Self Training Wrapper Induction with Linked Data*

Anna Lisa Gentile, Ziqi Zhang, and Fabio Ciravegna

Department of Computer Science, University of Sheffield, UK
{a.l.gentile,z.zhang,f.ciravegna}@dcs.shef.ac.uk

Abstract. This work explores the usage of Linked Data for Web scale Information Extraction, with focus on the task of Wrapper Induction. We show how to effectively use Linked Data to automatically generate training material and build a self-trained Wrapper Induction method. Experiments on a publicly available dataset demonstrate that for covered domains, our method can achieve F measure of 0.85, which is a competitive result compared against a supervised solution.

1 Introduction

Information Extraction (IE) is the process of transforming unstructured or semi-structured textual data into structured representation that can be understood by machines. IE is a crucial technique to deal with the continuously growing data published on the Web. Many websites use scripts to generate pages, which get populated with values from an underlying database. These automatically generated pages have high structural similarity. The technique to extract information from this kind of pages is commonly referred as Wrapper Induction (WI) and consists of identifying a set of rules which enables the systematic extraction of specific data records from the pages. In general, WI addresses the extraction of data from *detail* Web pages [3], which are pages corresponding to a single data record (or entity) of a certain type or *concept* (e.g. in a collection of detail pages about films, each page describes a single film).

An extensive range of work has been carried out to study WI, with a mainstream of research focusing on supervised WI [9,11,5], i.e. using a collection of manually annotated Web pages as training data to generate extraction patterns. To reduce the number of required annotations, porting techniques have been proposed to adapt wrappers learnt on one specific website to other websites of the same domain [14,7]. Completely unsupervised methods [4,1] overcome the need of manual annotation but the semantic of produced results is to be interpreted by the user. Hybrid methods [6] propose to automatically generate annotations (exploiting available resources, e.g semantic data on the Web) to be used in the learning phase.

Our solution adopts the methodology proposed in [6], which focus on the usage of *Linked Data*[1] (*LD*) to automatically generate training data to learn extraction patterns. We extend the method by focusing on *limiting noise* in the generation of annotations, and the selection of *reliable* patterns. Experiments report an an average F measure of 0.85.

* Part of this research has been sponsored by the EPSRC funded project LODIE: Linked Open Data for Information Extraction, EP/J019488/1

[1] A collection of interrelated datasets on the Web `http://tinyurl.com/lbncdjl`

P. Sojka et al. (Eds.): TSD 2014, LNAI 8655, pp. 285–292, 2014.

2 State of the Art

Using Wrapper Induction to extract information from structured Web pages has been studied extensively. Early studies focused on the DOM-tree representation of Web pages and learn a template that wrap data records in HTML tags, such as [9,12,13]. The challenges related to WI concern three main aspects: the need of training data in the case of supervised methods, the lack of semantic for extracted values in the case of unsupervised methods and the robustness problem in both cases. Supervised methods require manual annotation on example pages to learn wrappers for similar pages [9,11,5]. The number of required annotations can be drastically reduced by annotating pages from a specific website and then adapting the learnt rules to previously unseen websites of the same domain [14,7]. Completely unsupervised methods (e.g. RoadRunner [4] and EXALG [1]) do not require any training data, nor an initial extraction template (indicating which concepts and attributes to extract), and they only assume the homogeneity of the considered pages. If homogeneity is not assumed, cluster techniques can be used to obtain homogeneous pages [2]. The drawback of unsupervised methods is that the semantic of produced results is left as a post-process to the user. Hybrid methods [6] intend to find a tradeoff with these two limitations by proposing a supervised strategy, where the training data is automatically generated exploiting *LD*. The methodology proposed by [6], which we will adopt in this work, consists of three steps: (i) dictionary generation, (ii) annotation generation and (iii) pattern extraction. It suggests to build pertinent dictionaries from *LD* and then use them to automatically generate annotations of pages. These automatically generated annotations, potentially incomplete and imprecise, are used to discover the common structural patterns in the Web pages that encapsulate the target information. A limitation of this approach is that extremely noisy annotations can lead to learn incorrect patterns, and the strategy do not propose any check for the reliability of produced patterns.

3 Methodology

Our Wrapper Induction method has two inputs: a schema, defining objects to extract and set of homogenous Web pages. The schema specifies a set of *concepts* of interest $C = \{c_1, \ldots, c_i\}$ with their attributes $\{a_{i,1}, \ldots, a_{i,k}\}$. A set of homogenous Web pages W_{c_i} contains pages from the same website, describing entities of type c_i. They share a similar structure since they are generated using the same script.

