Parameter Learning in Probabilistic Databases: A Least Squares Approach

Bernd Gutmann BERND.GUTMANN@CS.KULEUVEN.BE
Angelika Kimmig ANGELIKA.KIMMIG@CS.KULEUVEN.BE
Luc De Raedt LUC.DERAEDT@CS.KULEUVEN.BE
Dept. of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, POBox 2402, BE-3001 Heverlee, Belgium

Kristian Kersting KRISTIAN.KERSTING@IAIS.FRAUNHOFER.DE
Fraunhofer IAIS, Schloß Birlinghoven, 53754 Sankt Augustin, Germany

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 compute the probabilities attached to facts that have a low approximation error on the training data as well as on unseen examples. Assuming Gaussian error terms on the observed success probabilities, this naturally leads to a least squares optimization problem. Experiments on real world data show the usefulness and effectiveness of this least squares calibration of probabilistic databases.

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 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 and we report on some experiments in this domain that validate the approach.

Appearing in the 6^{th} *International Workshop on Mining and Learning with Graphs*, Helsinki, Finland, 2008.

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 encoded in ProbLog in the following, but the entire discussion carries over to arbitrary ProbLog programs. Figure 1(a) shows a small example that can be encoded in ProbLog as follows:

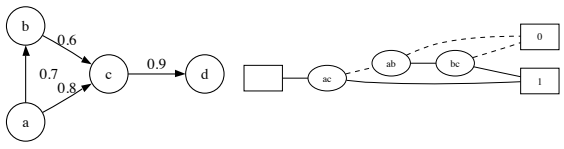
$$0.8 : \texttt{edge(a,c)}. \quad 0.7 : \texttt{edge(a,b)}.$$
$$0.6 : \texttt{edge(b,c)}. \quad 0.9 : \texttt{edge(c,d)}.$$

Such a probabilistic graph can be used to sample subgraphs by tossing a biased coin for each edge. The corresponding ProbLog program $T = \{p_1 : c_1, \cdots, p_n : c_n\}$ therefore defines a probability distribution over

subgraphs $L \subseteq L_T = \{c_1, \cdots, c_n\}$ 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).



(a) (b)

Figure 1. Probabilistic graph and BDD for query path(a,c).

Formally, the *success probability* $P_s(q|T)$ of a query q in a ProbLog program T is defined as

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

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.

Due to simultaneous presence of multiple proofs (paths) in sampled programs (graphs), evaluating the success probability of ProbLog queries is computationally hard. In (De Raedt et al., 2007), an approximation algorithm is proposed to tackle this problem by employing a reduction to the computation of the probability of a monotone DNF formula, an NP-complete problem. The approximative inference engine performs an iterative deepening search for proofs and also relies on Binary Decision Diagrams (BDDs) (Bryant, 1986), cf. Figure 1(b).

3. Parameter Learning

The semantics of ProbLog does not provide a generative model for sampling queries (e.g. paths between given nodes). We therefore cannot directly apply standard maximum likelihood techniques for parameter estimation based on the EM algorithm as is usually done for statistical relational learning models (Getoor & Taskar, 2007). We therefore consider parameter learning for ProbLog as a function optimization problem, where we seek a set of parameters for our program that approximates actual query probabilities well, yielding:

Definition 1 (ProbLog 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* $h : \mathcal{H} \to [0, 1]$ *with low approximation error on the training data as well as on unseen examples.* $\mathcal{H}$ *comprises all parameter assigments for a given logical program* T.

The example space considered here thus comprises logical queries, and we want to approximate a real-valued function (a probability) given a set of training examples together with their desired outcome. The error function that we minimize is the mean squared error:

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

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 in $\tilde{p}_i$, 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.

Gradient descent is a standard way of minimizing a given error function. We now derive the gradient of the MSE. Applying the sum and chain rule to Eq. (2) yields the partial derivative $\partial MSE(T)/\partial p_j =$

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

Part 1 can be calculated by a ProbLog inference call computing (1). It does not depend on j and has to be calculated only once in every iteration of a gradient descent algorithm. Part 2 is calculated as following

$$\frac{\partial P_s(q_i|T)}{\partial p_j} = \sum_{\substack{S \subseteq L_T \\ S \models q_i}} \delta_{jS} \prod_{\substack{c_x \in S \\ x \neq j}} p_x \prod_{\substack{c_x \in L_T \setminus S \\ x \neq j}} (1 - p_x)\ , \quad (4)$$

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 clauses, which is straightforward, and then summing over all subsets S where q_i can be proven.

All p_j values are probabilities. To maintain this property during gradient descent, we reparameterize the search space to arbitrary real numbers 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 $a_j \in \mathbb{R}$.

This technique has also been used for Bayesian networks and in particular for sigmoid belief networks (Saul et al., 1996). We can derive the partial derivative $\partial P_s(q_i|T)/\partial a_j$ in the same way as (4) but we have to apply the chain rule one more time due to the σ function

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

We also have to replace every p_j in Eq. (1) by $\sigma(p_j)$. Going over all subprograms S in the last equation is infeasible, but by traversing the BDDs corresponding to the query q_i, the value of the gradient can be calculated efficiently.

Given the gradient (3), we can minimize the MSE by running a standard gradient descent algorithm. We initialize the parameters a_j of the ProbLog program T randomly, calculate in every iteration the gradient, and add the negative gradient multiplied by the learning rate η. A small η will lead to a slow convergence whereas larger values can provoke oscillation.

4. Experiments

We implemented the gradient descent algorithm in Prolog (Yap-5.1.3) and used CUDD[1] for BDD operations. 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. The graph was obtained by taking the union of subgraphs of radius 3 from each gene, producing weights as described in (Sevon et al., 2006).

We randomly sampled 100 node pairs in the graph and calculated the probability that there exists a path between them using approximative ProbLog inference. We split the set in 5 folds of 20 query-probability-tuple each. Then we used 4 folds as training set and the remaining fold as test set. We initialized the parameters a_j randomly but kept the seed constant for succeeding experiments. The learning rate η was set to 80, the size of the training set, which simply retracts the scaling induced by the $1/K$ factor of the MSE. We found that this value yields a good trade-off between convergence speed and tendency to oscillate.

Figure 2 shows the learning curve for the test set. After 50 iterations, the MSE is 0.00016 ± 0.00001 on the training set and 0.00107 ± 0.00065 on the test set. This answers our question affirmatively.

[1] `http://vlsi.colorado.edu/~fabio/CUDD`

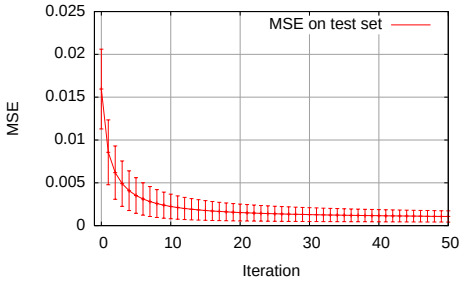


Figure 2. Learning curve on the test set; the error bars indicate the standard deviation over the 5 folds.

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

References

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, *35*, 677–691.

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). ProbLog: 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.

Saul, L., Jaakkola, T., & Jordan, M. (1996). Mean field theory for sigmoid belief networks. *JAIR*, *4*, 61–76.

Sevon, P., Eronen, L., Hintsanen, P., Kulovesi, K., & Toivonen, H. (2006). Link discovery in graphs derived from biological databases. *Proceedings 3rd International Workshop on Data Integration in the Life Sciences* (pp. 35–49). Springer.