Variational Bayes via Propositionalization (extended abstract)

Taisuke SATO, Yoshitaka KAMEYA, and Kenichi KURIHARA

Tokyo Institute of Technology
2-12-1 Ôokayama Meguro-ku Tokyo Japan 152-8552

1 Introduction

In this paper, we propose a unified approach to VB (variational Bayes) [1] in symbolic-statistical modeling via propositionalization[1]. By propositionalization we mean, broadly, expressing and computing probabilistic models such as BNs (Bayesian networks) [2] and PCFGs (probabilistic context free grammars) [3] in terms of propositional logic that considers propositional variables as binary random variables.

Our proposal is motivated by three observations. The first one is that propositionalization, or more precisely *PPC (propositionalized probability computation)*, i.e. probability computation formulated in a propositional setting, has turned out to be general and efficient when variable values are sparsely interdependent. Examples include (discrete) BNs, PCFGs and more generally PRISM [4, 5] which is a probabilistic logic programming language we have been developing that computes probabilities using graphically represented AND/OR boolean formulas. Efficacy of PPC is already proved by the Inside-Outside algorithm in the case of PCFGs and by recent PPC approaches to BNs such as the one by Chavira et al. that exploits 0 probability and CSI (context specific independence) [6]. Mateescu et al. introduced AND/OR search tress which is a propositional representation of bucket trees and revealed that PPC is a general computation machanism for BNs [7].

Second of all, while VB has been around for sometime as a powerful tool for Bayesian modeling [1], it's use is restricted to somewhat simple models such as BNs and HMMs (hidden Markov models) [8] though its usefulness is established through a variety of applications from model selection to prediction. On the other hand it is already proved that VB can be extended to PCFGs and efficiently implementable using dynamic programming [9]. Note that PCFGs are just one class of PPC and much more general PPC is already realized in PRISM. Accordingly if VB is combined with PRISM's PPC, we will obtain VB for general probabilistic models, far wider than BNs and PCFGs.

The last observation is that currently deriving and implementing a VB algorithm is an error-prone time-consuming process. In addition ensuring its correctness beyond PCFGs seems a non-trivial task. However once VB becomes available in PRISM, it will save considerable time and effort. That is, we do not have to derive a new VB algorithm from scratch nor have to implement it. All we have to do is just to write a probabilistic

[1] In this paper, models are assumed to be discrete.

model at predicate level. The rest of work will be carried out automatically in a uniform manner by the PRISM system as is the case with EM learning in PRISM.

Hence introducing VB to PRISM will make Bayesian modeling much less painful and also make exploring order-made Bayesian models much easier. In the following, we first review PPC in PRISM in Section 2 and then derive a variational Bayes algorithm for PRISM in Section 3. Section 4 is conclusion.

2 Propositionalized probability computation in PRISM

PRISM[2] is a Prolog-based modeling language to describe and learn symbolic-statistical models. PRISM programs define distributions in terms of clauses and a probabilistic built_in predicate msw/2, the only probabilistic predicate in PRISM, that simulates probabilistic choices such as coin tossing. $\mathtt{msw}(i, v)$ asserts that sampling a discrete random variable named i returns a value v. i and v are arbitrary (ground) terms. The following PRISM program describes how one's ABO blood type is determined by genes inherited from the parents.

```
values_x(gene,[a,b,o],[0.5,0.2,0.3]).
bloodtype(P) :-   % genotype <X,Y> determins phenotype P
   genotype(X,Y),
   ( X=Y -> P=X ; X=o -> P=Y ; Y=o -> P=X ; P=ab ).
genotype(X,Y) :-  % gene X from father, Y from mother
   msw(gene,X),msw(gene,Y).
```

Fig. 1. ABO blood type program

values_x(gene,[a,b,o],[0.5,0.2,0.3]) declares that msw(gene,X) returns in the logical variable X one of {a, b, o} with probabilities 0.5, 0.2 and 0.3 respectively. In other words we define $P(\mathtt{msw}(\mathtt{gene},\mathtt{a})) = 0.5$ $P(\mathtt{msw}(\mathtt{gene},\mathtt{b})) = 0.2$ and $P(\mathtt{msw}(\mathtt{gene},\mathtt{o})) = 0.3$. These probabilities are called *parameters*. Given a top-goal ?-bloodtype(a), the program is executed exactly like Prolog except that msw(gene,X) returns in X a value probabilistically chosen from {a, b, o}.

