Mach Learn (2012) 89:279–297
DOI 10.1007/s10994-012-5305-8

Online Bayesian inference for the parameters of PRISM programs

James Cussens

Received: 5 February 2012 / Revised: 4 April 2012 / Accepted: 8 June 2012 / Published online: 27 July 2012

Abstract This paper presents a method for approximating posterior distributions over the parameters of a given PRISM program. A sequential approach is taken where the distribution is updated one datapoint at a time. This makes it applicable to online learning situations where data arrives over time. The method is applicable whenever the prior is a mixture of products of Dirichlet distributions. In this case the true posterior will be a mixture of very many such products. An approximation is effected by merging products of Dirichlet distributions. An analysis of the quality of the approximation is presented. Due to the heavy computational burden of this approach, the method has been implemented in the Mercury logic programming language. Initial results using a hidden Markov model and a probabilistic graph are presented.

Keywords Inductive logic programming · Bayesian statistics · Statistical relational learning · PRISM · Mixture models · Missing data

1 Introduction

In the Bayesian approach to 'parameter estimation' the goal is to return the joint posterior distribution over all parameters, rather than return the single 'best estimate' of the parameters. The motivation for attempting this complex task is that the posterior captures the combined information given by observed data and prior knowledge, and so provides a much fuller picture of the state of our knowledge about the parameters than can a point estimate.

Unfortunately, many posterior distributions are hard even to represent let alone compute efficiently. This is certainly generally the case for posterior distributions over the parameters of PRISM programs. PRISM programs define distributions over finite or countably infinite sample spaces using potentially complex generative processes. Generally the steps

Editors: S. Muggleton and J. Chen.

J. Cussens (✉)
Dept. of Computer Science & York Centre for Complex Systems Analysis, University of York, Deramore Lane, York, YO10 5GE, UK
e-mail: james.cussens@york.ac.uk

taken in the generative process are not discernible from any observed output—a hidden data situation—which leads to a posterior distribution with many local modes.

Fortunately, if the prior over PRISM parameters is a mixture of products of Dirichlet distributions, then at least the form of the posterior will be known: it will also be a mixture of products of Dirichlet distributions. However, the number of mixture components will usually be large. This paper presents an exact technique for finding all these mixture components for small scale problems and considers an approximate method for cases where the exact approach is infeasible.

The paper is organised as follows. The statistical aspects of the PRISM formalism are explained in Sect. 2. Section 3 derives an expression for the posterior distribution when the prior is a mixture of products of Dirichlet distributions. Section 4 describes the approximation method and presents results on the quality of the approximation. Initial results are presented in Sect. 5 followed by conclusions and pointers to further work in Sect. 6.

2 PRISM

In this section a description of PRISM will be given which includes an explanation of the basics of the Bayesian approach to parameter estimation in PRISM. All of this material has been previously presented elsewhere and is included here merely as a convenience for the reader. In particular, the following is closely based on work by Cussens (2005, 2007) and Sato et al. (2008).

2.1 Defining distributions in PRISM

A PRISM program is a logic program together with a probabilistic built-in predicate `msw/2`. See Fig. 1 for the PRISM program `hmm.psm` which comes as an example with the PRISM system and implements a simple 2-state hidden Markov model which will be used as a running example throughout the paper. From a procedural logic programming perspective PRISM programs are very easy to grasp: if, for example, the goal `msw(init,S)` is called then it will always succeed and the logical variable `S` will be instantiated by sampling from a probability distribution defined by the parameters of the PRISM program. These parameters will be denoted by $\boldsymbol{\theta}$ in this paper. In `hmm.psm` the distribution associated with `msw(init,S)` is $(0.9, 0.1)$ (as defined by the `set_sw/2` built-in) and the corresponding possible values for `S` are `s0` and `s1` (as specified by the builtin-in `values/2`). All other predicates behave in the normal logic programming fashion. The remainder of this section just expands and formalises this behaviour.

Considering now the general case, a ground fact such as `msw('X1',x)` is actually an abbreviation for a fact `msw('X1',j,x)` which is a statement that it is true that the random variable $X_{1,j}$ is instantiated to have a value x, where $j \in \mathbb{N}$. For any $j, j' \in \mathbb{N}$, where $j \neq j'$, $X_{1,j}$ and $X_{1,j'}$ must be independent and identically distributed (which motivates the abbreviation just mentioned and explains why the index j is not represented in actual PRISM programs). The common distribution of the $X_{1,j}$ is an arbitrary discrete distribution defined by the parameter vector $\boldsymbol{\theta}_1 = (\theta_{1,1}, \ldots, \theta_{1,v}, \ldots \theta_{1,n(1)})$ where $\theta_{1,v}$ is the probability that $X_{1,j}$ takes value v. A family of iid random variables such as $\{X_{1,j}\}_{j \in \mathbb{N}}$ is known as a *switch*.

Typically a PRISM program has more than one switch, each switch defining a different family of iid variables. `hmm.psm` has 5 switches. These are `init`, `tr(s0)`, `out(s0)`, `tr(s1)` and `out(s1)` defining: the initial state distribution, state transitions from state `s0`, emissions from state `s0`, state transitions from state `s1` and emissions from state `s1`,

```
values(init,[s0,s1]).  % there are two initial states {s0,s1}.
values(out(S),[a,b]).  % output symbols are {a,b} in every state.
values(tr(S),[s0,s1]). % transition destinations are {s0,s1} in every
                         state.

set_params :-          % set parameters manually to the model.
   set_sw(init,   0.9+0.1),
   set_sw(tr(s0), 0.2+0.8),
   set_sw(out(s0),0.5+0.5),
   set_sw(tr(s1), 0.8+0.2),
   set_sw(out(s1),0.6+0.4).

hmm(L):-
   msw(init,S),        %
   hmm(1,5,S,L).       % length of 5 hard-coded in

hmm(T,M,_,[]) :-T>M,!.
hmm(T,M,S,[Ob|Y]) :-   % current state is S, current time is T.
   msw(out(S),Ob),     % output in the state S is Ob.
   msw(tr(S),Next),    % transition from S to Next.
   T1 is T+1,          % count up time
   hmm(T1,M,Next,Y).   % recursion
```

Fig. 1 `hmm.psm`: a PRISM program implementing a hidden Markov model

respectively. This collection of discrete probability distributions $\boldsymbol{\theta}_i$, one for each switch i, makes up the parameter set $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_i, \ldots, \boldsymbol{\theta}_n)$ for a given PRISM program. Crucially, any two distinct switches are mutually independent so that $\forall i, i', j, j'\ X_{i,j}$ is independent of $X_{i',j'}$ whenever $i \neq i'$. Given any finite subset of the $\{X_{i,j}\}_{i,j}$ a product distribution can be defined on its joint instantiations in the obvious way. As noted by Sato and Kameya (2001) it then follows that there is a probability distribution which assigns a probability to any (measurable) set of infinite joint instantiations of the $\{X_{i,j}\}_{i,j}$. This is known as the *basic distribution* and is consistent with all the finite product distributions.

Any instantiation of all the (infinitely many) $X_{i,j}$ together with the rest of the clauses of the PRISM program, defines a least Herbrand model: a *possible world*. The probability of any possible world is the probability of the set of infinite instantiations of the $X_{i,j}$ which entail it. Thus the parameters of the PRISM program define a distribution over a set of possible worlds. This distribution determines a (marginal) probability for each ground atomic formulae in the first-order language defined by the PRISM program; it is the sum of the probabilities of the worlds which satisfy the formula.

Usually a particular *target predicate* is distinguished to represent an 'output' distribution defined by a PRISM program. The target predicate for `hmm.psm` is `hmm/1`. Typically a target predicate is defined so that exactly one ground atomic formula with it as predicate symbol is true in each possible world, thereby defining a distribution over ground instances of the target predicate. (In Sect. 5.3 we will see that is possible to relax this condition and still do Bayesian parameter estimation.) This is certainly the case for `hmm.psm`. It is not too difficult to see that however the switches are instantiated (i.e. which possible world is chosen) exactly one ground instance of `hmm(L)` is logically entailed by the program. Such ground instances are viewed as *outputs* of the PRISM program and will be generically denoted by y.

