
Parameter Learning in Probabilistic Databases: A Least Squares Approach

Bernd Gutmann
Angelika Kimmig
Luc De Raedt

Dept. of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, POBox 2402, BE-3001 Heverlee, Belgium

BERND.GUTMANN@CS.KULEUVEN.BE
ANGELIKA.KIMMIG@CS.KULEUVEN.BE
LUC.DERAEDT@CS.KULEUVEN.BE

Kristian Kersting

Fraunhofer IAIS, Schloß Birlinghoven, 53754 Sankt Augustin, Germany

KRISTIAN.KERSTING@IAIS.FRAUNHOFER.DE

Keywords: Learning, Graphs, Probabilistic Databases, Logic

Abstract

Probabilistic databases compute the success probabilities of queries. We introduce the problem of learning the parameters of the probabilistic database ProbLog. Given the observed success probabilities of a set of queries, we use a least squares approach to compute the probabilities attached to facts that have a low approximation error on the training data as well as on unseen examples.¹

1. Introduction

The statistical relational learning community has devoted a lot of attention to learning both the structure and parameters of probabilistic logics, cf. (Getoor & Taskar, 2007; De Raedt et al., 2008), but so far seems to have devoted little attention to the learning of probabilistic database formalisms. Probabilistic databases (Dalvi & Suciu, 2004; De Raedt et al., 2007) associate probabilities to facts, indicating the probabilities with which the facts hold. This information is then used to define and compute the success probability of queries or derived facts or tuples. Because probabilistic databases do not constitute a generative model, it has – so far – been unclear as how to learn such databases. In this paper, we introduce the problem of learning the parameters of probabilistic databases from a set of queries together with their target probabilities. The approach is incorporated in

¹This is a shortened version of an extended abstract submitted to the 6th International Workshop on Mining and Learning with Graphs (MLG2008)

the probabilistic database ProbLog (De Raedt et al., 2007), though it can easily be integrated in other probabilistic databases as well. ProbLog has been designed to support life scientists that mine a large network of biological entities in interactive querying sessions..

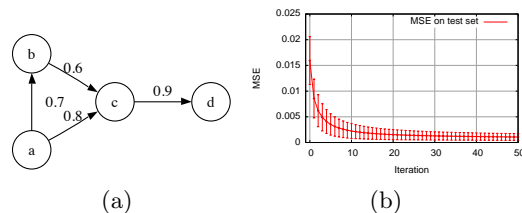


Figure 1. (a) Probabilistic graph. (b) Learning curve with standard deviation, on test set for graph with 88 edges.

2. ProbLog

ProbLog is a simple probabilistic extension of Prolog introduced in (De Raedt et al., 2007). A ProbLog program consists – as Prolog – of a set of definite clauses. However, in ProbLog every fact c_i is labeled with the probability p_i that it is true, and those probabilities are assumed to be mutually independent. For ease of illustration, we will consider probabilistic graphs like the one in Figure 1(a) in the following, but the entire discussion carries over to arbitrary ProbLog programs. Such a probabilistic graph can be used to sample subgraphs by tossing a biased coin for each edge. The corresponding ProbLog program $T = \{p_1 : c_1, \dots, p_n : c_n\}$ therefore defines a probability distribution over subgraphs $L \subseteq L_T = \{c_1, \dots, c_n\}$ in the following way:

$$P(L|T) = \prod_{c_i \in L} p_i \prod_{c_i \in L_T \setminus L} (1 - p_i).$$

It is straightforward to add background knowledge in the form of Prolog clauses, say, the definition of a path by combining edges. We can then ask for the probability that there exists e.g. a path between nodes a and c in our probabilistic graph, i.e. the probability that a randomly sampled subgraph contains the edge from a to c , or the path from a to c via b (or both of them). Formally, the *success probability* $P_s(q|T)$ of a query q in a ProLog program T is defined as

$$P_s(q|T) = \sum_{L \subseteq L_T} P(q|L) \cdot P(L|T), \quad (1)$$

where $P(q|L) = 1$ if there exists a θ such that $L \models q\theta$, and $P(q|L) = 0$ otherwise. The success probability of query q corresponds to the probability that the query q is *provable* in a randomly sampled logic program. Due to presence of multiple paths in samples, evaluating the success probability of ProLog queries is computationally hard, see (De Raedt et al., 2007) for an approximation algorithm.

