A Sparse Parameter Learning Method for Probabilistic Logic Programs

Masaaki Nishino[†*] and **Akihiro Yamamoto**[*] and **Masaaki Nagata**[†]
[†]NTT Communication Science Laboratories, NTT Corporation
2-4, Hikaridai, Seika, Souraku, Kyoto, 239-0153, Japan
[*]Graduate School of Informatics, Kyoto University
Yoshida Honmachi, Sakyo, Kyoto, 606-8501, Japan
{nishino.masaaki,nagata.masaaki}@lab.ntt.co.jp, akihiro@i.kyoto-u.ac.jp

We propose a new parameter learning algorithm for ProbLog, which is an extension of a logic program that can perform probabilistic inferences. Our algorithm differs from previous parameter learning algorithms for probabilistic logic program (PLP) models on the point that it tries to reduce the number of probabilistic parameters contained in the estimated program. Since the amount of computation required for a probabilistic inference with a ProbLog program can be exponential with respect to the number of probabilistic parameters, programs with fewer parameters are preferable. Our algorithm tries to reduce the number of parameters by adding a penalty term to the objective function, and then minimizing it to encourage the estimated parameters to take either 0 or 1. If a parameter takes value 0 or 1, we can delete the corresponding probabilistic fact, or treat the corresponding fact as a deterministic one, to remove the parameter from the obtained model. We also show that the optimization problem can be solved efficiently by applying the projected gradient method to a compiled knowledge representation. We confirm experimentally that the proposed algorithm is comparable with the state-of-the-art algorithm, while it can reduce the number of probabilistic parameters contained in the estimated program.

Introduction

A probabilistic logic program (PLP) is an extension of a logic program that can perform probabilistic inferences. A PLP is a kind of statistical relational model (Getoor and Taskar 2007), and was developed for modeling complex and uncertain relationships. Many PLP models have been proposed (e.g., (Muggleton 1996; Sato and Kameya 2001; De Raedt, Kimmig, and Toivonen 2007)), and most of existing PLP models are based on Sato's distribution semantics (Sato 1995), which defines a probability distribution over possible worlds by introducing probabilistic ground facts into a logic program.

A problem with these PLP models is the difficulty related to inferences. In the worst case, a PLP model that is based on the distribution semantics may require computational time exponential to the number of probabilistic parameters contained in the model. It prevents PLP models from being applied to large problems. Although some efficient exact and approximate inference algorithms have been proposed for these models (De Raedt, Kimmig, and Toivonen 2007), inference is still difficult and hence PLP models with fewer probabilistic parameters are preferable.

In this paper, we propose a parameter learning algorithm for probabilistic logic programs. The proposed algorithm can reduce the number of probabilistic factors contained in the estimated model. Parameter estimation algorithms for PLP models proposed in previous research (e.g, (Gutmann, Thon, and De Raedt 2011; Sato and Kameya 2001)) are not intended for estimating compact models, i.e., a model with fewer probabilistic parameters. In order to estimate a compact model, we add penalty terms to the negative log-likelihood function and then minimize it to estimate probabilistic parameters. Penalty terms are often introduced into machine learning algorithms to impose some restrictions on the estimated parameters. A well-known penalty term is ℓ_1 norm, which is applied for obtaining sparse solutions (Bach et al. 2009), but the ℓ_1 or other sparsity inducing norms cannot be directly applied to the parameter learning problem of PLP models. The new penalty term we propose in this paper induces the learned parameters to take either 0 or 1. When a probabilistic parameter is 0 or 1, we can remove it or treat it as deterministic to obtain a PLP model with fewer number of probabilistic parameters. We also give an efficient optimization algorithm that can run on a compiled knowledge representation. Given an objective function with penalty terms, we minimize it by applying a projected gradient algorithm (Bertsekas 1999). A projected gradient algorithm is an efficient method for solving an optimization problem with constraints that a solution must be contained in a convex set, and we present a method to run it with a compiled knowledge representation, which is obtained by transforming a logical model into another form, and reduces the computational time. As the transformation, we use a deterministic and decomposable negation normal form (d-DNNF) (Darwiche and Marquis 2002) so that we make optimization problems tractable.

In the following, we use ProbLog (De Raedt, Kimmig, and Toivonen 2007) as a concrete example of PLP model and we propose a parameter learning algorithm on it. However, with slight modification our method may also be applicable to other PLP models. We give a class of PLP models

to which our parameter learning algorithm can be applied.

Preliminaries

We first briefly introduce some basic notations used in this paper. A term is a variable, a constant, or a function applied to terms. Let $\mathtt{q}(\mathtt{t}_1,\ldots,\mathtt{t}_k)$ be an atom, where $\mathtt{t}_1,\ldots,\mathtt{t}_k$ are terms and q is a predicate of arity k. Definite clauses are universally quantified expressions of the form `h :- b1, ..., bn`, where $\mathtt{h},\mathtt{b}_1,\ldots,\mathtt{b}_n$ are all atoms, and h is the head of a clause and $\mathtt{b}_1,\ldots,\mathtt{b}_n$ are the body. A clause without a body is a fact. A substitution θ is an expression of the form $\{V_1/\mathtt{t}_1,\ldots,V_m/\mathtt{t}_m\}$ where V_i are different variables and $\mathtt{t}_i$ are terms. If a substitution θ is applied to an expression e, then the instantiated expression $e\theta$ is made by simultaneously replacing the variables V_i in e with $\mathtt{t}_i$. An expression is called ground if it has no variables. The semantics of a set of definite clauses is given by its least Herbrand model, i.e., the set of all ground facts entailed by the theory.