To compute the probability $P(\mathtt{bloodtype}(\mathtt{a}))$, bloodtype(a) is reduced to an equivalent AND/OR propositional formula shown in Fig. 2 through SLD search applied to bloodtype(a) using the program in Fig. 1.[3]

$P(\mathtt{bloodtype}(\mathtt{a}))$ is then computed isomorphically to Fig. 2 from the probabilities (parameters) which are declared by values_x/3 predicate as follows.

[2] http://sato-www.cs.titech.ac.jp/prism/.

[3] When atoms contain variables, they are expanded to ground conjunctions during the SLD search. So q(a,Y) is expanded to $\mathtt{q}(\mathtt{a},\mathtt{b}_1) \vee \cdots \vee \mathtt{q}(\mathtt{a},\mathtt{b}_n)$ where $\mathtt{b}_1,\ldots,\mathtt{b}_n$ are appropriate ground terms.

```
bloodtype(a)  <=> genotype(a,a) v genotype(a,o) v genotype(o,a)
genotype(a,a) <=> msw(gene,a) & msw(gene,a)
genotype(a,o) <=> msw(gene,a) & msw(gene,o)
genotype(o,a) <=> msw(gene,o) & msw(gene,a)
```

Fig. 2. Reduced propositional formula for `bloodtype(a)`

$$\begin{aligned} P(\texttt{bloodtype(a)}) &= P(\texttt{genotype(a,a)}) + P(\texttt{genotype(a,o)}) + P(\texttt{genotype(o,a)}) \\ P(\texttt{genotype(a,a)}) &= P(\texttt{msw(gene,a)}) \cdot P(\texttt{msw(gene,a)}) \\ &= 0.5 * 0.5 \\ &\cdots \end{aligned}$$

In general, to compute $P(G)$, the probability of a goal G defined by the distribution semantics of PRISM [5], we first reduce G to a propositional AND/OR formula such that $G \Leftrightarrow E_1 \vee \cdots \vee E_n$[4] ($n \geq 0$) where each E_i ($1 \leq i \leq n$) is a conjunction $\texttt{msw}_1 \wedge \cdots \wedge \texttt{msw}_{k_i}$ ($k_i \geq 0$) of ground `msw` atoms. E_i is called an *explanation* for G. We use $\varphi(G) = \{E_1, \ldots, E_n\}$ for the set of explanations for G. Then $P(G)$ is computed by

$$P(G) = \sum_{i=1}^{n} P(E_i)$$

$$P(E_i) = \prod_{j=1}^{k_i} P(\texttt{msw}_j) \quad \text{where} \quad E_i = \texttt{msw}_1 \wedge \cdots \wedge \texttt{msw}_{k_i}$$

Notice first that we need the probabilities of `msw` atoms (parameters), to compute $P(G)$ but they are declared by `values_x` in a program or learned from data.

Notice second that this way of probability computation is justified only when the exclusiveness of the explanations and the independence of the `msw` atoms in every explanation are guaranteed [5]. They are guaranteed if, roughly speaking, the program describes a *generative process* of possible outcomes in which every choice is probabilistically made by `msw` atoms and there is no failed or infinite computation.

There is another concern. Converting a given goal to an equivalent propositional formula may blows up the size of expressions. In the case of PCFGs for example, there are exponentially many parse trees for one sentence. So if a goal G representing a sentence is reduced to a set $\varphi(G)$ of explanations such that each explanation in $\varphi(G)$ represents a possible probabilistic derivation, the size of $\varphi(G)$ will also be exponential. Fortunately we can suppress the explosion, though not always successful, by reorganizing $\varphi(G)$. We introduce intermediate atoms representing subexpressions of in $\varphi(G)$ and eliminate redundancy by having formulas in $\varphi(G)$ share common subexpressions. The compressed expression, represented as a set propositional AND/OR formulas, is called

[4] This equivalence is two-fold. First both sides have the same truth value and they are equal as binary random variables in terms of the distribution semantics of PRISM [5].

an *explanation graph* for G and denoted by $Expl(G)$. There is more than one way of constructing $Expl(G)$ but we construct it in a top-down manner using tabled search that avoids repetition of the same search [10].

