Supplementary materials for this article are available online. Please click the TAS link at http://pubs.amstat.org.

Statistical Computing and Graphics

Using Infer.NET for Statistical Analyses

S. S. J. WANG and M. P. WAND

We demonstrate and critique the new Bayesian inference package Infer.NET in terms of its capacity for statistical analyses. Infer.NET differs from the well-known BUGS Bayesian inference packages in that its main engine is the variational Bayes family of deterministic approximation algorithms rather than Markov chain Monte Carlo. The underlying rationale is that such deterministic algorithms can handle bigger problems due to their increased speed, despite some loss of accuracy. We find that Infer.NET is a well-designed computational framework and offers significant speed advantages over BUGS. Nevertheless, the current release is limited in terms of the breadth of models it can handle, and its inference is sometimes inaccurate. Supplemental materials accompany the online version of this article.

KEY WORDS: Bayesian inference; Expectation propagation; Mean field approximation; Variational Bayes.

1. INTRODUCTION

Infer.NET (Minka et al. 2010) is a new computational framework for approximate Bayesian inference in hierarchical Bayesian models. The first beta version of Infer.NET was released in December 2008. Infer.NET can be downloaded from *research.microsoft.com/infernet*. This website also contains a two-minute mini-clip that explains the essential features of Infer.NET. It points out, for example, that Infer.NET can be used from any of the so-called .NET languages, a family that includes C♯, C++, Visual Basic, and Iron Python. We work with the first of these languages in the present article. At the time of this writing, the current version of Infer.NET is 2.4 Beta 2 and all advice given in this article is based on that version. Since Infer.NET is in its infancy it is anticipated that new and improved versions will be released quite regularly in the coming years.

S. S. J. Wang is Doctoral Candidate, Centre for Statistical and Survey Methodology, School of Mathematics and Applied Statistics, University of Wollongong, Wollongong 2522, Australia. M. P. Wand is Distinguished Professor, School of Mathematical Sciences, University of Technology, Sydney, Broadway 2007, Australia (E-mail: *matt.wand@uts.edu.au*). The authors are grateful for advice received from Tom Minka and John Ormerod during the course of this project. The authors also thank an associate editor and three referees for their feedback on earlier versions of this article. This research was supported in part by Australian Research Council Discovery Project DP0877055.

Many readers from the statistical community will be familiar with the BUGS (Bayesian inference Using Gibbs Sampling) software products (Spiegelhalter et al. 2003), with WinBUGS (Lunn et al. 2000) being the most popular of these. Infer.NET is similar to BUGS in that both facilitate the fitting of hierarchical Bayesian models. They differ in their methods for approximate inference. BUGS uses Markov chain Monte Carlo (MCMC) samples from the posterior distributions of parameters of interest. Infer.NET instead uses deterministic approximation methods, known as *variational message passing* and *expectation propagation*, to approximate posterior distributions. Deterministic approximate inference methods have the advantage of being quite fast in comparison with MCMC, and not requiring laborious convergence checks. However, they can be considerably less accurate than MCMC—with the latter having the advantage of improved accuracy through larger samples. Infer.NET has a Gibbs sampling option, which means that it can also perform MCMC-based approximate inference. However, this is for a much narrower class of models compared with BUGS.

Variational message passing (VMP) (Winn and Bishop 2005) is a special case of *variational Bayes* (also known as *mean field approximation*) which, in turn, is a special case of variational approximation. The basic idea of variational Bayes is to approximate joint posterior densities such as $p(\alpha, \beta, \gamma|\text{observed data})$ by product density forms such as

$$\begin{aligned} &\text{(a)} \quad q_{\alpha,\beta}(\alpha,\beta)q_\gamma(\gamma), \\ &\text{(b)} \quad q_\alpha(\alpha)q_{\beta,\gamma}(\beta,\gamma), \qquad \text{or} \\ &\text{(c)} \quad q_\alpha(\alpha)q_\beta(\beta)q_\gamma(\gamma). \end{aligned}$$

The choice among (a), (b), and (c) usually involves a trade-off between tractability and minimal imposition of product assumptions. Once the form is settled upon, the optimal q-densities are chosen to minimize the Kullback–Liebler distance from the exact joint posterior. Further details are provided by Bishop (2006, chap. 10) and Ormerod and Wand (2010). VMP takes advantage of the analytic forms that arise for exponential family distributions with conjugate priors.

