Learning a generative failure-free PRISM clause

Waleed Alsanie[1] and James Cussens[2]

[1] Department of Computer Science
[2] York Centre for Complex Systems Analysis
& Department of Computer Science
{walsanie,jc}@cs.york.ac.uk
University of York, York, United Kingdom

Abstract. PRISM is a probabilistic logic programming formalism which allows learning parameters from examples through its graphical EM algorithm. PRISM is aimed at modelling generative processes in the compact first-order logic representation. It facilitates model selection by providing three scoring functions *Bayesian Information Criterion (BIC)*, *Cheeseman-Stutz (CS)* and *Variational free energy.* This paper considers learning failure-free single clause PRISM program by searching and scoring possible models built from observations and Background Knowledge (BK).

Keywords: Probabilistic Inductive Logic Programming, Statistical Relational Learning, PRISM, Structure Learning

1 Introduction

PRISM is a probabilistic logic programming formalism in which a probability distribution is defined over possible worlds [4]. Given a logic program $DB = F \cup R$ where F is a set of ground facts and R is a set of rules (definite clauses), a probability distribution P_F is defined over all possible assignments of truth values of the ground facts in F. Sampling from P_F will lead to a set $F' \subseteq F$ where the least Herbrand model of $F' \cup R$ represents a sample from all true ground atoms in DB. Thus P_F can be extended to P_{DB} which is knows as *Distribution Semantics* [4, 5]. With this semantic, *generative models* can be represented in such a way that each sample of the ground atoms in DB explains a generation process of an observation. PRISM allows learning parameters from observations through its graphical EM algorithm. It also facilitates model (structure) scoring with three scoring functions *Bayesian Information Criterion (BIC)*, *Cheeseman-Stutz* and *Variational free energy.* However, general structure learning of PRISM has not been addressed much.

When learning generative models as *Probabilistic Logic Programs (PLP)*, variables in the head are assumed to be output of some predicates in the body, so each observation is fully explained by the program. This is different from learning *predictive models* where, typically, the aim is to learn the simplest (most general) model that is *complete* and *consistent* according to the positive E^+ and negative E^- examples, which is normally the setting in learning with *Inverse Entailment (IE)* [3, 2].

In subsequent sections we explain a technique to learn generative failure-free single clause PRISM programs. Such programs can represent static Bayesian networks. In section 2, we briefly explain the Variational free energy scoring function. Section 3 explains the formulation of the hypothesis space and the search strategies. Section 4 shows an experiment of learning the Asia network. Finally we conclude with discussion and future directions.

2 Scoring with Variational Free Energy

In the *Variational Bayesian (VB)* approach, the marginal log-likelihood $L(M)$ is used in models scoring where $L(M) \overset{def}{=} \log p(D|M)$. As explanations Z of goals are hidden, the marginal log-likelihood can be written as:

$$L(M) = \log \sum_{Z} \int_{\theta} p(D, Z, \theta|M) d\theta \tag{1}$$

$$L(M) \geq F[q] \overset{def}{=} \sum_{Z} \int_{\theta} q(Z, \theta|D, M) \log \frac{p(D, Z, \theta|M)}{q(Z, \theta|D, M)} d\theta \tag{2}$$

$F[q]$ is the Variational free energy and it is a lower bound of $L(M)$. The full derivation of approximating the distribution p by maximising the Variational free energy is explained by Sato et al. [6]. In VB learning, a further assumption is made that $q(Z, \theta|D, M) \approx q(Z|D, M)q(\theta|D, M)$. It can be noticed from this assumption and from the integral above that this score penalises complex models (ones that have large parameter space). This trade-off between the complexity of the model and fitting the data is needed to avoid overfitting.

3 Building and Searching the Hypothesis Space

3.1 Building the Hypothesis Space

The search space is defined with respect to the observations and some *Background Knowledge (BK)*. The BK considered so far is PRISM switch definitions only. The outcomes of a switch are then added along with the switch separately as facts. For example, given the following BK:

```
values(smoking,[0,1]).
values(visitAsia,[0,1]).
```

BK will be transformed into facts as follows:

```
msw(smoking,0).
msw(smoking,1).
msw(visitAsia,0).
msw(visitAsia,1).
```