In PRISM programs 'with failure' the condition on the target predicate is weakened so that *at most* one output is true in each possible world; sometimes the program may 'fail' to generate an output. Adding, for example, the literal `L = [A,B,A,C,D]` to the clause defining `hmm/1`, which effects the constraint that the 1st and 3rd symbols of the HMM

output are the same, would be enough to make `hmm.psm` a program 'with failure', since then some instantiations of the switches would not logically entail any ground instance of `hmm(L)`. Unless the probability of failure is zero the probabilities of ground instances of the target predicate will sum to less than one, so that to define a distribution over them it is necessary to normalise. All PRISM programs discussed in this paper will be *failure-free*, i.e. **not** 'with failure'.

In the normal non-failure case any instantiation of the infinitely many $X_{i,j}$ determines exactly one output. However, it is a requirement of PRISM programs that a finite subset of any such infinite instantiation is enough to determine which output this is. A finite instantiation which is *minimal* with respect to determining output is called an *explanation*. Explanations will be generically denoted by x. An explanation x must entail an output $y = f(x)$ and be such that any of its proper subsets do not. f is the function mapping explanations to outputs which is encoded by the structure of the PRISM program. A further restriction on PRISM programs is that any output has only a finite number of associated explanations: $\forall y : |f^{-1}(y)| < \infty$. This is known as the *finite support condition* and $f^{-1}(y)$ is known as the *support set* for y.

Let $C_{i,v}(x)$ be the count of how often switch i takes the value v in explanation x. The probability of an explanation is then a simple monomial:

$$P(x|\boldsymbol{\theta}) = \prod_{i,v} {\theta_{i,v}}^{C_{i,v}(x)} \tag{1}$$

The probability of an observation y is then the sum of the probabilities of its explanations:

$$P(y|\boldsymbol{\theta}) = \sum_{x:f(x)=y} P(x|\boldsymbol{\theta}) = \sum_{x:f(x)=y} \prod_{i,v} {\theta_{i,v}}^{C_{i,v}(x)} \tag{2}$$

2.2 Existing approaches to parameter estimation in PRISM

The simplest way to fit the parameters of a PRISM program is to do maximum likelihood estimation (MLE) using the EM algorithm. The key to doing this efficiently is to find the explanations $f^{-1}(y)$ of an observed data point y just *once*, rather than on each iteration of the EM algorithm. In addition, it is highly advantageous to use a compact representation of explanations called *explanation graphs*. All of this is done in the *graphical EM algorithm* which is explained by Sato and Kameya (2001). This algorithm is provided via builtin predicates in the current PRISM distribution. The EM approach to parameter estimation has been extended to PRISM programs 'with failure' (Sato et al. 2005) by incorporating the *failure-adjusted maximisation* approach, originally devised by the current author Cussens (2001) for stochastic logic programs (Muggleton 1996).

More recently Sato and colleagues have looked at estimating PRISM parameters using a Bayesian approach, arguing (as here) that MLE can result in overfitting. In work (Sato et al. 2008) closely related to the current paper they present a variational approach to Bayesian parameter estimation (and approximation of the marginal likelihood). A Dirichlet prior distribution is defined for the probabilities of each switch in the PRISM program. A variational Bayes approach (the VB-EM algorithm) is then used which produces an approximate posterior distribution which is also a Dirichlet distribution for each switch. Rather than perform the necessary calculations naïvely (as is done in this paper), Sato et al. use the *graphical VB-EM algorithm* which exploits explanation graphs similarly to the graphical EM algorithm.

Approximate Bayesian inference is often done via sampling, and PRISM is no exception. Sato (2011) uses MCMC to sample explanations of observed data (rather than find all

such explanations as is done in this paper). These samples can then be used to estimate the marginal likelihood and also to estimate the most likely explanation of some new datapoint. As an alternative to MCMC the current author has used *Approximate Bayesian Computation (ABC)* to estimate PRISM parameters (Cussens 2011). ABC methods are often called 'likelihood-free' since they avoid explicitly considering the likelihood function: instead data is generated from candidate parameter values and this synthetic data is compared to the actually observed data. Cussens (2011) the current author applies the ABC sequential Monte Carlo (ABC-SMC) algorithm of Toni et al. (2009) to PRISM, implemented using the current PRISM distribution. (In later unpublished work a Mercury implementation was used.)

Evidently the current paper is most closely related to previous work which took a Bayesian approach. There is an obvious difference with the sample-based approaches: here we update a closed form approximation to the posterior as each datapoint is considered. This sort of online updating is more difficult with sampling approaches.

A key difference to all the existing work in this area is that the current approach is informed by the knowledge that the true posterior distribution for PRISM parameters is guaranteed to be a mixture of products of Dirichlet distributions whenever the prior is also of this form. A special case of such a prior is a single product of Dirichlet distributions which is the prior used in all previous work on Bayesian inference for PRISM parameters. The hypothesis here is that the true mixture distribution is best approximated by a distribution which is also a mixture (albeit with fewer components). This will at least be a distribution with many local modes like the true one. Approximating with a single product of Dirichlet distributions (as in Sato et al. 2008) cannot do this. On the negative side, aspects of the algorithm presented here are naïve when compared to, say, Sato et al. (2008), since explanations are collected and stored without exploiting explanation graphs. This is an obvious direction for future improvement.

2.3 Bayesian inference for PRISM parameters

When using PRISM programs for statistical inference, observed data will be a sequence of outputs $\mathbf{y} = y_1, \ldots, y_T$. For example, when considering `hmm.psm` the data might be the 3 datapoints `hmm([a,b,a,a,a])`, `hmm([b,b,a,a,b])` and `hmm([b,a,a,b,b])`. Each such datapoint is viewed as having been independently sampled from an unknown PRISM program. This paper concerns only parameter estimation and so the structure of the program will be known, only the parameters defining switch distributions will be unknown.

The key point about statistical inference for PRISM programs is that, barring exceptional cases, the values of the switches—the explanation—associated with any observed datapoint will be unobserved. For example, there are 2^5 distinct explanations for the observed datapoint `hmm([a,b,a,a,a])` corresponding to the 2^5 different state trajectories associated with this sequence of symbols. We are thus faced with latent variables which can be considered as 'missing data'.

Now consider a Bayesian approach where a prior $P(\boldsymbol{\theta})$ is given and the goal is to compute the posterior $P(\boldsymbol{\theta}|y_1, \ldots, y_T)$ where, of course:

$$P(\boldsymbol{\theta}|y_1, \ldots, y_T) = \frac{P(\boldsymbol{\theta}) \prod_{t=1}^{T} P(y_t|\boldsymbol{\theta})}{P(y_1, \ldots, y_T)} \tag{3}$$

As indicated by (2), $P(y_t|\boldsymbol{\theta})$ is generally a complex polynomial function of $\boldsymbol{\theta}$ which means that the likelihood function $\prod_{t=1}^{T} P(y_t|\boldsymbol{\theta})$ is an even more complex polynomial. This rules out computing an exact closed form representation for $P(\boldsymbol{\theta}|y_1, \ldots, y_T)$ irrespective

of the chosen prior $P(\boldsymbol{\theta})$. However, as mentioned in the introduction, $P(\boldsymbol{\theta}|y_1,\ldots,y_T)$ will be a mixture of products of Dirichlet distributions whenever the prior $P(\boldsymbol{\theta})$ is a product of Dirichlet distributions. So mixtures of products of Dirichlet distributions are 'closed' under updating. This is an example of the well-known phenomenon of the 'weak' conjugacy of mixtures (Bernardo and Girón 1988). (Note also that using a mixture of distributions from a particular class as a prior is always more flexible than using a single distribution from that class.) The next section has the necessary derivation.