Expectation propagation (EP) is a different class of deterministic approximation methods. An early reference is the work of Minka (2001) although similar approaches such as *assumed density filtering* and *moment matching* have a longer history. For certain models, EP has been seen to achieve greater accuracy than VMP (e.g., Bishop 2006, sec. 10.7.1). On the other hand, models for which EP admits analytic solutions are fewer compared with VMP.

 DOI: 10.1198/tast.2011.10169

While Infer.NET is geared toward machine learning applications it is, nonetheless, a Bayesian inference engine and therefore has numerous potential statistical applications. In this article we demonstrate and critique the use of Infer.NET in statistical analyses. We start with four elementary examples, and then treat two advanced statistical problems: multivariate classification and additive model analysis. Each example is accompanied by a C♯ script, which is available in the supplemental materials that accompany the online version of this article.

We find Infer.NET to be a well-structured framework and are attracted by it being a script-based environment. However, most statisticians will find its model specification syntax to be less friendly than that of BUGS. Compared with BUGS, it is also limited in terms of the scope of statistical models it currently can handle, and it is inherently less accurate. The main advantage of Infer.NET is speed, and our examples demonstrate that it can be significantly faster than BUGS—especially for larger problems.

We introduce the use of Infer.NET for statistical analyses via four simple examples in Section 2. Two advanced examples are described in Section 3. Section 4 explain options for usage of Infer.NET. Some brief comparisons with BUGS are made in Section 5. Our overall evaluation of the current version of Infer.NET as a tool for statistical analysis is given in Section 6.

2. FOUR SIMPLE EXAMPLES

We start with four examples involving simple Bayesian models. The first of these is Bayesian simple linear regression. We then describe extensions to (a) binary responses, and (b) random effects. Our last example in this section is concerned with the classical finite normal mixture fitting problem. The simplicity of the examples allows the essential aspects of Infer.NET to be delineated more clearly.

All continuous variables are first transformed to the unit interval and and weakly informative hyperparameter choices are used. The resulting approximate posterior densities are then back-transformed to the original units.

Each example has an accompanying C♯ program containing calls to Infer.NET classes. The full scripts are contained in the supplemental materials that accompany the online version of this article. We highlight the salient code here. An Appendix provides additional details.

The notation $x_i \overset{\text{ind.}}{\sim} D_i$, $1 \le i \le n$, means that the random variables x_i have distribution D_i and are mutually independent. The notation $x \sim \text{Gamma}(A, B)$ means that the random variable x has a Gamma distribution with shape parameter $A > 0$ and rate parameter $B > 0$. The corresponding density function is $p(x) \propto x^{A-1}e^{-Bx}$ for $x > 0$. We use $\mathbf{y}$ to denote the $n \times 1$ vector with entries $y_1, \ldots, y_n$. Analogous notation applies to other vectors.

2.1 Simple Linear Regression

Our first example involves the Bayesian simple linear regression model

$$y_i|\beta_0, \beta_1, \tau \overset{\text{ind.}}{\sim} N(\beta_0 + \beta_1 x_i, \tau^{-1}), \qquad 1 \le i \le n, \quad (1)$$

$$\beta_0, \beta_1 \overset{\text{ind.}}{\sim} N(0, \sigma_\beta^2), \qquad \tau \sim \text{Gamma}(A, B), \quad (2)$$

where $\sigma_\beta^2, A, B > 0$ are hyperparameters to be specified by the analyst.

A full description of Infer.NET fitting of this model is given in an Appendix to this article. Here we will confine discussion to the Infer.NET code for specification of (1) and (2). Specification of the prior distributions (2) is achieved through the following Infer.NET code:

```
Variable<double> beta0 =
  Variable.GaussianFromMeanAndVariance(0.0,
  sigsqBeta).Named("beta0");
Variable<double> beta1 =
  Variable.GaussianFromMeanAndVariance(0.0,
  sigsqBeta).Named("beta1");
Variable<double> tau =
  Variable.GammaFromShapeAndScale(A,
  1/B).Named("tau");
```
(3)

For example, the last command corresponds to the prior specification $\tau \sim \text{Gamma}(A, B)$. But since B is a rate parameter, rather than a scale parameter, its reciprocal appears as the second argument in the call to `Variable.GammaFromShapeAndScale()`. The likelihood (1) is specified via the loop-type structure:

```
Range index = new Range(n).Named("index");
VariableArray<double> x =
  Variable.Array<double>(index).Named("x");
VariableArray<double> y =
  Variable.Array<double>(index).Named("y");
VariableArray<double> mu =
  Variable.Array<double>(index).Named("mu");
mu[index] = beta0 + beta1*x[index];
y[index] =
  Variable.GaussianFromMeanAndPrecision(
  mu[index],tau);
```
(4)

