Generalized stochastic tree automata for multi-relational data mining

Amaury Habrard, Marc Bernard, and François Jacquenet

EURISE – Université de Saint-Etienne – 23, rue du Dr Paul Michelon
42023 Saint-Etienne cedex 2 – France
{Amaury.Habrard,Marc.Bernard,Francois.Jacquenet}@univ-st-etienne.fr

Abstract. This paper addresses the problem of learning a statistical distribution of data in a relational database. Data we want to focus on are represented with trees which are a quite natural way to represent structured information. These trees are used afterwards to infer a stochastic tree automaton, using a well-known grammatical inference algorithm. We propose two extensions of this algorithm: use of sorts and generalization of the infered automaton according to a local criterion. We show on some experiments that our approach scales with large databases and both improves the predictive power of the learned model and the convergence of the learning algorithm.

Keywords. Stochastic tree automata, multi-relational data mining, generalization, sorts.

1 Introduction

For many years, the use of relational databases is continually increasing in all domains of activities. Companies want to store more and more information in their databases and their activities now really depend on them. It is now well known that a large amount of knowledge may be hidden inside these large databases. That is the reason why machine learning and knowledge discovery in databases have known a fast expansion during the last decade.

Most of existing techniques work on a flat representation of the database, that is on one table built from a join on multiple tables [9, 2, 20]. Unfortunately, flattening data leads to lose some information since each example must have the same representation. A few attempts have been made to learn from databases without flattening data [12, 10, 16]. That leads to the emergence of an active field of research, called multi-relational learning [11]. For example the ILP framework [21] allows learning from a representation of the database as a subset of first order logic, keeping the structuring of data. [13] proposed the concept of probabilistic relational models that extends the standard attribute-based Bayesian network representation to take into account the relational structure on different entities.

Our approach can be placed in the same context. We try to collect in a tree all the information relative to each example. This is done using relations between tables. This kind of representation is quite natural and allows to have trees of variable size for various examples, depending on the information we have about them. In this paper we present a method for learning a distribution of data stored in a particular table of a relational database, taking into account the related information stored in other tables. Our method follows several steps. We first generate a learning sample containing one tree per record of the table we want to focus on. Then we infer a stochastic tree automata following the distribution of the learning sample. We improve the inference algorithm proposing the use of sorts during the inference, and introducing a local generalization technique. We illustrate the relevance of these two contributions on some experiments.

The paper focuses on grammatical inference aspects of this approach. We first present the generation of trees from a relational database. Then we define the inference of stochastic many-sorted tree automata and we introduce generalized tree automata. Finally we evaluate our approach on a relational database of the PKDD'01 discovery challenge [4].

2 Capturing the information in a tree

A relational database stores information about multiple entities. In order to learn some knowledge from one entity of the database, we focus on a particular table that we call the root table. We use a tree structure to capture in a tree all the information related to one row of the root table. The basic idea to build such a tree is to generate a root node labelled with the name of the root table. Then for each attribute value of the row we construct a subtree. The process computes recursively the subtrees corresponding to foreign keys. For example

we may want to discover some knowledge about products stored in the database relying on the schema of figure 1.

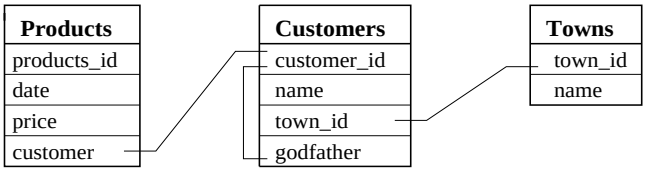

Fig. 1. Example database

If the database contains data of figure 2, the information related to the product 1 is represented by the tree of figure 3.

Products			
product_id	date	price	customer
1	11/11/2001	120	2
...	...	...	...

Customers			
customer_id	name	town_id	godfather
1	Jones	2	
2	Smith	1	1
...	...	...	...

Towns	
town_id	name
1	hills
2	city
...	...