A ProbLog theory $\mathcal{T}$ consists of both a set of labeled facts F and a set of definite clauses KB. Let f_i be a fact contained in F and $w_i \in [0,1]$ be the label of f_i. w_i represents the probability that each ground substitution $f_i\theta$ is true in the theory. We refer to an annotated fact $w_i :: f_i$ as a *probabilistic fact*.

Example 1. *The following is an example of ProbLog program.*

```
0.1::burglary.    0.2::earthquake.
0.7::hears_alarm(X) :- person(X).
person(mary).    person(john).
alarm :- burglary.    alarm :- earthquake.
calls(X) :- alarm, hears_alarm(X).
```

In this program, `burglary.` *and* `earthquake.` *are facts and their probabilities are* 0.1 *and* 0.2*, respectively.* `0.7 :: hears_alarm(X) :- person(X).` *is a notation that represents two probabilistic facts,* `0.7::hears_alarm(mary)` *and* `0.7::hears_alarm(john)`.

Given a finite number of possible ground substitutions $\{\theta_{i,1},\ldots,\theta_{i,K_i}\}$ for each probabilistic fact $w_i :: f_i$, a ProbLog program $\mathcal{T}$ defines probability distribution over total choices L, where L is a subset of the set of all ground facts $L_\mathcal{T} = \{f_1\theta_{1,1},\ldots,f_1\theta_{1,K_1},\ldots,f_N\theta_{N,1},\ldots,f_N\theta_{N,K_N}\}$.

$$P(L|\mathcal{T}) = \prod_{f_i\theta_{i,k}\in L} w_i \prod_{f_i\theta_{i,k}\in L_\mathcal{T}\setminus L} (1-w_i)\ . \qquad (1)$$

Using the above definition of probability $P(L|\mathcal{T})$, we define the success probability of a query literal q as

$$P(q|\mathcal{T}) = \sum_{L\subseteq L_\mathcal{T}, L\cup KB\models q} P(L|\mathcal{T}) \sum_{L\subseteq L_\mathcal{T}} \delta(q, KB\cup L)\cdot P(L|\mathcal{T})$$

where $\delta(q, KB\cup L) = 1$ if there exists a substitution θ such that $KB\cup L \models q\theta$, and 0 otherwise.

Example 2. *For the program in Example 1, $L_\mathcal{T}$ contains four probabilistic facts, and hence there are $2^4 = 16$ possible $L \subseteq L_\mathcal{T}$. If $q =$* `alarm`*, $\delta(q, KB\cup L) = 1$ if either* `burglary` *or* `earthquake` *is contained in L, and its probability is $P(\mathtt{alarm}|\mathcal{T}) = 1 - P(\neg\mathtt{burglary}\wedge\neg\mathtt{earthquake}|\mathcal{T}) = 1 - 0.9\times 0.8 = 0.28$.*

Parameter Learning

Motivating Examples

Before presenting our parameter learning algorithm, we first show what it aims to do. What we want is a compact ProbLog program, but a compact program is not just a program with fewer clauses. At this point, our algorithm differs slightly from sparse learning algorithms (Bach et al. 2009); sparse learning algorithms try to obtain sparse models by letting many parameters take zero. By contrast, our learning algorithm induces many parameters to take either 0 or 1. This setting is motivated by the following two examples.

$$w_1\mathtt{:: q.} \quad w_2\mathtt{:: r.} \quad \mathtt{p :- q.} \quad \mathtt{p :- r.}$$

Suppose that we are given training examples $\mathcal{D}$ that only contains literal p, and we want to set parameters w_1, w_2 so as to maximize the log-likelihood of $\mathcal{D}$. If training examples follow probabilistic distribution $P(\mathtt{p}) = 0.5$, then any combination of parameters w_1 and w_2 that satisfies $1.0-(1-w_1)(1-w_2) = 0.5$ will maximize log-likelihood. However, if we set $w_1 = 0.0$ and $w_2 = 0.5$ (or equivalently, set $w_1 = 0.5$ and w_2 = 0.0), then we can remove one probabilistic fact from the program. This clearly reduces the size of the obtained program. The above approach is equivalent to theory compression (De Raedt et al. 2008), which allows some parameters of a ProbLog program to be zero to obtain a more concise logic program.