Finally we must point out that `msw` atoms are enough to represent discrete random variables and use their values in a program. Suppose there is a random variable X taking a value $v_i \in V_X = \{v_1, \ldots, v_n\}$ with probability θ_i ($1 \leq i \leq n$, $\theta_1 + \cdots + \theta_n = 1$). Corresponding to X, we declare `values_x("X",[`$v_1, \ldots, v_n$`], [`$\theta_1, \ldots, \theta_n$`])` in the program where "X" is a constant naming X, and introduce n binary random variables `msw("X",`v_i`)` which is true with probability θ_i if-and-only-if $X = v_i$ happens ($1 \leq i \leq n$). We require that those `msw("X",`v_i`)`s are exhausting and exclusive. I.e. `msw("X",`v_1`)` $\vee \cdots \vee$ `msw("X",`v_n`)` and $\neg($`msw("X",`v_i`)` $\wedge$ `msw("X",`v_j`))` ($1 \leq i \neq j \leq n$) hold.

Hence, although only `msw` atoms, binary random variables, are available in PRISM, which does not pose any restriction on probabilistic modeling as long as models are discrete. In addition, the use of `msw` atoms provides not only a uniform way of probability computation by way of explanation graphs but statistical learning such as ML(maximum likelihood) estimation as well [5].

3 Parameter learning and VB in PRISM

In this section, assuming the reader is familiar with VB [1], we explain VB in PRISM.

3.1 ML estimation

Suppose there is a program DB that describes how observable atoms are generated by a series of probabilistic choices made by `msw` atoms. Let `msw(`i`,`$\cdot$`)` be a `msw` predicate used in DB. It corresponds to a random variable whose name is i that takes on a value (term) in the value set $V_i = \{v_1, \ldots, v_{|V_i|}\}$. We call $\theta_{i,v} = P($`msw(`i`,`v`)`$)$ ($v \in V_i$) the *parameter associated with* `msw(`i`,`v`)` and use $\boldsymbol{\theta}_i = \langle v_1, \ldots, v_{|V_i|} \rangle$ to denote the set of parameters associated with `msw(`i`,`$\cdot$`)` and $\boldsymbol{\theta}$ as the set of all parameters in a program.

Let $\varphi(G)$ be the set of explanations for an observed goal G. We do not know which explanation E in $\varphi(G)$ is true. Forgetting for the moment that E is a conjunction of `msw` atoms, we consider E as a hidden variable and G as an observed variable, respectively. Let I be the set of names for `msw` atoms used in a program and V_i ($i \in I$) be the value set for `msw(`i`,`$\cdot$`)`. Denote by $\sigma_{i,v}(E_t)$ the number of occurrences of `msw(`i`,`v`)` in E. Then according to the distribution semantics of PRISM [5], the distribution of E and G is given by

$$P(E, G \mid \boldsymbol{\theta}) = \begin{cases} \prod_{i \in I} \prod_{v \in V_i} \theta_{i,v}^{\sigma_{i,v}(E)} \\ \quad \text{if } E \text{ is an explanation for } G \\ 0 \quad \text{otherwise} \end{cases}$$

Now the problem of parameter learning is stated as follows: Given a sequence $\mathcal{G} = G_1, \ldots, G_T$ of observed goals (repetition possible), estimate parameters $\widehat{\boldsymbol{\theta}}$ associated with `msw` atoms in a program for the above distribution by ML estimation, i.e. $\widehat{\boldsymbol{\theta}} =$

argmax$_{\boldsymbol{\theta}} \prod_{t=1}^{T} P(G \mid \boldsymbol{\theta})$. We solve this estimation problem by using the EM algorithm generalized for logic programs which considers explanations as hidden variables. The ML estimation can be efficiently done by the propositionalized probability computation approach described in Section 2 using dynamic programming [5].