Fig. 2. Data example

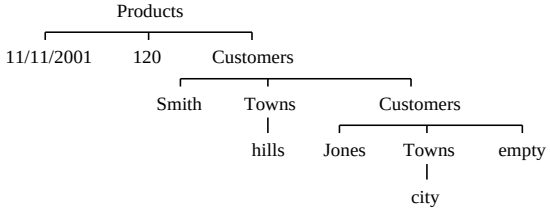

Fig. 3. Generated tree

To present the algorithm which computes the generation of trees we introduce some notations. We consider a database $\mathcal{T}$ with n tables $T_1, \ldots, T_n$. Each table is made up of a set of attributes $A(T_i) = \{a_{i_1}, \ldots, a_{i_{n(i)}}\}$ where $n(i)$ is the number of attributes of the table T_i. We denote $T_i.a_{i_k}$ the attribute a_{i_k} of the table T_i. We define the set of foreign keys $F(T_i) = \{a_{i_1}, \ldots, a_{i_m}\}$ such that $F(T_i) \subseteq A(T_i)$ and each $a_{i_k} \in F(T_i)$ is a reference to an attribute $T_j.a_{j_l}$ (what we denote $T_i.a_{i_k} : T_j.a_{j_l}$).

Algorithm 1 describes the construction of a tree associated with one row of the root table. For example the tree of figure 3 is obtained calling Generate_tree($Products, product\_id, 1$).

Generate_tree(T_i,$target\_attribute$,$value$)
Result: Node
begin
  $p_1, \ldots, p_{n(i)} \leftarrow$ SELECT $a_{i_1}, \ldots, a_{i_{n(i)}}$ FROM T_i
    WHERE $T_i.target\_attribute$=$value$
  $N \leftarrow T_i(p_1, \ldots, p_{n(i)})$
  foreach $a_{i_k} \in F(T_i)$ *such that* $T_i.a_{i_k} : T_j.a_{j_l}$ **do**
    $p_k \leftarrow Generate\_tree(T_j, a_{j_l}, p_k)$
  end
  Return N
end

Algorithm 1: Tree generation

A bias language, that we do not detail here, allows us to control the generation of trees. A SQL query can be specified to select only a part of rows of the root table. We may ignore a subset of attributes of each table. We can also bound the depth of trees to avoid infinite trees.

Generating trees from the database produces a training set of trees representing a concept stored in the database. Based on this set we now aim at learning a probabilistic tree grammar represented by a stochastic tree automaton.

3 Inference of stochastic tree automata

Tree automata [15, 8] define a regular language on trees as finite automata define a regular language on strings. Note that we consider ordered trees, that is trees where left to right order between siblings is significant. Stochastic tree automata are an extension of tree automata which define a statistical distribution on the tree language recognized by the automata. Learning tree automata has received many attention for some years. For example [14] and [19] proposed some algorithms for learning tree automata. In the probabilistic framework, [23] proposed a framework for inducing probabilistic grammars. In the context of tree automata, [6] proposed an efficient algorithm to learn stochastic tree automata. [1] dealt with learning stochastic tree grammars to predict protein secondary structure. [22] presented a generalization of k-gram models for stochastic tree languages.

Our inference procedure is an extension of [6], that takes sorts into account, defining stochastic many-sorted tree automata (SMTA). We briefly present SMTA before their inference procedure.

Definition 1 *A signature Σ is a 4-tuple (S, X, α, σ). S is a finite set whose elements are called sorts. X is a finite set whose elements are called function symbols. α is a mapping from X into $\mathbb{N}$. $\alpha(f)$ will be called the arity of f. σ is a mapping from X into S. $\sigma(s)$ will be called the sort of s.*

Note that in our approach the signature is learned from the learning sample using a basic procedure [3]. A symbol a appears at position (f, n) if it is the n^{th} argument of a tree constructed on the symbol f. We define equivalence classes in the following way: two symbols are equivalent if they appear at least one time at the same position. A sort is then associated with each equivalence class.

A set of sorted trees can be recognized by a stochastic many-sorted tree automaton.