The first of these commands sets up the variable `index` to range over the integers between 1 and n. Note that `index` corresponds to the subscript i in (1). The next three commands set up arrays of size n for storage of the x_i, y_i, and $\mu_i \equiv \beta_0 + \beta_1 x_i$. The fifth command accordingly fills up the array `mu`. The last line corresponds to the specification $y_i|\mu_i, \tau \overset{\text{ind.}}{\sim} N(\mu_i, \tau^{-1})$. The `mu` array is needed because Infer.NET does not permit algebraic forms in the argument list of `Variable.GaussianFromMeanAndPrecision()`.

The joint posterior density of the model parameters $p(\beta_0, \beta_1, \tau|\mathbf{y})$ does not have a closed-form expression and Infer.NET will fit the product density approximation

$$p(\beta_0, \beta_1, \tau|\mathbf{y}) \approx q_{\beta_0}(\beta_0) q_{\beta_1}(\beta_1) q_\tau(\tau). \quad (5)$$

The milder product density restriction

$$p(\beta_0, \beta_1, \tau|\mathbf{y}) \approx q_{\beta_0,\beta_1}(\beta_0, \beta_1) q_\tau(\tau) \quad (6)$$

can be achieved by treating the regression coefficients as a block, that is, working with $\boldsymbol{\beta} = [\beta_0 \ \ \beta_1]^T$ rather than the individual coefficients. The required Infer.NET code matches the following alternative expression of (1) and (2):

$$y_i|\boldsymbol{\beta}, \tau \sim N(\boldsymbol{\beta}^T \mathbf{x}_i, \tau^{-1}),$$

$$\boldsymbol{\beta} \sim N(\mathbf{0}, \sigma_\beta^2 \mathbf{I}), \tau \sim \text{Gamma}(A, B),$$

where $\mathbf{x}_i = [1 \;\; x_i]^T$. The prior for $\boldsymbol{\beta}$ is specified via

```
PositiveDefiniteMatrix SigmaBeta =
  PositiveDefiniteMatrix.IdentityScaledBy(2,
  sigsqBeta);
Variable<Vector> beta =
  Variable.VectorGaussianFromMeanAndVariance(
  new Vector(new double[]{0.0,0.0}),
  SigmaBeta).Named("beta");
```

while the likelihood specification is

```
Range index =
  new Range(n).Named("index");
VariableArray<double> y =
  Variable.Array<double>(index).Named("y");
VariableArray<Vector> xvec =
  Variable.Array<Vector>(index).Named("xvec");
y[index] =
  Variable.GaussianFromMeanAndPrecision(
  Variable.InnerProduct(
  beta,xvec[index]),tau);
```

The Appendix explains how Infer.NET produces approximate posterior distributions for data on the age and price of $n = 39$ Mitsubishi cars (source: Smith 1998) with the hyperparameters set at $\sigma^2_{\boldsymbol{\beta}} = 10^8$, $A = B = 0.01$.

The standard benchmark for Bayesian inference is MCMC since, with sufficiently high Monte Carlo sample sizes, it produces accurate posterior densities. We ran MCMC with initial samples of size 5000 and then retained samples of size 1 million from the posterior distributions of β_0, β_1, and τ. This was achieved using BUGS, with model specification code as follows:

```
for (i in 1:n)
{
  mu[i] <- beta0+beta1*x[i]
  y[i] ~ dnorm(mu[i],tau)
}
tauBeta <- 1/sigsqBeta
beta0 ~ dnorm(0,tauBeta) ;
  beta1 ~ dnorm(0,tauBeta)
tau ~ dgamma(A,B) ; sigsq <- 1/tau
```

The approximate posterior densities produced by Infer.NET and BUGS are shown in Figure 1, but for the error variance $\sigma^2 \equiv 1/\tau$ instead of τ. As explained in the Appendix, the Infer.NET posterior densities have simple algebraic expressions. The BUGS posterior densities are based on kernel density estimation with plug-in bandwidth selection, obtained via the R package KernSmooth (Wand and Ripley 2010). For a sample size of 1 million such estimates are very accurate.