We adopt a three steps methodology which consists of (i) dictionary generation, (ii) dictionary based annotation and (iii) pattern extraction, originally proposed by [6] and we introduce novel methods to implement each step. The intuition behind the methodology is that large scale dictionaries will help produce a good number of annotations. Dictionary based annotation is applied in a brute force way that can over-annotate, creating false positives. Good seeds in the dictionary will contribute to the creation of true positive annotations while bad seeds will create false annotations. If there are no good seeds in the dictionary only false positive annotations are created. The number of false positives ca be reduced, by reducing the number of bad seeds in the dictionaries. In the absence of true positives, we cannot generate a useful extraction

pattern, but we propose a strategy to detect unreliable patterns and we rather do not propose any for the attribute than proposing an incorrect one. In remaining of this Section we will describe our solution in detail.

Dictionary generation The goal of this step is to create sufficiently large gazetteers that are good representation of attributes of each concept. *LD* contains billions of facts for specific domains, and can be used as a large entity knowledge base in many tasks [8,10]. [6] propose to query available SPARQL endpoints[2], with a pertinent query for each concept c_i – attribute $a_{i,k}$ pair, thus obtaining a dictionary $d_{i,k}$ for each attribute $a_{i,k}$ of each concept c_i. We propose two simple strategies to limit noise in each dictionary. First, we classify each entry in the result set of the query according to very broad data types, i.e. *NUMBER*, *DATE*, *SHORT TEXT* and *LONG TEXT* using simple regular expressions. We assume the majority type to be the correct one for the dictionary and we discard all entries other than the majority type. After that, for each concept c_i we check for intersections in the dictionaries $d_{i,k}$ of all considered attributes $a_{i,k}$. Values present in more than one dictionary are likely to be either noise (as the wrong usage of properties is common on *LD* [6]) or ambiguous examples, therefore likely to generate misleading annotations. The dictionary generation process is completely independent from the data in W_{c_i}. No a priori knowledge about the data is introduced to this process and thus the dictionary $d_{i,k}$ is unbiased and universal for any extraction tasks concerning the pair c_i–$a_{i,k}$.

Web page annotation We generate annotations for each pair c_i - $a_{i,k}$, using the dictionary $d_{i,k}$. Each W_{c_i} can contain a number of Web pages varying from a few hundreds to several thousand (e.g. W_{c_i} could be the set of pages on the website `http://www.imdb.com/` describing films). Instead of generating annotations for the whole W_{c_i} we propose a simple strategy to reduce the number of required annotations. Although the annotation process is automatic and no human intervention is needed, limiting the number of pages to annotate is crucial to speed up the process at Web scale.

Algorithm 1 *annotate*$(W_{c_i}, d_{i,k})$

1: $M \leftarrow xpathDensity(W_{c_i})$
2: $M_{filt} \leftarrow filterXpath(M)$
3: $M_{match} \leftarrow matchXpath(M_{filt}, d_{i,k})$

Algorithm 2 *xpahDensity*(W_{c_i})

1: $trainingSize \leftarrow 0$; $M \leftarrow \emptyset$; $W_{train} \leftarrow \emptyset$
2: **while** new nodes are added to M **do**
3: $W_{train} \leftarrow selectRandom(W_{c_i}, n)$
4: $trainingSize \leftarrow trainingSize + n$
5: $M \leftarrow M + indexNodes(W_{train})$
6: **end while**

Algorithm 1 illustrates the main steps of the annotation procedure. The *xpahDensity* function takes as input all the Web pages in W_{c_i}, determines a sufficient subset $W_{train} \subseteq$