Definition 2 *A stochastic many-sorted tree automaton is a 5-tuple $(\Sigma, Q, r, \delta, p)$. Σ is a signature (S, X, α, σ). $Q = \cup_{s \in S} Q^s$ is a finite set of states, each state having a sort in S. $r : Q \longrightarrow [0, 1]$ is the probability for the state to be an accepting state. $\delta : X \times Q^* \longrightarrow Q$ is the transition function. $p : X \times Q^* \longrightarrow [0, 1]$ is the probability of a transition.*

A SMTA parses a tree using a bottom-up strategy. A state and a probability are associated with each node of the tree. The labelling of each node is defined by the transition function. The tree is accepted if the probability of its root node is strictly positive. Given a SMTA A, the probability of a tree t is computed as follows:

$$p(t \mid A) = r(\delta(t)) \times \pi(t)$$

where $\pi(f(t_1, \ldots, t_n))$ is recursively computed by:

$$\pi(t) = p(f, \delta(t_1), \ldots, \delta(t_n)) \times \pi(t_1) \times \cdots \times \pi(t_n)$$

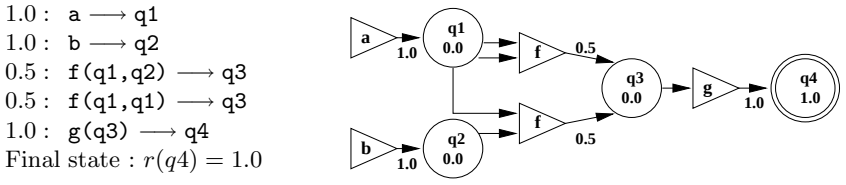

Fig. 4. An example of stochastic tree automaton

Example The automaton defined on figure 4 recognizes the tree $t = g(f(a, b))$ with the following probability:

$$\begin{aligned} p(t|A) &= r(\delta(g(f(a,b)))) \times \pi(g(f(a,b))) \\ &= r(q4) \times p(g, \delta(f(a,b))) \times \pi(f(a,b)) \\ &= r(q4) \times p(g, q3) \times p(f, \delta(a), \delta(b)) \times \pi(a) \times \pi(b) \end{aligned}$$

$$= r(q4) \times p(g, q3) \times p(f, q1, q2) \times p(a) \times p(b)$$
$$= 1.0 \times 1.0 \times 0.5 \times 1.0 \times 1.0$$
$$= 0.5$$

We are interested in producing consistent SMTA. Such automata define a statistical distribution over all trees built on a given alphabet. Therefore we have to ensure the sum of the probabilities of all these trees equals to one.

Algorithm 2 gives an idea of the main steps of inference. We do not detail the procedure here, the interested reader may refer to [6]. The input of the algorithm is a training set S of trees and the output is a SMTA which respects the distribution over S. The algorithm computes the transition function considering

Data: S: a training set,
$\Sigma = (X, S, \sigma, \alpha)$: a signature
Result: $A = (\Sigma, Q, \delta, p, r)$: a SMTA
begin
  $W \leftarrow$ Subtrees of S
  $States \leftarrow \emptyset$
  while $W \neq \emptyset$ **do**
    $x \leftarrow g(t_1, \ldots, t_n) = min\ W$
    $W \leftarrow W \backslash \{x\}$
    if $\exists y \in States \mid \sigma(x) = \sigma(y)$ *and* $comp(x, y, \alpha)$ **then**
      $\delta(g, [t_1], \ldots, [t_n]) = y$
    else
      $States \leftarrow States \cup \{x\}$
      $\delta(g, [t_1], \ldots, [t_n]) = x$
    end
  end
  $compute\_probabilities(S, \delta, p, r)$
end

Algorithm 2: Inference of a SMTA

all the subtrees of the training set. A total order is defined on subtrees comparing their depth. Each subtree is mapped to a state, taking into account the fact that if two subtrees are similar in the training set, then they have the same state. We denote $[t]$ the state mapped to the subtree t. To compute the similarity of two subtrees (*comp* function), the algorithm uses a statistical test [18] depending on a parameter $0 \leq \alpha \leq 1$. Intuitively α represents a tolerance parameter for the merging of two trees on a same state. The probabilities are then computed counting the subtree occurrences on the training set. The algorithm has the following properties: it is polynomial in the number of different subtrees of the training set, it infers a consistent automaton [7], and it converges to the limit under the Gold paradigm [17].

