Parameterless Information Extraction Using (k,l)-Contextual Tree Languages

Stefan Raeymaekers, Maurice Bruynooghe

K.U.Leuven, Dept. of Computer Science, Celestijnenlaan 200A, B-3001 Leuven

Abstract

Recently, several wrapper induction algorithms for structured documents have been introduced. They are based on contextual tree languages and learn from positive examples only but have the disadvantage that they need parameters. To obtain the optimal parameter setting, they use precision and recall. This goes in fact beyond learning from positive examples only. In this paper, a parameter estimation method for a wrapper based on (k, l)-contextual tree languages is introduced that is solely based on a few positive examples. Experiments show that the quality of the wrappers is very close to that of wrappers with the optimal parameter setting.

1 Introduction

Information extraction(IE) from structured documents (HTML, XML), aims at extracting specific information from structurally similar documents. Often it is referred to as *wrapper induction*. Examples are [10, 12, 14, 3, 8, 9, 15]. Most interest is in learning from positive examples only. The elements of a HTML-page can be divided between those to be extracted and the others. We define a positive example as a HTML page where one of the elements to be extracted is marked with a special marker and all other elements are unmarked. A HTML page with multiple elements to be extracted can therefore result in multiple examples. As a seminal result of [6] states that regular languages cannot be learned from positive examples only, researchers focussed on identifying less expressive, but learnable, subclasses of regular languages. Learnable subclasses of regular string languages are defined in [2, 11, 5, 1]. More recently, also learnable subclasses of regular tree languages have been introduced [4, 7, 15]. It is argued in [8, 9] that IE from structured documents benefits from using the tree structure in the wrapper. Indeed, the authors obtain superior results on various benchmarks. In [15], the class of (k,l)-contextual tree languages, a subclass of the regular unranked tree languages, is introduced. An inference algorithm for the class is given and it is shown that a wrapper induction algorithm based on a variant of (k,l)-contextual tree languages gives the best results to date.

The algorithms in [8, 9] need one parameter; that of [15] needs two. The common solution is to select the parameter(s) that give the best results when evaluated on a test set. In [8], the optimal parameter is selected by cross-validation on a **completely** marked training set. This implies that a field that is never marked is in fact a negative example. Hence, at the conceptual level, one can argue that the learning is not from positive examples only. At the practical level, when using the algorithm to extract information from a new data set, one has to mark every field to be extracted in each document used for training. In a truly learning from positive examples setting, it should suffice to mark a few fields only.

In this paper we propose a true parameterless algorithm. We use the algorithm presented in [15] and add a parameter estimation algorithm in an initial step, in which the parameters are estimated on the training set alone. This algorithm uses as extra information the number of extractions from some evaluation set (consisting of unmarked trees) made with a wrapper learned from the training set. This evaluation set consists of a number of **unmarked** documents. This set can be the training set, with the markers stripped off. Or some other unmarked documents from the domain of the extraction task.

We start in Section 2 with a brief sketch of the method from [15]. In Section 3 we give some observations about the evolution of the number of extractions and introduce our method. We describe the details of our implementation in Section 4 and present an experimental validation of our approach in Section 5. We conclude in Section 6.

2 The IE Algorithm

2.1 Preliminaries

We define the alphabet Σ as a finite set of symbols. The set of all unranked trees over Σ is defined as $T(\Sigma) = \{f(s) \mid f \in \Sigma, s \in T(\Sigma)^*\}$. We usually denote $f(\epsilon)$, where ϵ is the empty sequence, by f. A *tree language* is any subset of $T(\Sigma)$. The *depth* of a tree is the number of nodes on the longest path from the root, i.e., $depth(f) = 1$ and $depth(f(t_1, \ldots, t_n)) = 1 + max(\{depth(t_1), \ldots, depth(t_n)\})$. The set of (k,l)-roots of a tree $f(t_1, \ldots, t_n)$ is the singleton $\{f\}$ if l=1, otherwise it is the set of trees obtained by extending the root f with k $(k,l-1)$-roots taken from successive children of f (all children if $k < n$).