3 The posterior distribution when the prior is a mixture of products of Dirichlet distributions

Let

$$P(\boldsymbol{\theta}) = \sum_{\ell=1}^{L} P(\ell)P(\boldsymbol{\theta}|\ell) = \sum_{\ell=1}^{L} P(\ell)\prod_i \mathrm{Dir}(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_{\ell,i}) \tag{4}$$

be the prior distribution where (as shown) each $P(\boldsymbol{\theta}|\ell)$ is a product of Dirichlet distributions (one Dirichlet distribution for each switch i in the PRISM program) and the $P(\ell)$ define an arbitrary discrete distribution. $P(\boldsymbol{\theta})$ is thus a mixture distribution. The $P(\boldsymbol{\theta}|\ell)$ are the *mixture components* and the corresponding $P(\ell)$ are the *mixture weights*. $\boldsymbol{\alpha}_{\ell,i}$ is $(\alpha_{\ell,i,v})_v$ the vector of Dirichlet parameters for the Dirichlet distribution for the ith switch in the ℓth mixture component. This distribution is defined as follows:

$$\mathrm{Dir}(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_{\ell,i}) = \big(B(\boldsymbol{\alpha}_{\boldsymbol{\ell},\boldsymbol{i}})\big)^{-1}\prod_v {\theta_{i,v}}^{\alpha_{\ell,i,v}-1} \tag{5}$$

where each $\alpha_{\ell,i,v}$ must be positive and B is the multinomial beta function:

$$B(\boldsymbol{\alpha}_{\boldsymbol{\ell},\boldsymbol{i}}) = \frac{\prod_v \Gamma(\alpha_{\ell,i,v})}{\Gamma(\sum_v \alpha_{\ell,i,v})} \tag{6}$$

and the Γ function is defined as $\Gamma(x) \equiv \int_0^\infty e^{-t}t^{x-1}\,dt$.

As is well-known (Gelman et al. 1995), the Dirichlet is the *conjugate* prior for multi-valued discrete distributions, such as those defined over PRISM switches. Consider conditioning on a single explanation x and let $\boldsymbol{C}_i(x)$ be the vector of counts recording how often switch i took on each of its possible values in x (recall that an explanation x is an instantiation of switch values). Since the Dirichlet is a conjugate prior for such data, we have that the posterior distribution is also a product of Dirichlet distributions:

$$P(\boldsymbol{\theta}|\ell,x) = \prod_i \mathrm{Dir}\big(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_{\ell,i} + \boldsymbol{C}_i(x)\big) \tag{7}$$

Equation (7) is the key reason why mixtures of product of *Dirichlet* distributions are used. Consider now conditioning on a single observed datapoint y. To do this note first that:

$$P(x|\ell) = \prod_i \frac{B(\boldsymbol{\alpha}_{\ell,i} + \boldsymbol{C}_i(x))}{B(\boldsymbol{\alpha}_{\ell,i})} \tag{8}$$

Equation (8) is obtained by marginalising over $\boldsymbol{\theta}$ with respect to the density $\mathrm{Dir}(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_{\ell,i})$. We now use it to derive an expression for $P(\boldsymbol{\theta}, y|\ell)$.

$$
\begin{aligned}
P(\boldsymbol{\theta}, y|\ell) &= \sum_{x:f(x)=y} P(\boldsymbol{\theta}, x|\ell) \\
&= \sum_{x:f(x)=y} P(x|\ell) P(\boldsymbol{\theta}|x, \ell) \\
&= \sum_{x:f(x)=y} \prod_i \frac{B(\boldsymbol{\alpha}_{\ell,i} + \boldsymbol{C}_i(x))}{B(\boldsymbol{\alpha}_{\ell,i})} \prod_i \mathrm{Dir}\big(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_{\ell,i} + \boldsymbol{C}_i(x)\big)
\end{aligned} \tag{9}
$$

From (8) it follows that:

$$
P(y|\ell) = \sum_{x:f(x)=y} \prod_i \frac{B(\boldsymbol{\alpha}_{\ell,i} + \boldsymbol{C}_i(x))}{B(\boldsymbol{\alpha}_{\ell,i})} \tag{10}
$$

and so

$$
P(\boldsymbol{\theta}|y, \ell) = \frac{\sum_{x:f(x)=y} \prod_i \frac{B(\boldsymbol{\alpha}_{\ell,i}+\boldsymbol{C}_i(x))}{B(\boldsymbol{\alpha}_{\ell,i})} \prod_i \mathrm{Dir}(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_{\ell,i} + \boldsymbol{C}_i(x))}{\sum_{x:f(x)=y} \prod_i \frac{B(\boldsymbol{\alpha}_{\ell,i}+\boldsymbol{C}_i(x))}{B(\boldsymbol{\alpha}_{\ell,i})}} \tag{11}
$$

Finally, since $P(\boldsymbol{\theta}|y) = \sum_\ell P(\ell) P(\boldsymbol{\theta}|y, \ell)$ we have:

$$
P(\boldsymbol{\theta}|y) = \sum_\ell P(\ell) \frac{\sum_{x:f(x)=y} \prod_i \frac{B(\boldsymbol{\alpha}_{\ell,i}+\boldsymbol{C}_i(x))}{B(\boldsymbol{\alpha}_{\ell,i})} \prod_i \mathrm{Dir}(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_{\ell,i} + \boldsymbol{C}_i(x))}{\sum_{x:f(x)=y} \prod_i \frac{B(\boldsymbol{\alpha}_{\ell,i}+\boldsymbol{C}_i(x))}{B(\boldsymbol{\alpha}_{\ell,i})}} \tag{12}
$$

So, as (12) shows, using a mixture of products of Dirichlet distributions as a prior with weights $P(\ell)$ and conditioning on a single datapoint y results in a new mixture of products of Dirichlet distributions with one component for each pair (ℓ, x) where x is an explanation of y. The weight for each such component is:

$$
P(\ell) \frac{\prod_i \frac{B(\boldsymbol{\alpha}_{\ell,i}+\boldsymbol{C}_i(x))}{B(\boldsymbol{\alpha}_{\ell,i})}}{\sum_{x':f(x')=y} \prod_i \frac{B(\boldsymbol{\alpha}_{\ell,i}+\boldsymbol{C}_i(x'))}{B(\boldsymbol{\alpha}_{\ell,i})}} = \frac{P(\ell) \prod_i B(\boldsymbol{\alpha}_{\ell,i} + \boldsymbol{C}_i(x))}{\sum_{x':f(x')=y} \prod_i B(\boldsymbol{\alpha}_{\ell,i} + \boldsymbol{C}_i(x'))} \tag{13}
$$

and the associated product of Dirichlet distributions is:

$$
\prod_i \mathrm{Dir}\big(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_{\ell,i} + \boldsymbol{C}_i(x)\big) \tag{14}
$$

4 Sequential approximate computation of posterior distributions for PRISM parameters

Using (12) the posterior $P(\boldsymbol{\theta}|\mathbf{y}) = P(\boldsymbol{\theta}|y_1, \ldots, y_T)$ could be sequentially computed by conditioning on each of the y_t in turn. However, since the number of mixture components increases by a factor of $|\{x : f(x) = y_t\}|$ for each y_t this is clearly impractical for all but the smallest values of T. It is true that if $\prod_i \mathrm{Dir}(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_{\ell,i} + \boldsymbol{C}_i(x)) = \prod_i \mathrm{Dir}(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_{\ell',i} + \boldsymbol{C}_i(x'))$ for two pairs (ℓ, x), and (ℓ', x') then the components are identical and can be merged, but it is unrealistic to depend upon such coincidences to keep the number of mixture components manageable.

So instead an approximate sequential approach will be taken. The idea is to compute approximations to the following sequence of posterior distributions:

$$P(\boldsymbol{\theta}|y_1), P(\boldsymbol{\theta}|y_1, y_2), \ldots, P(\boldsymbol{\theta}|y_1, y_2 \ldots, y_T)$$

Since the number of mixture components grows exponentially with t, at each point only an approximation to the distribution $P(\boldsymbol{\theta}|y_1, \ldots, y_t)$ is maintained. A limit K on the number of mixture components is set; if a mixture distribution is constructed with more than K components then it is approximated by successively finding the component with the smallest weight and then 'merging' it with the 'nearest' other component. The limit K puts an upper bound on the time and space needed for each update and so the approach is *linear* in the size of the data.

This approach to mixture reduction was used by Cowell et al. (1995) and is also discussed by Cowell et al. (1999). It is an instance of *assumed density filtering* where a complex posterior is approximated with something more convenient at each observation. The method presented here is also related to the clustering method of West (1992). Note that for each y_t, *all* explanations x are searched for, although only the associated count vectors $\boldsymbol{C}_i(x)$ are recorded. It follows that for this method to be practical the number of explanations for any single datapoint cannot be too great.

