TLP **15** (2): 147–168, 2015. © Cambridge University Press 2014
doi:10.1017/S1471068413000677 First published online 28 January 2014

Viterbi training in PRISM

TAISUKE SATO
Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro, Tokyo, Japan
(*e-mail:* sato@mi.cs.titech.ac.jp)

KEIICHI KUBOTA
Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro, Tokyo, Japan
(*e-mail:* kubota@mi.cs.titech.ac.jp)

submitted 10 June 2012; revised 11 December 2012; accepted 7 March 2013

Abstract

VT (Viterbi training), or hard expectation maximization (EM), is an efficient way of parameter learning for probabilistic models with hidden variables. Given an observation y, it searches for a state of hidden variables x that maximizes $p(x, y \mid \theta)$ by coordinate ascent on parameters θ and x. In this paper we introduce VT to PRogramming In Statistical Modeling (PRISM), a logic-based probabilistic modeling system for generative models. VT improves PRISM in three ways. First, VT in PRISM converges faster than EM in PRISM due to VT's termination condition. Second, parameters learned by VT often show good prediction performance compared with those learned by EM. We conducted two parsing experiments with probabilistic grammars while learning parameters by a variety of inference methods, i.e. VT, EM, MAP and VB. The result is that VT achieved the best parsing accuracy among them in both experiments. Also, we conducted a similar experiment for classification tasks where a hidden variable is not a prediction target unlike probabilistic grammars. We found that in such a case VT does not necessarily yield superior performance. Third, since VT always deals with a single probability of a single explanation, Viterbi explanation, the exclusiveness condition imposed on PRISM programs is no more required if we learn parameters by VT. Last but not least, we can say that as VT in PRISM is general and applicable to any PRISM program, it largely reduces the need for the user to develop a specific VT algorithm for a specific model. Furthermore, since VT in PRISM can be used just by setting a PRISM flag appropriately, it makes VT easily accessible to (probabilistic) logic programmers.

KEYWORDS: Viterbi training, PRISM, exclusiveness condition

1 Introduction

Viterbi training (VT) has been used for a long time as an efficient parameter learning method in various research fields such as machine translation (Brown *et al.* 1993), speech recognition (Juang and Rabiner 1990; Strom *et al.* 1999), image analysis (Joshi *et al.* 2006), parsing (Spitkovsky *et al.* 2010) and gene finding (Lomsadze *et al.* 2005). Although VT is NP-hard even for probabilistic context free grammars (PCFGs), which is proved by encoding the 3-SAT problem into PCFGs (Cohen and

Smith 2010), and is biased unlike maximum likelihood estimation (MLE) (Lember and Koloydenko 2007), it often outperforms and runs faster than the conventional expectation maximization (EM) algorithm.

We introduce this VT to PRogramming In Statistical Modeling (PRISM), which is a probabilistic extension of Prolog (Sato and Kameya 2001; Sato and Kameya 2008).[1] There are already multiple parameter learning methods available in PRISM. One is the EM algorithm, or more generally maximum a posteriori (MAP) estimation (Sato and Kameya 2001). Another is variational Bayes (VB) (Sato *et al.* 2009), which approximately realizes Bayesian inference and learns pseudo counts assuming Dirichlet priors over parameters. These are implemented on PRISM's data structure called *explanation graph*s representing AND/OR Boolean formulas made up of probabilistic ground atoms. Probabilities used in EM, MAP and VB are all computed by running the generalized inside–outside algorithm (Sato and Kameya 2001) or its variant on explanation graphs.

Viterbi training in PRISM runs on explanation graphs just like EM, MAP and VB, but always deals with a single probability of a single explanation called *Viterbi explanation* or most probable explanation. Compared with EM that updates only parameters, VT alternately updates the Viterbi explanation and parameters, computing one from the other and *vice versa*, until the Viterbi explanation stops changing. Note that this results in the earlier termination of the algorithm than EM because a small perturbation in parameters does not change the Viterbi explanation whereas it keeps EM running. Actually, we found in our experiments in Section 3 that EM required eight to 15 times more cycles to stop than VT. Also, since VT updates parameters so that they maximize the probability of the Viterbi explanation, it is possible and probable that the final parameters by VT give a higher probability to the Viterbi explanation than those learned by EM. This intuitively explains why VT tends to yield superior performance to EM in prediction tasks such as parsing that computes the Viterbi explanation as a predicted value, as we see in Section 4.

In addition, VT brings about a favorable side effect on PRISM. VT does not require *the exclusiveness condition* imposed on PRISM programs to ensure efficient sum–product probability computation. This is because VT always deals with a single probability of Viterbi explanation, and hence there is no need for summing up probabilities of the non-exclusive explanations. Consequently, PRISM can learn parameters by VT for programs that do not satisfy the exclusiveness condition. We will discuss more about the exclusiveness condition in Section 6.

Viterbi training thus improves PRISM as follows:

- Faster convergence due to a less number of iterations compared with EM.
- Ability to learn parameters good for prediction.
- Elimination of the exclusiveness condition imposed on programs.

From the viewpoint of statistical machine learning and probabilistic logic programming (PLP), we can first say that PRISM generalizes VT. That is, the VT

[1] Viterbi training is available in PRISM2.1. PRISM2.1 is the latest version of PRISM, downloadable from `http://sato-www.cs.titech.ac.jp/prism/`

algorithm implemented in PRISM works for arbitrary probabilistic models described by PRISM, a Turing complete language, including Bayesian networks (BNs), Hidden Markov Models (HMMs) and PCFGs, and hence eliminates the need for the user to derive and implement a specific VT algorithm for a specific model that can be described as a PRISM program. Also, it makes VT easily accessible to probabilistic logic programmers because they can use VT just by setting `learn_mode`, one of PRISM's flags, appropriately. As a result, by switching the `learn_mode` flag he/she can choose the best parameter learning method for their models from EM, MAP, VB and VT, all available in PRISM2.1, without rewriting and adapting their programs to each parameter learning method. Indeed, the exhaustive comparisons among EM, MAP, VB and VT done in our experiments seem quite costly in other environments.

In what follows, we first review PRISM in Section 2 and then explain the basic idea of VT and reformulate it for PRISM in Section 3. We then apply VT to two probabilistic grammars in Section 4 using the ATR corpus, where a hidden variable in a model is a prediction target. In Section 5, we deal with a different situation using a naive Bayes with a hidden variable (NBH) model whose hidden variable is *not* a prediction target. We explain the implication of VT on the exclusiveness condition in Section 6. Section 7 discusses the related work. Section 8 contains the conclusions.

2 Reviewing PRISM

For the self-containedness we review PRISM focusing on its computation mechanism. PRISM is one of the statistical relational learning (SRL)/probabilistic logic learning (PLL) languages (Getoor and Taskar 2007; De Raedt and Kersting 2008) which aim at using rich expressions such as relations and first-order logic for complex probabilistic modeling. It is a probabilistic extension of Prolog enhanced with various built-in predicates for statistical machine learning such as predicates for parameter learning, Viterbi inference, model scoring, Markov Chain Monte Carlo (MCMC) sampling and so on in addition to standard predicates equipped with Prolog.

2.1 Probability naively computed

Syntactically a PRISM program *DB* looks like a usual Prolog program except the use of probabilistic built-in predicate of the form `msw(i,v)` called "multi-valued random switch" (with switch name i) that represents a probabilistic choice using simple probabilistic events such as dice throwing; `msw(i,v)` says that throwing a dice named i yields an outcome v. Let $V_i = \{v_1, \ldots, v_{|V_i|}\}$ be the set of possible outcomes for i. The set $\texttt{msw}(i,\cdot) \stackrel{\text{def}}{=} \{\texttt{msw}(i,v) \mid v \in V_i\}$ of `msw` atoms is given a joint distribution such that one of the $\texttt{msw}(i,\cdot)$'s, say `msw(i,v)`, becomes exclusively true (others false) with probability $\theta_{i,v}$ ($v \in V_i$) where $\sum_{v \in V_i} \theta_{i,v} = 1$. In other words, $\texttt{msw}(i,\cdot)$ stands for a discrete random variable X_i taking v with probability $\theta_{i,v}$ ($v \in V_i$). In this sense we identify $\texttt{msw}(i,\cdot)$ with X_i and its distribution.