3.2 The graphical VB-EM algorithm

It is well-known that ML suffers the over-fitting problem when there is not enough data. One can avoid this problem by adopting the Bayesian approach that imposes a prior distribution on the set of model parameters. With the same purpose in mind we introduce a Dirichlet distribution $Dir(\alpha_{i,v_1}, \ldots, \alpha_{i,v_{|V_i|}})$ for each parameter set (vector) $\boldsymbol{\theta}_i = \langle \theta_{i,v_1}, \ldots, \theta_{i,v_{|V_i|}} \rangle$ associated with `msw(`i`,`$\cdot$`)`. So the prior distribution is written as

$$P(\theta_{i,v_1}, \ldots, \theta_{i,v_{|V_i|}} \mid \boldsymbol{\alpha}_i) = \frac{1}{Z_i} \prod_{v \in V_i} \theta_{i,v}^{\alpha_{i,v}-1}$$

$$Z_i = \frac{\prod_{v \in V_i} \Gamma(\alpha_{i,v})}{\Gamma\left(\sum_{v \in V_i} \alpha_{i,v}\right)}$$

where $\boldsymbol{\alpha}_i = \langle \alpha_{i,v_1}, \ldots, \alpha_{i,v_{|V_i|}} \rangle$ is the set of hyper parameters for $\langle \theta_{i,v_1}, \ldots, \theta_{i,v_{|V_i|}} \rangle$.

The next task in the Bayesian approach is to integrate out parameters to compute the marginal likelihood but it is intractable. We therefore resort to a deterministic approximation by VB [1]. VB applied to the exponential family yields an iterative algorithm called the *VB-EM algorithm* which is similar to the EM algorithm but iteratively updates hyper parameters.

While (discrete) VB was applied to fairly simple models initially [1], their extensions have been pursued and now it is proved VB is applicable to PCFGs [9]. We further extend VB and generalize the VB-EM algorithm described in [9] to the distribution semantics of PRISM, yielding the following *naive VB-EM* algorithm.

```
1:  procedure naive VB-EM($DB, \mathcal{G}$)
2:  begin
3:     foreach $i \in I, v \in V_i$ do
4:        $\alpha_{i,v}^{(0)} = \alpha_{i,v}$ (initial values);
5:        $m := 0$;
6:     repeat
7:        for $t := 1$ to $T$ do
8:           foreach $E_t \in \varphi(G_t)$ do
9:              $q(E_t \mid G_t) \propto \exp\left(\sum_{i \in I} \sum_{v \in V_i} \sigma_{i,v}(E_t) \left(\Psi(\alpha_{i,v}^{(m)}) - \Psi\left(\sum_{v' \in V_i} \alpha_{i,v'}^{(m)}\right)\right)\right)$
10:    $\alpha_{i,v}^{m+1} = \alpha_{i,v} + \sum_{t=1}^{T} \sum_{E_t \in \varphi(G_t)} q(E_t \mid G_t) \sigma_{i,v}(E_t)$;
11:       $m := m + 1$
12:    until every $\alpha_{i,v}^{(m)} - \alpha_{i,v}^{(m-1)} < \varepsilon$
```

```
13:     foreach $i \in I, v \in V_i$ do
14:         $\alpha^*_{i,v} := \alpha^{(m-1)}_{i,v}$
15: end
```

where $q(E_t \mid G_t)$ is a probability computed by $q(E_t \mid G_t) = \frac{q(E_t,G_t)}{\sum_{E'_t \in \varphi(G_t)} q(E'_t,G_t)}$[5] and $\Psi(x)$ is the digamma function defined by $\Psi(x) = \frac{d}{dx}\Gamma(x)$.