4.1 Merging components

As previously mentioned, the number of components is reduced by successively merging the lowest-weighted component with the nearest other component. Following Cowell et al. (1995), distance between components is simply the Euclidean distance between mean vectors. Other, perhaps better justified, metrics are possible: Euclidean distance was chosen since it is computationally cheap and has been used previously by Cowell et al. (1995) with some success. Once a nearest neighbour has been found according to this metric, merging is done using (essentially) the moment-matching method of Cowell et al. (1995).

The Dirichlet distributions for the ith switch in each of the two components are merged into a new single Dirichlet distribution without regard for the other switches and so it is enough to explain how the two Dirichlet distributions for switch i are merged. For simplicity, and without loss of generality label the two distributions to be merged as component $\ell = 1$ and $\ell = 2$. Let $P'(\ell = 1) = P(\ell = 1)/(P(\ell = 1) + P(\ell = 2))$ and $P'(\ell = 2) = P(\ell = 2)/(P(\ell = 1) + P(\ell = 2))$. First consider the mean value for $\theta_{i,v}$ given by the mixture of these two components. Denote it by $m_{i,v}$, we have:

$$m_{i,v} = \sum_{\ell=1}^{2} P'(\ell) \frac{\alpha_{\ell,i,v}}{\sum_{v'} \alpha_{\ell,i,v'}} \tag{15}$$

Consider next the second moment of $\theta_{i,v}$ *about 0*. Denote this by $m^{(2)}_{i,v}$. We have, from Cowell et al. (1995):

$$m^{(2)}_{i,v} = \sum_{\ell=1}^{2} P'(\ell) \frac{\alpha_{\ell,i,v}(\alpha_{\ell,i,v} + 1)}{(\sum_{v'} \alpha_{\ell,i,v'})(\sum_{v'} \alpha_{\ell,i,v'} + 1)} \tag{16}$$

Let $n(i)$ be the number of values for the ith switch, then the average second moment about 0 for the ith switch is:

$$\bar{m}^{(2)}_i = \frac{1}{n(i)} \sum_{v} m^{(2)}_{i,v} \tag{17}$$

Let $\tilde{\alpha}_{i,v}$ be the Dirichlet parameter for $\theta_{i,v}$ in the merged distribution. As long as $\tilde{\alpha}_{i,v} = \beta m_{i,v}$ for some β then the mean of the merged distribution will match that of the 2-component mixture. To ensure the average second moments also match β is set as follows:

$$\beta = \frac{\sum_v (m_{i,v} - m_{i,v}^{(2)})}{\sum_v (m_{i,v}^{(2)} - (m_{i,v})^2)} \tag{18}$$

So, in this way, two Dirichlet distributions for a switch are merged into one. Doing this for all switches produces one product of Dirichlet distributions from a pair of such products. The weight of the merged component is simply $P(\ell = 1) + P(\ell = 2)$, the sum of the weights of the two original components.

4.2 Accuracy of the sequential approximation

The true posterior is approximated by merging mixture components. A key issue is, of course, how good this approximation is. One option is to measure the quality of the approximation using KL-divergence. The KL-divergence, $D(p||q)$, between two probability densities $p(z)$ and $q(z)$ is:

$$D(p||q) = \int_z p(z) \log \frac{p(z)}{q(z)}\, dx \tag{19}$$

First consider the general question of the KL-divergence between two mixtures $\sum_{\ell=1}^N w_\ell p_\ell$ and $\sum_{\ell=1}^N w_\ell q_\ell$ with equal component weights. (Here $P(\ell)$ is written as w_ℓ for compactness.) An upper bound on $D(\sum_{\ell=1}^N w_\ell p_\ell || \sum_{\ell=1}^N w_\ell q_\ell)$ can be found using the log-sum inequality as shown in Cover and Thomas (1991):

$$D\left(\sum_{\ell=1}^N w_\ell p_\ell || \sum_{\ell=1}^N w_\ell q_\ell\right) \leq \sum_{\ell=1}^N w_\ell D(p_\ell||q_\ell) \tag{20}$$

Now consider the KL-divergence between a mixture $\sum_{\ell=1}^N w_\ell p_\ell$ and an approximation to it constructed by merging components. Let $\sum_{\kappa=1}^K w_\kappa q_\kappa$ (where $K < N$) be the approximation. Let g be a function such that $g(\ell) = \kappa$ if and only if component ℓ in the original mixture ends up 'in' component κ in the approximation. Since weights are added when merging, we have $w_\kappa = \sum_{\ell \in g^{-1}(\kappa)} w_\ell$ and so $w_\kappa q_\kappa = \sum_{\ell \in g^{-1}(\kappa)} w_\ell q_\kappa$ and thus $\sum_{\kappa=1}^K w_\kappa q_\kappa = \sum_{\ell=1}^N w_\ell q_{g(\ell)}$. Plugging this into (20) gives:

$$D\left(\sum_{\ell=1}^N w_\ell p_\ell || \sum_{\kappa=1}^K w_\kappa q_\kappa\right) \leq \sum_{\ell=1}^N w_\ell D(p_\ell||q_{g(\ell)}) = \sum_{\kappa=1}^K \sum_{\ell \in g^{-1}(\kappa)} w_\ell D(p_\ell||q_\kappa) \tag{21}$$

The inequality (21) shows that, unsurprisingly, it is important that high weight components in the original mixture are mapped into 'nearby' components in the approximation, where nearness is measured by KL-divergence. Clearly if $g^{-1}(g(\ell)) = \{\ell\}$ so that no other components are merged with p_ℓ then $q_{g(\ell)}$ will be p_ℓ and so $D(p_\ell||q_{g(\ell)}) = D(p_\ell||p_\ell) = 0$. In particular, consider the KL-divergence between a mixture and an approximation to it produced by merging the smallest weight component into another component. If this smallest weight is $w_{\ell'}$ then (21) shows that the KL-divergence between the original mixture and its

approximation is at most $w_{\ell'} D(p_{\ell'}||q_{g(\ell')})$ where $q_{g(\ell')}$ is the merged component. This motivates the greedy step in the sequential algorithm where the smallest weight component is chosen for merging.

In this paper, both p_ℓ and q_κ will be products of Dirichlet distributions. Fortunately, the KL-divergence between product distributions is straightforwardly connected to the KL-divergences between the relevant factors (Cover and Thomas 1991):

$$D(p_\ell||q_\kappa) = \sum_i D(p_{\ell,i}||q_{\kappa,i}) \tag{22}$$

It remains to examine the KL-divergence between two Dirichlet distributions. This is given by Penny (2001). Let $\alpha_{\ell,i,0} = \sum_v \alpha_{\ell,i,v}$, then

$$D(p_{\ell,i}||q_{\kappa,i}) = \log \frac{B(\boldsymbol{\alpha}_{\ell,i})}{B(\boldsymbol{\alpha}_{\kappa,i})} + \sum_v [\alpha_{\ell,i,v} - \alpha_{\kappa,i,v}]\big[\Psi(\alpha_{\ell,i,v}) - \Psi(\alpha_{\ell,i,0})\big] \tag{23}$$

where Ψ is the Digamma function: $\Psi(x) = (\log \Gamma(x))' = \Gamma'(x)/\Gamma(x)$.

Putting together inequality (21) with (22) and (23) provides an upper bound on the KL-divergence between any given mixture of products of Dirichlet distributions and any given approximation to it produced by merging components. However it is difficult to see how to use this result to formulate a good merging strategy.