3. Parameter Learning

ProLog does not provide a generative model for sampling queries (e.g. paths between nodes). Thus, we cannot directly apply standard maximum likelihood techniques for parameter estimation based on the EM algorithm. We consider parameter learning for ProLog as a function optimization problem:

Definition 1 (ProLog Parameter Learning)

Given a set of training examples $\{q_i, \tilde{p}_i\}_{i=1}^K$, $K > 0$, where each $q_i \in \mathcal{H}$ is a logical query with success probability \tilde{p}_i , **find** a function $h : \mathcal{H} \rightarrow [0, 1]$ with low approximation error on the training data as well as on unseen examples. \mathcal{H} comprises all parameter assignments for a given logical program T .

We want to minimize the mean squared error (MSE):

$$MSE(T) = \frac{1}{K} \sum_{1 \leq i \leq K} (P_s(q_i|T) - \tilde{p}_i)^2. \quad (2)$$

It is easy to show that minimizing the squared error in this case corresponds to finding a maximum likelihood hypothesis, provided that for each training example (q_i, \tilde{p}_i) , a Gaussian error is included, i.e. $\tilde{p}_i = p(q_i) + e_i$, with $p(q_i)$ the actual probability of query q_i and e_i drawn independently from a Gaussian with mean zero. We now derive the gradient of the MSE. Applying the sum and chain rule to Eq. (2) yields the partial derivative $\partial MSE(T)/\partial p_j =$

$$\frac{2}{K} \sum_{1 \leq i \leq K} \underbrace{(P_s(q_i|T) - \tilde{p}_i)}_{\text{Part 1}} \cdot \underbrace{\frac{\partial P_s(q_i|T)}{\partial p_j}}_{\text{Part 2}}. \quad (3)$$

We apply standard gradient descent to minimize the MSE. (3) can be evaluated by ProLog inference directly (Part 1) and by slightly adapting the underlying techniques (Part 2).

4. Experiments

We implemented the gradient descent algorithm in Prolog (Yap-5.1.3). Since this is ongoing work, we primarily try to answer the question: *does the gradient descent minimize the MSE?*

As our test graph G , we used a real biological graph around 3 random Alzheimer genes, with 45 nodes and 88 edges, cf. (De Raedt et al., 2007). We randomly sampled 100 node pairs and calculated the probability that there exists a path between them using approximate ProLog inference. We performed 5-fold cross-validation, initializing the parameters randomly, with fixed seed for succeeding experiments. Figure 1(b) shows the learning curve for the test set. After 50 iterations, the MSE averaged over 5 folds is 0.00016 ± 0.00001 on the training set and 0.00107 ± 0.00065 on the test set, answering our question positively.

5. Conclusions

We introduced an approach to parameter learning for the probabilistic database ProLog and successfully showed it at work on a real biological application. Interesting directions for future research include optimizing the learning algorithm and regularization-based cost functions. Those enable domain experts to refine probabilities of a database by stating examples.

Acknowledgments AK, BG are supported by the Research Foundation-Flanders (FWO-Vlaanderen), KK by a Fraunhofer ATTRACT fellowship.

References

- Dalvi, N. N., & Suciu, D. (2004). Efficient query evaluation on probabilistic databases. *VLDB* (pp. 864–875).
- De Raedt, L., Frasconi, P., Kersting, K., & Muggleton, S. (Eds.). (2008). *Probabilistic inductive logic programming - theory and applications*, vol. 4911 of *LNAI*. Springer-Verlag.
- De Raedt, L., Kimmig, A., & Toivonen, H. (2007). ProLog: A probabilistic Prolog and its application in link discovery. *IJCAI* (pp. 2462–2467).
- Getoor, L., & Taskar, B. (Eds.). (2007). *Statistical relational learning*. The MIT press.