Networks</i> 19(6-7): 723-733 (2006)
2004	
23	Krista Lagus, Samuel Kaski, Teuvo Kohonen: Mining massive document collections by the WEBSOM method. <i>Inf. Sci.</i> 163(1-3): 135-156 (2004)
2002	
22	Teuvo Kohonen, Panu Somervuo: How to make large self-organizing maps for nonvectorial data. <i>Neural Networks</i> 15(8-9): 945-952 (2002)
2000	
21	Panu Somervuo, Teuvo Kohonen: Clustering and Visualization of Large Protein Sequence Databases by Means of an Extension on the Self-Organizing Map. <i>Discovery Science 2000</i> : 76-85
20	Teuvo Kohonen: Self-Organizing Maps of Massive Document Collections. <i>IJCNN (2) 2000</i> : 3-12

Fig. 1. Página web obtenida de DBLP

celda de una tabla; $previous(x, y)$: indica que el anterior de x es y ; por último, $next(x, y)$: indica que el siguiente de x es y .

Además se han definido un conjunto de ejemplos positivos que contienen coautores válidos, y ejemplos negativos con elementos que no son coautores:
Positivos : *Krista Lagus, SamuelKaski, Panu Somervuo, ...*

Negativos : *Self – organizing neural projections, Neural Networks, ...*

La ejecución de FOIL en forma de árbol para determinar reglas que extraigan los coautores de una página web puede observarse en la figura 2. El número de nodos generados viene determinado por el número de predicados de soporte definidos, mientras que el coste de calcular la ganancia de cada predicado está determinado por el tamaño del conjunto de entrenamiento.

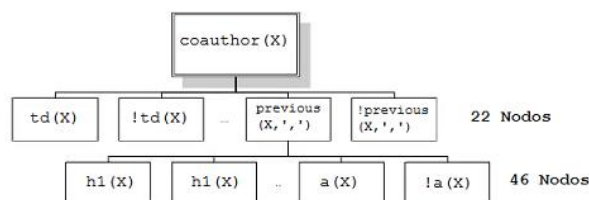


Fig. 2. Árbol de ejecución de FOIL

4 Optimizaciones

Las optimizaciones aplicables a la versión actual de FOIL, pueden clasificarse en dos tipos: independientes del dominio y en el contexto concreto de la Extracción de Información en la Web.

4.1 Independiente del dominio

Test T-Students pareado. Un test T-Students pareado es una fórmula sencilla, que determina si dos nodos estadísticamente pueden considerarse iguales. Dos ramas dentro de un árbol, aunque sean completamente distintas para un experto humano, estadísticamente pueden ser iguales.

Se trata de reducir la expansión de nodos a clases de equivalencia, de manera que, a cada clase pertenecieran todos aquellos nodos que estadísticamente parecen iguales. De esta forma, sólo calculamos la ganancia de un único nodo por clase. El número de literales explorados se reduce al número de clases encontradas.

Asignación de semántica a los predicados. La búsqueda de una definición en FOIL es ciega, de modo que no tiene en cuenta la semántica de los predicados. Existen casos en los que no tendría sentido explorar ciertos literales. Por ejemplo, el literal $previous(X, X)$ es correcto, sin embargo, para un observador humano está claro que no tiene interés ya que es imposible ser el antecesor de sí mismo.

Podemos así añadir para cada predicado de soporte información semántica que permita prescindir de literales que, aunque son válidos, carecen de sentido práctico. Por ejemplo, podemos indicar que los predicados $previous(X, Y)$ y $next(X, Y)$, no tienen sentido cuando las variables X e Y son iguales.