Figure 2 displays the fitted regression line, pointwise credible intervals, and Bayesian prediction intervals.

Several comments are in order. First, the error variance posterior approximation is unaffected by the type of variational Bayes restriction. But this is far from the case for the regression coefficients. The comparisons with the accurate MCMC-based posterior approximations, given in Figure 1, demonstrate that variational Bayes approximation (6) is quite accurate, but variational Bayes approximation (5) is poor. The good performance of (6) is to be expected for diffuse independent priors because of the orthogonality between $\boldsymbol{\beta}$ and τ in likelihood-based inference. However, β_0 and β_1 are far from orthogonal, and this affects the accuracy of (5). In particular, both variational Bayes approximation (6) and MCMC lead to (for the pre-transformed data)

$$\text{posterior correlation between } \beta_0 \text{ and } \beta_1 \approx -0.92,$$

but variational Bayes approximation (5) forces this value to zero. The posterior covariance matrix of $\boldsymbol{\beta}$ is poorly approximated under (5), leading to posterior density functions with incorrect amounts of spread and strange behavior in the 95% pointwise credible intervals. The prediction intervals are less affected by the differences between (5) and (6) since the error variance posterior contribution dominates.

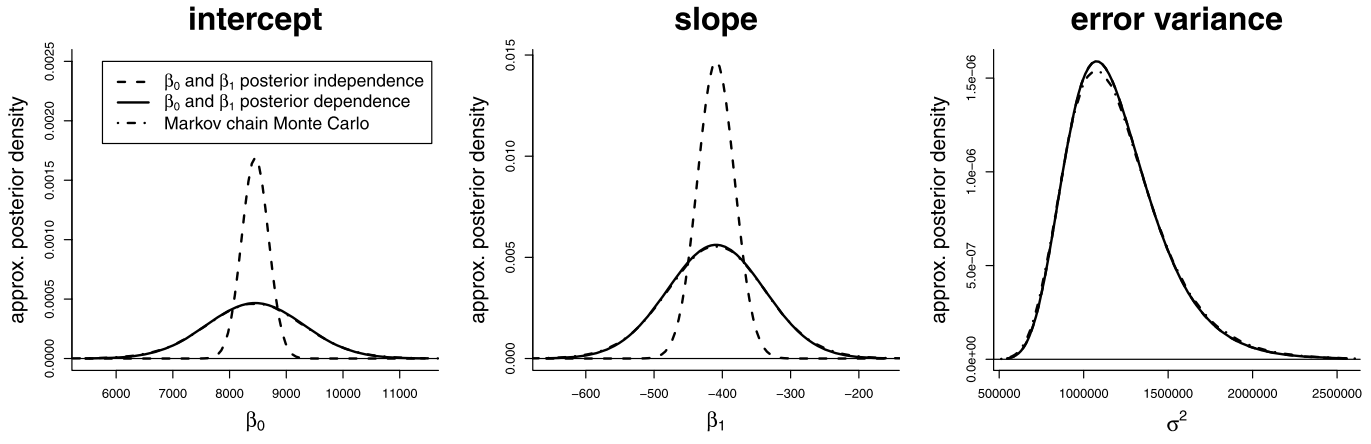

Figure 1. Variational Bayes approximate posterior density functions produced by Infer.NET for the simple linear regression fit to the Mitsubishi car price/age data. The dashed curves correspond to the variational Bayes restriction of posterior independence between β_0 and β_1, given by (5). The solid curves correspond to a relaxation of this restriction, given by (6). The dot-dash curves correspond to MCMC-based posterior density functions. The approximate posterior densities for σ^2 are identical under both (5) and (6).