As we consider many-sorted trees, the algorithm only computes the similarity of subtrees built from symbols of same sort. Thus using sorts speeds up the convergence of the algorithm and improves its predictive power as experimentally shown in section 6.

4 Generalization of SMTA

A SMTA generalizes examples of the learning sample. Nevertheless, it is possible to generalize even more. Actually, looking at the transition rules of a learned SMTA, we may observe local regularities. For example if we consider $R_{f,q}$, the set of all rules of the form $f(q_1, \ldots, q_n) \to q$:

$$R_{f,q} = \{f(q1, q2, q3) \to q,\ f(q1, q4, q3) \to q,\ f(q1, q5, q3) \to q\}$$

Here we can see that whatever the state at position $(f, 2)$ is, we reach the state q given that $q1$ is in position $(f, 1)$ and $q3$ is in position $(f, 3)$. Therefore we would like to generalize these three rules in the rule $f(q1, X, q3) \to q$. This leads us to introduce generalized stochastic many-sorted tree automata which locally generalize SMTA. Note the α parameter of the inference algorithm is set globally and thus cannot have the local effect we described above.

4.1 Generalized SMTA

Definition 3 *A generalized SMTA is 6-tuple* $(\Sigma, Q, \delta, p, r, V)$. $\Sigma = (S, X, \sigma, \alpha)$ *is a signature.* Q *is a finite set of states, each state having a sort in* S. $r : Q \to [0, 1]$ *is the probability for a state to be an accepting state.* $\delta : X \times \{Q \cup V\}^* \to Q$ *is the transition function.* $p : X \times \{Q \cup V\}^* \to [0, 1]$ *is the probability of a transition.* V *is a set of variables such that* $V \cap X = \emptyset$.

The parsing of a tree with a generalized SMTA is similar to the one performed by a SMTA. It processes a tree using a bottom-up strategy, trying to label each node with a state. A variable stands for any state or any unrecognized state. In our framework, the parsing is deterministic. If two rules or more can be used, we choose the one of highest probability.

Given a generalized tree automaton A, the probability associated with a tree $t = f(t_1, \ldots, t_n)$ is defined by:

$$p(t \mid A) = r(\delta(t)) \times \pi'(t_1) \times \ldots \times \pi'(t_n)$$

where π' is recursively computed by:

$$\pi'(t) = p(f, \delta(t_1), \ldots, \delta(t_n)) \times \nu_1(\pi'(\delta(t_1))) \times \ldots \times \nu_n(\pi'(\delta(t_n)))$$

where

$$\nu_i(b) = \begin{cases} 1 \text{ if } \delta(t_i) \text{ instantiates a variable} \\ b \text{ otherwise} \end{cases}$$

Example Let A be the generalized automaton defined by:

$$\begin{array}{lll} f(q1, X, q2) \rightarrow q3 : 0.4 & & a \rightarrow q1 \quad : 0.2 \\ b \rightarrow q2 \quad : 0.1 & & r(q3) = 1.0 \end{array}$$

The probability associated with the tree $f(a, f(a, b, c), b)$ is:

$$p(f(a, f(a, b, c), b)|A) = 1.0 \times 0.4 \times 0.2 \times 1.0 \times 0.1 = 0.008$$

4.2 Generalization algorithm

We propose a method to generalize a stochastic tree automata A according to a parameter γ ($0 \leq \gamma \leq 1$). The idea consists in locally generalizing the tree automaton by replacing a set of rules $R_{f,q}$ with more general rules.

Definition 4 *Let r_1 and r_2 be two rules: $r_1 = f(x_1, \ldots, x_n) \rightarrow q$ and $r_2 = f(x'_1, \ldots, x'_n) \rightarrow q'$ such that $x_i \in Q \cup V$ and $x'_i \in Q \cup V$. r_1 is more general than r_2 (we note $r_1 > r_2$) if and only if there exists a substitution θ such that $f(x_1, \ldots, x_n)\,\theta = f(x'_1, \ldots, x'_n)$. We say that r_1 subsumes r_2.*

Definition 5 *A rule r is called a generalized rule if there exists a rule r' in $R_{f,q}$ such that $r > r'$.*

This generalization relation defines an upper semi-lattice for each set $R_{f,q}$. The upper bound is the rule containing only variables and the lower bounds are the rules of $R_{f,q}$.

