Online Bayesian inference for the parameters of PRISM programs

James Cussens

Dept of Computer Science & York Centre for Complex Systems Analysis
University of York
Deramore Lane, York, YO10 5GE, UK
jc@cs.york.ac.uk

Abstract. This paper presents a method and working implementation for approximating posterior distribution 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. Due to the heavy computational burden of this approach, the method has been implemented in the Mercury logic programming language. The method is evaluated using a hidden Markov model example.

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 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 approximate methods for cases where the exact approach is infeasible.

2 PRISM

A PRISM program is a logic program together with a probabilistic built-in predicate `msw/2`. 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 associated with the *switch name* `init`. All other predicates behave in the normal logic programming fashion.

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; its values are defined by the structure of program, the associated probabilities are parameters of the PRISM program. 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. 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 in [2] 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.

Usually a particular *target predicate* is distinguished in a PRISM program. In this paper, we assume that the target predicate is defined so that exactly one ground atomic formula with it as predicate symbol is logically entailed by any given joint instantiation of the switches. Such ground instances are viewed as *outputs* of the PRISM program and will be generically denoted by y. It is a PRISM requirement that associated with each output there is a finite set of *explanations*. An explanation is a finite collection of switch instantiations which together logically imply the observed datum. Because an explanation x determines an output y, the logical structure of the PRISM program defines a function f such that $f(x) = y$. This can generalised to vectors of explanations determining vectors of outputs: $f(\mathbf{x}) = \mathbf{y}$.

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

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. So this is a 'hidden data' situation. If Dirichlet prior distributions are used for multinomial switch distributions then,

because the switch values are not observed, the posterior will be a mixture of Dirichlet distributions, with one component for each possible value of the hidden switch instantiations. This motivates using mixtures of Dirichlet distributions as priors. Let $\mathbf{y} = y_1, \ldots, y_T$ be the observed data. Let

$$P(\boldsymbol{\theta}) = \sum_{\ell=1}^{L} P(\ell) P(\boldsymbol{\theta}|\ell) \tag{1}$$

be the prior distribution where each $P(\boldsymbol{\theta}|\ell)$ is a product of Dirichlet distributions:

$$P(\boldsymbol{\theta}|\ell) = \prod_i \mathrm{Dir}(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_{\ell,i}) \tag{2}$$

where $\boldsymbol{\theta}_i$ are the parameters for the ith switch, and $\boldsymbol{\alpha}_{\ell,i}$ is the associated vector of Dirichlet parameters. $P(\ell)$ defines 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*. Due to space constraints it is necessary to just state what the posterior distribution is:

$$P(\boldsymbol{\theta}|\mathbf{y}) = \frac{1}{P(\mathbf{y})} \sum_{\ell=1}^{L} \frac{P(\ell)}{\prod_i B(\boldsymbol{\alpha}_{\ell,i})} \sum_{\mathbf{x}:f(\mathbf{x})=\mathbf{y}} \prod_i \mathrm{Dir}(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_{\ell,i}+C_i(\mathbf{x})) \prod_i B(\boldsymbol{\alpha}_{\ell,i}+C_i(\mathbf{x})) \tag{3}$$

In (3), B is the multinomial Beta function, $\mathbf{x}$ denotes a vector of hidden explanations for the observed data vector $\mathbf{y}$ and $C_i(\mathbf{x})$ are the counts associated with switch i associated with $\mathbf{x}$.

4 Sequential approximate computation of posterior distributions for PRISM parameters

Following (3) the posterior $P(\boldsymbol{\theta}|\mathbf{y})$ could be eventually computed exactly by finding each $\mathbf{x}$ such that $f(\mathbf{x}) = \mathbf{y}$, constructing the $|\{\mathbf{x} : f(\mathbf{x}) = \mathbf{y}\}| \times L$ products of Dirichlet distributions and computing their weights. If $\prod_i \mathrm{Dir}(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_{\ell,i}+C_i(\mathbf{x})) = \prod_i \mathrm{Dir}(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_{\ell',i}+C_i(\mathbf{x}'))$ for some $\ell, \ell', \mathbf{x}, \mathbf{x}'$ then evidently the components are identical and can be merged into one by adding the weights. In a final step the weights can be normalised, effectively computing $P(\mathbf{y})$.