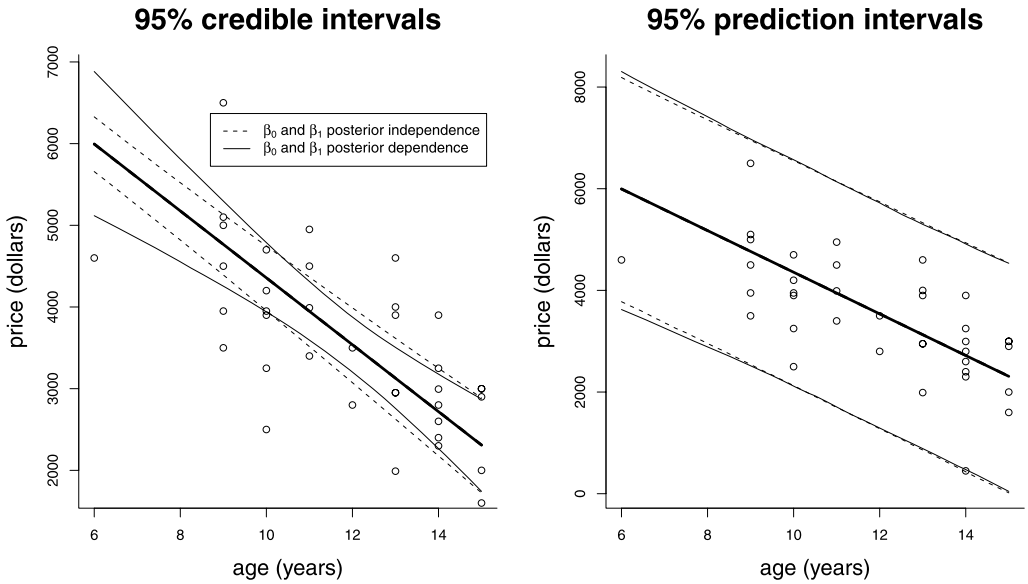

Figure 2. Fitted regression line, pointwise 95% credible intervals, and pointwise 95% Bayesian prediction for data on the age and price of 39 Mitsubishi cars (source: Smith 1998). The dashed-curve intervals are based on variational Bayes restriction (5). The solid-curve intervals are based on variational Bayes restriction (6).

2.2 Binary Response Regression

Suppose we still observe predictor/response pairs (x_i, y_i) but the $y_i \in \{0, 1\}$. Then appropriate regression models take the form

$$P(y_i = 1|\boldsymbol{\beta}) = F(\boldsymbol{\beta}^T \mathbf{x}_i), \qquad \boldsymbol{\beta} \sim N(\mathbf{0}, \sigma_\beta^2 \mathbf{I}),$$

where $F : \mathbb{R} \to (0, 1)$ is an inverse link function. The most common choices of F correspond to *logistic* regression and *probit* regression. Infer.NET is able to handle both varieties of binary response regression, although the most suitable deterministic approximation method differs between the two. Table 1 summarizes this state of affairs, with Φ denoting the standard normal cumulative distribution function.

In Infer.NET, both types of binary regression models require that the response variable is converted to be of type Boolean. The likelihood specification for the logistic regression model is

```
Range index =
  new Range(n).Named("index");
VariableArray<bool> y =
  Variable.Array<bool>(index).Named("y");
VariableArray<Vector> xvec =
  Variable.Array<Vector>(index).Named("xvec");
y[index] = Variable.BernoulliFromLogOdds(
  Variable.InnerProduct(beta,xvec[index]));
```

For probit regression, the last line should be replaced by

```
y[index] = Variable.IsPositive(
  Variable.GaussianFromMeanAndVariance(
  Variable.InnerProduct(beta,xvec[index]),1));
```

Note that the auxiliary variable version of probit regression (Albert and Chib 1993) is being used here.

For logistic regression, the inference engine specification is

```
InferenceEngine engine =
  new InferenceEngine();
engine.Algorithm =
  new VariationalMessagePassing();
```

But for probit regression, the `engine.Algorithm` assignment should be

```
engine.Algorithm =
  new ExpectationPropagation();
```

Figure 3 shows the fitted probability curves and pointwise 95% credible sets for data on birthweight (grams) and indicator of bronchopulmonary dysplasia (BPD) (source: Pagano and Gauvreau 1993) and $\sigma_\beta^2 = 10^8$. The two probability curves are very similar, but the credible interval curves differ substantially for higher birthweights.

2.3 Random Intercept Model

Our second extension of the linear regression model involves the addition of a random intercept, and represents a simple ex-

Table 1. Summary of Infer.NET handling of binary response regression.