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

The $\theta_{i,v}$'s are called *parameter*s associated with i. These are directly specified by the user, or learned from data. We define $P_{\mathtt{msw}}(\cdot \mid \boldsymbol{\theta})$ as an infinite product of such distributions for msws, where $\boldsymbol{\theta}$ stands for the set of all parameters. Then $P_{\mathtt{msw}}(\cdot \mid \boldsymbol{\theta})$ is uniquely extended by way of the least model semantics for logic programs to a σ-additive probability measure $P_{DB}(\cdot \mid \boldsymbol{\theta})$ over possible Herbrand interpretations of DB, which we consider as the denotation of DB (*distribution semantics*) (Sato 1995; Sato and Kameya 2001). In the following we omit $\boldsymbol{\theta}$ when the context is clear for the sake of brevity.

Let G be a non-msw atom which is ground. $P_{DB}(G)$, the probability of G, can be naively computed as follows. First reduce the top-goal G using Prolog's exhaustive top-down proof search to an equivalent propositional DNF formula $\mathrm{expl}_0(G) = \epsilon_1 \vee \cdots \vee \epsilon_k$,[2] [3] where ϵ_i $(1 \leqslant i \leqslant k)$ is a conjunction $\mathtt{msw}_1 \wedge \cdots \wedge \mathtt{msw}_n$ of msw atoms such that $\mathtt{msw}_1 \wedge \cdots \wedge \mathtt{msw}_n, DB \vdash G$. Each ϵ_i is called an *explanation for* G. Then assuming

[Independence condition] msw atoms in an explanation are independent:
$P_{DB}(\mathtt{msw} \wedge \mathtt{msw}') = P_{DB}(\mathtt{msw}) P_{DB}(\mathtt{msw}')$
[Exclusiveness condition] Explanations are exclusive:
$P_{DB}(\epsilon_i \wedge \epsilon_j) = 0$ if $i \neq j$

we compute $P_{DB}(G)$ as

$$\begin{aligned} P_{DB}(G) &= P_{DB}(\epsilon_1) + \cdots + P_{DB}(\epsilon_k) \\ P_{DB}(\epsilon_i) &= P_{DB}(\mathtt{msw}_1) \cdots P_{DB}(\mathtt{msw}_n) \quad \text{for} \quad \epsilon_i = \mathtt{msw}_1 \wedge \cdots \wedge \mathtt{msw}_n. \end{aligned}$$

Recall here that msws with different switch names are independent by construction of $P_{\mathtt{msw}}(\cdot \mid \boldsymbol{\theta})$. We may further assume that msw atoms with the same switch name are independent and identically distributed (iid). That is, when $\mathtt{msw}(i,v)$ and $\mathtt{msw}(i,v')$ occur in a program, we consider they are the results of sampling the same $\mathtt{msw}(i,\cdot)$ twice. This is justified by hypothetically adding an implicit argument, trial-id t (Sato and Kameya 2001) to $\mathtt{msw}(i,\cdot)$ and assuming that $\mathtt{msw}(i,t,\cdot)$s have a product of joint distributions just like the case of msw/2, which makes $\mathtt{msw}(i,t,\cdot)$ and $\mathtt{msw}(i,t',\cdot)$ $(t \neq t')$ iid. So, in what follows we assume that the independence condition is automatically satisfied.

Contrastingly, the exclusiveness condition cannot be automatically satisfied. It needs to be satisfied by the user, for example, by writing a program so that it generates an output solely as a sequence of probabilistic choices made by msw atoms (modulo auxiliary non-probabilistic computation). Although most generative models, including BNs, HMMs and PCFGs, are written this way naturally, but there are models which are unnatural or difficult to write this way (De Raedt *et al.* 2007). Relating to this, observe that *Viterbi explanation*, i.e. the most likely explanation ϵ^* for G, is computed similarly to $P_{DB}(G)$ just by replacing sum with argmax: $\epsilon^* \stackrel{\text{def}}{=} \mathrm{argmax}_{\epsilon \in \mathrm{expl}_0(G)} P_{DB}(\epsilon)$, and does not require the exclusiveness condition to

[2] The equivalence means that G and $\mathrm{expl}_0(G)$ denote the same Boolean random variable in view of the distribution semantics of PRISM.
[3] When convenient, we treat $\mathrm{expl}_0(G)$ as a bag $\{\epsilon_1, \ldots, \epsilon_k\}$ of explanations.

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

compute because it only deals with the probability of a single explanation. We will discuss more about the exclusiveness condition in Section 6.

2.2 *Tabled search, dynamic programming, probability computation and Viterbi inference*

So far our computation is naive. Since there can be exponentially many explanations, naive computation would lead to exponential time computation. PRISM avoids this by adopting tabled search in the exhaustive search for all explanations for the top-goal G and applying dynamic programming to probability computation. By tabling, a goal once called and proved is stored (tabled) in memory with answer substitutions and later calls to the goal return with stored answer substitutions without processing further. Tabling is important to probability computation because tabled goals factor out common sub-conjunctions in $\mathrm{expl}_0(G)$, which results in sharing probability computation for the common sub-conjunctions, thereby realizing dynamic programming, which gives exponentially faster probability computation compared with naive computation.

As a result of exhaustive tabled search for all explanations for G, PRISM obtains a set of propositional formulas called *defining formula*s of the form $H \Leftrightarrow B_1 \vee \ldots \vee B_h$ for every tabled goal H[4] that directly or indirectly calls `msw`s. We call the heads of defining formulas *defined goal*s. Each B_i $(1 \leqslant i \leqslant h)$ is recursively a conjunction $C_1 \wedge \ldots \wedge C_m \wedge \mathtt{msw}_1 \wedge \ldots \wedge \mathtt{msw}_n$ $(0 \leqslant m, n)$ of defined goals $\{C_1, \ldots, C_m\}$ and `msw` atoms $\{\mathtt{msw}_1, \ldots, \mathtt{msw}_n\}$. We introduce a binary relation $H \succ C$ over defined goals such that $H \succ C$ holds if H is the head of some defining formula and C occurs in the body. Assuming "$\succ$" is acyclic, we extend it to a partial ordering over the defined atoms. We denote by $\mathrm{expl}(G)$ the whole set of defining formulas and call $\mathrm{expl}(G)$ the *explanation graph* for G like the non-tabled case.

Once $\mathrm{expl}(G)$ is obtained, since defined goals are layered by the "$\succ$" relation by our assumption where the defining formula in the bottom layer has only `msw`s in the body whose probabilities are known, we can compute probabilities by sum-product operation[5] for all defined goals from the bottom layer upward in a dynamic programming manner in time linear in the size of $\mathrm{expl}(G)$, i.e. the number of atoms appearing in $\mathrm{expl}(G)$.

Compared with naive computation, dynamic programming on $\mathrm{expl}(G)$ can reduce time complexity for probability computation from exponential time to polynomial time. For example, PRISM's probability computation for HMMs takes $O(L)$ time for a given sequence with length L and coincides with the well-known forward–backward algorithm for HMMs. Likewise, PRISM's probability computation for PCFGs takes $O(L^3)$ time for a sentence with length L and coincides with the computation of inside probability for PCFGs. More interestingly, belief propagation (BP), one of the standard algorithms for probability computation for BNs, coincides

[4] The top-goal G is a tabled goal. Tabled goals except the top-goal are called "intermediate goals" in Sato and Kameya (2001) and Zhou *et al.* (2008).

[5] The exclusiveness and independence conditions are inherited from the naive case.

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

with PRISM's probability computation applied to PRISM programs that describe junction trees (Sato 2007).

Viterbi inference, i.e. the computation of the Viterbi explanation and its probability, is similarly performed on expl(G) in a bottom-up manner like probability computation stated above. The only difference is that we use argmax instead of sum. In what follows, we look into how the Viterbi explanation is computed. We use $\boldsymbol{\theta}$ for the set of all parameters. Let H be a defined goal and $H \Leftrightarrow B_1 \vee \ldots \vee B_h$ the defining formula for H in expl(G). Write $B_i = C_1 \wedge \ldots \wedge C_m \wedge \mathtt{msw}(i_1, v_1) \wedge \ldots \wedge \mathtt{msw}(i_n, v_n)$ $(1 \leqslant i \leqslant h)$ and suppose recursively that the Viterbi explanation $\epsilon^*_{C_j}$ $(1 \leqslant j \leqslant m)$ has been already calculated for each defined goal in C_j in B_i. Then the Viterbi explanation $\epsilon^*_{B_i}$ for B_i and the Viterbi explanation ϵ^*_H for H are respectively computed by

$$\begin{aligned} \epsilon^*_{B_i} &= \epsilon^*_{C_1} \wedge \cdots \wedge \epsilon^*_{C_m} \wedge \mathtt{msw}(i_1, v_1) \wedge \ldots \wedge \mathtt{msw}(i_n, v_n) \\ \epsilon^*_H &= \mathrm{argmax}_{B_i} P_{DB}(\epsilon^*_{B_i} \mid \boldsymbol{\theta}) \\ &\qquad \text{where } P_{DB}(\epsilon^*_{B_i} \mid \boldsymbol{\theta}) = P_{DB}(\epsilon^*_{C_1}) \cdots P_{DB}(\epsilon^*_{C_m}) \theta_{i_1, v_1} \cdots \theta_{i_n, v_n}. \end{aligned}$$

Here θ_{i_1, v_1} is a parameter associated with $\mathtt{msw}(i_1, v_1)$ and so on. In this way the Viterbi explanation for the top-goal G is computed in a bottom-up manner by scanning expl(G) once in time linear in the size of expl(G), i.e. exactly the same time complexity as probability computation; for example, $O(L)$ for HMMs and $O(L^3)$ for PCFGs, where L is respectively the length of sequence and that of sentence.