The transformed BK constitutes the base for building the search space. The search space is built by first approximating the *relative least general generalisation (RLGG)* of the first two observations relative to the BK. Then, after deleting all observations explained by the RLGG, an approximation of the *least general generalisation (LGG)*[3] of the resulting clauses (RLGG at first) and the clause formed by the first unexplained observation as a head and the BK as a body is built. The steps are repeated until all observations are explained. LGG is approximated and not built exactly because some resulting predicates violate the PRISM condition that switches must be ground terms. For example, $lgg(msw(smoking, 1), msw(visitAsia, 1)) = msw(X, 1)$ which is not acceptable in PRISM. Some heuristics can also be used to approximate the LGG in order to reduce the resulting clause. Building the hypothesis space goes as follows:

1. Build the LGG of the two clauses Obs_1*:-Conj(BK)* and Obs_2*:-Conj(BK)*, where Obs_1 is the first observation, Obs_2 is the second observation and *Conj(BK)* is a conjunction of the facts in BK. The result of the LGG is a clause *Head:-Body*.
2. Delete all observations that are explained by the resulting clause(*Head:-Body*) .
3. Build the LGG of the resulting clause (*Head:-Body*) and the clause Obs_{n-m}*:-Conj(BK)*, where n is the number of observations, m is the number of observation explained by the previously resulting clause and Obs_{n-m} is the first unexplained observation.
4. repeat step 3 until all observations are explained.

Though the final clause explains all observations, it is extremely large and contains many predicate which would cause failures in most of the samples drawn from it. However, it defines a search space in which more salient clauses are contained which can be reached either by deleting literals or selecting some literals in a way that a generative process is achieved. This is the basic idea of the search algorithms which work on reducing the final clause resulting from the steps above until they reach a clause which gives the highest score amongst the visited clauses.

3.2 Search

The search algorithms aim at finding a clause with the highest score. As the aimed clause is assumed to be generative, two main conditions need to be met:

– All variables are range restricted.
– All switches are ground when they are invoked (this condition is imposed by PRISM).

[3] The idea of approximating the LGG was proposed by Idestam-Almquist [1] in solving ILP problems

Greedy Hill Climbing: The hill climbing search works by choosing the best candidate in the neighbourhood which is constituted by clauses with one literal deleted from the current clause. First it takes the massive clause generated by the steps described in section 3.1, and then start searching by selecting the best choice according the Variational free energy score. Given n literals in the body, hill climbing algorithm finds the best choice by deleting each one in turn, thus it runs VB learning n times and then $n-1$ times and so on. So the cost of the search is $O(n^2)$ (this does not include the cost of the VB learning). Due to the large number of literals at the start of the search, this search strategy is considerably inefficient for big problems.

Random Generation Search: Random generation search takes the output of the steps in section 3.1. It works by building generative clauses randomly. First, it considers the head of the resulting clause, and then it samples on body literal at a time until a generative clause it built. This clause is then scored and the process is repeated a pre-specified number of times. Finally it produces the clause that has the best score. At each sampling stage, the set of body literals that are considered should not include any none ground switch. For example, the literal `msw(s(A),V)` must not be considered for sampling unless a literal that generates `A` has been sampled before in order for the switch `s(A)` to be ground. So the first sampling step works on literals with only ground switches (and other none `msw` predicates). Next, switches that are grounded by the sampled literal are added to the list that the algorithm will sample from, and the process is repeated until a generative clause is produced.

4 Experiment

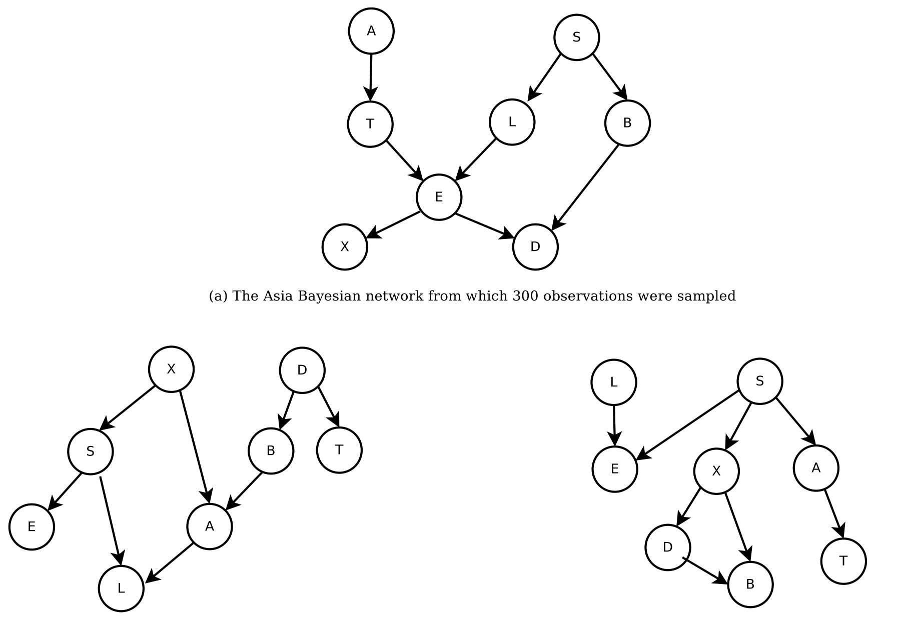
In order to test the learning algorithms, we sampled 300 observations from the PRISM program shown in Table 2 representing the Asia network shown in Figure 1 (a). The observations were fed to the learning algorithm as training data along with the switch declaration part in Table 2 as background knowledge. The learning algorithm is run twice with the two search strategies *greedy hill climbing* and *random generation*. The *random generation* search was set to generates 1000 sample programs and returns the program that with the highest score. The result is shown in Table 1. The *greedy hill climbing* search scored slightly higher than the *random generation* search. However, the time taken by the *greedy hill climbing* search is unsatisfactory. Taken into account that *random generation* search is completely random, this strategy could be improved with some guiding mechanisms.