Type	Inverse link	Approximation method	Infer.NET engine algorithm name
Logistic	$e^x/(1+e^x)$	variational Bayes	`VariationalMessagePassing()`
Probit	$\Phi(x)$	expectation propagation	`ExpectationPropagation()`

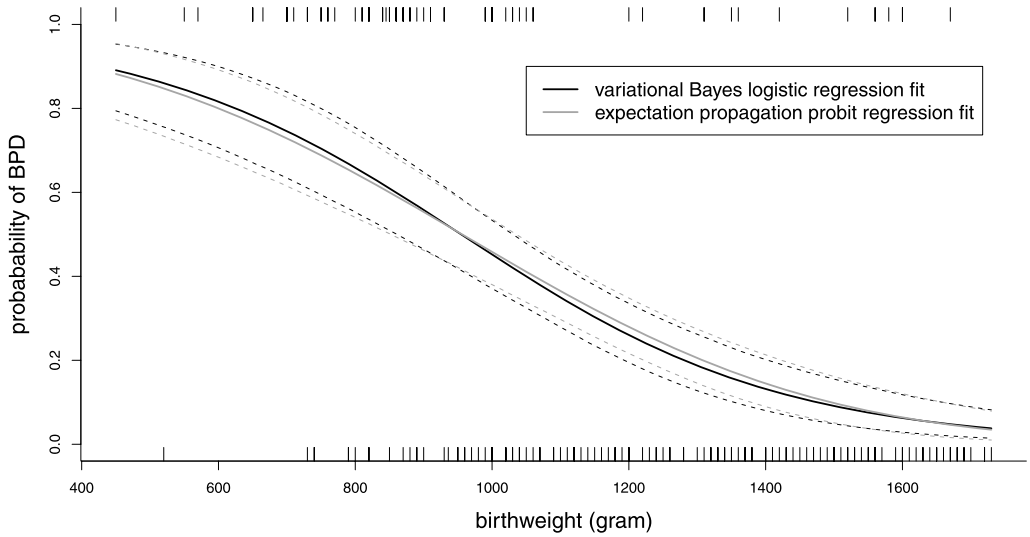

Figure 3. Binary response regression fits to the bronchopulmonary dysplasia (BPD) data obtained using Infer.NET. For each method the solid line corresponds to the posterior probability of BPD for a given birthweight. The dashed lines are pointwise 95% credible sets.

ample of a *mixed* model for grouped data. Specifically, we now consider models of the form

$$\begin{aligned} y_{ij}|\boldsymbol{\beta}, u_i, \tau_\varepsilon &\overset{\text{ind.}}{\sim} N(\boldsymbol{\beta}^T\mathbf{x}_{ij} + u_i, \tau_\varepsilon^{-1}), \\ u_1, \ldots, u_m|\tau_u &\overset{\text{ind.}}{\sim} N(\mathbf{0}, \tau_u^{-1}), \\ \boldsymbol{\beta} &\sim N(\mathbf{0}, \sigma_\beta^2\mathbf{I}), \\ \tau_u &\sim \text{Gamma}(A_u, B_u), \\ \tau_\varepsilon &\sim \text{Gamma}(A_\varepsilon, B_\varepsilon), \end{aligned} \tag{7}$$

where y_{ij} is the jth response measurement in the ith group and m is the number of groups. Note that u_i is a random intercept specific to the ith group. Let $\mathbf{u}$ denote the vector of u_i's.

Figure 4 shows the approximate posterior density obtained from Infer.NET with the variational Bayes product assumption:

$$q_{\boldsymbol{\beta},\mathbf{u},\tau_u,\tau_\varepsilon}(\boldsymbol{\beta}, \mathbf{u}, \tau_u, \tau_\varepsilon) = q_{\boldsymbol{\beta},\mathbf{u}}(\boldsymbol{\beta}, \mathbf{u})q_{\tau_u,\tau_\varepsilon}(\tau_u, \tau_\varepsilon). \tag{8}$$

(It turns out that the *induced* factorization $q_{\tau_u,\tau_\varepsilon}(\tau_u, \tau_\varepsilon) = q_{\tau_u}(\tau_u)q_{\tau_\varepsilon}(\tau_\varepsilon)$ arises in the optimal solution.) The input data correspond to four longitudinal orthodontic measurements on each of $m = 27$ children (source: Pinheiro and Bates 2000). The data are available in the R computing environment (R Development Core Team 2010) via the package nlme (Pinheiro et al. 2009), in the object `Orthodont`. The y_{ij} correspond to distances from the pituitary to the pterygomaxillary fissure (mm) and

$$\boldsymbol{\beta}^T\mathbf{x}_{ij} = \beta_0 + \beta_1\texttt{age}_{ij} + \beta_2\texttt{male}_i,$$

where $\texttt{age}_{ij}$ is the age of the child when y_{ij} was recorded and $\texttt{male}_i$ is an indicator variable for the child being male. Note that the posterior densities are for the variances $\sigma_u^2 \equiv 1/\tau_u$ and $\sigma_\varepsilon^2 \equiv 1/\tau_\varepsilon$ rather than for the precisions τ_u and τ_ε. The MCMC-based posterior density functions, obtained the same way as those shown in Figure 1, are also shown in Figure 4. Variational Bayes is seen to be quite accurate for this model and dataset.