The lattice is traversed searching for the maximally specific general rules having a score greater than γ. The score of a generalized rule is equal to the sum of the probabilities of all less general rules.

Example Consider the set of rules $R_{f,q}$ of a stochastic tree automaton:

$$\begin{array}{ll} f(q1, q2, q3) \rightarrow q : 0.3 & f(q1, q4, q3) \rightarrow q : 0.3 \\ f(q1, q2, q4) \rightarrow q : 0.2 & f(q5, q2, q3) \rightarrow q : 0.2 \end{array}$$

The generalized rules having a score greater than 0.6 are:

$$\begin{array}{lll} f(\_, \_, \_) \quad : 1.0 & f(q1, \_, \_) : 0.8 & f(q1, \_, q3) : 0.6 \\ f(\_, \_, q3) : 0.8 & f(\_, q2, \_) : 0.7 & \end{array}$$

As we only keep the maximally specific generalized rules, we only keep the rules $f(q1, \_, q3)$ and $f(\_, q2, \_)$ for state q and symbol f. Note that we use the symbol '$\_$' to denote any anonymous variable. The lattice associated with $R_{f,q}$ is represented on figure 5.

Algorithm 3 generalizes a SMTA. It first constructs a set of generalized rules. In this set, some rules may subsume some others. In this case the more general rule is deleted. When two generalized rules, reaching a different state, subsume the same non generalized rule, one of the two general rules is deleted. Finally all

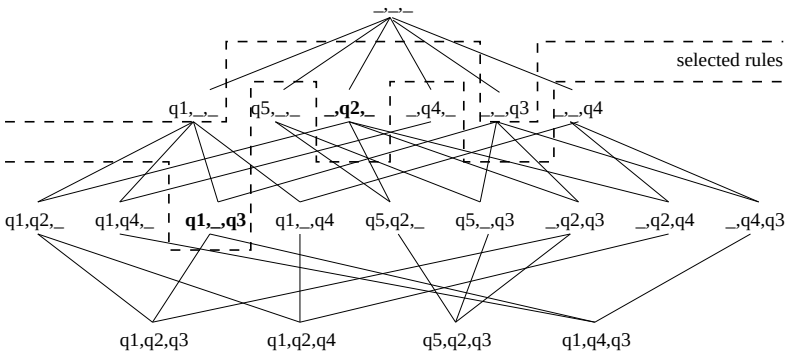

Fig. 5. Lattice of generalization

the rules of the SMTA that are not subsumed by generalized rules are added to the set of transitions of the generalized SMTA.

Then the probabilities of generalized rules are computed from their scores using the function *update_probabilities*. If a rule r of the initial automaton is subsumed by n generalized rules $r_1, \ldots, r_n$, its probability is shared among the generalized rules. Therefore the score $p(r_i)$, $1 \leq i \leq n$, must be updated adding $\frac{p(r)}{n} - p(r)$.

Data: $A = (\Sigma, Q, \delta, p, r_s)$: a SMTA
γ: a generalization parameter
Result: $A_g = (\Sigma, Q, \delta_g, p_g, r_s, V)$: a generalized SMTA
begin
  $F \leftarrow \emptyset$
  foreach $R_{f,q}$ *such that* $q \in Q, f \in X, \alpha(f) > 1$ **do**
    $E \leftarrow \{\ f(q_1, \ldots, q_n) \mid f(q_1, \ldots, q_n) \rightarrow q\ \}$
    $F_{f,q} \leftarrow find\_rules(E, \gamma)$
    $F \leftarrow F \cup F_{f,q}$
  end
  $F \leftarrow F \backslash \{r \mid r \in F_{f,i}, \exists r' \in F_{f,j}, i \neq j, r' > r\}$
  $F \leftarrow F \backslash \{r \mid r \in F_{f,i}, \exists r' \in F_{f,j}, i \neq j, \exists d \in \delta \text{ such that } r > d, r' > d\}$
  $F \leftarrow F \cup \{d \in \delta \mid \nexists r \in F, r > d\}$
  $p_g \leftarrow update\_probabilities(F)$
  $\delta_g \leftarrow F$
  Return A_g
end

Algorithm 3: SMTA generalization

The function $find\_rules$ searches for all most specific rules of $R_{f,q}$ with a score greater than γ.