Since, in common with other work in this area, we are adopting a greedy approach where the approximation is constructed by successively merging pairs of components, the key issues are (1) which two components to merge and (2) what the new merged component should be. To this end consider the KL-divergence between a 2-component mixture of Dirichlet distributions $r = \lambda p_1 + (1-\lambda) p_2$ $(0 < \lambda < 1)$ and a single Dirichlet distribution q. In the context of this paper these 3 distributions would all be Dirichlet distributions for the parameters of some switch i, and so we would have $p_1 = p_{1,i}$, $p_2 = p_{2,i}$ and $q = q_{\kappa,i}$ with $\lambda = w_1/(w_1 + w_2)$. For notational convenience the switch indicator i will be suppressed in what follows, except that the number of values the switch can take will remain denoted by $n(i)$. We have:

$$\begin{aligned} D(r||q) &= \int r \log \frac{r}{q} \\ &= \int r \log r - \int r \log q \\ &= -H(r) - \int \big(\lambda p_1 + (1-\lambda) p_2\big) \log q \\ &= -H(r) - \lambda \int p_1 \log q - (1-\lambda) \int p_2 \log q \end{aligned} \tag{24}$$

where $H(r)$ is the entropy of the mixture distribution. From (24) it is clear that the key quantities to evaluate are $\int p_1 \log q$ and $\int p_2 \log q$. So let p be a Dirichlet distribution with parameters $\alpha_{p,1}, \ldots, \alpha_{p,n(i)}$ and q be a Dirichlet distribution with parameters $\alpha_{q,1}, \ldots \alpha_{q,n(i)}$. Let $\alpha_{p,0} = \sum_{v=1}^{n(i)} \alpha_{p,v}$ and $\alpha_{q,0} = \sum_{v=1}^{n(i)} \alpha_{q,v}$. Drawing on some observations in Blei et al. (2003), we have:

$$\int_{\boldsymbol{\theta}} p(\boldsymbol{\theta}) \log q(\boldsymbol{\theta})\, d\boldsymbol{\theta} = \int_{\boldsymbol{\theta}} p(\boldsymbol{\theta}) \left[\sum_{v=1}^{n(i)} (\alpha_{q,v} - 1) \log \theta_v + \log \Gamma(\alpha_{q,0}) - \sum_{v=1}^{n(i)} \log \Gamma(\alpha_{q,v}) \right] d\boldsymbol{\theta}$$

$$= \sum_{v=1}^{n(i)} (\alpha_{q,v} - 1) \int_{\boldsymbol{\theta}} p(\boldsymbol{\theta}) \log \theta_v \, d\boldsymbol{\theta} + \log \Gamma(\alpha_{q,0}) - \sum_{v=1}^{n(i)} \log \Gamma(\alpha_{q,v})$$

$$= \sum_{v=1}^{n(i)} (\alpha_{q,v} - 1)\big(\Psi(\alpha_{p,v}) - \Psi(\alpha_{p,0})\big) + \log \Gamma(\alpha_{q,0}) - \sum_{v=1}^{n(i)} \log \Gamma(\alpha_{q,v})$$

$$= -\log B(\boldsymbol{\alpha_q}) + \sum_{v=1}^{n(i)} (\alpha_{q,v} - 1)\big(\Psi(\alpha_{p,v}) - \Psi(\alpha_{p,0})\big) \tag{25}$$

Plugging (25) into (24) gives:

$$D(r||q) = -H(r) - \sum_{v=1}^{n(i)} (\alpha_{q,v} - 1)\big[\lambda\big(\Psi(\alpha_{p_1,v}) - \Psi(\alpha_{p_1,0})\big) + (1-\lambda)\big(\Psi(\alpha_{p_2,v}) - \Psi(\alpha_{p_2,0})\big)\big] - \log \Gamma(\alpha_{q,0}) + \sum_{v=1}^{n(i)} \log \Gamma(\alpha_{q,v}) \tag{26}$$

Our goal is to choose q to minimise $D(r||q)$. Differentiating (26) we have:

$$\frac{dD(r||q)}{d\alpha_{q,v}} = \lambda\big(\Psi(\alpha_{p_1,v}) - \Psi(\alpha_{p_1,0})\big) + (1-\lambda)\big(\Psi(\alpha_{p_2,v}) - \Psi(\alpha_{p_2,0})\big) - \Psi(\alpha_{q,0}) + \Psi(\alpha_{q,v}) \tag{27}$$

So that $D(r||q)$ is minimised when:

$$\forall v : \Psi(\alpha_{q,v}) - \Psi(\alpha_{q,0}) = \lambda\big[\Psi(\alpha_{p_1,v}) - \Psi(\alpha_{p_1,0})\big] + (1-\lambda)\big[\Psi(\alpha_{p_2,v}) - \Psi(\alpha_{p_2,0})\big] \tag{28}$$

Interestingly, as shown by Blei et al. (2003), $\Psi(\alpha_{q,v}) - \Psi(\alpha_{q,0})$ is the expected value of $\log \theta_v$ with respect to the Dirichlet distribution q, so KL-divergence is minimised by a choice of q where:

$$\forall v : E_q[\log \theta_v] = \lambda E_{p_1}[\log \theta_v] + (1-\lambda) E_{p_2}[\log \theta_v] \tag{29}$$

Given λ and Dirichlet distributions p_1 and p_2, there seems no simple way of finding a distribution q which satisfies (29). However, it is possible to minimise (26) numerically. Some experiments have been conducted in this direction using simple Beta distributions (Dirichlet distributions where there are only two (switch) values.) For example, setting $p_1 = \mathrm{Dir}(\theta_1, \theta_2|1, 4)$, $p_2 = \mathrm{Dir}(\theta_1, \theta_2|3, 5)$ and $\lambda = 0.5$, the R (R Development Core Team 2011) `optim` function was used to find that $\alpha_q = (1.278, 3.242)$ is the minimising choice, giving a KL-divergence of $-H(r) - 0.3651454$. Moment-matching, in contrast produces a value of $\alpha_q = (1.444, 3.579)$ which gives a KL-divergence of $-H(r) - 0.3604162$. Changing λ to 0.1 leads to a minimising choice of $\alpha_q = (2.279, 4.146)$ with a KL-divergence of $-H(r) - 0.3629558$, whereas moment matching here produces $\alpha_q = (2.488, 4.471)$ with a KL-divergence of $-H(r) - 0.3607265$. In both cases the minimising solution did indeed satisfy (29) as expected.

These two comparisons are typical of results that have been obtained by preliminary numerical experimentation: moment-matching does not minimise KL-divergence, but it approximates the minimising choice quite well. Given the computational savings of the moment-matching approach it has been decided to use this as the merging method in this paper. Another less pragmatic argument in favour of moment-matching is given by Cowell

(1998) who argues that the KL-divergence between distributions is the wrong score to minimise. Instead the KL-divergence between the *predictive performance* of the distributions is what should be minimised. PRISM programs can be used to make predictions about likely explanations x or likely observations y. The predictive performance of approximating q when r is the true distribution is $\sum_x r(x) \log \frac{r(x)}{q(x)}$ for explanations and $\sum_y r(y) \log \frac{r(y)}{q(y)}$ for observations.

4.3 Approximating the marginal likelihood

The main focus of this paper is on approximating the posterior distribution $P(\boldsymbol{\theta}|\mathbf{y})$, but another important quantity to approximate is

$$P(\mathbf{y}) = \int_{\boldsymbol{\theta}} P(y|\boldsymbol{\theta}) P(\boldsymbol{\theta})\, d\boldsymbol{\theta}$$

the *marginal likelihood*. This can be used as a score for the statistical model encoded by the structure of the PRISM program. By integrating over possible values of the parameters it avoids the danger of overfitting which a fitted likelihood score has. Marginal likelihood can be multiplied by a prior probability to get (up to normalisation) a posterior probability for a PRISM program structure.

Given its importance for model choice it is not surprising that there is existing work on computing marginal likelihood for PRISM programs. Sato et al. (2008) have used a variational approach to approximating $P(\mathbf{y})$, which is implemented in the current version of PRISM. In this approach the joint distribution $P(\boldsymbol{\theta}, \mathbf{x}|\mathbf{y})$ over parameters and hidden data is approximated by a product $q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) q_{\mathbf{x}}(\mathbf{x})$ and an EM-like algorithm is used to find choices for $q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$ and $q_{\mathbf{x}}(\mathbf{x})$ which optimise this approximation. More recently, Sato (2011) has used MCMC to approximate $P(\mathbf{y})$.

With the current approach online approximation of $P(\mathbf{y})$ is straightforward. We have $P(\mathbf{y}) = P(y_1)P(y_2|y_1)\ldots P(y_T|y_1\ldots y_{T-1})$. Let $P(\boldsymbol{\theta}|y_1\ldots y_{t-1})$ be approximated by a mixture $\sum_\ell P(\ell) \prod_i \mathrm{Dir}(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_{\ell,i})$ then from (10) we have:

$$P(y_t|y_1\ldots y_{t-1}) \approx \sum_\ell P(\ell) P(y_t|\ell) = \sum_\ell P(\ell) \sum_{x:f(x)=y_t} \prod_i \frac{B(\boldsymbol{\alpha}_{\ell,i} + \boldsymbol{C}_i(x))}{B(\boldsymbol{\alpha}_{\ell,i})} \tag{30}$$

So the marginal likelihood can be approximated easily using quantities like $\frac{B(\boldsymbol{\alpha}_{\ell,i}+\boldsymbol{C}_i(x))}{B(\boldsymbol{\alpha}_{\ell,i})}$ which have to be computed anyway.