We give another example.

$$w_1\mathtt{:: q.} \quad w_2\mathtt{:: r.} \quad \mathtt{p :- q, r.}$$

With this program, the probability $P(\mathtt{p})$ is represented as $P(\mathtt{p}) = w_1w_2$. As same as the previous example, suppose that we are given training examples $\mathcal{D}$ that only contains literal p, and we want to set parameters w_1, w_2 so as to maximize log-likelihood of $\mathcal{D}$. If training examples follow probabilistic distribution $P(\mathtt{p}) = 0.5$, many combinations of parameters are possible. However, if we set $w_1 = 1.0$ and $w_2 = 0.5$ (or $w_1 = 0.5$ and $w_2 = 1.0$), it means we treat the probabilistic fact q as a deterministic (i.e., non probabilistic) fact. Since the complexity of probabilistic inference with a ProbLog program depends on the number of probabilistic facts, treating some probabilistic facts as deterministic facts also helps to reduce the cost of probabilistic inference.

Our algorithm uses a penalty function to obtain these two types of reduction of probabilistic parameters, namely, (i) removing probabilistic facts and (ii) substituting probabilistic facts with deterministic ones.

Learning Algorithm

Our parameter learning algorithm follows the learning from interpretation setting that has been proposed in (Gutmann, Thon, and De Raedt 2011) since it is more general than the learning from entailment settings as used in other learning algorithms (Gutmann et al. 2008; Sato and Kameya 2001; Cussens 2001). Let I be a partial interpretation of ground

atoms in $L_{\mathcal{T}}$, which determines the truth values of some atoms in $L_{\mathcal{T}}$. We represent a partial interpretation as $I = (I^+, I^-)$ where I^+ contains all true atoms and I^- all false atoms. We define the probability of a partial interpretation I in a way that is similar to that for an atom q:

$$P(I|\mathcal{T}) = \sum_{L \subseteq L_{\mathcal{T}}} \delta(I^+, KB \cup L)\bar{\delta}(I^-, KB \cup L)P(L|\mathcal{T}),$$

where $\delta(I^+, KB \cup L) = 1$ if $KB \cup L \models q$ for all atoms $q \in I^+$, and $\bar{\delta}(I^-, KB \cup L) = 1$ if $KB \cup L \not\models q$ for all atoms $q \in I^-$.

Parameter learning is formalized as the task of finding a set of parameters $\hat{\mathbf{w}} = \{\hat{w}_1, \ldots, \hat{w}_N\}$ that minimizes the objective function given training examples. Let a set of interpretation $\mathcal{D} = I_1, \ldots, I_M$ be training examples. We make the objective function as a combination of the negative log-likelihood and a penalty function that encourages parameters to take either 0 or 1. We therefore define the objective function $g(\mathcal{T}(\mathbf{w}), \mathcal{D})$ as

$$g(\mathcal{T}(\mathbf{w}), \mathcal{D}) = \ell(\mathcal{T}(\mathbf{w}), \mathcal{D}) + \lambda h(\mathbf{w}) \ ,$$

where $\mathcal{T}(\mathbf{w})$ represents a ProbLog program whose parameters are $\mathbf{w}$, and $\ell(\mathcal{T}(\mathbf{w}), \mathcal{D})$ represents a negative log-likelihood function and $h(\mathbf{w})$ is a penalty function. A parameter λ controls the effect of the penalty term. We minimize the above objective function by considering that $0 \leq w_i \leq 1$ for all $1 \leq i \leq N$. We define the negative log-likelihood function $\ell(\mathcal{T}(\mathbf{w}), \mathcal{D})$ as

$$\ell(\mathcal{T}(\mathbf{w}), \mathcal{D}) = -\sum_{j=1}^{M} \log P(I_j|\mathcal{T}(\mathbf{w})) \ .$$

It is simply defined as the logarithm of the product of the probabilities $P(I_j|\mathcal{T}(\mathbf{w}))$ for $j = 1, \ldots, M$. We define the penalty function $h(\mathbf{w})$ as

$$h(\mathbf{w}) = \sum_{i=1}^{N} \{\log(w_i + \varepsilon) + \log(1 - w_i + \varepsilon)\} \ , \qquad (2)$$

where ε is a small positive value that is added to avoid the function becoming $-\infty$ at $w_i = 0$ or $w_i = 1$. The penalty term takes smaller values when w_i is near 0 or 1, hence we can encourage w_i to take 0 or 1 by adding $h(\mathbf{w})$ to the negative log-likelihood $g(\mathcal{T}(\mathbf{w}), \mathcal{D})$. Hence this penalty term can contribute to reduce the number of parameters. We show an example of $h(\mathbf{w})$ when $N = 1$ in Fig.1.

The minimization problem of only negative log-likelihood function $\ell(\mathcal{T}(\mathbf{w}))$ is the same as the problem solved in the LFI-ProbLog algorithm shown in (Gutmann, Thon, and De Raedt 2011), and it can be solved efficiently by using the Expectation Maximization (EM) algorithm. The difference between the LFI-ProbLog algorithm and ours is the use of the penalty term $h(\mathbf{w})$. This addition makes EM algorithm inapplicable for our problem. We therefore introduce a new optimization method that is based on the projected gradient algorithm.