5 Experimental evaluation

We have evaluated our approach using a relational database in the medical domain. The database was collected at Chiba University Hospital and stores data about collagen diseases and thrombosis. The original version is available on the UCI repository [4]. In this paper we use a more recent version of this database proposed for the PKDD'01 discovery challenge[1].

5.1 Perplexity

The evaluation of non-probabilistic models is often based on the classification rate. Probabilistic models are rather assessed on their ability to correctly predict the probability of a test sample. In this paper we evaluate the learned automata by measuring the perplexity criterion.

In the case of tree automata, the quality of a model A can be evaluated by the average likelihood on a set of trees S relatively to the distribution defined by A:

$$LL = (\frac{1}{\|S\|} \sum_{j=1}^{|S|} \log P(t_j))$$

This is linked to the Kullback-Leibler divergence which compares the distribution of an unknown target to a distribution evaluated on a training set [5]. A perfect model can predict each element of the sample with a probability equal to one, and so $LL = 0$. In a general way we consider the perplexity of the test set which is defined by $PP = 2^{LL}$. A minimal perplexity ($PP = 1$) is reached when the model can predict each element of the test sample. Therefore we consider that a model is more predictive than another if its perplexity is lower.

A problem occurs when a tree of the test sample cannot be recognized by the automaton. Actually the probability of this example is 0 and the perplexity cannot be computed. To avoid this problem we smooth the distribution of the learned model. We define a universal (generalized) SMTA A_0 with only one state s. The probabilities of the rules are computed from the training set. If a transition involves an unknown symbol, its probability is fixed to a small ratio of the size of the training set. In the smoothed model, a tree t has the probability:

$$\hat{P}(t) = \beta.p(t|A) + (1 - \beta).p(t|A_0)$$

where $0 \leq \beta \leq 1$. The results presented below are computed with $\beta = 0.9$.

[1] http://lisp.vse.cz/challenge/pkdd2001/

5.2 Experimental results

The database is made up of seven relations. The table *patient_info* stores information about patients. The three tables *antibody_exam*, *ana_pattern* and *lab_exam* report laboratory examinations. The table *thrombosis* stores data about all thrombosis attacks. The table *diagnosis* lists all diseases a patient is suffering from. Finally, the table *disease* includes all the diseases diagnosed at the hospital.

The idea is to learn a distribution on the patient examinations, taking into account information stored in other tables. Thus we focus our experiments on the *lab_exam* table. The table has 57542 records that we split into a set of 30000 trees and a test set of 17542 trees. 30000 trees are used during the training phases and the remaining ones form the test set. For each perplexity measure we consider five random learning samples of size 1000 to 30000. For each learned model we assess the perplexity according to the size of the learning sample.

In a first series of experiments, we aim at measuring the benefit of sorts in tree automata. For each training sample we represent examples either with trees or many-sorted trees. This leads to learn either stochastic tree automata or stochastic many-sorted tree automata. Figure 6 shows that perplexity is significantly reduced when we used many-sorted trees.