4.4 Implementation of the sequential algorithm

Due to the computationally demanding nature of the task, the fastest logic programming language available was used, namely Mercury (Somogyi et al. 1996). In all cases Mercury was used in a standard fashion: the Mercury compiler generated low-level C which was compiled to native code by gcc. The GNU scientific library was linked in to provide log beta functions. A collection of 6 Mercury modules were developed: `model.m`, `prior.m`, `data.m`, `sequential.m`, `params.m` and `vectors.m`. The first three of these are problem-specific and define, respectively, the PRISM model, prior distribution and observed data for a particular Bayesian inference problem. The other modules implement the sequential approximation algorithm. These modules 'know' the bare minimum about the PRISM model: just the types of observed datapoints, the number and type of switches and (via a predicate

```
:- module model.
:- interface.

:- import_module list.
:- import_module vectors.

:- type hmm_symbol ---> a;b.
:- type datum == list(hmm_symbol).
:- type explanation_counts == vectors.ivecs22222.

:- pred model(datum,explanation_counts).
:- mode model(in,out) is multi.
```

Fig. 2 Interface for the HMM model

`model/2` exported from `model.m`) which $\boldsymbol{C}_i(x)$ are associated with a particular datapoint. This modularity ensures the algorithm can be applied to any (failure-free) PRISM program. An additional advantage of using Mercury is that it is possible to use the Mercury compiler to check that a PRISM program is indeed failure-free. To do this one writes the `model/2` predicate to be a relation between data and explanations and declares that if an explanation is given as input, exactly one datum can result as output (a `det` mode declaration). Note that in this paper, `model.m` has been written as a relation between data and explanation *counts* since this is slightly more efficient if there is no need to check that there is no failure.

5 Results for the approximate sequential approach

This section reports on some initial empirical results using the approximate sequential approach. Most results were produced using Mercury version rotd-2010-04-17, configured for i686-pc-linux-gnu using a dual-core 3 GHz machine running Linux. In all cases only a single core was used. For some of the bigger experiments it was necessary to increase the default size of the Mercury `det` stack using the runtime `-detstack-size-kwords` option.

5.1 An initial experiment

In a first experiment a single datapoint `hmm([b,b,a,a,a])` was sampled from `hmm.psm` (Fig. 1) using the standard PRISM implementation. A prior with a single product of Dirichlet distributions was used. The parameter vector for this product of Dirichlet distributions was $\boldsymbol{\alpha}_1 = ((1,1),(1,1),(1,1),(1,1),(1,1))$ for the switches `init`, `tr(s0)`, `tr(s1)`, `out(s0)`, `out(s1)`. In what follows, the parameter vector for all products of Dirichlet distributions will be given in this order. For individual Dirichlet distributions the parameters for the switch values will be given in lexicographic order so that, for example, a parameter vector of $(2,3)$ for switch `tr(s0)` has the 2 corresponding to the value `s0` and the 3 corresponding to `s1`. For those familiar with Mercury the interface of `model.m` is given in Fig. 2. Note that the `multi` mode declaration for `model/2` ensures that a compile-time error is thrown if the Mercury compiler cannot establish that any possible `datum` y has at least one explanation x (and thus at least one vector of counts $\boldsymbol{C}_i(x)$, $i = 1,\ldots,n$). Note also that the model is a 'black box' which returns vectors of counts in response to datapoints, the approach cannot know that the model is an HMM and so does not exploit any known properties of HMMs.

```
                      init     tr(s0)   tr(s1)   out(s0) out(s1)
0.0786713286713288    ((2, 1), (2, 2), (1, 4), (1, 3), (4, 1))
0.0786713286713288    ((1, 2), (4, 1), (2, 2), (4, 1), (1, 3))
0.0629370629370632    ((2, 1), (6, 1), (1, 1), (4, 3), (1, 1))
0.0629370629370632    ((1, 2), (1, 1), (1, 6), (1, 1), (4, 3))
0.05664335664335645   ((2, 1), (1, 2), (1, 5), (1, 2), (4, 2))
0.05664335664335645   ((1, 2), (5, 1), (2, 1), (4, 2), (1, 2))
0.028321678321678295  ((2, 1), (4, 2), (2, 1), (3, 3), (2, 1))
0.028321678321678295  ((1, 2), (1, 2), (2, 4), (2, 1), (3, 3))
0.026223776223776234  ((2, 1), (2, 2), (2, 3), (1, 3), (4, 1))
0.026223776223776234  ((2, 1), (1, 4), (3, 1), (3, 2), (2, 2))
```

Fig. 3 Ten most probable products of the exact posterior distribution using 1 datapoint drawn from `hmm.psm` (Took 0.013 seconds)

Table 1 Mean values of HMM probabilities according to the posterior distribution conditional on data `[a,b,a,b,b]`, `[a,b,a,a,b]`, `[a,b,a,a,a]`, `[a,a,a,a,a]` and three approximations to it. Table headings have been abbreviated, so that, for example, `init = s0` is short for $E[P(\texttt{init} = \texttt{s0})]$

	`init=s0`	`tr(s0)=s0`	`tr(s1)=s0`	`out(s0)=a`	`out(s1)=a`
Exact	0.5000	0.4660	0.5340	0.6487	0.6487
$K = 100$	0.5001	0.4662	0.5334	0.6410	0.6402
$K = 100$	0.5003	0.4676	0.5327	0.6593	0.6589
$K = 10$	0.4924	0.4643	0.5342	0.6857	0.6885

The approximate sequential algorithm was run with a component limit set to 1,000,000 mixture components. There are $2^6 = 64$ explanations for a single datapoint and 44 distinct explanation count vectors. Since the number of components in the posterior is thus only 44 this limit had the effect of imposing no approximation, and so an exact representation of the posterior was obtained. The ten most probable components are show in Fig. 3. The first thing to note is the symmetrical nature of the posterior. The components come in equally probable pairs. These pairs correspond to explanations with the two hidden states swapped round. Evidently, since this symmetry is obvious from our knowledge of HMMs, this could be exploited. Here this is not done since the goal is to test the algorithm and implementation in the general case.

5.2 Comparing approximate and exact solutions

In a second HMM experiment the dataset `[a,b,a,b,b]`, `[a,b,a,a,b]`, `[a,b,a,a,a]`, `[a,a,a,a,a]` was used. Since the posterior distribution in this case only has 10,445 components it was possible to compute this exactly by setting a very high component limit. To examine the accuracy of the sequential approach, it was run twice more with different orderings of the datapoints but a component limit of 100 in each case. In addition a single run using a component limit of just 10 was done. Using the exact posterior and the three approximations, mean values of the following probabilities: $P(\texttt{init} = \texttt{s0})$, $P(\texttt{tr(s0)} = \texttt{s0})$, $P(\texttt{tr(s1)} = \texttt{s0})$, $P(\texttt{out(s0)} = \texttt{a})$ and $P(\texttt{out(s1)} = \texttt{a})$ were computed. The results are given in Table 1 and show that there has been little loss in accuracy in producing these mean values when $K = 100$ but that accuracy is noticeably worse when $K = 10$.