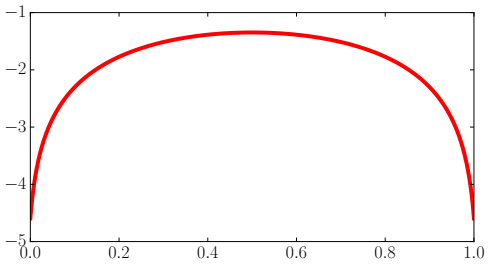

Figure 1: Shape of the penalty term $h(\mathbf{w})$, where $N = 1$ and $\varepsilon = 0.001$.

Algorithm 1 A projected gradient parameter learning algorithm

1: Initialize $\mathbf{w}^0$, set $k \leftarrow 0$.
2: **while** k is less than the iteration limit **do**
3: $\quad \mathbf{x}^k \leftarrow \mathbf{w}^k - \alpha_k \nabla g(\mathbf{w}^k)$
4: $\quad \mathbf{w}^{k+1} \leftarrow \mathbf{proj}(\mathbf{x}^k)$
5: $\quad$ **if** converged($\mathbf{w}^k$, $\mathbf{w}^{k+1}$) **then**
6: $\quad\quad$ **return** $\mathbf{w}^k$
7: $\quad k \leftarrow k + 1$
8: **return** $\mathbf{w}^k$

Projected Gradient Algorithm

The minimization problem we introduced in the previous section cannot be solved with the EM algorithm, we therefore propose a new method that is based on the projected gradient algorithm. The projected gradient algorithm (Bertsekas 1999) is used for minimization problems with the constraint that the variables must be contained in a convex set. In the present case, w_i must satisfy $0 \leq w_i \leq 1$, and hence the region in which $\mathbf{w}$ is contained is convex. We can therefore employ the projected gradient method for our problem.

The projected gradient algorithm is an extension of a gradient descent algorithm, and it solves optimization problems by repeating gradient computation and projection onto a convex set. Algorithm 1 shows the projected gradient algorithm. After initializing $\mathbf{w}^0$ (line 1), it repeatedly updates $\mathbf{w}^k$ until it converges (lines 2 to 7). First we compute $\mathbf{x}^k$ from the current $\mathbf{w}^k$ and the gradient $\nabla g(\mathbf{w}^k)$ (line 3). This step is the same as an ordinary gradient descent algorithm, then we project $\mathbf{x}^k$ into the domain that satisfies $0 \leq w_i \leq 1$ $(1 \leq i \leq N)$ to obtain $\mathbf{w}^{k+1}$. Here, function $\mathbf{proj}(\mathbf{x})$ is a projection function that maps $\mathbf{x}$ to $\mathbf{w}$ such that satisfies $0 \leq w_i \leq 1$ for all $1 \leq i \leq N$ and minimizes $||\mathbf{x} - \mathbf{w}||^2$. In this case, the i-th element $[\mathbf{proj}(\mathbf{x})]_i$ is defined as

$$[\mathbf{proj}(\mathbf{x})]_i = \begin{cases} 0 & \text{if } x_i \leq 0 \\ x_i & \text{if } 0 < x_i < 1 \\ 1 & \text{if } x_i \geq 1 \end{cases} \ , \qquad (3)$$

i.e., we simply map x_i to 0 or 1 if it is not in the $0 \leq x_i \leq 1$ range.

α_k is the step size used for the k-th iteration. Setting an appropriate α_k is important since it determines the convergence of the projected gradient algorithm. We set α^k by using a simple line search based procedure called *the Armijo*

rule along the projection arc, as described in (Bertsekas 1999; 1976). It defines α_k as $\alpha_k = \beta^{t_k}$, where $\beta \in (0, 1)$, and t_k is the first non-negative integer t that satisfies

$$g(\mathbf{w}^k) - g(\mathbf{w}^k(\beta^{t_k})) \leq \sigma \nabla g(\mathbf{w}^k)^T(\mathbf{w}^k - \mathbf{w}^k(\beta^{t_k})) \ , \quad (4)$$

where $\mathbf{w}^k(\beta^{t_k})$ is $\mathbf{proj}(\mathbf{w}^k - \beta^{t_k}\nabla g(\mathbf{w}^k))$, and σ is a parameter that satisfies $\sigma \in (0, 1)$. In the experiments, we use parameters $\sigma = 0.2$ and $\beta = 0.5$, as these values are suggested in the textbook (Bertsekas 1999). By using α_k selected with this rule, it is proved that the projected gradient algorithm converges to a stationary point after several iterations(Bertsekas 1999; Calamai and Moré 1987) even if the objective function is nonconvex. Hence Alg. 1 converges after finite numbers of iterations.