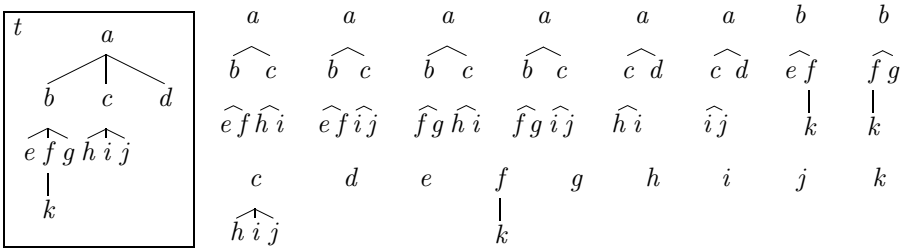
Formally (using $r_{k,l}(t)$ to denote the elements of $R_{k,l}(t)$):

$$R_{(k,l)}(f(t_1 \ldots t_n)) = \begin{cases} \{f\} & \text{if } l = 1 \\ \{f(r_{(k,l-1)}(t_1), \ldots, r_{(k,l-1)}(t_n))\} & \text{if } l > 1 \text{ and } k > n \\ \{f(r_{(k,l-1)}(t_p), \ldots, r_{(k,l-1)}(t_{p+k-1})) \mid p = 1, \ldots, n-k+1\} & \text{otherwise} \end{cases}$$

Finally, a (k,l)-root of a subtree of t is a (k,l)-fork of t; hence the set of (k,l)-forks is given by:

$$F_{(k,l)}(f(t_1 \ldots t_n)) = \{t \mid t \in R_{(k,l)}(f(t_1 \ldots t_n))\} \bigcup_{i=1}^{n} F_{(k,l)}(t_i)$$

Example 1 *Below we show graphically the (2,3)-forks of a tree t. The first 6 of these forks, are the (2,3)-roots of t.*



2.2 Approach

In the learning phase, a wrapper is learned from positive examples. The learned wrapper is expected to accept only correctly marked documents (This is with a marker on a required element). In the extraction phase, several runs are performed on each document. In each run, one single candidate target is replaced by the marker; it is extracted when the automaton accepts the marked document.

The basic idea[15] is to represent the wrapper by a set of (k,l)-forks (generalizing the notion of k-grams[1]). To obtain better generalization, only forks that contain the marker are taking into account. Learning the wrapper simply consists of collecting all forks (with a marker) from the examples. A run accepts a document when all the forks that contain a marker occur in the wrapper; k and l are the parameters of the algorithm, they determine the depth and width of the context that is used to recognize a target field.

Another generalization concerns the text nodes in the leaves of the tree. Typically, many unseen text strings will occur during the extraction. Therefor, all text fields are replaced by the same symbol. One exception is made for the so called *distinguishing context* [9]. If all examples have the same text field on a fixed distance to the target field, then such field (if several occur, the one most close to the target is selected) is retained during learning and extraction.

3 Parameter Estimation

We define *maxl* as the maximal depth found in the trees of the training set, we define *maxk* as the maximum number of children of a node found in the training set. The search space of our parameters starts at (1,1)[1]; k and l are bounded by respectively *maxk* and *maxl*. To evaluate the quality of a pair of parameters we use the metrics precision(P), recall(R) and $F1$ score. Let E be the number of text nodes extracted from the test set, C the number of correctly extracted text nodes, and T the total number of text nodes in the test set. Then we can define these metrics as $P = C/E$, $R = C/T$, and $F1 = 2PR/(P + R)$, the harmonic mean of P and R. Given a training set and a test set, we define the **optimal parameters** (k,l) as those parameters that result in the best $F1$ score when a wrapper, learned

[1]With $l = 1$, k does not matter; the wrapper consists of a single fork with the marker as single node.

from the given training set with these parameters, is evaluated on the given test set. As stated before, we do not have the test set at our disposal, therefor we have to estimate these optimal parameters based on the training set only.