Exploración de n predicados. Una de las formas más simples es ampliar la filosofía greedy a la creación de los nuevos predicados a evaluar. Nuestra propuesta es reducir este número. Se establece así una profundidad límite p , un número de caminos c , y un número de predicados a explorar n . Se llevan de forma simultánea los c mejores caminos de todos los posibles. De cada camino se obtienen los n nuevos predicados y de entre todos ellos se eligen los $c - 1$ mejores y del resto de literales descartados se elige uno al azar para garantizar una cierta capacidad de exploración. Estos predicados constituirán los nuevos caminos a explorar. Una vez que se ha alcanzado una profundidad p (se han añadido p predicados) el algoritmo se centrará sólo en el mejor de todos los caminos que se ha estado considerando.

Selección del primer predicado cuya ganancia supere un umbral. Esta propuesta se basa en generar todos los posibles nuevos predicados y seleccionar el primero con una ganancia que supere cierto umbral, dejando de evaluar el resto de predicados candidatos en ese nivel del árbol de exploración. El problema de esta optimización es la generación de reglas poco expresivas. No obstante, nuestro objetivo es desarrollar herramientas que hagan transparentes las reglas a los usuarios.

Catálogo completo de predicados de soporte. Se trata de disponer de un conjunto útil de predicados de soporte, catalogados por dominio, para ayudarnos

a la hora de enfrentarnos a un nuevo sitio web. Además, dando prioridad a predicados que tras un análisis previo, ya sea experimentalmente o proporcionados por un experto en el dominio se sepan que pueden alcanzar una mayor precisión si se incluyen en la regla.

Determinar si una cadena de predicados está generada por azar. En Lerman [21], se usa el test de hipótesis de Popoulis[22] donde mediante detección de regularidades, es capaz de determinar si una secuencia de texto es fruto o no del azar. Sería interesante incorporar este test a FOIL, ya que así podríamos ser capaces de detectar, antes de evaluar, si la regla en construcción es aleatoria o no, y descartarla en caso afirmativo para centrarnos en otras más prometedoras.

4.2 Dependiente del dominio de Extracción de Información

Uso de los predicados Left y Right. El objetivo de esta optimización es hacer uso de predicados que caractericen el contexto por la derecha y después por la izquierda (o al revés) de la información a extraer. Se pretende explotar la estructura de las páginas Web. A este tipo de extractores de información se les denominan landmark [21].

Un ejemplo de este tipo de extractores son los de Kushmerick[8]. Por tanto, si la información se puede extraer a partir de su contexto, es francamente una buena idea darle prioridad a predicados que tienen como objetivo caracterizar el contexto por la derecha (p.e predicado *next*) y una vez que la ganancia no sea significativa, dar prioridad a predicados que tengan por objetivo fijar el contexto por la izquierda (p.e *previous*).

Chequear si se pueden emplear este tipo de reglas landmark para extraer información de un sitio web, es tan fácil como tomar n tokens por la izquierda y n tokens por la derecha de la información a extraer y aplicar un algoritmo tipo *longest common subsequence* ²

Priorización de Constantes. Durante la fase de etiquetado de páginas Webs que servirán de entrenamiento al sistema, el experto humano puede observar patrones de tokens que aparecen a izquierda y derecha de la información a extraer. Si esto ocurre, podríamos ordenar las clases de tokens en función de la distancia en la que aparezcan de la información relevante, siendo prioritarias aquellas clases más cercanas a dicha información. De esta manera, en reglas en las que se hayan añadido literales del tipo *next*(X, Y) o *previous*(X, Y) los literales más prometedores para evaluar en la siguiente iteración son los predicados predefinidos del tipo $Y = C$ siendo C un token de clase preferente. Incluso si el literal más prometedor fuera *next*(X, Y), añadir el siguiente más prometedor del tipo $Y = C$ en un solo paso. En el resto de predicados de soporte habría que estudiar su utilidad.

² http://en.wikipedia.org/wiki/Longest_common_subsequence_problem/

Predicados que implementan modelos de características de la información. Además de las reglas de tipo landmark, disponemos de reglas de tipo content-based [21]. En este tipo de reglas se explota la estructura de la propia información a extraer. La idea es usar predicados de soporte que hagan referencia a las características de la información a extraer.