5 Conclusion and Future Work

Probabilistic Inductive Logic Programming (PILP) is a challenging problem. PRISM is a well developed formalism which generalises probabilistic models

	Variational free energy	**Learning time**
Original Net	-1459	N/A
Net learned by greedy hill climbing	-1464	$>$ 3 hours
Net learned by random generation	-1475	$\approx$ 5 minutes

Table 1. Result of learning a single clause PRISM program encoding a Bayesian network with two search strategies



(a) The Asia Bayesian network from which 300 observations were sampled

(b) The Bayesian network learned with greedy hill climbing

(c) The Bayesian network learned with random generation

Fig. 1. Learning Bayesian network encoded as a PRISM program

(e.g. Hidden Markov Model (HMM), Bayesian network, Stochastic Context-free Grammar (SCFG) . . . etc) and Logic Programming (LP). This paper presents a technique to learn single clause failure-free PRISM programs and shows promising results. However, some challenging problem has yet to be solved. As the presented technique is based on constructing the LGG, though LGG is approximated and not constructed exactly, it could be problematic for big problems due to the combinatoric explosion of the number of literals. This problem needs further investigation.

There are different areas of improvement. One of which is learning recursive definitions as some models cannot be represented without recursive, HMM with variable length for instance. *Predicate Invention (PI)* can also be addressed as in some problems new predicates need to be invented to build the final theory.

```
%Part of the switches declaration which also served as BK in learning Bayesian network:
values(a,[0,1]).          values(s,[0,1]).          values(t_a(0),[0,1]).
values(t_a(1),[0,1]).     values(l_s(0),[0,1]).     values(l_s(1),[0,1]).
values(e_tl(0,0),[0,1]). values(e_tl(0,1),[0,1]).
.
.
%Bayesian network modelling part:
asia(A,T,E,S,L,B,X,D):-
   msw(a,A),
   msw(s,S),
   msw(t_a(A),T),
   msw(l_s(S),L),
   msw(b_s(S),B),
   msw(e_tl(T,L),E),
   msw(d_eb(E,B),D),
   msw(x_e(E),X).
```

Table 2. A PRISM program encoding the Bayesian network where the training data was sampled from. The switch declaration part was used as a background knowledge fed to the learning algorithm.

References

1. Idestam-Almquist, P.: Generalization of Clauses Relative to a Theory. Machine Learning 26(2), 213–226 (Feb 1997)
2. Muggleton, S.: Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming 13(3-4), 245–286 (1995)
3. Muggleton, S., De Raedt, L.: Inductive Logic Programming: Theory and Methods. Journal of Logic Programming 19(20), 629–679 (1994)
4. Sato, T., Kameya, Y.: PRISM: A language for symbolic-statistical modeling. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence. pp. 1330–1339 (1997)
5. Sato, T., Kameya, Y.: New advances in logic-based probabilistic modeling by PRISM. In: De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S.H. (eds.) Probabilistic Inductive Logic Programming, Lecture Notes in Computer Science, vol. 4911. Springer, New York (2008)
6. Sato, T., Kameya, Y., Kurihara, K.: Variational Bayes via propositionalized probability computation in PRISM. Annals of Mathematics and Artificial Intelligence 54(1-3), 135–158 (2008)