Computation of gradient

To compute gradient $\nabla g(\mathcal{T}(\mathbf{w}), \mathcal{D})$, we must compute $\nabla \ell(\mathcal{T}(\mathbf{w}), \mathcal{D})$ and $\nabla h(\mathbf{w})$. Let $[\nabla \ell(\mathcal{T}(\mathbf{w}), \mathcal{D})]_i$ be the i-th element of the gradient $\nabla \ell(\mathcal{T}(\mathbf{w}), \mathcal{D})$, then it becomes

$$[\nabla \ell(\mathcal{T}(\mathbf{w}), \mathcal{D})]_i = -\sum_{j=1}^{N} \frac{\sum_{L \in \mathcal{L}_j} [\nabla P(L|\mathcal{T}(\mathbf{w}))]_i}{P(I_j|\mathcal{T}(\mathbf{w}))} \ . \quad (5)$$

Here we use $\mathcal{L}_j$ as the set that contains all possible assignments $L \subseteq L_{\mathcal{T}}$ that are consistent with I_j. We define $\nabla P(L|\mathcal{T}(\mathbf{w}))$ as

$$[\nabla P(L|\mathcal{T}(\mathbf{w}))]_i = \sum_{k=1}^{K_i} (2\delta(f_i\theta_{i,k} \in L) - 1) \prod_{f_n\theta_{n,k} \in L^{-i,k}} w_n \prod_{f_n\theta_{n,k} \in L_{\mathcal{T}}^{-i,k} \setminus L} (1 - w_n),$$

where $\delta(f_i\theta_{i,k} \in L) = 1$ if the condition is satisfied, otherwise 0, and $L^{-i,k}$ is $L \setminus \{f_i\theta_{i,k}\}$. $\nabla h(\mathbf{w})$ is easy to compute and $[\nabla h(\mathbf{w})]_i$ becomes

$$[\nabla h(\mathbf{w})]_i = \frac{1}{w_i + \varepsilon} - \frac{1}{1 - w_i + \varepsilon} \ .$$

The computation of gradient $\nabla \ell(\mathcal{T}(\mathbf{w}), \mathcal{D})$ involves summation over $L \in \mathcal{L}_j$. Since $\mathcal{L}_j$ consists of all the possible combinations of ground probabilistic facts $L_{\mathcal{T}}$, its size becomes $2^{L_{\mathcal{T}}}$ in the worst case and the naive computation of gradients is intractable with a large $L_{\mathcal{T}}$. Hence we use the knowledge compilation technique to compute them.

Knowledge compilation (Darwiche and Marquis 2002) is an approach that can be used for efficient computation involving propositional models. It first compiles a propositional model so as to represent it in a form that is suitable for specific operations such as probabilistic inference. Although knowledge compilation incurs the additional cost of compiling a model, once a compiled model is obtained, we can efficiently perform several operations by using it. In previous work, a ProbLog program was compiled into *binary decision diagrams (BDD)* (Bryant 1986), and *a deterministic, decomposable negation normal form (d-DNNF)* (Darwiche and Marquis 2002; Darwiche 2009). With a ProbLog program compiled into a BDD or a d-DNNF, we can compute the probability $P(I|\mathcal{T}(\mathbf{w}))$ in time proportional to the size of the compiled representation. This computation is also used when employing EM style algorithms (Fierens et al. 2013; Gutmann, Thon, and De Raedt 2011).

We use compiled knowledge representations for computing the gradients of the negative log-likelihood function $\nabla \ell(\mathcal{T}(\mathbf{w}), \mathcal{D})$. Although both BDD and d-DNNF can be used for computing gradients, we chose d-DNNF as a compiled knowledge representation since it performed well in the previously reported parameter learning algorithms (Fierens et al. 2013).

We briefly introduce d-DNNF. d-DNNF is a kind of negation normal form (NNF) that satisfies decomposability and determinism. An NNF is a rooted directed acyclic graph in which each leaf node is labeled with a literal, true or false, and each internal node is labeled with a conjunction or disjunction. For any node n in an NNF graph, $vars(n)$ denotes all propositional variables that appear in the subgraph rooted at n, and $\Delta(n)$ denotes the formula represented by n and its descendants. We say an NNF satisfies decomposability if $vars(n_i) \cap vars(n_j) = \emptyset$ holds for any two children n_i and n_j $(i \neq j)$ of an and-node n, and NNF satisfies determinism when $\Delta(n_i) \wedge \Delta(n_j)$ is logically inconsistent for any two children n_i and n_j $(i \neq j)$ for an or-node n.

Given the set of interpretations $\mathcal{D}$, the process of compiling a ProbLog program into d-DNNFs is the same as for the parameter learning method proposed in (Fierens et al. 2013): We first make a ground ProbLog program and convert it into a conjunctive normal form (CNF), and then convert it into d-DNNF with a d-DNNF compiler that converts a CNF into the corresponding d-DNNF. After obtaining N different d-DNNFs that correspond to each interpretation $I_1, \ldots, I_N$, we use them to compute the probability $P(I_j|\mathcal{T}(\mathbf{w}))$ and $\nabla P(L, \mathcal{T}(\mathbf{w}))$, which is the numerator of (5). We use the inference algorithm shown in (Darwiche 2009), which was originally used for computing conditional probabilities and gradients for graphical models. These algorithms can compute $P(I_j|\mathcal{T}(\mathbf{w}))$ by traversing all the nodes of a d-DNNF once, and can compute $\nabla P(L|\mathcal{T}(\mathbf{w}))$ by traversing all the nodes twice. As a result, we can efficiently compute $\nabla g(\mathcal{T}(\mathbf{w}), \mathcal{D})$.