To get further insight into the quality of the approximation the exact posterior density and the first $K = 100$ approximate density were evaluated at 3,125 evenly spaced

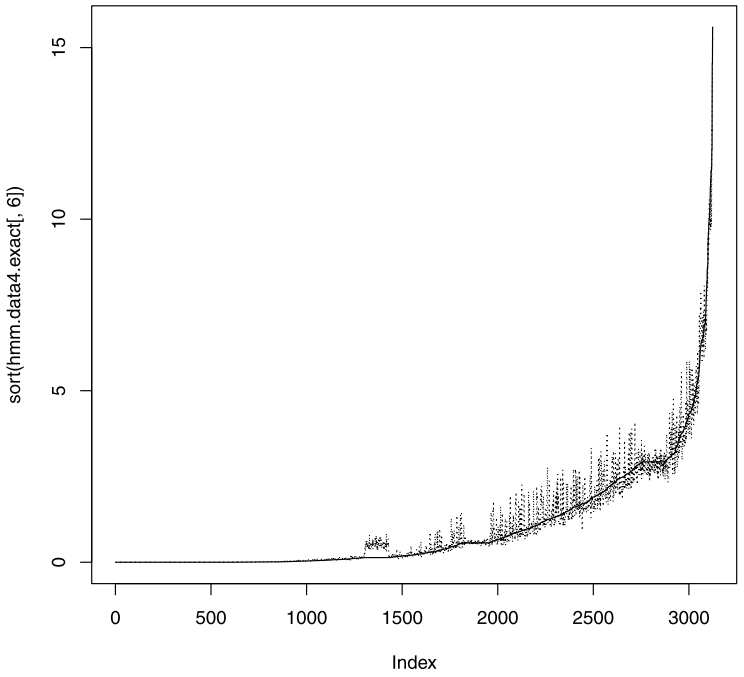

Fig. 4 Exact (*solid line*) and approximate (*dotted line*) density values for 3,125 points

points in the support of the distributions. These points were ordered according to their exact posterior density value and their exact and approximate density values were plotted against each other in Fig. 4. As the plot indicates the approximation is reasonably good. In particular, the highest exact density point from the 3,125 evaluated is for $P(\texttt{init} = \texttt{s0}) = 0.1$, $P(\texttt{tr(s0)} = \texttt{s0}) = 0.3$, $P(\texttt{tr(s1)} = \texttt{s0}) = 0.9$, $P(\texttt{out(s0)} = \texttt{a}) = 0.5$ and $P(\texttt{out(s1)} = \texttt{a}) = 0.9$ with a value of 15.59. This is also the highest point for the $K = 100$ approximation which gives it a value of 15.32.

In a third experiment, a larger dataset of 10 points (also sampled from `hmm.psm`) was used with the same prior. Note that there are $(2^6)^{10} = 1.2 \times 10^{18}$ possible joint explanations. A component limit of just 10 components was used. The final approximation to the posterior took 9.4 seconds to compute and is shown in Fig. 5. The experiment was then re-run but with the order of the datapoints reversed. Evidently such a reversal does not alter the true posterior but may impact on an approximation which uses a sequential approach. The final approximation from the run with reversed data are shown in Fig. 6. A different, albeit similar, approximation has been produced.

In a final experiment 100 datapoints were sampled from the HMM with parameters set to these values: $P(\texttt{init} = \texttt{s0}) = 0.9$, $P(\texttt{tr(s0)} = \texttt{s0}) = 0.4$, $P(\texttt{tr(s1)} = \texttt{s0}) = 0.8$, $P(\texttt{out(s0)} = \texttt{a}) = 0.2$ and $P(\texttt{out(s1)} = \texttt{a}) = 0.7$. The algorithm was run with component limits of $K = 100$ and $K = 200$, which took 93 s and 388 s, respectively. In both cases the resulting mixture components were then split into two sets depending on whether the mean value for $P(\texttt{init} = \texttt{s0})$ was above or below 0.5. In effect this creates a 2-component mixture where each component is itself a mixture distribution. For both of these 2 components one can compute mean values for the 5 probabilities mentioned above and compute the component weight. These results are provided in Table 2 where $<$ indicates the compo-

```
          init       tr(s0)       tr(s1)       out(s0)     out(s1)
0.520 ((6.0,6.0),(14.5,12.3),(12.6,14.5),(10.9,16.0),(14.1,13.0))
0.248 ((7.4,4.6),(19.1,11.3),(10.0,13.6),(11.5,18.9),(13.5,10.1))
0.087 ((5.8,6.2),(16.9,13.6),(13.7, 9.7),(15.7,14.9),( 9.3,14.1))
0.042 ((5.4,6.6),( 9.9,14.3),(15.0,14.7),(10.7,13.6),(14.3,15.4))
0.023 ((3.4,8.6),(11.3, 9.4),(12.0,21.3),(10.7,10.1),(14.3,18.9))
0.019 ((4.7,7.3),(17.4, 7.5),( 9.4,19.6),(10.3,14.7),(14.7,14.3))
0.019 ((6.2,5.8),(17.8,11.2),(11.1,13.9),(16.0,12.9),( 9.0,16.1))
0.018 ((4.7,7.3),(16.2, 8.7),(10.0,19.1),(13.5,11.4),(11.5,17.6))
0.017 ((4.9,7.1),(10.6,12.4),(13.0,18.0),(13.2, 9.8),(11.8,19.2))
0.007 ((7.9,4.1),(17.9,13.8),(12.4, 9.9),(10.7,21.0),(14.3, 8.0))
```

Fig. 5 Products of Dirichlet distributions computed using the approximate sequential approach using 10 datapoints drawn from `hmm.psm` and with a component limit of 10 (Took 9.4 seconds)

```
          init       tr(s0)       tr(s1)       out(s0)     out(s1)
0.525 ((6.5,5.5),(13.1,12.6),(12.0,16.3),(11.7,14.0),(13.3,15.0))
0.116 ((7.1,4.9),(16.3,11.0),( 9.3,17.4),(13.6,13.7),(11.4,15.3))
0.093 ((4.8,7.2),(15.1,12.2),(13.9,12.8),(15.9,11.4),( 9.1,17.6))
0.068 ((4.9,7.1),(18.0, 8.4),(10.2,17.4),(11.5,15.0),(13.5,14.0))
0.068 ((6.2,5.8),(16.7,10.6),(11.0,15.7),( 8.6,18.7),(16.4,10.3))
0.055 ((5.0,7.0),(13.8,12.5),(13.9,13.9),(10.3,15.9),(14.7,13.1))
0.040 ((5.1,6.9),( 9.4,13.7),(14.5,16.4),(12.1,11.0),(12.9,18.0))
0.017 ((4.8,7.2),( 9.7,12.4),(13.6,18.4),(6.91, 5.1),(18.1,13.9))
0.017 ((8.2,3.8),(19.1,13.5),(11.6, 9.8),(16.2,16.4),( 8.8,12.6))
0.001 ((8.7,3.3),(18.4,13.0),( 9.9,12.7),(14.5,16.9),(10.5,12.1))
```

Fig. 6 Products of Dirichlet distributions computed using the approximate sequential approach using 10 datapoints drawn from `hmm.psm` and with a component limit of 10 Datapoints have reverse order from those used to produce the results in Fig. 5

Table 2 Mean values of HMM probabilities according to various distributions. Table heading have been abbreviated, so that, for example, `init = s0` is short for $E[P(\texttt{init} = \texttt{s0})]$

K		w	`init=s0`	`tr(s0)=s0`	`tr(s1)=s0`	`out(s0)=a`	`out(s1)=a`
100	$<$	0.528	0.074	0.391	0.812	0.618	0.152
100	$>$	0.471	0.921	0.196	0.610	0.146	0.627
200	$<$	0.490	0.081	0.410	0.821	0.611	0.147
200	$>$	0.510	0.923	0.240	0.620	0.144	0.653

nent containing all the original components with a mean for $P(\texttt{init} = \texttt{s0})$ below 0.5 and $>$ contains those where this mean is above 0.5.

It is clear that in both the $K = 100$ and $K = 200$ case two (near) equally weighted (near) 'symmetrical' distributions have been returned; with the hidden states being swapped between them. Note, for example that the mean value for $P(\texttt{out(s0)} = \texttt{a})$ in the 1st row is close to that of $P(\texttt{out(s1)} = \texttt{a})$ in the second. The $<$ components are the 'correct' ones since the true probability of $P(\texttt{init} = \texttt{s0})$ is indeed above 0.5. Note that in these components the mean value of $P(\texttt{init} = \texttt{s0})$ is very close to the true value of 0.9 and the means for the emission probabilities are also good estimates. The transition probabilities are less well estimated by these mean values though.

```
0.8 :: edge(a, c).   0.7 :: edge(a, b).   0.8 :: edge(c, e).
0.6 :: edge(b, c).   0.9 :: edge(c, d).   0.5 :: edge(e, d).
```

Fig. 7 Probabilistic graph taken from (Gutmann et al. 2010)

```
:- module model.
:- interface.

:- import_module bool.
:- import_module vectors.

:- type vertex ---> a;b;c;d;e.
:- type datum == {vertex,vertex,bool}.
:- type explanation_counts == vectors.ivecs222222.

:- pred model(datum,explanation_counts).
:- mode model(in,out) is multi.
```

Fig. 8 Interface for the probabilistic graph model

5.3 Probabilistic graph model

In a second experiment an example used to illustrate the ProbLog system (Gutmann et al. 2010) was used. Given a probabilistic directed graph, such as the one represented in Fig. 7, there is a certain probability of a path between any two vertices. One can imagine probabilistically generating graphs by asserting each of the 6 possible edges in Fig. 7 with the associated probability. Once a graph has been generated it is simple to check whether there is a path between any two given vertices.