Parameter learning in PRISM, be it EM, MAP, VB or VT (explained next), is based on computation by dynamic programming on expl(G). For example, EM in PRISM computes generalized inside probabilities and generalized outside probabilities for defined goals in expl(G) using dynamic programming and calculates expectations of the number of occurrences of msw atoms in a Selective Linear Definite (SLD) proof for the top-goal to update parameters in each iteration, similar to the inside–outside algorithm for PCFGs (Sato and Kameya 2001). MAP estimation and VB inference are also performed similarly (Sato and Kameya 2008; Sato *et al.* 2009).

3 Viterbi training and PRISM

In this section we adapt VT to the distribution semantics of PRISM and derives the VT algorithm for PRISM.

3.1 Viterbi training

Here we explain the basic idea of VT without assuming specific distributions. Let x be hidden variables, y observed ones and $p(x, y \mid \theta)$ their joint distribution with parameters θ. We assume x and y are discrete. MLE estimates parameters θ from y as the maximizer of the (log) likelihood function $L_{EM}(y \mid \theta)$:

$$L_{EM}(y \mid \theta) \stackrel{\text{def}}{=} \log \sum_x p(x, y \mid \theta).$$

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

In the case of MAP estimation, we add a prior distribution $p(\theta)$ and use $L_{MAP}(y \mid \theta)$ below as an objective function:

$$L_{MAP}(y \mid \theta) \stackrel{\text{def}}{=} \log \sum_x p(x, y \mid \theta)p(\theta).$$

What VT does is similar to MLE and MAP, but it uses a different objective function $L_{VT}(y \mid \theta)$ defined as

$$L_{VT}(y \mid \theta) \stackrel{\text{def}}{=} \log \max_x p(x, y \mid \theta)p(\theta).$$

Viterbi training estimates parameters as the maximizer of $L_{VT}(y \mid \theta)$ by coordinate ascent that alternates the maximization of $\log p(x, y \mid \theta)$ w.r.t. x and the maximization of $\log p(x, y \mid \theta)$ w.r.t. θ:

$$x^{(n)} = \operatorname{argmax}_x \log p(x, y \mid \theta^{(n)}) \quad (1)$$

$$\theta^{(n+1)} = \operatorname{argmax}_\theta \log p(x^{(n)}, y \mid \theta)p(\theta) \quad (2)$$

Starting with appropriate initial parameters $\theta^{(0)}$, VT iterates the above two steps and terminates when $x^{(n+1)} = x^{(n)}$ holds (recall that random variables x and y are discrete). Proving the convergence property of VT is straightforward,

$$\begin{aligned} L_{VT}(y \mid \theta^{(n+1)}) &= \log p\left(x^{(n+1)}, y \mid \theta^{(n+1)}\right) p(\theta^{(n+1)}) \\ &\geqslant \log p\left(x^{(n)}, y \mid \theta^{(n+1)}\right) p(\theta^{(n+1)}) \\ &\geqslant \log p\left(x^{(n)}, y \mid \theta^{(n)}\right) p(\theta^{(n)}) \\ &= L_{VT}\left(y \mid \theta^{(n)}\right) \end{aligned}$$

So $L_{VT}(y \mid \theta^{(n)}) \leqslant L_{VT}(y \mid \theta^{(n+1)}) \leqslant 0$ for every $n = 0, 1, \ldots$ Since $\{L_{VT}(y \mid \theta^{(n)})\}_n$ is a monotonically increasing sequence with an upper bound, it converges as n goes to infinity.

3.2 VT for PRISM

Here we reformulate VT in the context of PRISM. Let *DB* be a PRISM program with parameters $\boldsymbol{\theta}$ and $P_{DB}(\cdot \mid \boldsymbol{\theta})$ a probability measure defined by *DB*. Also, let $\boldsymbol{G} = G_1, \ldots, G_T$ be observed goals, and $\mathrm{expl}(G_t)$ $(1 \leqslant t \leqslant T)$ be the set of all explanations ϵ_t for G_t such that $\epsilon_t, DB \vdash G_t$. $\boldsymbol{G} = G_1, \ldots, G_T$ corresponds to observed variables y and $\epsilon_1, \ldots, \epsilon_T$ corresponds to hidden variables x in $p(x, y \mid \theta)$ respectively in equations (1) and (2) in Section 3.1.

Let `msw(i,·)` be the set of `msw` atoms for a multi-valued random switch i as before that represents a probabilistic choice from a finite set V_i of possible outcomes such that `msw(i,v)` $(v \in V_i)$ becomes exclusively true with probability $\theta_{i,v}$.[6] Since $\sum_{v \in V_i} \theta_{i,v} = 1$ holds, $\boldsymbol{\theta}_i$ is a point in the probability simplex. We put $\boldsymbol{\theta}_i = \{\theta_{i,v}\}_{v \in V_i}$ and $\boldsymbol{\theta} = \bigcup_i \boldsymbol{\theta}_i$ where i ranges over possible switch names.

[6] In PRISM, V_i is declared by `values/2-3` predicate.

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

We introduce as a prior distribution Dirichlet distribution $P_{\mathrm{Dir}}(\boldsymbol{\theta}_i) \propto \prod_{v \in V_i} \theta_{i,v}^{\alpha_{i,v}-1}$ with hyper parameters $\{\alpha_{i,v}\}_{v \in V_i}$ over $\boldsymbol{\theta}_i$ and their product distribution $P_{\mathrm{Dir}}(\boldsymbol{\theta}) \stackrel{\mathrm{def}}{=} \prod_i P_{\mathrm{Dir}}(\boldsymbol{\theta}_i)$. In the following, to avoid the difficulty of zero-probability encountered in parameter learning, we assume *pseudo count* $\delta_{i,v} \stackrel{\mathrm{def}}{=} \alpha_{i,v} - 1 > 0$ and use $\delta_{i,v}$ in place of $\alpha_{i,v}$.

Finally, recall that the *Viterbi explanation* ϵ_t^* for a goal G_t is the most probable explanation for G_t given by

$$\epsilon_t^* \quad = \quad \mathrm{argmax}_{\epsilon_t \in \mathrm{expl}(G_t)} P_{DB}(\epsilon_t \mid \boldsymbol{\theta}). \tag{3}$$

By substituting $\boldsymbol{G} = G_1, \ldots, G_T$ for y and $\epsilon_1, \ldots, \epsilon_T$ for x in the definition of $L_{VT}(y \mid \theta)$, the objective function $L_{VT}(\boldsymbol{G} \mid \boldsymbol{\theta})$ for VT in PRISM is now computed as follows:

$$\begin{aligned}
& L_{VT}(\boldsymbol{G} \mid \boldsymbol{\theta}) \\
&= \log \max_{\epsilon_1 \in \mathrm{expl}(G_1), \ldots, \epsilon_T \in \mathrm{expl}(G_T)} \prod_{t=1}^{T} P_{DB}(\epsilon_t, G_t \mid \boldsymbol{\theta}) P_{\mathrm{Dir}}(\boldsymbol{\theta}) \\
&= \log \prod_{t=1}^{T} \max_{\epsilon_t \in \mathrm{expl}(G_t)} P_{DB}(\epsilon_t \mid \boldsymbol{\theta}) P_{\mathrm{Dir}}(\boldsymbol{\theta}) \\
&= \log \prod_{t=1}^{T} P_{DB}(\epsilon_t^* \mid \boldsymbol{\theta}) P_{\mathrm{Dir}}(\boldsymbol{\theta}) \\
&= \log \prod_{i,v} \theta_{i,v}^{\sum_{t=1}^{T} \sigma_{i,v}(\epsilon_t^*) + \delta_{i,v}} \\
&= \sum_{i,v} \left(\sum_{t=1}^{T} \sigma_{i,v}(\epsilon_t^*) + \delta_{i,v} \right) \log \theta_{i,v}
\end{aligned} \tag{4}$$

where "i, v" ranges over these such that `msw(i,v)` appears in some ϵ_t^* and $\sigma_{i,v}(\epsilon_t^*)$ is the count of `msw(i,v)` in ϵ_t^*.