The results shown in Figure 4 actually correspond to a slight modification of model (7), since Infer.NET does not support its direct fitting under product restriction (8). We will now explain, and justify, the modified model. First note that (7) can be written in matrix form as

$$\begin{aligned} \mathbf{y}|\boldsymbol{\beta}, \mathbf{u}, \tau_\varepsilon &\sim N(\mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{u}, \tau_\varepsilon^{-1}), \\ \begin{bmatrix}\boldsymbol{\beta}\\ \mathbf{u}\end{bmatrix}\Big|\tau_u &\sim N\left(\mathbf{0}, \begin{bmatrix}\sigma_\beta^2\mathbf{I} & \mathbf{0}\\ \mathbf{0} & \tau_u^{-1}\mathbf{I}\end{bmatrix}\right), \\ \tau_u &\sim \text{Gamma}(A_u, B_u), \\ \tau_\varepsilon &\sim \text{Gamma}(A_\varepsilon, B_\varepsilon), \end{aligned} \tag{9}$$

where $\mathbf{X}$ contains the $\mathbf{x}_{ij}$ and $\mathbf{Z} = \mathbf{I}_{27} \otimes \mathbf{1}_4$ is the indicator matrix for matching the $\mathbf{x}_{ij}$'s with their corresponding u_i ($1 \le i \le 27, 1 \le j \le 4$). Note that $\mathbf{1}_4$ is the 4×1 vector of ones. The posterior density function of $\boldsymbol{\beta}, \mathbf{u}$ satisfies

$$\begin{aligned} p(\boldsymbol{\beta}, \mathbf{u}|\mathbf{y}) \propto \int_0^\infty \int_0^\infty \exp\Big\{&-\frac{1}{2}\tau_\varepsilon\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta} - \mathbf{Z}\mathbf{u}\|^2 \\ &- \frac{1}{2\sigma_\beta^2}\|\boldsymbol{\beta}\|^2 - \frac{1}{2}\tau_u\|\mathbf{u}\|^2\Big\} p(\tau_\varepsilon)\, p(\tau_u)\, d\tau_\varepsilon\, d\tau_u, \end{aligned}$$

where $p(\tau_\varepsilon)$ and $p(\tau_u)$ are the posterior density functions of τ_ε and τ_u. An alternative model, which employs an auxiliary data vector $\mathbf{a}$, is

$$\begin{aligned} \mathbf{y}|\boldsymbol{\beta}, \mathbf{u}, \tau_\varepsilon &\sim N(\mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{u}, \tau_\varepsilon^{-1}), \\ \mathbf{a}|\boldsymbol{\beta}, \mathbf{u}, \tau_u &\sim N\left(\begin{bmatrix}\boldsymbol{\beta}\\ \mathbf{u}\end{bmatrix}, \begin{bmatrix}\sigma_\beta^2\mathbf{I} & \mathbf{0}\\ \mathbf{0} & \tau_u^{-1}\mathbf{I}\end{bmatrix}\right), \\ \begin{bmatrix}\boldsymbol{\beta}\\ \mathbf{u}\end{bmatrix} &\sim N(\mathbf{0}, \kappa^{-1}\mathbf{I}), \\ \tau_u &\sim \text{Gamma}(A_u, B_u), \\ \tau_\varepsilon &\sim \text{Gamma}(A_\varepsilon, B_\varepsilon), \end{aligned} \tag{10}$$