[2] A service to query a knowledge base via SPARQL `http://tinyurl.com/n9h3kce`

W_{c_i} and produces an index M of all text nodes in W_{train} and their values. In detail, *xpahDensity* (algorithm 2) starts by randomly selecting a small fixed number n of pages from W_{c_i} (e.g. $n = 15$). Each page is parsed into a DOM tree and, for each leaf node containing text, *xpahDensity* extracts (i) its text value and (ii) its *xpath*[3], which are added to the index M (function *indexNodes*). M will contain all possible *xpaths* (identifying textual nodes) found in the pages and all different content values for them. The process of adding pages to M is repeated iteratively, on the the next n random pages from W_{c_i}, until no new *xpath* are added to M by the current set of pages W_{train}. The stopping criteria does not consider the values of the nodes, but only checks if new structural elements are introduced by the set of pages. The intuition behind this procedure is that once we cover all structural patterns in the pages, we do not need to generate more annotations. The stop criteria also provides a useful information: the number of pages used before covering all possible path in the website gives an idea of the complexity of structure of website itself. On simple websites, the structure will be covered after using a small number of pages (as little as 45 pages, i.e. 3 iteration with $n = 15$) while for more complex websites the number of pages used can go up to 1000. The *filterXpath* function (algorithm 1) implements simple heuristics to filter out non useful *xpath* in M. The goal of this function is to make sure that boilerplate in the pages does not contribute to the generation of spurious annotations. *FilterXpath* removes all *xpath* that have the same value on all the training pages: these are likely to be formatting and menu items that do not contribute to the extraction process. The *matchXpath* function (algorithm 1) is the one generating the annotation examples. This function simulates a human identifying which nodes in the page contain the information we are interested in. *MatchXpath* generates a positive example every time that the text contained by a particular node matches a value in the dictionary $d_{i,k}$. If there is an exact match between the text content of a node and any item in $d_{i,k}$, the annotation is saved for the page, as a pair $\langle$*xpath, text value*$\rangle$.

Pattern Extraction The intuition beyond this step is that useful *xpaths* will be likely to match a bigger variety of dictionary entries, on a sufficiently large sample of web pages. Therefore the same *xpaths* producing different annotations over the set of Web pages are more likely to be useful than the ones with limited variety of annotations. For a particular attribute $a_{i,n}$ and a website collection W_{c_i}, starting from all generated annotations (M_{match}) and exploiting the *xpath* density (M_{filt}) we attribute a score to each *xpath* in M_{match}. Based on the hypothesis of structural consistency in W_{c_i}, we expect the majority of true positives to share the same or similar *xpath*. The collection of pages considered to calculate the score is the final set W_{train} in algorithm 2), from which M_{match} and M_{filt} are produced. For each *xpath*, using M_{match} and M_{filt}, the score takes into account three factors: (i) the number of different results produced and (ii) the number of matches with the dictionary, with respect to (iii) the number of pages in the collection. This is to favour *xpaths* which both produce values which are consistent with the dictionaries and apply to the majority of web pages, i.e. are likely to produce a different value for each page. The scoring function *score* is detailed in algorithm 3.

[3] `http://www.w3.org/TR/xpath/`

Algorithm 3 $score(M_{match}, M_{filt}, trainingSize)$

1: **for all** $xpath_i \in M_{match}$ **do**
2: $allVal_i \leftarrow |values(M_{filt}, xpath_i)|$
3: $margin_i \leftarrow allVal_i / trainingSize$
4: $matchVal_i \leftarrow |values(M_{match}, xpath_i)|$
5: $score_i \leftarrow matchVal_i / allVal_i + margin_i$
6: $X \leftarrow addPair(xpath_i, score_i)$
7: **end for**

Using M_{filt} we count the number of different values $allVal_i$ corresponding to $xpath_i$, regardless if they match or do not match the dictionary. We then calculate what we call a *margin* for $xpath_i$, which is the ratio between $allVal_i$ and the total number number of pages in the collection W_{train}. The *margin* indicates if an *xpath* is (i) highly applicable, produces a result for most pages, and (ii) informative , produces different results for different pages. Then we count the number $matchVal_i$ of different values corresponding to $xpath_i$ matching the dictionary, using M_{match}. The final $score_i$ is given by the linear combination of the margin $margin_i$ and the ratio between $matchVal_i$ and $allVal_i$.

We introduce a reliability strategy for extracted *xpaths*. We repeat Algorithm 1 multiple times, while applying the *score* function at each iteration (in this work we set the number of iterations to 20). In each iteration, we verify if the best scoring *xpath* and check if it converges; since pages for generating annotations are selected randomly by *xpathDensity* function, this might not be the case and different *xpaths* can be produced. If the highest scoring *xpath* is consistent, we mark it as reliable. In case different *xpaths* are produced at different runs, we check the compatibility of results produced by each *xpath* on all the pages and we pick the *xpath* with highest overlap with the pertinent dictionary.