Likewise, by substituting $\boldsymbol{G} = G_1, \ldots, G_T$ for y and $\epsilon_1, \ldots, \epsilon_T$ for x in equations (1) and (2) respectively and using the definition of ϵ_t^*, we obtain the VT algorithm for PRISM which alternately executes (5) and (6) where $\boldsymbol{\theta}^{(n)}$ stands for the set of parameters $\{\theta_{i,v}^{(n)}\}$ at step n,

$$\epsilon_t^{*(n)} \quad = \quad \mathrm{argmax}_{\epsilon_t \in \mathrm{expl}(G_t)} P_{DB}(\epsilon_t \mid \boldsymbol{\theta}^{(n)}) \quad (1 \leqslant t \leqslant T) \tag{5}$$

$$\theta_{i,v}^{(n+1)} \quad \propto \quad \sum_{t=1}^{T} \sigma_{i,v}(\epsilon_t^{*(n)}) + \delta_{i,v} \tag{6}$$

Here (5) corresponds to (1) and (6) to (2) respectively.

Using (5) and (6), VT in PRISM is performed as follows. Given observed goals $\boldsymbol{G} = G_1, \ldots, G_T$, we first perform tabled search for all explanations to build explanation graphs $\mathrm{expl}(G_t)$ for each t $(1 \leqslant t \leqslant T)$. Then starting from the initial parameters $\boldsymbol{\theta}^{(0)}$, we repeat (5) and (6) alternately while computing the Viterbi explanations

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

Table 1. *Average number of iterations and learning time for convergence*

	Iterations		Learning time (sec)	
	VT	EM	VT	EM
PCFG	8.10 (2.28)	123.6 (3.23)	0.45 (0.11)	6.29 (0.16)
PLCG	15.80 (4.73)	144.2 (43.51)	1.55 (0.36)	11.686 (3.64)

$\epsilon_t^{*(n)}$ in (5) by dynamic programming over expl(G_t) as explained in Section 2 until $\epsilon_t^{*(n+1)} = \epsilon_t^{*(n)}$ holds for all t $(1 \leqslant t \leqslant T)$. $\{\theta_{i,v}^{(n+1)}\}$ are then learned parameters.

Having derived the VT algorithm for PRISM, we examine the effect of the termination condition $\epsilon_t^{*(n+1)} = \epsilon_t^{*(n)}$ $(1 \leqslant t \leqslant T)$ on the convergence of VT. As we have remarked in Section 1, this condition means that VT terminates as soon as the Viterbi explanations converge, i.e. there is no change in the Viterbi explanations between steps n and $n + 1$ whereas EM always runs until the convergence of parameters. As a result, since a small change of parameters does not affect the Viterbi explanation but keeps EM running, VT tends to converge in much less number of iterations than EM.

To empirically check this, we conducted parameter learning of probabilistic grammars by VT and EM using PRISM and compared their convergence behavior.[7] We used two probabilistic grammars, PCFG and probabilistic left-corner grammar (PLCG), for the ATR corpus (Uratani *et al.* 1994) (their details are described in the next section), and measured the average number of iterations and learning time[8] required for convergence over 10 runs. Table 1 summarizes the results with standard deviations in parentheses.

Looking at the table, we see that VT required only a small number of iterations to converge compared with EM; the ratio of average number of iterations of VT to EM is 1:15.2 w.r.t. the PCFG and 1:8.3 w.r.t. the PLCG. We also note that the ratio of average learning time[9] is similar to that of iterations: 1:13.8 w.r.t. the PCFG and 1:7.4 w.r.t. the PLCG. It therefore seems natural to conclude that VT learns parameters with much less number of iterations, and thereby much faster than EM.[10]

Since VT is a local maximizer, it is sensitive to the initial condition such as EM. So we need to carefully choose $\boldsymbol{\theta}^{(0)}$. Uniform distributions for $\boldsymbol{\theta}^{(0)}$ (Spitkovsky *et al.*

[7] All experiments in this paper are done on a single machine with Core i7 Quad 2.67 GHz×2 CPU and 72-GB RAM running OpenSUSE 11.2 using PRISM2.1.

[8] We used a built-in predicate `prism_statistics(em_time,x)` to measure learning time, which returns in x time used by the learning algorithm.

[9] Learning time displayed by the PRISM system after `learn` is "total learning time" which includes search time for explanations and other overhead time such as copying `msw`s in the memory, in addition to actual learning time reported by `prism_statistics(em_time,x)`. Since such extra-time accounts for a large percentage of total learning time, it can happen that the difference in total learning time between EM and VT is smaller than that in Table 1.

[10] In the table, the difference of VT and EM in the number of iterations is statistically significant for both grammars by unpaired t-test at the 5% significance level with the Bonferroni correction. This applies to learning time as well.

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

```
values('S',[['S','S'],[a],[b]],[0.4,0.3,0.3]).

pcfg(L):- pcfg(['S'],L,[]).
pcfg([A|R],L0,L2):-
  ( get_values(A,_) ->  % msw(A,_) exists, so
      msw(A,RHS),       % A is a nonterminal
      pcfg(RHS,L0,L1)
  ; L0=[A|L1] ),
  pcfg(R,L1,L2).
pcfg([],L,L).
```

Fig. 1. A PCFG program.

2010) and $\epsilon_t^{*(0)} (1 \leqslant t \leqslant T)$ (Cohen and Smith 2010) are possible choices. In practice, we further add random restart to alleviate the sensitivity problem. For example, in the experiments in the next section, we repeated parameter learning 50 times with random restart for each learning and selected the parameter set giving the largest value of the objective function $L_{VT}(\boldsymbol{G} \mid \boldsymbol{\theta})$ computed by (4).

4 Learning experiments with probabilistic grammars

In this section we apply VT to parsing tasks in natural language processing where observable variables are sentences and hidden variables are parse trees. We predict parse trees for given sentences using probabilistic grammars (PCFG and PLCG) whose parameters are learned by VT and compare the parsing performance with EM, MAP and VB.[11]

4.1 VT for PCFGs

Prior to describing the parameter learning experiment with a PCFG by VT, we briefly review how to write PCFGs in PRISM. In PCFGs, sentence derivation is carried out probabilistically. When there are k PCFG rules $\theta_1 : A \to \beta_1, \ldots, \theta_k : A \to \beta_k$ for a nonterminal A with probabilities $\theta_1, \ldots, \theta_k$ $(\theta_1 + \cdots + \theta_k = 1)$, A is expanded by $A \to \beta_i$ into β_i with probability θ_i. The probability of a parse tree τ is the product of probabilities associated with occurrences of CFG rules in τ, and the probability of a sentence is the sum of probabilities of parse trees for the sentence.

Writing PCFG programs is easy in PRISM. Figure 1 is a PRISM program for a PCFG { `0.4:S→S S, 0.3:S→a, 0.3:S→b` }. In general, PCFG rules such as $\{\theta_1 : A \to \beta_1, \ldots, \theta_k : A \to \beta_k\}$ are encoded by `values/3` declaration as

$$\texttt{values('A',[}\beta_1,\ldots,\beta_k\texttt{],[}\theta_1,\ldots,\theta_k\texttt{])}$$

where β_i $(1 \leqslant i \leqslant k)$ is a Prolog list of terminals and nonterminals.

[11] We assume, the reader is familiar with the basics of parsing theory.

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

We wrote a PCFG program as shown in Figure 1 for the ATR corpus (Uratani *et al.* 1994) using an associated CFG.[12] The corpus contains labeled parse trees for 10,995 Japanese sentences whose average length is about 10. The associated CFG[13] comprises 861 CFG rules (168 non-terminals and 446 terminals) and on average yields 958 parses/sentences. We applied four learning algorithms, i.e. VT, EM, MAP and VB (Sato *et al.* 2009) available in PRISM2.1 to the PCFG program for the ATR corpus[14] and compared the performance of VT with other learning methods.

