

Estimating the Parameters of Probabilistic Databases from Probabilistically Weighted Queries and Proofs [Extended Abstract]

Bernd Gutmann¹, Angelika Kimmig¹, Kristian Kersting², and Luc De Raedt¹

¹ Dept. of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, POBox 2402, BE-3001 Heverlee, Belgium {`firstname.lastname`}@`cs.kuleuven.be`

² Fraunhofer IAIS, Schloß Birlinghoven, 53754 Sankt Augustin, Germany
`kristian.kersting@iais.fraunhofer.de`

Abstract. We introduce the problem of learning the parameters of the probabilistic database ProbLog. Given the observed success probabilities of a set of queries, we compute the probabilities attached to facts that have a low approximation error on the training examples as well as on unseen examples. Assuming Gaussian error terms on the observed success probabilities, this naturally leads to a least squares optimization problem. Our approach, called LeProbLog, is able to learn both from queries and from proofs and even from both simultaneously. This makes it flexible and allows faster training in domains where the proofs are available. Experiments on real world data show the usefulness and effectiveness of this least squares calibration of probabilistic databases.³

1 Introduction

Many real-world application today depend on managing enormous volumes of uncertain data. Such "dirty" databases arise for example when integrating data from various sources, when analyzing social, biological, and chemical networks, within privacy-preserving data mining where only aggregated data is available, and within pervasive computing. These are only some of the many real-world applications showing the abundance of uncertain data and the need for probabilistic databases, i.e., generalizations of traditional relational databases that can deal with uncertainty.

Over the last years, the statistical relational learning community has devoted a lot of attention to learning both the structure and parameters of probabilistic logics, cf. [1, 2], but so far seems to have devoted little attention to the learning of probabilistic database formalisms. Probabilistic databases [3, 4] associate probabilities to facts, indicating the probabilities with which these facts hold. This information is then used to define and compute the success probability of queries

³ A longer version of this paper has been accepted at the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2008) under the title "Parameter Learning in Probabilistic Databases: A Least Squares Approach"

or derived facts or tuples, which are defined using background knowledge (in the form of predicate definitions). As one example, imagine a life scientist mining a large network of biological entities in an interactive querying session. The biological network is a probabilistic graph, in which the edges are represented by probabilistic facts about the biological entities [4]. Interesting questions can then be asked about the probability of the existence of *a* connection between two nodes, or the most reliable path between them.

The key contribution of the present paper is the introduction of a novel setting for learning the parameters of a probabilistic database from examples together with their target probability. The task then is to find those parameters that minimize the least squared error w.r.t. these examples. The examples themselves can either be queries or proofs, where a proof is a conjunction of all facts in the database needed to prove a query by SLD-resolution. This learning setting is then incorporated in the probabilistic database ProbLog [4], though it can easily be integrated in other probabilistic databases as well. This yields the second key contribution of the paper, namely an effective learning algorithm called *LeProbLog* (Least square estimation for ProbLog).

Within the probabilistic database community, parameter estimation has received surprisingly little attention. Nottelmann and Fuhr [5] consider learning probabilistic Datalog rules in a similar setting where the underlying distribution semantics is similar to ProbLog. However, their setting and approach also significantly differ from ours. A single probabilistic target predicate only is estimated whereas we consider estimating the probabilities attached to definitions of multiple predicates. Their approach employs the training probabilities only. Specifically, they generate training examples labeled with 0/1 randomly whereas we use the observed probabilities directly. Finally, the probabilistic database setting differs from the usual statistical relational learning approach in that there is no underlying generative model as for instance at PRISM programs [6].

We proceed as follows. After reviewing ProbLog in Section 2, we will define the parameter estimation problem for probabilistic databases and introduce our least-squares approach LeProbLog for solving it. Before concluding, we will present the results of an extensive set of experiments on a real-world data set.