4 Experiments

We perform experiments on a publicly available WI dataset [7], which covers 8 concepts, including *Autos, Books, Cameras, Jobs, Movies, NBA Players, Restaurants, and Universities*, with 10 websites per concept and around 2000 pages per website. The dataset is accompanied by ground truth values for 3 to 5 common attributes per concept. Our method depends on the availability of suitable dictionaries for each concept-attribute. For the concepts *Autos, Cameras* and *Jobs* our dictionary generation step could not generate dictionaries using *LD*, as we were not able to find a suitable class on *LD* to represent the concept. We tested the method on the concept-attributs: *Book* (*title, author, isbn, publisher, publish-date*), *Film* (*title, director, genre, rating*), *NBA player* (*name, team*),*Restaurant* (*name, address, phone, cuisine*), *University* (*name, phone, website*). To generate the dictionaries for the above concept-attributes we explore *LD* and we

manually create queries to retrieve possible values for all attributes of interest[4]. We executed the queries over different SPARQL endpoints and federated the results. We apply the noise filtering procedure (Section 3): we classify results of each query by data type, and only keep results classed as the majority type. For example, when creating the dictionary for *Book ISBN*, the majority of results are classified as number (e.g. 978-0-671-87743-9), while some are classified as short text (e.g. "0143017861(penguin group canada)"). We only keep in the dictionary the one classified as numbers. Once we created dictionaries for all attributes of each concept, we check for values present in more than one attribute dictionary; e.g. for the concept *Book*, "Allen Robert" is present both in the *author* and *publisher* dictionary, or "A Fictional Guide to Scotland" is both in *title* and *publisher*. We consider those as noise and remove them from all the dictionaries for the concept. The resulting dictionaries have varying sizes, ranging from less than 200 seeds for some attributes, to more than 70,000 seeds for others (an indication of the number of seeds for each dictionary can be found in Table 1).

Table 1. Results of our method in terms of Precision (P), Recall (R) and F measure (F). Results are reported as the average on all websites for each concept (or domain) and attribute.

Domain	attribute	P	R	F	Domain	attribute	P	R	F	Average		
book	author	0.99	0.86	0.88	nbaplayer	name	0.78	0.77	0.77	**P**	**R**	**F**
book	publisher	0.59	0.55	0.56	nbaplayer	team	1.00	0.93	0.95	0.78	0.74	0.75
book	title	0.90	0.90	0.90	restaurant	cuisine	0.39	0.37	0.38			
film	director	0.77	0.71	0.74	restaurant	name	0.88	0.85	0.86			
film	genre	0.66	0.64	0.65	university	name	1.00	1.00	1.00			
film	title	0.77	0.76	0.77	university	website	1.00	0.94	0.97			

Results in terms of Precision, Recall and F-measure are reported in table 1, for each concept and attribute. We excluded the attributes: *Book* (*isbn, publish-date*), *Film* (*rating*), *NBA player* (*name, team*), *Restaurant* (*address, phone*), *University* (*phone*), as the generated dictionaries had no coverage on all the websites of the dataset (some were too small to be representative). The *coverage* indicates the percentage of true answers (obtained from the groundtruth) which are contained in our generated dictionaries. In some cases, none of the true answers is covered by the specific dictionary, which means that the training data will not contain any true positive at all, but only false positive (i.e. dictionary seeds matching nodes of the pages other that the node of interest). This is a limitation of our current method, partially overcome by our reliability strategy (Section 3) which checks the compatibility of results with the dictionary (in terms of datatype). This will discard highly incorrect patterns, although it will not avoid the selection of incorrect patterns that extract results of compatible types. Our method achieves an average (micro-average) F measure of 0.78 (with P = 0.81 and R = 0.77)

[4] An example of SPARQL query to retrieve film titles could be "SELECT DISTINCT ?title WHERE{ ?film a ⟨`http://schema.org/Movie`⟩; ⟨`http://dbpedia.org/property/name`⟩?title . FILTER (langMatches(lang(?title), 'EN')).}".

considering all results, while when excluding websites where the dictionaries have no coverage (marked as bold in Table 1) the F measure rises to F 0.85 (with P = 0.87 and R = 0.84).

Table 2. Precision (P), Recall (R) and F measure (F) by concept, including and excluding websites with no coverage. Comparison with state of the art methods.

	All results			Covered websites			SoA	
CONCEPT	P	R	F	P	R	F	F[7]	F[6]
book	0.83	0.77	0.78	0.88	0.83	0.84	0.87	0.78
film	0.73	0.70	0.72	0.79	0.76	0.78	0.79	0.76
NBA player	0.88	0.84	0.86	1.00	0.96	0.97	0.82	0.87
restaurant	0.62	0.60	0.61	0.69	0.67	0.68	0.96	0.69
university	1.00	0.97	0.98	1.00	0.97	0.98	0.83	0.91
	0.81	0.78	0.79	0.87	0.84	0.85	0.85	0.80

Table 2 reports the average of results by concept (both including and excluding websites with no coverage) and the overall macro-average. It also compare results with available figures from [7] and [6]. When comparing to [6], figures excluding no coverage websites should be considered, as they discarded them as well for computing final figures. Our method (F=0.85) outperforms [6] (F=0.80), and achieves same performance as [7] (which is a supervised method).