We conducted eight-fold cross-validation (CV) for each algorithm[15] to evaluate the quality of learned parameters in terms of three performance metrics, i.e. labeled tree (LT), bracketed tree (BT) and zero crossing brackets (0-CB) (Goodman 1996). These metrics are computed from T_c, the set of parse trees in a test corpus which are considered correct and T_g, the set of parse trees predicted for sentences in the test corpus by a parsing algorithm. LT is defined as $|T_c \cap T_g|/N$, where $|S|$ denotes the number of elements in a set S and $N = |T_g| = |T_c|$. It is the ratio of correctly predicted labeled parse trees to the total number of labeled parse trees. Compared with LT, BT is a less strict metric that ignores nonterminals in parse trees. Let T'_g be the set of unlabeled trees obtained by removing nonterminals from T_g which coincide with the corresponding unlabeled trees in T_c. Then BT is defined as $|T'_g|/N$. Finally, 0-CB is the least strict metric in the three metrics. We say brackets $(w_i, \ldots, w_j)$ in a tree τ is inconsistent with another tree τ' if τ' contains brackets $(w_s, \ldots, w_t)$ such that $s < i \leqslant t < j$ or $i < s \leqslant j < t$, otherwise they are consistent with τ'. Let T''_g be the set of trees in T_g which have no inconsistent brackets with the corresponding trees in T_c. Then 0-CB is given by $|T''_g|/N$.

To perform CV, the entire corpus is partitioned into eight sections. In each fold, one section is used as a test corpus and sentences in the remaining sections are used as training data. For each of EM, MAP, VT and VB, parameters (or pseudo counts) are learned from the training data. A parse tree is predicted, i.e. the Viterbi explanation is computed for each sentence in the test corpus using learned parameters or using the approximate posterior distribution learned by VB. The predicted trees are compared with answers, i.e. the labeled trees in the test corpus to compute LT, BT and 0-CB respectively. The final performance figures are calculated as averages over eight folds and summarized in Table 2 with standard deviations in parentheses.

[12] In the experiment, to speed up parsing, we partially evaluated the PCFG program with individual CFG rules and used the resulting specialized program.

[13] Copyrights protected.

[14] In PRISM, EM is a special case of MAP inference. We used random but almost uniform initialization of parameters and set uniformly pseudo counts $\delta_{i,v}$ to 1.0^{-9} for EM and 1.0 for MAP and VT, respectively. Similarly, we uniformly set hyper parameters $\alpha_{i,v}$ to 1.0 for VB. The number of candidates for re-ranking in VB (Sato *et al.* 2009) was set to 5.
In all the cases, we set the number of random restart to 50 and used the best parameter set that gave the largest value of objective functions, i.e. L_{EM} for EM, L_{MAP} for MAP and L_{VT} for VT. For the case of VB that learns pseudo counts, we chose the best set of pseudo counts giving the highest free energy (Sato *et al.* 2009).

[15] We chose eight-fold CV for parallel execution of learning by our machine.

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

Table 2. *Parsing performance by PCFG*

Metric	Learning method			
	VT	EM	MAP	VB
LT (%)	**74.69** (0.87)	70.02 (0.88)	70.31 (1.13)	72.13 (1.10)
BT (%)	**77.87** (0.84)	73.10 (1.01)	73.45 (1.20)	75.46 (1.13)
0-CB (%)	83.78 (0.92)	84.44 (0.89)	84.89 (0.84)	**87.08** (0.87)

We statistically analyzed the parsing performance by Dunnett's test.[16] The result is that VT outperformed EM, MAP and VB in terms of LT and BT at the 5% level of significance, but not in terms of 0-CB. This is understandable if we assume that there are many parse trees that can give high scores in terms of less restrictive metrics such as 0-CB, but since VT concentrates probability mass on a single tree, those promising trees are allocated little probability mass by VT, which results in relatively low performance of VT in terms of 0-CB.

So far we have examined parsing performance by parameters obtained from incomplete data (sentences in the corpus). We also examined parsing performance using 8-fold CV by parameters learned from complete data, i.e. by parameters obtained by counting occurrences of CFG rules in the corpus. The result is LT: 79.06% (1.25), BT: 85.28% (0.69), 0-CB: 95.37% (0.26) (figures in parentheses are standard deviations). These figures are considered as the best possible performance. We note that the gap in parsing performance between the complete data case and the incomplete data case tends to become wider as the performance metric gets less restrictive in the order of LT, BT and 0-CB.

Another thing to note is that the objective functions for EM, MAP and VB are similar in the sense that they all sum out hidden variables whereas the objective function for VT retains them. This fact together with Figure 2 seems to suggest that parsing performance is more affected by the difference among objective functions than the difference among learning methods.

4.2 VT for PLCGs

Probabilistic context free grammars assume top-down parsing. Contrastingly, there is a class of probabilistic grammars based on bottom-up parsing for CFGs called probabilistic left-corner grammars (PLCGs) (Manning 1997; Roark and Johnson 1999; Van Uytsel *et al.* 2001). Although they use the same set of CFG rules as PCFGs, they attach probabilities not to expansion of nonterminals but to three elementary operations in bottom-up parsing, i.e. shift, attach and project. As a result they define a different class of distributions from PCFGs. In this section we conduct an experiment for parameter learning of a PLCG by VT.

[16] We used Dunnett's test for multiple comparisons of means with VT as the control to avoid inflating the significance level. Figures in bold indicate the best performance.

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

```
values(lc('S','S'),[rule('S',['S','S'])]).
values(lc('S',a),[rule('S',[a])]).
values(lc('S',b),[rule('S',[b])]).
values(first('S'),[a,b]).
values(att('S'),[att,pro]).

plcg(L):- g_call(['S'],L,[]).

g_call([],L,L).
g_call([G|R],[Wd|L],L2):-
   ( G = Wd -> L1 = L     % shift operation
   ; msw(first(G),Wd),lc_call(G,Wd,L,L1) ),
   g_call(R,L1,L2).

lc_call(G,B,L,L2):-       % B-tree is completed
   msw(lc(G,B),rule(A,[B|RHS2])),
   ( G = A -> true ; values(lc(G,A),_) ),
   g_call(RHS2,L,L1),     % complete A-tree
   ( G = A -> att_or_pro(A,Op),
     ( Op = att -> L2 = L1 ; lc_call(G,A,L1,L2) )
   ; lc_call(G,A,L1,L2) ).

att_or_pro(A,Op):-
   ( values(lc(A,A),_) -> msw(att(A),Op) ; Op=att ).
```

Fig. 2. A PLCG program.

The objective of this section is two-fold. One is to apply VT to a PLCG, which, as far as we know, is not attempted before, and to examine the parsing performance. The other is to empirically demonstrate the universality of our approach to VT that subsumes differences in probabilistic models as differences in explanation graphs and applies a single VT algorithm to the latter.

Programs for PLCGs look very different from those for PCFGs. Figure 2 is a PLCG program, a dual version of the PCFG program in Figure 1 with the same underlying CFG {S→S S, S→a, S→b}. It generates sentences using the first set of `'S'` and the left-corner relation for this CFG (`values/2` there only declares the space of outcomes). The program works as follows: Suppose nonterminals `G` and `B` are in the left-corner relation and `G` is waiting for a `B`-tree, i.e. a subtree with the root node labeled `B`, to be completed. When a `B`-tree is completed, the program probabilistically chooses a CFG rule of the form $\mathtt{A} \rightarrow \mathtt{B}\beta$ to further grow the `B`-tree using this rule. Upon the completion of the `A`-tree and if $\mathtt{G} = \mathtt{A}$, the attach operation or the projection is probabilistically chosen. By replacing `values` declarations appropriately, this program is applicable to any PLCG.

We have developed a PLCG program similarly to the PCFG program for the ATR corpus and applied VT, EM, MAP and VB to it to learn parameters. We measured parsing performance by learned parameters in terms of LT, BT and 0-CB by eight-fold CV for each of VT, EM, MAP and VB as shown in Table 3 (standard deviations in parentheses). We compared the parsing performance of VT with EM, MAP and VB by Dunnett's test at the 5% level of significance similar to the PCFG

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

Table 3. *Parsing performance by PLCG*

Metric	Learning method			
	VT	EM	MAP	VB
LT (%)	**76.26** (0.96)	71.81 (0.91)	71.17 (0.93)	71.15 (0.90)
BT (%)	**78.86** (0.70)	75.17 (1.15)	74.28 (1.12)	74.28 (1.00)
0-CB (%)	**87.45** (1.00)	86.49 (0.97)	86.03 (0.67)	86.04 (0.71)

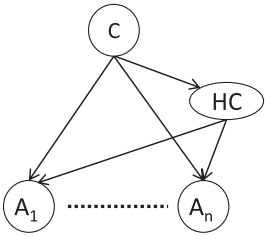

Fig. 3. Bayesian network for an NBH model.

case. This time, however, VT outperformed all of EM, MAP and VB by all metrics, i.e. LT, BT and 0-CB.

5 Applying VT to classification tasks

In the previous section, we conducted learning experiments with PCFG and PLCG in which the prediction target was parse trees that coincide with a hidden variable in a probabilistic model. In this section, we deal with a different situation where a prediction target differs from a hidden variable. We apply VT to classification tasks using an NBH model whose hidden variable is summed out and instead an observable variable, class label, is predicted for the given data.