As is the case for other local languages (k-contextual languages[11, 1], k-testable languages[5], and k-testable tree languages[4, 7]), (k,l)-contextual tree languages[15] are monotone; i.e., if L and L' are the languages learned (from a given training set), for parameter settings (k,l) and (k',l') respectively , then $k \leq k'$ and $l \leq l'$ implies that $L' \subseteq L$. This property also holds for the variant we use to learn wrappers. The k and l parameters define a kind of window around the marked node. The language learned with the smallest window ($k = 1, l = 1$) is the most general; it extracts everything hence recall is 100% and precision is low. As the (k,l)-window is enlarged, more and more of the features surrounding the marked node are taken into account, less fields are extracted and precision increases, however, at least in the beginning, recall is likely to stay at 100%. If the extraction task can be expressed with a contextual language then —if enough examples are given— precision will reach 100% while recall is still at 100%. Typically, there will be a region of parameter settings for which the $F1$-score remains 100%. In that region, the number of extracted fields remains constant. When the (k,l)-window is further enlarged, a point will be reached where the number of extracted fields start to drop again; while precision remains at 100%, recall (and hence the $F1$-score) start to drop again at a faster pace.

Hence, for the parameter setting $(1, 1)$, all fields are extracted. As k and l increase, the number of extracted fields quickly goes down; however one can expect the number of extracted fields to stabilize in the region with optimal $F1$-score and then to further go down again. This behavior is illustrated in Example 2.

Example 2 *Table 1 and Table 2 show the number of extractions for two data sets where the wrapper is learned from five random examples (see Section 5 for details about the data sets used). The extractions are performed on the (unmarked) pages that contained the examples. In Table 1 we see a large region with 115 extractions (corresponding to the optimal $F1$-score). In Table 2 there is a region with 98 extracted fields (corresponding to the optimal $F1$-score) that stretches over 2 l-values, but it does not cover a 2×2 region.*

Table 1: # of Extractions for okra-1

l \ k	1	2	3	4	5	6	7
1	1267						
2	141	141	141	141	141	141	141
3	126	115	115	115	115	115	115
4	120	115	115	115	115	115	115
5	120	115	115	115	115	115	115
6	120	115	115	111	111	111	111
7	115	115	98	97	90	90	75

Table 2: # of Extractions for bigbook-3

l \ k	1	2	3	4	5	6	7
1	969						
2	412	402	402	402	402	402	402
3	309	299	299	299	299	299	299
4	309	98	98	98	98	98	98
5	309	98	82	51	38	31	27

Starting from a parameter setting $(1, 2)$, our implementation searches for a region where the number of extracted fields changes minimally. However, the

search is interrupted when it becomes clear that the model becomes too specific. Two tests are used to check this:

- A clear indication of a too specific model is when the number of extractions equals the number of examples.
- Another test checks whether a value of l is reached for which the number of extractions keeps decreasing for increasing k-values. This behavior is seen in lines 7 and 5 of respectively Table 1 and Table 2. The explanation is that the depth of the marked forks (l) is becoming so large that they contain a common ancestor of the target fields. For such l-value, the subtrees containing a target have a different position (first, second, ..., last) as a child of the common ancestor. With increasing k, the automaton can distinguish more and more of these positions and the number of extractions decreases unless there is an example for every position.

Precision will never reach 100% when (k,l)-contextual tree languages cannot model the extraction task. When it is expressive enough, but not enough examples are available for learning, the $F1$-score will not reach 100%. However, still, there is likely a region where the $F1$-score is optimal and the number of extracted fields stabilizes.

4 Implementation

Let E be the training set, $W(k,l)$ the wrapper learned for the parameters k and l and T the evaluation set. The number of extractions made on T with $W(k,l)$ is denoted as $ne(k,l)$. To evaluate the quality of a wrapper we consider a (2×2)-region and calculate the sum of the differences between $ne(k,l)$ and three of its specializations, i.e., we define $\mathit{diff}(k,l) = (ne(k,l) - ne(k,l+1)) + (ne(k,l) - ne(k+1,l)) + (ne(k,l) - ne(k+1,l+1))$.