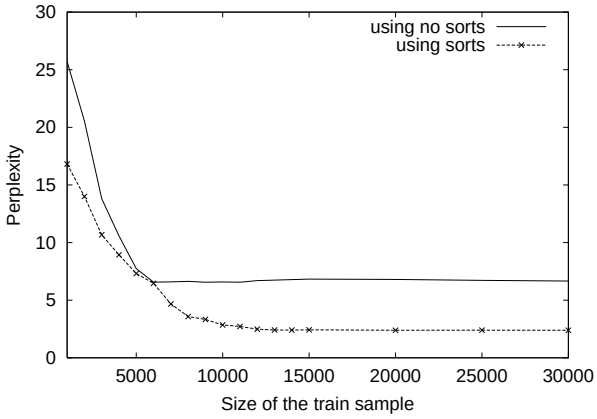

Fig. 6. Contribution of sorts

In a second series of experiments, we measure the predictive ability of the generalized SMTA comparing the one of SMTA. Let us recall the generalization depends on a parameter γ. Figure 7 and 8 show the evolution of the perplexity with various values of γ. In order to improve the readability we present these results on four charts. We may observe that for some values of γ, the generalized model performs better than ungeneralized one. First it improves the predictive

power of the automaton and secondly it requires less examples to converge. These two points seem to show the interest of local generalization.

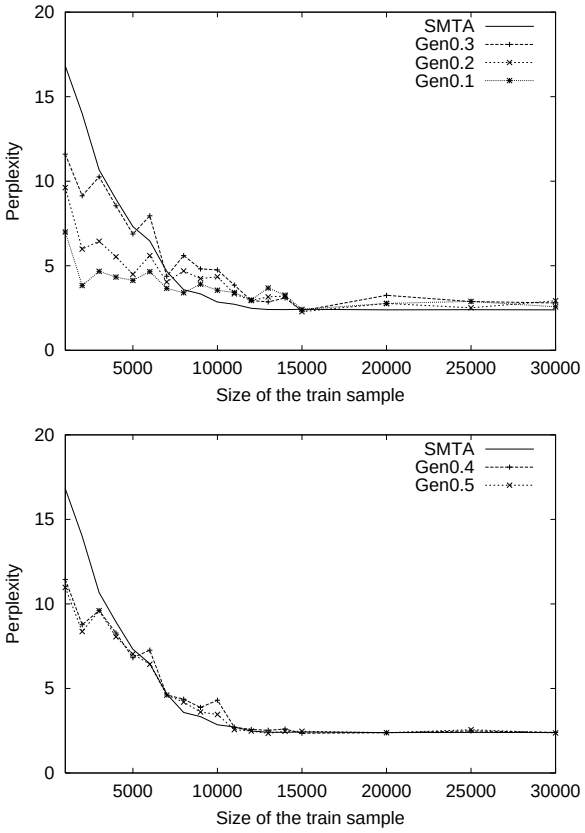

Fig. 7. Evaluation of the generalized SMTA

6 Conclusion

In this paper we have presented a grammatical inference approach to learn probabilistic models on relational databases. This approach is based on the inference of stochastic many-sorted tree automata. Experiments have shown our approach is able to address large amount of data (up to 30000 trees which corresponds to 230000 records of the database). We have shown that incorporating sorts really improves the efficiency of the learned automaton and speeds up the convergence of the inference algorithm. Moreover we have proposed generalized stochastic