2 ProbLog

In this work, the probabilistic database employed is ProbLog, a simple probabilistic extension of Prolog introduced in [4]; alternative formalisms that could be used include those of [3] or [5]. We repeat the main ideas of ProbLog here, see [4] for details. A ProbLog program consists of a set of labeled facts $p_i :: c_i$ together with a set of definite clauses encoding *background knowledge* (BK). Each ground instance (that is, each instance not containing variables) of such a fact c_i is true with probability p_i , where all probabilities are assumed mutually independent. The corresponding ProbLog program $T = \{p_1 :: c_1, \dots, p_n :: c_n\} \cup BK$ defines a probability distribution over sets of facts $L \subseteq L_T = \{c_1, \dots, c_n\}$ as follows:

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

Based on this distribution, the *success probability* $P_s(q|T)$ of a query q in a ProbLog 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. In other words, the success probability of query q corresponds to the probability that the query q is *provable* in a randomly sampled logic program. Alternatively, it can be calculated based on the set of all proofs of query q , see [4] for details. The probability of a *specific* proof, also called explanation, corresponds to that of sampling a logic program L that contains all the facts needed in that proof. The *explanation probability* $P_x(q|T)$ is then defined as the probability of the most likely explanation or proof of the query q :

$$P_x(q|T) = \max_{e \in E(q)} P(e|T) = \max_{e \in E(q)} \prod_{c_i \in e} p_i \quad (2)$$

where $E(q)$ is the set of all explanations for query q [7]. Finally, the k -probability $P_k(q|T)$ approximates the success probability by restricting the set of proofs used in the calculation to the k most likely proofs. Note that $P_1(q|T) = P_x(q|T)$.

3 Parameter Learning in Probabilistic Databases

Data added to a probabilistic database is often uncertain, e.g. information extraction algorithms yield results with probabilities. Furthermore, there is uncertainty about how to explain the data, e.g. there is a connection between genes and diseases but the exact dependency is unknown. This results in a learning task which can be formulated as following:

Definition 1 (Parameter Learning in Probabilistic Databases). *Given a set of training examples $\{q_i, \tilde{p}_i\}_{i=1}^M$, $M > 0$, where each $q_i \in \mathcal{H}$ is a query or proof and \tilde{p}_i is the k -probability of q_i , **find** a function $h : \mathcal{H} \rightarrow [0, 1]$ with low approximation error on the training examples as well as on unseen examples. \mathcal{H} comprises all parameter assignments for a given database T .*

This framework allows to naturally combine *learning from entailment* and *learning from proofs*, two learning settings that so far have been considered separately. In ProbLog, proofs correspond to conjunctions of probabilistic facts, and can be seen as a conjunction of queries. Therefore, a learning algorithm can use examples of both forms, (atomic) queries and proofs, at the same time. The *error function* that we want to minimize is the mean squared error:

$$MSE(T) = \frac{1}{M} \sum_{1 \leq i \leq M} (P_s(q_i|T) - \tilde{p}_i)^2. \quad (3)$$

To do so, we use a standard gradient descent approach. To obtain the gradient of the MSE, one has to apply the sum and chain rule to Eq. (3)

$$\frac{\partial MSE(T)}{\partial p_j} = \frac{2}{M} \sum_{1 \leq i \leq M} \underbrace{(P_s(q_i|T) - \tilde{p}_i)}_{\text{Part 1}} \cdot \underbrace{\frac{\partial P_s(q_i|T)}{\partial p_j}}_{\text{Part 2}}. \quad (4)$$

Algorithm 1 Evaluating the gradient of a query efficiently by traversing the corresponding BDD, calculating partial sums, and adding only relevant ones.

```

function GRADIENT(BDD  $b$ , fact to derive for  $n_j$ )
   $(val, seen) = \text{GRADIENTEVAL}(\text{root}(b), n_j)$ 
  If  $seen = 1$  return  $val \cdot \sigma(a_j) \cdot (1 - \sigma(a_j))$ 
  Else return 0
function GRADIENTEVAL(node  $n$ , target node  $n_j$ )
  If  $n$  is the 1-terminal return  $(1, 0)$ 
  If  $n$  is the 0-terminal return  $(0, 0)$ 
  Let  $h$  and  $l$  be the high and low children of  $n$ 
   $(val(h), seen(h)) = \text{GRADIENTEVAL}(h, n_j)$ 
   $(val(l), seen(l)) = \text{GRADIENTEVAL}(l, n_j)$ 
  If  $n = n_j$  return  $(val(h) - val(l), 1)$ 
  ElseIf  $seen(h) = seen(l)$  return  $(\sigma(a_n) \cdot val(h) + (1 - \sigma(a_n)) \cdot val(l), seen(h))$ 
  ElseIf  $seen(h) = 1$  return  $(\sigma(a_n) \cdot val(h), 1)$ 
  ElseIf  $seen(l) = 1$  return  $((1 - \sigma(a_n)) \cdot val(l), 1)$ 