Algorithm 1 shows a high level description of our implementation. The initialization computes the bounds $maxk$ and $maxl$ on respectively k and l and an upper bound $\mathit{maxdiff}$ on $\mathit{diff}(k,l)$. The algorithm searches for the parameter setting with the minimal difference and returns the values closest to the origin $(1,1)$ in case the corresponding parameter setting is not unique. The Manhattan distance is used to measure the distance to the origin and the outer loop of the algorithm iterates over the Manhattan distances.

The inner loop iterates over the possible l-values for a given Manhattan distance (starting from $l = 2$ since $l = 1$ returns all candidate targets). After calculating the corresponding k-value, $\mathit{diff}(k,l)$ is computed. Doing so requires knowledge about the number of extractions by $W(k,l)$, $W(k+1,l)$, $W(k,l+1)$, and $W(k+1,l+1)$. The values $ne(k,l), ne(k+1,l), \ldots$ are tabled (not shown in the code), so that the number of extractions by each wrapper is counted only once. If the difference is better than the best value so far, then the best values are updated. The second if-clause checks whether the optimal solution has already been found. This is the case when $\mathit{bestdiff} = 0$ or one of the stopping criteria as described in Section 3 is met. The latter is computed by the function $\mathit{tooSpecific}(k,l)$ that also uses the tabled

$ne(\ldots)$ values. More precisely, the function succeeds when either $ne(k+1,l+1) = \#(E)$ or $ne(1,l+1)$, $ne(2,l+1)$, $\ldots$, $ne(k+1,l+1)$ contains more than 3 different values. The stopcriterion is always met before the loop boundaries are reached.

Algorithm 1 Parameter Estimation

Input: The training set E and the evaluation set T.
Output: The estimated parameters as a pair (k,l).

```
1: (maxk, maxl) := calcBoundaries(E)
2: bestdiff := 3 * ne(1,1)
3: for d = 3 to maxk + maxl do
4:    for l = 2 to d − 1 do
5:       k := d − l
6:       if diff(k,l) < bestdiff then
7:          (bestk, bestl) := (k,l); bestdiff := diff(k,l)
8:       end if
9:       if bestdiff = 0 or tooSpecific(k,l) then
10:         return (bestk, bestl)
11:      end if
12:   end for
13: end for
```

5 Experiments

We evaluate our approach on the WIEN data sets (these are available at the RISE repository[13]). Some of the sets are discarded since they do not include labels. The other sets are split in different tasks: a task aiming at the extraction of a n-tuple is splitted in n extraction tasks. We refer to each field extraction task, with the name of the original data set together with the index of the field in the tuple. From these tasks we keep only those that extract a complete node, resulting in a total number of 36 extraction tasks. These are the same data sets used in [15], where is shown that (k,l)-contextual tree languages are expressive enough to model each of these extraction tasks.

For each task, 5 randomly picked examples are used as training set E. The other examples are used as test set. The evaluation set T consists of the pages that contain the examples from E. This set is smaller than the number of examples in cases where several examples were selected from the same page. Each experiment is performed 5 times, each time with a different set of random examples and the mean $F1$-score is computed. The results of this experiment are shown in Table 3. The column *Data set* refers to an extraction task; the column *ctx* indicates whether a distinguishing context was used for that task. The column *opt.* gives the $F1$-score obtained with the optimal parameter setting as computed in [15] and the column *est.* the $F1$-score obtained with out estimated parameters.

The estimated parameters are suboptimal on those tasks where the optimal parameter setting gives an $F1$-score below 100% (for only 5 examples in the train-