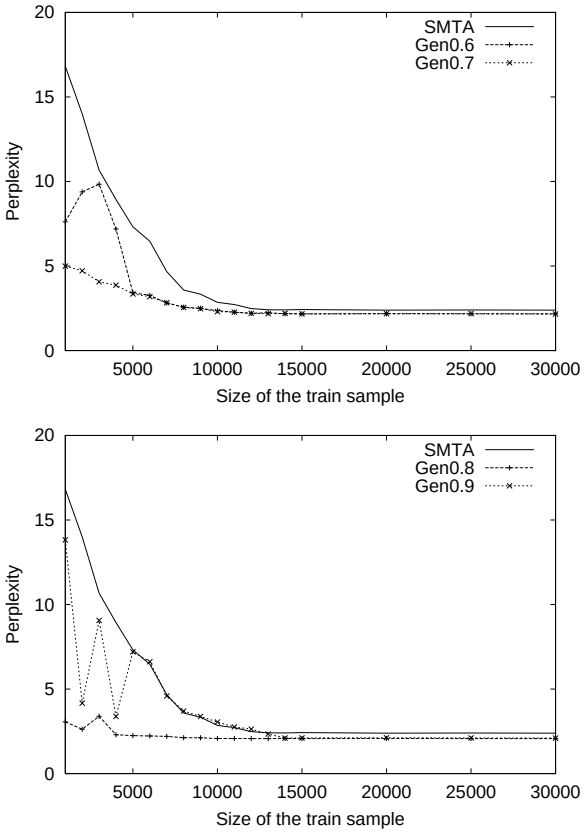

Fig. 8. Evaluation of the generalized SMTA

many-sorted tree automata which locally generalize stochastic many-sorted tree automata. The experiments have shown that this concept improves both the predictive power of the learned model and the convergence of the learning algorithm. We now aim at refining the local generalization studying various strategies to select generalized rules and to compute their probabilities.

References

1. N. Abe and H. Mamitsuka. Predicting protein secondary structure using stochastic tree grammars. *Machine Learning*, 29:275–301, 1997.
2. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In J. B. Bocca, M.s Jarke, and C. Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.
3. M. Bernard and C. de la Higuera. Apprentissage de programmes logiques par Inférence Grammaticale. *Revue d'Intelligence Artificielle*, 14(3–4):375–396, 2000. Hermes Sciences.

4. C.L. Blake and C.J. Merz. University of California Irvine repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/`, 1998.
5. J. Calera-Rubio and R. C. Carrasco. Computing the relative entropy between regular tree languages. *Information Processing Letters*, 68(6):283–289, 1998.
6. R. C. Carrasco, J. Oncina, and J. Calera. Stochastic Inference of Regular Tree Languages. *Machine Learning*, 44(1/2):185–197, 2001.
7. R. Chaudhuri and A. N. V. Rao. Approximating grammar probabilities: Solution of a conjecture. *Journal of the Association for Computing Machinery*, 33(4):702–705, 1986.
8. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications . Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997.
9. G. F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.
10. V. Crestana-Jensen and N. Soparkar. Frequent itemset counting across multiple tables. In *4th Pacific-Asian conference on Knowledge Discovery and Data Mining (PAKDD 2000)*, pages 49–61, April 2000.
11. L. De Raedt. Data mining in multi-relational databases. In *4th European Conference on Principles and Practice of Knowledge*, 2000. Invited talk.
12. L. Dehaspe and H. Toivonen. Discovery of frequent DATALOG patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.
13. N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1300–1307. Morgan Kaufmann, 1999.
14. P. Garcia and J. Oncina. Inference of recognizable tree sets. Research Report DSIC - II/47/93, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 1993.
15. F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.
16. L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of relational structure. In *18th International Conference on Machine Learning*, pages 170 – 177, Williamston, MA, June 2001. Morgan Kaufmann.
17. E. M. Gold. Language identification in the limit. *Information and Control*, 10(n5):447–474, 1967.
18. W. Hoeffding. Probabilities inequalities for sums or bounded random variables. *Journal of the American Association*, 58(301):13–30, 1963.
19. T. Knuutila and M. Steinby. Inference of tree languages from a finite sample: an algebraic approach. *Theoretical Computer Science*, 129:337–367, 1994.
20. H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In U. M. Fayyad and R. Uthurusamy, editors, *AAAI Workshop on Knowledge Discovery in Databases (KDD-94)*, pages 181–192, Seattle, Washington, 1994. AAAI Press.
21. S. Muggleton and L. De Raedt. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming*, 19–20:629–679, 1994.
22. J. R. Rico-Juan, J. Calera, and R. C. Carrasco. Probabilistic k-Testable Tree-Languages. In A. L. Oliveira, editor, *5th International Colloquium on Grammatical Inference (ICGI 2000), Lisbon (Portugal)*, volume 1891 of *Lecture Notes in Computer Science*, pages 221–228, Berlin, September 2000. Springer.
23. A. Stolcke and S. Omohundro. Inducing probabilistic grammars by bayesian model merging. In *2nd International Colloquium on Grammatical Inference (ICGI'94)*, volume 862 of *Lecture Notes in Artificial Intelligence*, pages 106–118, Alicante, Spain, 1994. Springer Verlag.