By "naive" we mean this algorithm is mathematically correct but needs computational refinement. This is because there are exponentially many explanations for a goal, and hence it will take exponential time per iteration (see line 8). However put

$$\pi^{(m)}_{i,v} = \exp\left(\Psi(\alpha^{(m)}_{i,v}) - \Psi\left(\sum_{v' \in V_i} \alpha^{(m)}_{i,v'}\right)\right)$$

and rewrite $q(E_t|G_t)$ as follows.

$$\begin{aligned}
q(E_t \mid G_t) &\propto \exp\left(\sum_{i \in I}\sum_{v \in V_i} \sigma_{i,v}(E_t)\left(\Psi(\alpha^{(m)}_{i,v}) - \Psi\left(\sum_{v' \in V_i} \alpha^{(m)}_{i,v'}\right)\right)\right) \\
&= \prod_{i \in I}\prod_{v \in V_i} \exp\left(\left(\Psi(\alpha^{(m)}_{i,v}) - \Psi\left(\sum_{v' \in V_i} \alpha^{(m)}_{i,v'}\right)\right)\right)^{\sigma_{i,v}(E_t)} \\
&= \prod_{i \in I}\prod_{v \in V_i} \left(\pi^{(m)}_{i,v}\right)^{\sigma_{i,v}(E_t)}
\end{aligned}$$

This rewriting suggests us that the quantity $q(E_t \mid G_t)$ can be computed exactly the same way as $p(E_t \mid G_t)$ is computed in PRISM. The only difference is that the model parameters $\theta_{i,v}$ specifying $p(E_t \mid G_t)$ are replaced by the $\pi^{(m)}_{i,v}$[6]. Therefore, just like the graphical EM algorithm is derived from the naive EM algorithm for PRISM [5], we can derive the graphical VB-EM algorithm (*gVB-EM algorithm*), a dynamic programming version of the naive VB-EM algorithm as shown in **Appendix: gVB-EM**. It computes inside and outside values similar to inside-outside probabilities using hyper parameters by dynamic programming, and the time complexity per iteration is proportional to the size of explanation graphs for the input goals.

4 Conclusion

We have derived the gVB-EM algorithm for the distribution semantics of PRISM. It is a generalization of the gEM algorithm used in PRISM for EM learning to discrete Bayesian learning using Dirichlet distributions. We believe that gVB-EM helps us avoid the over-fitting problem and gives us a reliable measure for model selection.

[5] $q(E_t, G_t)$ itself is computed according to the right-hand side of line 9 in the naive VB-EM algorithm.

[6] Note that $\sum_{v \in V_i} \pi^{(m)}_{i,v} = 1$ does not necessarily hold. So we cannot consider $\pi^{(m)}_{i,v}$ as probabilities.

References

1. Ghahramani, Z., Beal, M.: (Graphical models and variational methods)
2. Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann (1988)
3. Wetherell, C.S.: Probabilistic languages: a review and some open questions. Computing Surveys **12** (1980) 361–379
4. Sato, T., Kameya, Y.: PRISM: a language for symbolic-statistical modeling. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97). (1997) 1330–1335
5. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. Journal of Artificial Intelligence Research **15** (2001) 391–454
6. Chavira, M., Darwiche, A.: Compiling Bayesian networks with local structure. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05). (2005) 1306–1312
7. Mateescu, R., Dechter, R.: The relationship between AND/OR search spaces and variable elimination. In: Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI'05). (2005) 380–387
8. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE **77** (1989) 257–286
9. Kurihara, K., Sato, T.: An application of the variational bayesian approach to probabilistic context-free grammars. In: IJCNLP-04 Workshop Beyond shallow analyses. (2004)
10. Zhou, N.F., Sato, T.: Efficient fixpoint computation in linear tabling. In: Proceedings of the 5th ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'03). (2003) 275–283