Discussion

Our sparse parameter learning algorithm can be applied to other PLP models whose probabilistic parameters take values in $[0, 1]$. For example, PRISM, SLP, and ICL are in this class of PLP models. A major difference between ProbLog and these models is that these models employ the multinomial distribution, while ProbLog programs give probabilistic distribution based on the Bernoulli distribution defined on probabilistic facts.

With these multinomial distribution PLP models, the set of probabilistic parameters can be represented as $\mathbf{w} = \{\mathbf{w}_1, \ldots, \mathbf{w}_N\}$, where $\mathbf{w}_i$ $(i = 1, \ldots, N)$ is a K_i dimensional vector $\mathbf{w}_i = (w_{i1}, \ldots, w_{iK_i})$ and it satisfies $\sum_{j=1}^{K_i} w_{ij} = 1$ and $w_{ij} \geq 0$ for $j = 1, \ldots, K_i$. Here we make an assumption that we can compute the gradient of the negative log-likelihood function $\ell(\mathbf{w}, \mathcal{D})$, given a set of

training examples $\mathcal{D}$. Then we apply our algorithm by only modifying the penalty function $h(\mathbf{w})$ defined in (2) as

$$h(\mathbf{w}) = \sum_{i=1}^{N} \sum_{j=1}^{K_i} \log\left(w_{ij} + \varepsilon\right).$$

If $K_i = 2$ for all $i = 1, \ldots, N$, the above definition of $h(\mathbf{w})$ is equivalent to that in (2). It is also easy to compute $\nabla h(\mathbf{w})$.

we also need to modify the projection function $\mathbf{proj}(\mathbf{x})$ to perform our projected gradient algorithm. We have to project a K_i dimensional real value vector $\mathbf{x}_i$ onto a probability simplex, i.e., project onto a K_i dimensional vector $\mathbf{w}_i$ that satisfies $\sum_{j=1}^{K_i} w_{ij} = 1$ and $w_{ij} \geq 0$ for all $j = 1, \ldots, K_i$. This type of projection also can be efficiently performed (Parikh and Boyd 2014).

Experiments

Settings

We conducted experiments to evaluate our proposed learning algorithm. Our aim was to answer the following questions:

1. (**Q1**) Can we learn a compressed ProbLog program with our proposed algorithm?
2. (**Q2**) How do the estimated model changes when we change the parameter λ ?
3. (**Q3**) Can the proposed algorithm recover true distributions with a sufficient number of training examples?

We compare our proposed method with the LFI-ProbLog algorithm, which is an EM-style algorithm for estimating parameters from interpretations (Gutmann, Thon, and De Raedt 2011). We also compared our algorithm with an gradient projection algorithm, which simply minimizes the negative log-likelihood function.

Since our algorithm has much in common with the LFI-ProbLog algorithm, we implemented our algorithm on ProbLog2 (Renkens et al. 2012), a ProbLog implementation that supports several inference methods and parameter learning algorithms. We use grounding, CNF conversion, and a d-DNNF compilation algorithm implemented in ProbLog2, and run our projected gradient algorithm on a compiled d-DNNF to estimate the parameters. We also use the LFI-ProbLog algorithm implemented in ProbLog2.

We use two datasets, WebKB and Smokers for evaluating our parameter learning algorithm. The WebKB dataset[1] (Craven and Slattery 2001) is a real dataset that consists of labeled Web pages from the computer science departments of four universities. Every web page is marked with one of the following categories, *student*, *faculty*, *project*, *course*, *staff*, and *other*. The task is to predict the classes of pages given the words contained in each page and the link structure between pages. Following the setting used in (Fierens et al. 2013), we use the following ProbLog program.

```
p::link_class(P,P2,c1,c2) :- links_to(P,P2).
p::word_class(P,w1,c1) :- has_word(P,w1).
```

[1] `http://www.cs.cmu.edu/~webkb/`

Table 1: Negative Log-Likelihood (lower is better) and the number of probabilistic parameters contained on the WebKB learning experiment.

Method	NLL	Num. params
LFI	1387.28	39.0
PG	1299.30	38.0
PG+P ($\lambda = 0.001$)	1318.96	25.0
PG+P ($\lambda = 0.01$)	1445.37	18.0

```
p::learnable_prior(P,c1) :- page(P).
0.001::fixed_prior(P,c1):-page(P),class(c1).
has_class(P,C) :- word_class(P,W,C).
has_class(P,C) :- has_class(P2,C2),
                  link_class(P,P2,C,C2).
has_class(P,C) :- fixed_prior(P,C).
has_class(P,C) :- learnable_prior(P,C).
```

Where the probabilistic fact `link_class/4` represents the effect of the link structure, and `word_class/3` represents the effect of words contained in a page. We also added probabilistic facts `learnable_prior/2` to represent the probabilistic distribution on labels that are independent with the link structure and words. Finally we add `fixed_prior/2` for avoiding log-likelihood to become infinity in the test data. For computational reasons, we selected 20 words that show the highest information gain with the class labels. We therefore have in total $6 \times 20 + 6 \times 6 + 6 = 162$ probabilistic parameters to be estimated from the data. We conduct four-fold cross validation by using the dataset for three universities as a training set and use the other university as the test set.