Table 3: Results for data sets with 5 examples

Data set	ctx	opt.	est.	Data set	ctx	opt.	est.	Data set	ctx	opt.	est.
s1-1		100.0	100.0	s13-2		100.0	100.0	s25-2		100.0	100.0
s1-3		98.7	93.4	s13-4		100.0	100.0	s30-2		100.0	96.0
s1-4		100.0	100.0	s14-3		100.0	100.0	iaf-1	✔	100.0	100.0
s3-2		100.0	100.0	s15-2		100.0	100.0	iaf-2	✔	100.0	89.4
s3-3		100.0	100.0	s19-4		100.0	100.0	qs-1		96.6	65.6
s4-1		100.0	100.0	s20-3	✔	100.0	100.0	qs-2		87.8	36.8
s5-2		98.9	94.7	s20-4	✔	100.0	100.0	bigbook-2		100.0	97.3
s8-2		100.0	100.0	s20-5	✔	100.0	100.0	bigbook-3		100.0	96.9
s8-3		100.0	100.0	s20-6	✔	100.0	100.0	okra-1	✔	100.0	100.0
s10-2		100.0	100.0	s22-2		100.0	99.8	okra-2	✔	100.0	100.0
s10-4		100.0	100.0	s23-1		100.0	100.0	okra-3	✔	100.0	100.0
s12-2		98.5	98.4	s23-3		100.0	100.0	okra-4	✔	100.0	100.0

ing set). A perfect $F1$-score is obtained on all but 5 other tasks. Inspecting the outcome of the 180 individual experiments learned us that the same $F1$-score is obtained as with the optimal parameter setting for 153 experiments. The estimated k- and l-values ranged respectively from 1 to 5 and from 2 to 5.

We also compared our parameter estimation method for (k, l)-contextual tree languages with the method described in [8] (results reported in [15]). We obtained a better $F1$-score in 22 tasks out of the 36 and a worse score in only 3 tasks.

6 Conclusion

We have presented an parameterless wrapper induction algorithm. This algorithm is a combination of an existing wrapper induction algorithm [15], based on automata for (k,l)-contextual tree languages, and a heuristic to estimate its parameters.

The parameter estimation heuristic is based on the assumption that the differences in the number of extracted fields reaches a minimum around the optimal parameter setting.

The experimental results show the validity of this assumption. Most of the time we obtain a $F1$-score that is equal or very close to the score obtained with the optimal parameter setting. The strength of our approach is that it learns from a few positive examples only. Hence the user intervention in learning a new extraction task is very limited.

References

[1] Helena Ahonen. *Generating grammars for structured documents using grammatical inference methods.* PhD thesis, University of Helsinki, Department of Computer Science, 1996.

[2] Dana Angluin. Inference of reversible languages. *Journal of the ACM (JACM)*, 29(3):741–765, 1982.

[3] Boris Chidlovskii, Jon Ragetli, and Maarten de Rijke. Wrapper generation via grammar induction. In *Proc. 11th European Conference on Machine Learning (ECML)*, volume 1810, pages 96–108. Springer, Berlin, 2000.

[4] Pedro García. Learning k-testable tree sets from positive data. Technical report, Technical Report DSIC-ii-1993-46, DSIC, Universidad Politecnica de Valencia, 1993.

[5] Pedro García and Enrique Vidal. Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(9):920–925, 1990.

[6] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[7] Timo Knuutila. Inference of k-testable tree languages. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition: Proc. of the Intl. Workshop*, pages 109–120, Singapore, 1993. World Scientific.

[8] Raymondus Kosala, Maurice Bruynooghe, Hendrik Blockeel, and Jan Van den Bussche. Information extraction from web documents based on local unranked tree automaton inference. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 403–408. Morgan Kaufmann, 2003.

[9] Raymondus Kosala, Jan Van den Bussche, Maurice Bruynooghe, and Hendrik Blockeel. Information extraction in structured documents using tree automata induction. In *PKDD*, volume 2431 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2002.

[10] Nickolas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[11] S. Muggleton. *Inductive Acquisition of Expert Knowledge.* Addison-Wesley, 1990.

[12] Ion Muslea, Steve Minton, and Craig Knoblock. A hierarchical approach to wrapper induction. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 190–197, Seattle, WA, USA, 1999. ACM Press.

[13] Rise (1998). a repository of online information sources used in information extraction tasks. [http://www.isi.edu/info-agents/RISE/index.html]. University of Southern California, Information Sciences Institute.

[14] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.

[15] Raeymaekers Stefan, Maurice Bruynooghe, and Jan Van den Bussche. Learning (k,l)-contextual tree languages for information extraction. Technical Report CW-390, Department of Computer Science, Katholieke Universiteit Leuven, 2004.