```

To ensure that all p_j stay between 0 and 1 during gradient descent, we reparameterize the search space and express each $p_j \in]0, 1[$ in terms of the sigmoid function $p_j = \sigma(a_j) := 1/(1 + \exp(-a_j))$ applied to $a_j \in \mathbb{R}$. Part 1 can be calculated by a ProbLog inference call computing (1). Part 2 can be calculated as following

$$\frac{\partial P_s(q_i|T)}{\partial a_j} = \sigma(a_j) \cdot (1 - \sigma(a_j)) \cdot \sum_{\substack{S \subseteq L_T \\ L \models q_i}} \delta_{jS} \prod_{\substack{c_x \in S \\ x \neq j}} \sigma(a_x) \prod_{\substack{c_x \in L_T \setminus S \\ x \neq j}} (1 - \sigma(a_x)).$$

where $\delta_{jS} := 1$ if $c_j \in S$ and $\delta_{jS} := -1$ if $c_j \in L_T \setminus S$. It is derived by first deriving the gradient $\partial P(S|T)/\partial p_j$ for a fixed subset $S \subseteq L_T$ of facts, which is straight-forward, and then summing over all subsets S where q_i can be proven. Going over all subprograms S in the last equation is infeasible. But there is an efficient algorithm to compute $P_s(q_i|T)$ relying on BDDs [4]. We updated this towards the gradient as shown in algorithm 1. LeProbLog combines the BDD-based gradient calculation with a standard gradient descent search. Starting from parameters $\mathbf{a} = a_1, \dots, a_n$ initialized randomly, the gradient $\Delta \mathbf{a} = \Delta a_1, \dots, \Delta a_n$ is calculated, parameters are updated by subtracting the gradient, and updating is repeated until convergence. When using the k -probability with finite k , the set of k best proofs may change due to parameter updates. After each update, we therefore recompute the set of proofs and the corresponding BDD.

4 Experiments

We set up experiments to investigate the following questions:

- Q1** Does our approach reduce the mean squared error on training and test data?
- Q2** Is LeProbLog able to recover the original parameters?

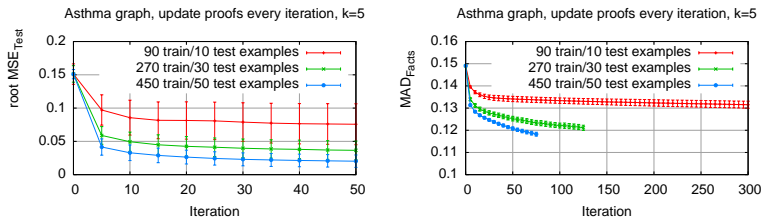


Fig. 1. \sqrt{MSE}_{Test} (left) and MAD_{facts} (right) for asthma using the 5 best proofs ($k = 5$) (**Q1** and **Q2**)

Q3 Does the algorithm perform better when parts of the training data are given as proof? (For instance the parse tree for a sentences when learning probabilistic grammars, or the most likely path in a probabilistic graph)

We extracted a graph around asthma disease from a collection of databases. We obtained a set of related genes by searching Entrez for human genes annotated with asthma; the asthma phenotype is from OMIM. Weights were assigned to edges as described in [8]. We used 7 randomly chosen asthma genes for graph extraction. The resulting graph contains 127 nodes and 241 edges. From this graph we sampled 500 random node pairs (a,b) and estimated the query probability for $\text{path}(a,b)$ using P_5 , the probability of the 5 best proofs (for **Q1**, **Q2**). We also sampled 300 node pairs and calculated P_1 for $\text{path}(a,b)$, the probability of the best path between a and b , and used it to answer **Q3**. We use both the root mean squared error on the test data and the mean absolute difference MAD_{facts} between learned and original fact probabilities to assess the results.