Despite the possibility of merging identical mixture components, the final number of mixture components will typically be too large for this exact approach to be of practical use. Instead an approximate sequential approach will be taken. The idea is to compute (approximations to) the following sequence of posterior distributions: $P(\boldsymbol{\theta}|y_1), P(\boldsymbol{\theta}|y_1, y_2), \ldots P(\boldsymbol{\theta}|y_1, y_2 \ldots y_T)$. The key to this idea is that (1) if a prior is a mixture of products of Dirichlet distributions then so will the posterior and that (2) the posterior $P(\boldsymbol{\theta}|y_1, \ldots, y_t)$ can be constructed by taking the 'prior' $P(\boldsymbol{\theta}|y_1, \ldots, y_{t-1})$ and conditioning on the single observation 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. This approach to mixture reduction was used in [1] and the important details of how merging is done and what counts as 'nearest' is described there.

To give more detail: for each y_t, *all* explanations x_t are searched for, although only the associated count vector $\bigotimes_i C_i(x_t)$ is recorded. It follows that for this method to be practical the number of explanations for any single datapoint cannot be too great. Concerning memory requirements, let $n(y_t)$ be the number of distinct values of $\bigotimes_i C_i(x_t)$, then the number of mixture components can temporarily grow to $Kn(y_t)$ immediately prior to the reduction just mentioned.

Due to the computationally demanding nature of the task, the fastest logic programming language available was used, namely Mercury [3]. 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 latter 3 are not problem-specific. `sequential.m` implements the main sequential updating algorithm, `params.m` just records the desired limit on components (K) and `vectors.m` contains utility predicates.

5 Results for the approximate sequential approach

This section reports empirical results using the approximate sequential approach. All results were produced using Mercury version rotd-2010-04-17, configured for i686-pc-linux-gnu. The machine was a dual-core 3GHz 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.

In a first experiment a single datapoint `hmm([b,b,a,a,a])` was sampled from a PRISM encoding of a simple 2-state 2-symbol hidden Markov model which uses 5 switches:`init`, `tr(s0)`, `tr(s1)`, `out(s0)`, `out(s1)`. These are, respectively, switches for: initial state selection, transitions from states `s0` and `s1` and emissions from these two states. A prior with a single product of Dirichlet distributions was used, its parameter vector being: $((1,1),(1,1),(1,1),(1,1),(1,1))$.

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 1 with the mixture weights in the leftmost column. 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.

	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. 1. Ten most probable products of the exact posterior distribution using 1 datapoint drawn from the HMM (Took 0.013 seconds)

To test the accuracy of the approximation method a larger dataset of 10 points 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 2.

	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. 2. Products of Dirichlet distributions computed using the approximate sequential approach using 10 datapoints drawn from the HMM and with a component limit of 10 (Took 9.4 seconds)

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 3. A different, albeit similar, approximation has been produced.

```
          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. 3. Products of Dirichlet distributions computed using the approximate sequential approach using 10 datapoints drawn from the HMM and with a component limit of 10 Datapoints have reverse order from those used to produce the results in Fig 2.

A mixture component limit of 100,000 was then used (on the same data) and an approximation to the posterior with this number of components was produced in 197 seconds. It is notable that the two most probable components account for less that 0.5% of the probability mass. Space prevents any further analysis of these results.

6 Conclusions

The initial results presented here are encouraging in that acceptable results have been produced for modest-sized problems. Evidently, it remains to conduct a proper empirical comparison with competing approaches to approximate Bayesian inference for PRISM parameters.

References

1. R. G. Cowell, A. P. Dawid, and P. Sebastiani. A comparison of sequential learning methods for incomplete data. In J. M. Bernado, J. Berger, A. P. Dawid, and A. F. M. Smith, editors, *Bayesian Statistics 5*, pages 533–541. Clarendon Press, Oxford, 1995.
2. Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15:391–454, 2001.
3. Zoltan Somogyi, Fergus Henderson, and Thomas Conway. The execution algorithm of Mercury: an efficient purely declarative logic programming language. *Journal of Logic Programming*, 29(1–3):17–64, October-December 1996.