The Smokers dataset (Domingos and Lowd 2009) represents the relationships between people, and contains the following probabilistic facts and rules.

```
0.2::stress(P) :- person(P).
0.3::influences(P1,P2) :- friend(P1,P2).
0.1::cancer_spont(P) :- person(P).
0.3::cancer_smoke(P) :- person(P).
smokes(P) :- stress(P).
smokes(P) :- smokes(P2), influences(P2, P).
cancer(P) :- cancer_spont(P).
cancer(P) :- smokes(P), cancer_smoke(P).
```

In addition to the above program, we add some ground facts person/1 and friend/2 into the program. We add them by first deciding the number of people in the domain, and then randomly deciding friend relationships between people. We set the number of people to 4.

Results

Table 1 shows the average negative log-likelihood and the number of parameters for WebKB dataset. Here we use LFI, PG, PG+P to represent the results of LFI-ProbLog, projected gradient algorithm, and projected gradient algorithm with penalty term (the proposed method), respectively. We can see that PG+P shows comparable performance comparing with the state-of-the-art method LFI when $\lambda = 0.001$, while it can reduce the average number of parameters contained in

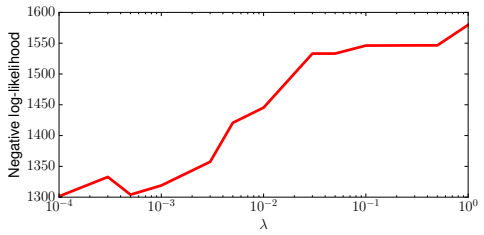

Figure 2: Negative log-likelihood for different λ.

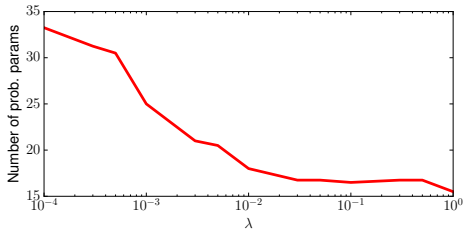

Figure 3: The number of estimated probabilistic parameters for different λ.

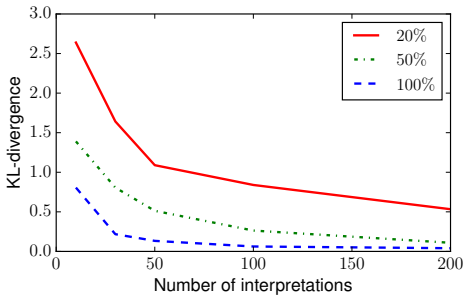

Figure 4: KL divergence on Smokers dataset with different numbers of evidences.

the learned model from 39 to 25. Following these results, we can answer **Q1** that the proposed method shows the inference performance that is comparable with the state-of-the-art algorithm, while it can reduce the number of parameters.

For answering **Q2**, we show the results of the proposed algorithm while changing the parameter λ in Fig.2 and Fig.3. From the result of Fig.3, we can see that the number of parameters contained in the learned program monotonically decreases when we use a large λ. This result reflects that the parameters tend to take 0 or 1 if we add more weight to the penalty term. Figure 2 shows that the performance decreases as we use a large λ. This result suggests the performance may decrease if we penalize too much.

To answer the **Q3**, we measured the KL divergence between the true probabilistic distribution and the distribution estimated by the proposed algorithm. We first make 10, 20, 50, 100, 200 different interpretations on `smokes/1` and `cancer/1` atoms of Smokers dataset, and then sample $20\%, 50\%$, and 100% of them to make training data. We evaluated the KL divergence between the true probabilistic distribution on `smokes/1` and `cancer/1` atoms and the distribution estimated from the data. Figure 4 shows the results. We can see that KL divergence decreases as the number of interpretations increases. We therefore can say that the proposed algorithm can estimate the true probabilistic distribution with a sufficient amount of training data (**Q3**).

Related Work

Many PLP models, such as ProbLog (De Raedt, Kimmig, and Toivonen 2007), PRISM (Sato and Kameya 2001), SLP (Muggleton 1996), ICL (Poole 2008), and parameter learning algorithms for these models have been proposed. Cussens proposed a parameter learning algorithm for SLP (Cussens 2001), Sato (Sato and Kameya 2001) proposed an EM learning algorithm for PRISM, and Gutmann et al. proposed two parameter learning algorithms for ProbLog (Gutmann, Thon, and De Raedt 2011; Gutmann et al. 2008). These algorithms exploit EM-learning or gradient descent methods to optimize an objective function for estimating parameters. Our proposed method differs in that we add penalty terms to induce parameters to take a zero or one probability. This feature resembles the sparse learning algorithms (Bach et al. 2009) used in many machine learning problems, but we believe that this paper is the first to apply a sparse learning method to a parameter learning problem for PLP models. ℓ_1-regularization is used in structure learning for Markov Logic Networks (Huynh and Mooney 2011), however, the algorithm cannot be directly applied to PLP models like ProbLog.