Before explaining classification tasks, we review NBH for completeness (Sato 2011). NBH is an extension of NB (naive Bayes) with a hidden class variable HC as illustrated in Figure 3. It defines a joint distribution

$$P(A_1,\ldots,A_n,HC,C \mid \boldsymbol{\theta}) = \prod_{j=1}^{n} P(A_j \mid HC,C,\boldsymbol{\theta})P(HC \mid C,\boldsymbol{\theta})P(C \mid \boldsymbol{\theta})$$

where $\boldsymbol{\theta}$ is model parameters, A_j's are attributes of the observed data,[17] C is a class and HC is a hidden class. It is easily seen from the equation (7) that NBH represents data distribution in class C as a mixture of data distributions indexed by HC.

$$\begin{aligned} P(A_1,\ldots,A_n \mid C,\boldsymbol{\theta}) &= \sum_{HC} P(A_1,\ldots,A_n \mid HC,C,\boldsymbol{\theta})P(HC \mid C,\boldsymbol{\theta}) \\ &= \sum_{HC}\prod_{j=1}^{n} P(A_j \mid HC,C,\boldsymbol{\theta})P(HC \mid C,\boldsymbol{\theta}) \end{aligned} \tag{7}$$

[17] We interchangeably use the attributes $A_1,\ldots,A_n$ as data when the context is clear.

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

```
values(class,[democrat,republican]). % class labels are democrat or republican
values(attr(_A,_C,_HC),[y,n]).       % attribute values are y or n

nbayes(C,Vals):-
   msw(class,C),msw(hclass(C),HC),nbh(1,C,HC,Vals).
nbh(J,C,HC,[V|Vals]):-
   ( V == '?' -> msw(attr(J,C,HC),_)  % '?' indicates missing value
   ; msw(attr(J,C,HC),V) ),
   J1 is J+1,
   nbh(J1,C,HC,Vals).
nbh(_,_,_,[]).
```

Fig. 4. An NBH program.

The role of HC is to cluster data in a class C so that a distribution $P(A_1,\ldots,A_n \mid HC,C,\boldsymbol{\theta})$ in each cluster HC satisfies the independent condition $P(A_1,\ldots,A_n \mid HC,C,\boldsymbol{\theta}) = \prod_{j=1}^{n} P(A_j \mid HC,C,\boldsymbol{\theta})$ imposed on NB as much as possible. NBH was introduced in Sato (2011) as a simple substitute for more complicated variants of NB such as TAN (Friedman *et al.* 1997), AODE (Webb *et al.* 2005), BNC (Castillo and Gama 2005), FBC (Su and Zhang 2006) and HBN (Jiang *et al.* 2009).

Given data $A_1,\ldots,A_n$, we classify $A_1,\ldots,A_n$ as a class C^* by

$$\begin{aligned} C^* &= \operatorname{argmax}_C P(C \mid A_1,\ldots,A_n,\boldsymbol{\theta}) \\ &= \operatorname{argmax}_C \sum_{HC} P(C \mid HC,A_1,\ldots,A_n,\boldsymbol{\theta}) P(HC \mid A_1,\ldots,A_n,\boldsymbol{\theta}). \end{aligned} \tag{8}$$

Note here that the hidden variable, HC, is *not* a prediction target unlike probabilistic grammars. It is just summed out. However, we expect that a sub-classifier $P(C \mid HC,A_1,\ldots,A_n,\boldsymbol{\theta})$ indexed by HC performs better than $P(C \mid A_1,\ldots,A_n,\boldsymbol{\theta})$, the original NB, in each cluster and so does their mixture (see equation (8)).

To evaluate the quality of parameters learned by VT for classification tasks, we conducted a learning experiment with NBH using 10 data sets from the UCI Machine Learning Repository (Bache and Lichman 2013). In the experiment training data are given as a set of tuples $C,A_1,\ldots,A_n$ consisting of a class C and attributes $A_1,\ldots,A_n$. Parameters (or pseudo counts) are learned by a PRISM program shown in Figure 4,[18] which is also used for predicting class labels in the test data. A `values/2` declaration `values(class,[democrat,republican])` in the program tells PRISM to introduce two `msw` atoms, `msw(class,democrat)` and `msw(class,republican)` that represent a probabilistic choice between `democrat` and `republican` as a `class`, implicitly together with their parameters $\theta_{class,democrat}$ and $\theta_{class,republican}$ such that $\theta_{class,democrat} + \theta_{class,republican} = 1$. This program assumes that attributes are numbered and missing values in a data set are replaced with `'?'`.

We obtained the classification accuracy of NBH for each combination of data set, learning method (VT, EM, MAP, VB), the number of clusters $\#HC$ in a class

[18] This program is for the vote data set from the repository.

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

Table 4. *Accuracy by VT, EM, MAP and VB*

Data set	Size	NB	NBH: Learning method			
		EM (%)	VT (%)	EM (%)	MAP (%)	VB (%)
nursery	12,960	90.23	92.93	99.40	**99.65**	97.45
mushroom	8,124	99.57	**100.00**	**100.00**	**100.00**	99.99
kr-vs-kp	3,196	87.86	88.69	91.59	**92.34**	88.90
car	1,728	85.86	90.97	97.67	**97.82**	94.68
votes	435	90.29	96.00	95.66	**96.51**	96.05
dermatology	336	97.73	97.98	97.51	98.06	**98.17**
glass	214	72.82	75.86	**76.84**	76.66	76.53
iris	150	94.40	95.07	**95.13**	95.07	95.07
breast cancer	150	72.52	72.52	70.07	72.76	**72.83**
zoo	101	95.07	96.55	**97.42**	96.95	96.62

C (from 2 to 15) and hyper parameters ($\{0.1,1.0\}$ as $\alpha_{i,v}$ for VB, and the same as pseudo counts $\delta_{i,v}$ for VT, MAP) as the average over 10 times ten-fold CV[19] except nursery, mushroom and kr-vs-kp data sets in which case ten-fold CV was used. We similarly obtained the classification accuracy of NB as baseline.[20]

Table 4 summarizes classification accuracies of NB and NBH. Accuracy for NBH in the table is the best accuracy obtained by varying $\#HC$ and hyper parameters as we mentioned for the given learning method and data set. Figures in bold indicate the best accuracy achieved in each data set. The table shows that for most data sets NBH performed better than NB as we expected. Actually the difference in accuracy between NB and the best one for NBH is statistically significant by unpaired t-test at the 5% level with the Bonferroni correction[21] for all data sets except dermatology, iris and breast cancer. The superiority of NBH over NB demonstrated in this experiment is interpreted as an effect of clustering in a class by introducing a hidden variable HC.

Comparing classification accuracies by four parameter learning methods applied to NBH, we note that VT's performance is comparable with the other three, i.e. EM, MAP and VB except for the case of nursery, kr-vs-kp and car data sets. For these data sets VT's accuracy is worse than the best one achieved by one of the three learning methods, which is statistically confirmed by unpaired t-test at the 5% level of significance with the Bonferroni correction. So from the viewpoint of a learning experiment with NBH, we cannot say that, regrettably, VT outperformed EM, MAP and VB for all data sets. However, the result is understandable if we recall that while the predication target in the experiment is a class variable C, VT optimizes parameters not for C but for the hidden variable HC which is summed out and hence only indirectly affects prediction.

[19] We used 10 times ten-fold CV when possible to have robust estimates, although computationally expensive (Japkowicz and Shah 2011).
[20] We used the EM algorithm for parameter learning of NB as there are missing data in some data sets.
[21] As 10 data sets are used, the significance level is set to $5\%/10 = 0.5\%$.

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

```
values(d_e(1,2),[on,off],[0.9,0.1]). values(d_e(2,3),[on,off],[0.8,0.2]).
values(d_e(3,4),[on,off],[0.6,0.4]). values(d_e(1,6),[on,off],[0.7,0.3]).
values(d_e(2,6),[on,off],[0.5,0.5]). values(d_e(6,5),[on,off],[0.4,0.6]).
values(d_e(5,3),[on,off],[0.7,0.3]). values(d_e(5,4),[on,off],[0.2,0.8]).

d_e(1,2):- msw(d_e(1,2),on). d_e(2,3):- msw(d_e(2,3),on).
d_e(3,4):- msw(d_e(3,4),on). d_e(1,6):- msw(d_e(1,6),on).
d_e(2,6):- msw(d_e(2,6),on). d_e(6,5):- msw(d_e(6,5),on).
d_e(5,3):- msw(d_e(5,3),on). d_e(5,4):- msw(d_e(5,4),on).

path(X,Y) :- path(X,Y,[X]).

path(X,X,_).
path(X,Y,A):- X\==Y, (d_e(X,Z) ; d_e(Z,X)), absent(Z,A), path(Z,Y,[Z|A]).

absent(_,[]).
absent(X,[Y|Z]):- X\==Y, absent(X,Z).
```

Fig. 5. A graph program violating the exclusiveness condition.