Q1, Q2: Sanity Check We attach probabilities to queries in the training set based on the best $k = 5$ proofs. The same approximation is used in the gradient descent algorithm. The left graph of Figure 1 shows the evolvement of the root mean squared error. LeProbLog reduces the MSE on both training and test data, with significant differences in all cases (two-tailed t-test, $\alpha = 0.05$). These results affirmatively answer **Q1**. The MAD_{facts} error is reduced as can be seen in the right plot of Figure 1. Again, all differences are significant (two-tailed t-test, $\alpha = 0.05$). Using more training examples results in faster error reduction. This answers **Q2** affirmatively.

Q3: Learning from Proofs and Queries To investigate the effect of using both proofs and queries as examples, we compute the best proof and its probability for 300 examples. For each example, we either use the query or the best proof, both with the probability of the best proof. Learning uses $k = 1$. We use proofs for 0, 50, \dots , 300 examples and queries for the remaining ones. Figure 2 shows the results of this experiment. The curve on the left side indicates that the error per fact (MAD_{facts}) goes down faster in terms of iterations when increasing the fraction of proofs. The curve on the right side shows that the root MSE on the test set decreases. These results answer **Q3** affirmatively.

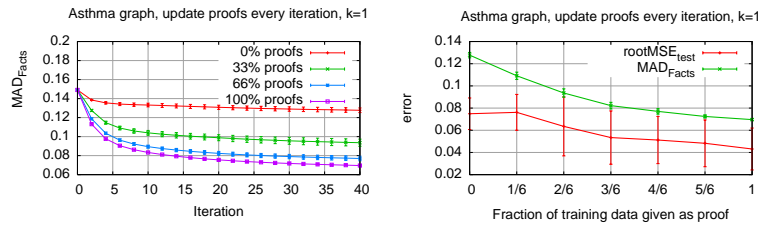


Fig. 2. MAD_{Facts} and $\sqrt{MSE_{Test}}$ after 40 iterations on the asthma graph when different fractions of the data are given as proof (**Q3**)

5 Conclusions

We have introduced an approach to learning the parameters of the probabilistic database ProbLog and successfully shown it at work on a real biological application. Interesting directions for future research include conjugate gradient techniques and regularization-based cost functions. Those enable domain experts to successively refine probabilities of a database by stating training examples.

Acknowledgments

Angelika Kimmig and Bernd Gutmann are supported by the Research Foundation-Flanders (FWO-Vlaanderen), Kristian Kersting by a Fraunhofer ATTRACT fellowship. This work is supported by the GOA project 2008/08 Probabilistic Logic Learning and uses HPC resources <http://ludit.kuleuven.be/hpc>.

References

1. Getoor, L., Taskar, B., eds.: Statistical Relational Learning. The MIT press (2007)
2. De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S., eds.: Probabilistic Inductive Logic Programming - Theory and Applications. Volume 4911 of LNAI. Springer (2008)
3. Dalvi, N.N., Suciu, D.: Efficient query evaluation on probabilistic databases. In: Proceedings of VLDB. (2004) 864–875
4. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discovery. In Veloso, M., ed.: IJCAI. (2007) 2462–2467
5. Nottelmann, H., Fuhr, N.: Learning probabilistic datalog rules for information classification and transformation. In: CIKM, ACM (2001) 387–394
6. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. J. Artif. Intell. Res. (JAIR) **15** (2001) 391–454
7. Kimmig, A., De Raedt, L., Toivonen, H.: Probabilistic explanation based learning. In: ECML. (2007) 176–187
8. Sevón, P., Eronen, L., Hintsanen, P., Kulovesi, K., Toivonen, H.: Link discovery in graphs derived from biological databases. In: DILS. (2006) 35–49