For the associated parameter estimation problem we have datapoints each of which is an ordered pair of vertices together with a Boolean value which indicates whether the two vertices are joined by some path. For example `a,d,yes` indicates that `a` and `d` are joined by a path, `a,d,no` would indicate that they are not. We can imagine that each such datapoint is generated by probabilistically generating a graph, taking an externally given ordered pair of vertices and then checking whether a path exists between the vertices. Note that here, in contrast to the HMM, there is no target predicate such that exactly one of its ground instances is true in each 'possible world'. Instead for each pair of vertices `x`, `y` either `(x,y,yes)` is true or `(x,y,no)` is—the model is discriminative whereas the HMM model is generative. For those familiar with Mercury the interface of `model.m` is given in Fig. 8.

A dataset of size 100 was produced by repeatedly generating a random graph according to the probabilities in Fig. 7, choosing an ordered pair of vertices and determining whether they were connected. Each ordered pair was uniformly selected from the set of all those which are connected by a path if all edges are present. This set is:

$$\{(a,b),(a,c),(a,d),(a,e),(b,c),(b,d),(b,e),(c,d),(c,e),(e,d)\}$$

The sequential approximation algorithm was run twice, with the order of the datapoints being reversed the second time. A single product of Dirichlet distributions was used as the prior, with each Dirichlet having both parameters set to 1. The component limit was 200. In contrast to earlier experiments Mercury 11.07 was used on 2.5 GHz laptop. The two runs took 370 s and 342 s.

Posterior means for all edge probabilities were computed for both runs and the results are given in Table 3. It is no surprise that these mean values are not always close to the true values given that only 100 datapoints were used. For example, the ordered pair (e,d)

Table 3 Mean values for edge probabilities as estimated by two runs of the sequential algorithm with the order of the data reversed between runs

$a \to c$	$b \to c$	$a \to b$	$c \to d$	$c \to e$	$e \to d$
0.872	0.512	0.536	0.628	0.830	0.799
0.835	0.469	0.515	0.611	0.782	0.768

appears 6 times in the data where, by chance, it has the label `yes` 5 times, even though this label has only 0.5 probability. The positive result is that these mean values are reasonably stable between the two runs.

6 Conclusions and future work

The principal contributions of this paper are theoretical. Starting from the easy result that priors which are mixture of Dirichlet products lead to posteriors of the same form, we move on to an analysis of the quality of the approximation effected by component merging. An upper bound on the KL-divergence for any such approximation is found although unfortunately this result has not been exploited to optimise the method. However, in the case of optimising the merger of any two given Dirichlet distributions an expression has been derived which can be solved numerically. This has allowed a comparison with the computationally convenient moment-matching method. In addition, it has been shown that marginal likelihood can be computed almost as a by-product of the presented sequential approximation approach. Some initial results have been produced using a Mercury implementation. Unfortunately space constraints prevent an account of the many interesting issues that arise when implementing a PRISM program as just a particular sort of Mercury program.

However, it is clear that much remains to be done. Most obviously extensive empirical evaluation would be useful, not least to compare marginal likelihood estimates with those produced using variational (Sato et al. 2008) and MCMC (Sato 2011) methods. It would also be useful to remove the restriction that each single datapoint does not produce very many explanations. A possible solution to this would be to search for only high weight explanations rather than just find all of them. Finally, although it generalises PRISM in an important respect, the ProbLog system (Gutmann et al. 2010) still satisfies (2) and so it would be interesting to do further investigation into using the method presented here for parameter estimation of ProbLog programs.

Acknowledgements Thanks to the anonymous reviewers for their comments and suggestions.

References

Bernardo, J. M., & Girón, F. J. (1988). A Bayesian analysis of simple mixture problems. In J. M. Bernardo, M. H. DeGroot, D. V. Lindley, & A. F. M. Smith (Eds.), *Bayesian statistics* (Vol. 3, pp. 67–78). London: Oxford University Press.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research*, *3*, 993–1022.

Cover, T. M., & Thomas, J. A. (1991). *Elements of information theory*. New York: Wiley.

Cowell, R. G., Dawid, A. P., & Sebastiani, P. (1995). A comparison of sequential learning methods for incomplete data. In J. M. Bernado, J. Berger, A. P. Dawid, & A. F. M. Smith (Eds.), *Bayesian statistics* (Vol. 5, pp. 533–541). Oxford: Clarendon Press.

Cowell, R. G. (1998). Mixture reduction via predictive scores. *Statistics and Computing*, *8*, 97–103.

Cowell, R. G., Dawid, A. P., Lauritzen, S. L., & Spiegelhalter, D. J. (1999). *Probabilistic networks and expert systems*. New York: Springer.

Cussens, J. (2001). Parameter estimation in stochastic logic programs. *Machine Learning*, *44*(3), 245–271.
Cussens, J. (2005). Integrating by separating: combining probability and logic with ICL, PRISM and SLPs. APRIL project report.
Cussens, J. (2007). Model equivalence of PRISM programs. In *Proceedings of the Dagstuhl seminar: probabilistic, logical and relational learning—a further synthesis*.
Cussens, J. (2011). Approximate Bayesian computation for the parameters of PRISM programs. In P. Frasconi & F. A. Lisi (Eds.), *Proc. 20th international conference on inductive logic programming (ILP 2010)* (pp. 38–46). Firenze: Springer.
Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (1995). *Bayesian data analysis*. London: Chapman & Hall.
Gutmann, B., Kimmig, A., Kersting, K., & De Raedt, L. (2010). Parameter estimation in ProbLog from annotated queries. Technical Report CW 583, Katholieke Universiteit Leuven, Belgium.
Muggleton, S. (1996). Stochastic logic programs. In L. De Raedt (Ed.), *Frontiers in artificial intelligence and applications: Vol. 32*. *Advances in inductive logic programming* (pp. 254–264). Amsterdam: IOS Press.
Penny, W. D. (2001). KL-divergences of Normal, Gamma, Dirichlet and Wishart densities. Technical report, University College London.
R Development Core Team (2011). *R: A language and environment for statistical computing*. Vienna: R Foundation for Statistical Computing. ISBN 3-900051-07-0
Sato, T. (2011). A general MCMC method for Bayesian inference in logic-based probabilistic modeling. In *Proceedings of the twenty-second international joint conference on artificial intelligence (IJCAI-2011)* (pp. 1472–1477).
Sato, T., & Kameya, Y. (2001). Parameter learning of logic programs for symbolic-statistical modeling. *The Journal of Artificial Intelligence Research*, *15*, 391–454.
Sato, T., Kameya, Y., & Kurihara, K. (2008). Variational Bayes via propositionalized probability computation in PRISM. *Annals of Mathematics and Artificial Intelligence*, *54*, 135–158.
Sato, T., Kameya, Y., & Zhou, N.-F. (2005). Generative modeling with failure in PRISM. In *Proceedings of the nineteenth international joint conference on artificial intelligence (IJCAI-05)*, Edinburgh, August.
Somogyi, Z., Henderson, F., & Conway, T. (1996). The execution algorithm of Mercury: an efficient purely declarative logic programming language. *The Journal of Logic Programming*, *29*(1–3), 17–64.
Toni, T., Welch, D., Strelkowa, N., Ipsen, A., & Stumpf, M. P. H. (2009). Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of the Royal Society Interface*, *6*(31), 187–202.
West, M. (1992). Modelling with mixtures. In J. M. Bernado, J. O. Berger, A. P. Dawid & A. F. M. Smith (Eds.), *Bayesian statistics* (Vol. 4, pp. 503–524). Oxford: Clarendon Press.