6 Removing the exclusiveness condition

PRISM assumes the exclusiveness condition on programs to simplify probability computation as explained in Section 2. It means, we cannot write a program clause $H \Leftarrow B \vee B'$ unless $P_{DB}(B \wedge B' \mid \boldsymbol{\theta}) = 0$ is guaranteed (Sato and Kameya 2001). Although most of generative probabilistic models such as BNs, HMMs and PCFGs are naturally described as PRISM programs satisfying the condition, removing it certainly gives us more freedom of probabilistic modeling. Theoretically, it is possible to remove it by introducing binary decision diagrams (BDDs) as ProbLog (De Raedt *et al.* 2007; Kimmig *et al.* 2008) and PITA (Riguzzi and Swift 2011) do, and their related systems, LeProbLog (Gutmann *et al.* 2008), LFI-ProbLog (Gutmann *et al.* 2011) and EMBLEM (Bellodi and Riguzzi 2012), offer parameter learning based on probability computation by BDDs, although with different learning frameworks from PRISM. If, however, we are only interested in obtaining the Viterbi explanation after parameter learning as we were in many cases, VT gives us a way of doing it without BDDs even for programs that do not satisfy the exclusiveness condition. This is because VT does not require the exclusiveness condition to execute equations (5) and (6) that always deal with a single explanation and a single probability.

We next give an example of parameter learning by VT followed by the computation of the Viterbi explanation for a program that violates the exclusiveness condition. Figure 5 is a PRISM program translated from a ProbLog program[22] that computes a path between two nodes (and its probability) in a graph. The graph has six nodes. Edges are assigned probabilities and we express this fact by attaching an `msw` atom to an atom `d_e(`x`,`y`)` representing an edge $x - y$ in the program. For example, (directed) edge `d_e(1,2)` between nodes 1 and 2 is assigned probability `0.9` as indicated by `msw(d_e(1,2),on)` following its value declaration `values(d_e(1,2),[on,off],[0.9,0.1])` in the program.

[22] The program is taken from the tutorial at `http://dtai.cs.kuleuven.be/problog/`

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

```
?- viterbif(path(1,4),P,_X),viterbi_switches(_X,VE)
P = 0.432
VE = [msw(d_e(1,2),on),msw(d_e(2,3),on),msw(d_e(3,4),on)]

?- set_prism_flag(learn_mode,ml_vt).

?- learn([path(1,4),path(1,3),path(2,4),path(2,5),path(3,6)]).
...

?- viterbif(path(1,4),P,_X),viterbi_switches(_X,VE)
P = 0.104
VE = [msw(d_e(1,6),on),msw(d_e(6,5),on),msw(d_e(5,4),on)]
```

Fig. 6. A sample session.

Observe that a ground top-goal `path(X,Y)` causes a call to `d_e(X,Z)` with `X` ground and `Z` free that calls more than one clause, which leads to the violation of the exclusiveness condition. Nonetheless, we can learn parameters by VT and compute the Viterbi explanation for this program.

Figure 6 is a sample session doing this. In this figure we first compute the Viterbi path `VE`, i.e. the most probable path between nodes 1 and 4 and its probability `P` by applying the built-in predicate `viterbif/3` to goal `path(1,4)`.[23] We next renew parameters by learning them using VT from observed goals `path(1,4)`,`path(1,3)`...[24] Finally, we compute the Viterbi path again that is determined by learned parameters and see whether the learning changes the Viterbi path or not. In the session, the Viterbi path changed after learning from `1 -> 2 -> 3 -> 4` to `1 -> 6 -> 5 -> 4` together with their probabilities from 0.432 to 0.161.

Viterbi training thus enables us to learn parameters from programs that violate the exclusiveness condition. However, we have to recall at this point that VT has an objective function different from likelihood and is biased (Lember and Koloydenko 2007), and the effect of removing the exclusiveness condition on the quality of parameters estimated by VT is unknown at the moment, which remains as a future research topic.

7 Related work and discussion

Viterbi training is closely related to K-means (MacQueen 1967), which is a standard clustering method for continuous data. If we apply VT to a Gaussian mixture for the clustering of continuous data with an assumption of a common variance to all composite Gaussian distributions, the resulting algorithm is identical to K-means. In this sense, the usefulness of VT is established. Actually, VT has been used in

[23] `viterbif(path(1,4),P,_X)` returns the Viterbi explanation in `_X` and its probability in `P`. `viterbi_switches(_X,VE)` extracts the Viterbi path `VE` as a conjunction of `msw` atoms.

[24] `set_prism_flag(learn_mode,ml_vt)` tells the PRISM system to use VT when `learn/1` is invoked.

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

various settings (Juang and Rabiner 1990; Brown *et al.* 1993; Strom *et al.* 1999; Lomsadze *et al.* 2005; Joshi *et al.* 2006; Spitkovsky *et al.* 2010) and also in the SRL frameworks that deal with structured data (Singla and Domingos 2005; Huynh and Mooney 2010) where the algorithmic essence of VT, coordinate ascent on parameters and target variables with argmax operation applied to the latter are used.

Despite its popularity, however, it seems that VT so far has been model-specific and only model-specific VT algorithms have been implemented. To our knowledge, in this paper we have given a unified treatment to VT for discrete models for the first time, and derived the VT algorithm for PRISM which is a single generic algorithm applicable to any discrete model as long as the model is described by a PRISM program. Since our derivation of VT is based on the reduction of goals to AND-OR propositional formulas, it seems quite possible for other logic-based modeling languages that use BDDs such as ProbLog (De Raedt *et al.* 2007; Kimmig *et al.* 2008) and PITA (Riguzzi and Swift 2011) to introduce VT as a parameter learning routine.

One of the unique features of VT is its affinity with discriminative modeling. Write the VT's objective function $L_{VT}(y \mid \theta)$ as follows:

$$\begin{aligned} L_{VT}(y \mid \theta) &= \log\ p(x^*, y \mid \theta)p(\theta) \\ x^* &= \mathrm{argmax}_x\ p(x, y \mid \theta)p(\theta) \\ &= \mathrm{argmax}_x\ p(x \mid y, \theta)\cdot \end{aligned}$$

This means that although PRISM is intended for generative modeling, VT in PRISM computes the Viterbi explanation x^* that gives the highest conditional probability $p(x^* \mid y, \theta)$ for y whose form is identical to the objective function in discriminative modeling, and the Viterbi explanation is chosen in the same way as the discriminative modeling, provided the hidden variable is a prediction target. When this condition is met VT shows good performance as demonstrated by the experiments in Section 4, but if not, then VT does not necessarily outperform other parameter learning methods as exemplified in Section 5. It therefore seems reasonable to say that VT is effective for prediction tasks when the prediction target coincides with hidden variables in a probabilistic model, though we obviously need more experiments.

As a coordinate ascent local hill-climber, VT is sensitive to initial parameters and also to the Viterbi explanation. To mitigate the sensitivity problem with initial parameters, we used 50 time random restart in the learning experiments in Section 4. To cope with sensitivity to the Viterbi explanation, it is interesting to introduce k-best explanations as discussed in Gutmann *et al.* (2008) and replace the Viterbi explanation in VT with them. This approach will give us control over the sensitivity and computation time by choosing k and seems not very difficult to implement in PRISM as k-best explanations for a goal G are already computed by built-in predicates such as `n_viterbi(`k`,`G`)`.

Since VT in PRISM runs on explanation graphs obtained from all solution search, it requires time for all solution search (by tabling) and also space to store discovered explanation graphs. It is possible, however, to implement VT without explanation graphs, and to realize much more memory saving VT by repeating search for a

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

Viterbi explanation in each cycle of VT. We note that this approach particularly fits well with mode-directed tabling (Zhou *et al.* 2010). In mode-directed tabling, we can search for partial Viterbi explanations for subgoals efficiently without constructing explanation graphs and put them together to form a larger Viterbi explanation for the goal. Currently, however, mode-directed tabling is not available in PRISM. We are planing to incorporate it in PRISM in the near future.

8 Conclusions

We introduced VT to PRISM to enhance PRISM's probabilistic modeling power. To our knowledge, PRISM becomes the first SRL language (Getoor and Taskar 2007; De Raedt and Kersting 2008) in which VT is available for parameter learning.