One assumption of our work is that the method works as long as the concept is fairly covered on LD. Table 2 shows that results for the concept *Restaurant* are worst than average. This is compatible with the fact that dictionaries for this concept are less rich than others. For example we could collect only 629 restaurant names, against 27,720 book titles or 72,354 film titles. Given the continuous and exponential growth of *LD*, it is reasonable to believe that the availability and coverage of resources will get better and better.

5 Conclusions and Future Work

We propose a novel method for Wrapper Induction, where the training data is automatically generated using *Linked Data* and obtained results comparable with a supervised solution. Two strong assumptions are taken. First, that LD contains resources to generate training data; this is not always the case, but it is reasonable to believe that coverage will improve fast. Second, our method assumes that the values to extract are fully and exactly contained in a page node. Although this a common feature amongst websites for observed attributes, it is not always the case, e.g. in the used dataset, the film website *iheartmovies* includes the film title and release year in a single HTML node, therefore our method will not find a match, although it does not return a wrong pattern (none is returned). Ongoing work is aimed at addressing these limitations.

References

1. Arasu, A., Garcia-Molina, H.: Extracting structured data from web pages. In: ACM SIGMOD/PODS'03. pp. 337–348. ACM (2003), http://dl.acm.org/citation.cfm?id=872799
2. Blanco, R., Halpin, H., Herzig, D., Mika, P.: Entity search evaluation over structured web data. In: SIGIR'11 (2011), http://www.aifb.kit.edu/images/d/d9/EOS-SIGIR2011.pdf. pp. 65–71
3. Carlson, A., Schafer, C.: Bootstrapping information extraction from semi-structured web pages. ECML-PKDD'08 (2008), http://www.springerlink.com/index/m003r7p6h52k366k.pdf
4. Crescenzi, V., Mecca, G.: Automatic information extraction from large websites. Journal of the ACM 51(5), 731–779 (Sep 2004), http://portal.acm.org/citation.cfm?doid=1017460.1017462
5. Dalvi, N., Kumar, R., Soliman, M.: Automatic wrappers for large scale web extraction. VLDB'11 4(4), 219–230 (2011), http://dl.acm.org/citation.cfm?id=1938547
6. Gentile, A.L., Zhang, Z., Augenstein, I., Ciravegna, F.: Unsupervised wrapper induction using linked data. In: K-CAP'13. pp. 41–48. ACM (2013), http://doi.acm.org/10.1145/2479832.2479845
7. Hao, Q., Cai, R., Pang, Y., Zhang, L.: From One Tree to a Forest : a Unified Solution for Structured Web Data Extraction. In: SIGIR 2011. pp. 775–784 (2011), http://research.microsoft.com/pubs/152207/StructedDataExtraction_SIGIR2011.pdf
8. Kobilarov, G., Bizer, C., Auer, S., Lehmann, J.: DBpedia-A Linked Data Hub and Data Source for Web and Enterprise Applications. In: WWW'09. pp. 1–3 (2009), http://jens-lehmann.org/files/2009/dbpedia_www_developers.pdf
9. Kushmerick, N.: Wrapper Induction for information Extraction. In: IJCAI'97. pp. 729–735 (1997), http://www.icst.pku.edu.cn/course/mining/11-12spring/%E5%8F%82%E8%80%83%E6%96%87%E7%8C%AE/10-01WrapperInductionforInformationExtraction.pdf
10. Mulwad, V., Finin, T., Syed, Z., Joshi, A.: Using linked data to interpret tables. In: COLD'10 (2010). pp. 1–12
11. Muslea, I., Minton, S., Knoblock, C.: Active Learning with Strong and Weak Views: A Case Study on Wrapper Induction. IJCAI'03 pp. 415–420 (2003), http://www.isi.edu/integration/papers/muslea03-ijcai.pdf
12. Muslea, I., Minton, S., Knoblock, C.: Hierarchical wrapper induction for semistructured information sources. Auton Agents and Multi-Agent Syst pp. 1–28 (2001), http://www.springerlink.com/index/XMG5W31380116467.pdf
13. Soderland, S.: Learning information extraction rules for semi-structured and free text. Mach. Learn. 34(1-3), 233–272 (Feb 1999), http://dx.doi.org/10.1023/A:1007562322031
14. Wong, T., Lam, W.: Learning to adapt web information extraction knowledge and discovering new attributes via a Bayesian approach. Knowledge and Data Engineering, IEEE 22(4), 523–536 (2010), http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4906994