Las características pueden clasificarse en dos grandes grupos, características numéricas o características categóricas. Para el ejemplo de la figura 1 una posible regla obtenida por FOIL para extraer todos los coautores es: $coauthor(X) : -a(X), digitDensity(X, 0)$ donde el predicado de soporte $digitDensity$ se refiere a una características numérica de la información a extraer, para el ejemplo de la figura 1 puede generar una regla del tipo $coauthor(X) : -a(X), nounPhrase(X)$ donde el predicado de soporte $nounPhrase(X)$ se refiere a una características lingüística de la información a extraer. Las ventajas que ofrecen estas reglas es que son menos susceptibles a cambio que las landmark, el problema está en evaluar las características, ya que supone un mayor coste computacional.

Concatenar tokens next y previous. En FOIL, una variable en un predicado puede instanciarse a un token. Si en vez de instanciarse a un sólo token puede instanciarse a un conjunto de ellos, estamos englobando varias iteraciones en una sola. Sería como seleccionar más de un literal candidato en una misma iteración.

Supongamos que los ejemplos positivos que quedan por cubrir son los coautores *Krista Lagus* y *Samuel Kaski*. Empezaríamos a generar una nueva regla para cubrir estos ejemplos. Una posible regla de salida que extrayera esta información sin extraer ningún ejemplo negativo sería

$$coauthor(X) : -next(X, ' '), next(' ', Z), next(Z, ' ')$$

Si permitimos la concatenación de tokens, se podría generar una regla del tipo $coauthor(X) : -next(X, ' ', Z', ' ')$.

Inducción de patrones. Inducir prefijos y sufijos de la información relevante, puede ser muy útil cuando trabajamos con los predicados de soporte next y previous. Por ejemplo, si disponemos de la tupla $\langle P, X, S \rangle$ donde P es el prefijo de X inducido, S es el sufijo de X inducido y X es la información relevante, es apropiado dar asignar prioridad a predicados más específicos del tipo $next(X, Y), Y = S$ o $previous(X, Y), Y = P$ o, de forma simplificada $next(X, S)$ y $previous(X, P)$ respectivamente, porque aportarán una ganancia más alta. Tanto P como S hacen referencia a un token o clase de tokens.

Transferencia de predicados a expresiones regulares. Una vez tenemos las reglas de extracción, acceder a la información puede plantear importantes problemas de eficiencia. En el peor de los casos, tenemos que tokenizar toda la página y a continuación, para cada token, comprobar si se satisfacen todos los predicados de las reglas. En reglas de tipo landmark, este problema queda resuelto traduciendo la regla a una expresión regular, para una página web, podríamos

identificar la información a extraer mediante una expresión xpath. Por ejemplo, si tenemos la regla $coauthor(X) : -next(X, a), previous(X, b), previous(b, c)$ se podría traducir a la expresión regular: $*cb\$a*$, siendo $*$ la clausura de kleene, y $\$$ la información que queremos extraer.

5 Conclusiones

En este artículo hemos presentado 13 ideas originales que tienen por objetivo optimizar su eficiencia computacional de FOIL. Estas optimizaciones permiten la posibilidad de desarrollar APIs programáticas para integrar aplicaciones Web (no desmantelables), abaratando los costes del desarrollo de extractores de información para soluciones de integración.

De entre las optimizaciones propuestas, 6 son independientes del dominio y 6 dependen del contexto de Extracción de Información de la Web. En el primer caso, las optimizaciones sólo son aplicables al núcleo de FOIL, para reducir el número de nodos a evaluar. Mientras que las optimizaciones del segundo tipo, van desde una adecuada definición de la base de conocimiento, hasta la reducción del número de nodos a evaluar en el dominio de extracción de información y la adaptación del conjunto de reglas inducidas de salida.