Although VT has already been used in various models under various names (Juang and Rabiner 1990; Brown *et al.* 1993; Strom *et al.* 1999; Lomsadze *et al.* 2005; Joshi *et al.* 2006; Spitkovsky *et al.* 2010), we made the following contributions to VT. One is the generalization by deriving a generic VT algorithm for PRISM, thereby making it uniformly applicable to a very wide class of discrete models described by PRISM programs ranging from BNs to probabilistic grammars. The other is an empirical clarification of conditions under which VT performs well. We conducted learning experiments with PCFG and PLCG using VT, and confirmed VT's excellent parsing performance compared with EM, MAP and VB. We also conducted a learning experiment with NBH for classification tasks. Putting together the results of these experiments, we may say that VT performs well when hidden variables are a prediction target.

From the viewpoint of PRISM, VT improves PRISM, first by realizing faster convergence compared with EM, second by providing the user with a parameter learning method that can learn parameters good for prediction, and finally by providing a solution to the problem of exclusiveness condition that hinders PRISM programming. Thanks to VT, we are now able to use arbitrary programs with inclusive-or for probabilistic modeling.

Last but not least, we can say that as VT in PRISM is general and applicable to any PRISM program, it largely reduces the need for the user to develop a specific VT algorithm for a specific model. Furthermore, since VT in PRISM can be used just by setting a PRISM flag appropriately, it makes VT easily accessible to (probabilistic) logic programmers.

References

Bache, K. and Lichman, M. 2013. *UCI Machine Learning Repository* [`http://archive.ics.uci.edu/ml`]. Irvine, CA: University of California, School of Information and Computer Science.

Bellodi, E. and Riguzzi, F. 2012. Expectation maximization over binary decision diagrams for probabilistic logic programs. *Intelligent Data Analysis 16,* 6.

Brown, P., Pietra, V., Pietra, S. and Mercer, R. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics 19*, 263–311.

https://doi.org/10.1017/S1471068413000677 Published online by Cambridge University Press

Castillo, G. and Gama, J. 2005. Bias management of Bayesian network classifiers. In *Discovery Science – DS 2005, 8th International Conference*, Singapore, Lecture Notes in Artificial Intelligence, Vol. 3735. Springer-Verlag, New York, NY, 70–83.

Cohen, S. and Smith, N. 2010. Viterbi training for PCFGs: Hardness results and competitiveness of uniform initialization. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL'10)*. 1502–1511.

De Raedt, L. and Kersting, K. 2008. Probabilistic inductive logic programming. In *Probabilistic Inductive Logic Programming – Theory and Applications*, L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton, Eds. Lecture Notes in Computer Science, Vol. 4911. Springer, New York, NY, 1–27.

De Raedt, L., Kimmig, A. and Toivonen, H. 2007. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*. MIT Press, Cambridge, MA, 2468–2473.

Friedman, N., Geiger, D. and Goldszmidt, M. 1997. Bayesian network classifiers. *Machine Learning 29,* 2, 131–163.

Getoor, L. and Taskar, B., Eds. 2007. *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA.

Goodman, J. 1996. Parsing algorithms and metrics. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL'96)*. ACL, New York, NY, 177–183.

Gutmann, B., Kimmig, A., Kersting, K. and De Raedt, L. 2008. Parameter learning in probabilistic databases: A least squares approach. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD 2008), Part I*. Springer, New York, NY, 473–488.

Gutmann, B., Thon, I. and De Raedt, L. 2011. Learning the parameters of probabilistic logic programs from interpretations. In *Proceedings of European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD 2011), Part I,* LNCS, Vol. 6911. Springer, New York, NY, 581–596.

Huynh, T. and Mooney, R. 2010. Online max-margin weight learning with Markov logic networks. In *Proceedings of the AAAI-10 Workshop on Statistical Relational AI (Star-AI 10)*. 32–37.

Japkowicz, N. and Shah, M., Eds. 2011. *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, Cambridge, UK.

Jiang, L., Zhang, H. and Cai, Z. 2009. A novel Bayes model: Hidden naive Bayes. *IEEE Transactions on Knowledge and Data Engineering 21,* 10, 1361–1371.

Joshi, D., Li, J. and Wang, J. 2006. A computationally efficient approach to the estimation of two- and three-dimensional hidden Markov models. *IEEE Transactions on Image Processing 15,* 7, 1871–1886.

Juang, B. and Rabiner, L. 1990. The segmental K-means algorithm for estimating parameters of hidden Markov models. *IEEE Transactions on Signal Processing 38*, 1639–1641.

Kimmig, A., Costa, V., Rocha, R., Demoen, B. and De Raedt, L. 2008. On the efficient execution of ProbLog programs. In *Proceedings of the 24th International Conference on Logic Programming (ICLP'08)*. 175–189.

Lember, J. and Koloydenko, A. 2007. Adjusted viterbi training. *Probability in the Engineering and Informational Sciences 21,* 3, 451–475.

Lomsadze, A., Ter-Hovhannisyan, V., Chernoff, Y. and Borodovsky, M. 2005. Gene identification in novel eukaryotic genomes by self-training algorithm. *Nucleic Acids Research 33*, 6494–6506.

MacQueen, J. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1. 281–297.

Manning, C. 1997. Probabilistic parsing using left corner language models. In *Proceedings of the 5th International Conference on Parsing Technologies (IWPT-97)*. MIT Press, Cambridge, MA, 147–158.

Riguzzi, F. and Swift, T. 2011. The PITA system: Tabling and answer subsumption for reasoning under uncertainty. *Theory and Practice of Logic Programming (TPLP) 11,* 4–5, 433–449.

Roark, B. and Johnson, M. 1999. Efficient probabilistic top-down and left-corner parsing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*. 421–428.

Sato, T. 1995. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming (ICLP'95)*. Cambridge University Press, Cambridge, UK, 715–729.

Sato, T. 2007. Inside-outside probability computation for belief propagation. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI '07)*. 2605–2610.

Sato, T. 2011. A general MCMC method for Bayesian inference in logic-based probabilistic modeling. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI '11)*. 1472–1477.

Sato, T. and Kameya, Y. 2001. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research 15*, 391–454.

Sato, T. and Kameya, Y. 2008. New advances in logic-based probabilistic modeling by PRISM. In *Probabilistic Inductive Logic Programming*, L. De Raedt, P. Frasconi, K. Kersting and S. Muggleton, Eds. LNAI, Vol. 4911. Springer, New York, NY, 118–155.

Sato, T., Kameya, Y. and Kurihara, K. 2009. Variational Bayes via propositionalized probability computation in PRISM. *Annals of Mathematics and Artificial Intelligence 54*, 135–158.

Singla, P. and Domingos, P. 2005. Discriminative training of Markov logic networks. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, M. M. Veloso and S. Kambhampati, Eds. Kluwer, the Netherlands, 868–873.

Spitkovsky, V., Alshawi, H., Jurafsky, D. and Manning, C. 2010. Viterbi training improves unsupervised dependency parsing. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*. 9–17.

Strom, N., Hetherington, L., Hazen, T., Sandness, E. and Glass, J. 1999. Acoustic modeling improvements in a segment-based speech recognizer. In *Proceedings of IEEE ASRU Workshop (ASRU'99)*. IEEE Signal Processing Society, 139–142.

Su, J. and Zhang, H. 2006. Full Bayesian network classifiers. In *Proceedings of the 23rd International Conference on Machine Learning (ICML'06)*. 897–904.

Uratani, N., Takezawa, T., Matsuo, H. and Morita, C. 1994. *ATR Integrated Speech and Language Database*. Technical Report TR-IT-0056, ATR Interpreting Telecommunications Research Laboratories, Kyoto, Japan. (in Japanese).

Van Uytsel, D., Van Compernolle, D. and Wambacq, P. 2001. Maximum-likelihood training of the PLCG-based language model. In *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop (ASRU'01)*. IEEE Signal Processing Society, 210–213.

Webb, G., Boughton, J. and Wang, Z. 2005. Not so naive Bayes: Aggregating one-dependence estimators. *Machine Learning 58,* 1, 5–24.

Zhou, N.-F., Kameya, Y. and Sato, T. 2010. Mode-directed tabling for dynamic programming, machine learning, and constraint solving. In *Proceedings of the 22th International Conference on Tools with Artificial Intelligence (ICTAI-2010)*. IEEE Computer Society, 213–218.

Zhou, N.-F., Sato, T. and Shen, Y.-D. 2008. Linear tabling strategies and optimization. *Theory and Practice of Logic Programming (TPLP) 8,* 1, 81–109.