Our work is also similar to probabilistic theory compression (De Raedt et al. 2008) and the theorem revision methods (Zelle and Mooney 1994) in that it tries to compress a theory into a more concise form. Our algorithm differs in that it simultaneously removes probabilistic facts and infers parameters in one operation.

Our algorithm can be seen as a kind of structure learning algorithm for PLP models, since it outputs a new PLP model given a prototype program and training examples. A previously reported structure learning algorithm for a PLP program is a beam search based algorithm, and it makes it necessary to solve the EM style parameter learning algorithm many times (Bellodi and Riguzzi 2012). Obviously we must conduct empirical comparisons, but we believe our algorithm can be more efficient than the previous structure leaning approach since it can find a program by just performing projection gradient based optimization.

Conclusion

We proposed a novel parameter learning algorithm for PLP models that attempts to set the learned parameters so that they take either 0 or 1 by adding a penalty term to an optimization problem. With our algorithm, the learned ProbLog program will have fewer probabilistic components, and inference tasks performed with it become easier. We solved the optimization algorithm by combining the gradient projection algorithm and the computation of gradients in a d-DNNF based knowledge representation.

References

Bach, F.; Jenatton, R.; Mairal, J.; and Obozinski, G. 2009. Optimization with sparsity-inducing penalties. *Foundations and Trends in Theoretical Computer Science* 3(2-3).

Bellodi, E., and Riguzzi, F. 2012. Learning the structure of probabilistic logic programs. In *Inductive Logic Programming*, 61–75.

Bertsekas, D. 1976. On the goldstein-levitin-polyak gradient projection method. *Automatic Control, IEEE Transactions on* 21(2):174–184.

Bertsekas, D. P. 1999. *Nonlinear programming*. Athena Scientific.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Trans. on* 100(8):677–691.

Calamai, P. H., and Moré, J. J. 1987. Projected gradient methods for linearly constrained problems. *Mathematical Programming* 39(1):93–116.

Craven, M., and Slattery, S. 2001. Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning* 43(1-2):97–119.

Cussens, J. 2001. Parameter estimation in stochastic logic programs. *Machine Learning* 44(3):245–271.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *JAIR* 17:229–264.

Darwiche, A. 2009. *Modeling and reasoning with Bayesian networks*. Cambridge University Press.

De Raedt, L.; Kersting, K.; Kimmig, A.; Revoredo, K.; and Toivonen, H. 2008. Compressing probabilistic prolog programs. *Machine Learning* 70(2-3):151–168.

De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, 2462–2467.

Domingos, P., and Lowd, D. 2009. Markov logic: An interface layer for artificial intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 3(1).

Fierens, D.; Broeck, G. V. d.; Renkens, J.; Shterionov, D.; Gutmann, B.; Thon, I.; Janssens, G.; and De Raedt, L. 2013. Inference and learning in probabilistic logic programs using weighted boolean formulas. *arXiv preprint arXiv:1304.6810 (to appear in Theory and Practice of Logic Programming)*.

Getoor, L., and Taskar, B. 2007. *Introduction to statistical relational learning*. The MIT press.

Gutmann, B.; Kimmig, A.; Kersting, K.; and De Raedt, L. 2008. Parameter learning in probabilistic databases: A least squares approach. In *ECML/PKDD*, 473–488.

Gutmann, B.; Thon, I.; and De Raedt, L. 2011. Learning the parameters of probabilistic logic programs from interpretations. In *ECML/PKDD*, 581–596.

Huynh, T. N., and Mooney, R. J. 2011. Online structure learning for markov logic networks. In *ECML/PKDD*.

Muggleton, S. 1996. Stochastic logic programs. *Advances in inductive logic programming* 32:254–264.

Parikh, N., and Boyd, S. 2014. Proximal algorithms. *Foundations and Trends in Optimization* 1(3):123–231.

Poole, D. 2008. The independent choice logic and beyond. In *Probabilistic inductive logic programming*. Springer. 222–243.

Renkens, J.; Shterionov, D.; Van den Broeck, G.; Vlasselaer, J.; Fierens, D.; Meert, W.; Janssens, G.; and De Raedt, L. 2012. Problog2: From probabilistic programming to statistical relational learning. In *Proceedings of the NIPS Probabilistic Programming Workshop*.

Sato, T., and Kameya, Y. 2001. Parameter learning of logic programs for symbolic-statistical modeling. *JAIR* 15:391–454.

Sato, T. 1995. A statistical learning method for logic programs with distribution semantics. In *ICLP*, 715–729.

Zelle, J. M., and Mooney, R. J. 1994. Inducing deterministic prolog parsers from treebanks: A machine learning approach. In *AAAI*, 748–753.