Todas ellas, se pueden aplicar de forma independiente aunque nuestra propuesta es aplicarla de forma conjunta puesto que no son excluyentes. Para ello, estamos desarrollando un marco de trabajo teórico que nos permita realizar un estudio experimental exhaustivo que determine, en función de un problema, las optimizaciones a usar.

References

1. Raghavan, S., Garcia-Molina, H.: Crawling the hidden web. In: VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2001) 129-138.
2. Pan, A., Raposo, J., Álvarez, M., Carneiro, V., Bellas, F.: Automatically maintaining navigation sequences for querying semi-structured web sources. In: Data Knowl. Eng. 63(3) (2007) 795-810.
3. Lage, J.P., da Silva, A.S., Golgher, P.B., Laender, A.H.F.: Automatic generation of agents for collecting hidden web pages for data extraction. In: Data Knowl. Eng. 49(2) (2004) 177-196.
4. Kaye, M., Shaalan, K.F.: A survey of web information extraction systems. In: IEEE Trans. on Knowl. and Data Eng. 18(10) (2006) 1411-1428.
5. Arjona, J.L., Corchuelo, R., Ruiz, D., Toro, M.: From wrapping to knowledge. In: IEEE Trans. Knowl. Data Eng. 19(2) (2007) 310-323.
6. Kushmerick, N.: Regression testing for wrapper maintenance. In: AAAI '99/IAAI'99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence, Menlo Park, CA, USA, American Association for Artificial Intelligence (1999) 74-79.
7. Kushmerick, N.: Wrapper verification. In: World Wide Web 3(2) (2000) 79-94.

8. Kushmerick N.: Wrapper induction: Efficiency and expressiveness In: *Artif. Intell.* 118, 15-68, (2000).
9. Hunt, E., Marin, J., Stone P.: Experiments in induction. In: New York: Academic Press. (1996).
10. Quinlan J.: Induction of decision trees. In: *Machine Learning*, 1, 81-106. (1986).
11. Quinlan, J.: C4.5: Programs for Machine Learning. In: Morgan Kaufmann Publishers, 1993.
12. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and regression trees. In: Belmont: Wadsworth, (1984).
13. Quinlan J.: Learning First-Order Definitions of Functions. In: *Journal of Artificial Intelligence Research*, 5, 139-161, (1996).
14. Muggleton, S.: Inverse Entailment and Progol. In: *New Generation Computing, Special issue on Inductive Logic Programming*, 13, 1995
15. Craven, D., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., Slatery, S.: Learning to Extract Symbolic Knowledge from the World Wide Web. In: *Proceedings of the 15th National Conference on Artificial Intelligence*, (2001).
16. Michalski, R., Mozetic, I., Hong, J., Lavrac, N.: The multipurpose incremental learning system AQ15 and its testing application to three medical domains. In: *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, 1041-1045, (1986).
17. Crescenzi, V., Mecca, G., Merialdo, P.: RoadRunner: Towards Automatic Data Extraction from Large Web Sites. In: *Proceedings of 27th International Conference on Very Large Data Bases*, 109-118, (2001).
18. Zhai, Y., Liu, B.: Web data extraction based on partial tree alignment. In: *International World Wide Web Conference*, 76-85, (2005).
19. Ma, L., Goharian, N., Chowdhury, A., Chung, M.: Extracting unstructured data from template generated web documents. In: *Conference on Information and Knowledge Management*, 213-215, (2003).
20. Muslea I., Minton, S., Knoblock, C.: Knoblock: Hierarchical Wrapper Induction for Semistructured Information Sources In: *Journal of Autonomous Agents and Multi-Agent Systems* 4 (1/2), 93-114, (2001).
21. Lerman, K., Minton, S., Knoblock, C.: Wrapper Maintenance: A Machine Learning Approach. In: *Journal of Artificial Intelligence Research*, 2002.
22. Papoulis, A.: Probability and Statistics. In: Prentice Hall, Englewood Cliffs, NJ, (1990).
23. Mitchell, T.: Machine Learning. In: McGraw Hill, 1997.