Appendix: gVB-EM

```
procedure Learn-gVBEM($DB, \mathcal{G}$)
begin
  foreach $i \in I, v \in V_i$ do
    $\alpha^{(0)}_{i,v} = \alpha_{i,v}$ (initial values);
    $m := 0$;
  repeat
    Get-Inside-Values($DB, \mathcal{G}$);
    Get-Outside-Values($DB, \mathcal{G}$);
    Get-Expectations($DB, \mathcal{G}$);
    foreach $i \in I, v \in V_i$ do
      $\alpha^{m+1}_{i,v} = \alpha_{i,v} + \eta[i, v]$;
    $m := m + 1$;
  until every $\alpha^{(m)}_{i,v} - \alpha^{(m-1)}_{i,v} < \varepsilon$
  foreach $i \in I, v \in V_i$ do
    $\alpha^{*}_{i,v} := \alpha^{(m-1)}_{i,v}$
end
```

```
procedure Get-Inside-Values($DB, \mathcal{G}$)
begin
  for $t := 1$ to $T$ do begin
    Let $\tau^t_0 = G_t$;
    for $k := K_t$ downto $0$ do begin
      $\mathcal{P}[t, \tau^t_k] := 0$;
      foreach $\widetilde{S} \in \widetilde{\psi}_{DB}(\tau^t_k)$ do begin
        Let $\widetilde{S} = \{A_1, A_2, \ldots, A_{|\widetilde{S}|}\}$;
        $\mathcal{R}[t, \tau^t_k, \widetilde{S}] := 1$;
        for $l := 1$ to $|\widetilde{S}|$ do
          if $A_l = \mathtt{msw}(i, \cdot, v)$ then
            $\pi^{(m)}_{i,v} = \exp\Big(\Psi(\alpha^{(m)}_{i,v})-$
                $\Psi(\sum_{v' \in V_i} \alpha^{(m)}_{i,v'})\Big)$;
            $\mathcal{R}[t, \tau^t_k, \widetilde{S}] *= \pi^{(m)}_{i,v}$
          else $\mathcal{R}[t, \tau^t_k, \widetilde{S}] *= \mathcal{P}[t, A_l]$;
        $\mathcal{P}[t, \tau^t_k] += \mathcal{R}[t, \tau^t_k, \widetilde{S}]$
      end /* foreach $\widetilde{S}$ */
    end /* for $k$ */
  end /* for $t$ */
end
```

```
procedure Get-Expectations($DB, \mathcal{G}$)
begin
  foreach $i \in I, v \in V_i$ do
    $\eta[i, v] := 0$;
  for $t := 1$ to $T$ do  begin
    Let $\tau^t_0 = G_t$;
    for $k := 0$ to $K_t$
      foreach $\widetilde{S} \in \widetilde{\psi}_{DB}(\tau^t_k)$  do begin
        Let $\widetilde{S} = \{A_1, A_2, \ldots, A_{|\widetilde{S}|}\}$;
        for $l := 1$ to $|\widetilde{S}|$ do
          if $A_l = \mathtt{msw}(i, \cdot, v)$ then
            $\eta[i, v] +=$
              $\mathcal{Q}[t, \tau^t_k] \cdot \mathcal{R}[t, \tau^t_k, \widetilde{S}] / \mathcal{P}[t, G_t]$
      end /*  foreach $\widetilde{S}$ */
  end /*  for $t$ */
end
```

```
procedure Get-Outside-Values($DB, \mathcal{G}$)
begin
  for $t := 1$ to $T$ do begin
    Let $\tau^t_0 = G_t$ ; $\mathcal{Q}[t, \tau^t_0] := 1$;
    for $k := 1$ to $K_t$ do $\mathcal{Q}[t, \tau^t_k] := 0$;
    for $k := 0$ to $K_t$
      foreach $\widetilde{S} \in \widetilde{\psi}_{DB}(\tau^t_k)$  do begin
        Let $\widetilde{S} = \{A_1, A_2, \ldots, A_{|\widetilde{S}|}\}$;
        for $l := 1$ to $|\widetilde{S}|$ do
          if $A_l \neq \mathtt{msw}(i, \cdot, v)$ then
            $\mathcal{Q}[t, A_l] +=$
              $\mathcal{Q}[t, \tau^t_k] \cdot \mathcal{R}[t, \tau^t_k, \widetilde{S}] / \mathcal{P}[t, A_l]$
      end /* foreach $\widetilde{S}$ */
    end /* for $k$ */
  end /* for $t$ */
end
```