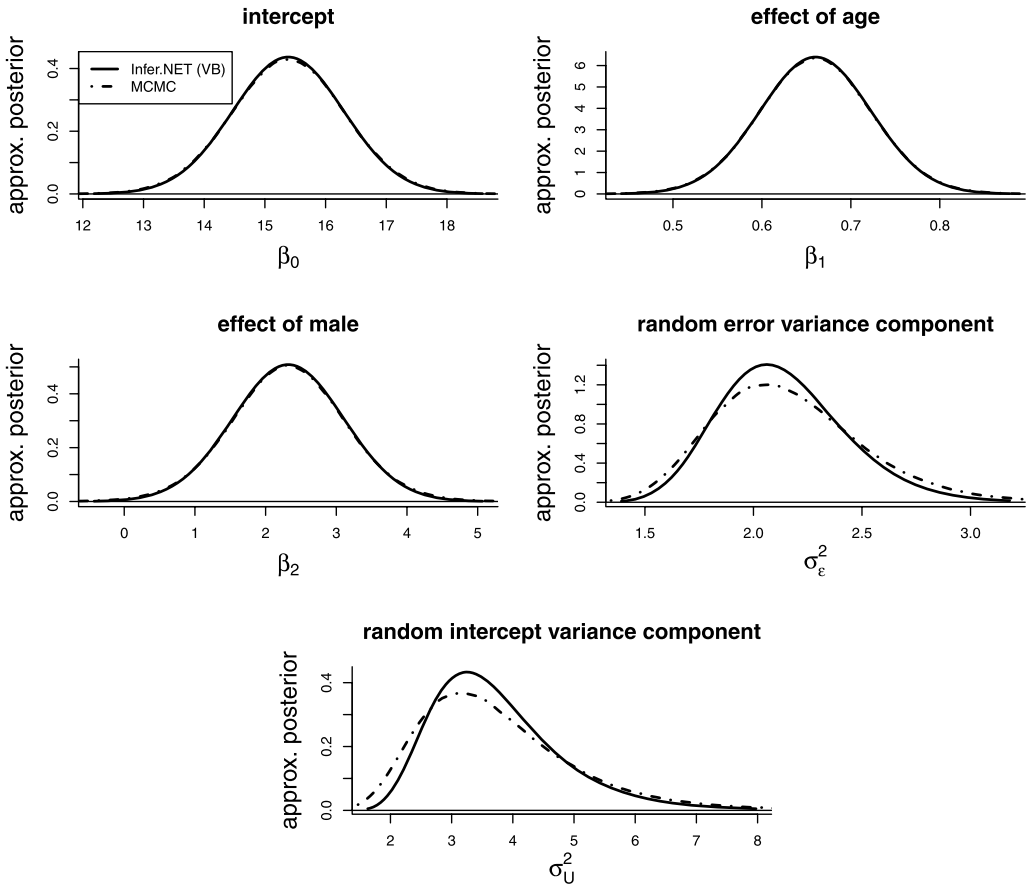

Figure 4. The solid curves are variational Bayes approximate posterior density functions produced by Infer.NET for the simple linear mixed model fit to the orthodontic data. The dot-dash curves correspond to MCMC-based posterior density functions.

where $\kappa > 0$ is a hyperparameter. Let $p_\kappa(\boldsymbol{\beta}, \mathbf{u}|\mathbf{y}, \mathbf{a})$ be the posterior density function of $(\boldsymbol{\beta}, \mathbf{u})$ under model (10). Then

$$p_\kappa(\boldsymbol{\beta}, \mathbf{u}|\mathbf{y}, \mathbf{a}) \propto \int_0^\infty \int_0^\infty \exp\left\{-\frac{1}{2}\tau_\varepsilon \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta} - \mathbf{Z}\mathbf{u}\|^2 - \frac{1+\kappa\sigma_{\boldsymbol{\beta}}^2}{2\sigma_{\boldsymbol{\beta}}^2}\|\boldsymbol{\beta}\|^2 - \frac{1}{2}\tau_u(1+\kappa/\tau_u)\|\mathbf{u}-\mathbf{a}\|^2\right\} p(\tau_\varepsilon)p(\tau_u)\,d\tau_\varepsilon\,d\tau_u.$$

It is apparent from this that

$$\lim_{\kappa\to 0} p_\kappa(\boldsymbol{\beta}, \mathbf{u}|\mathbf{y}, \mathbf{a} = \mathbf{0}) = p(\boldsymbol{\beta}, \mathbf{u}|\mathbf{y}).$$

Similar results hold for the other posterior density functions. Hence, using (10) with κ set to be a very small number and with the auxiliary vector $\mathbf{a}$ set to have an observed value $\mathbf{0}$ leads to essentially the same results. Figure 5 provides directed acyclic graph representations of the original and modified models.

2.4 Normal Mixture Model

The fourth simple example involves fitting a density function of the form

$$f(x) = \sum_{k=1}^{K} w_k(2\pi)^{-1/2}\tau_k \exp\left\{-\frac{1}{2}\tau_k(x-\mu_k)^2\right\},$$

$$w_k, \tau_k > 0, \sum_{k=1}^{K} w_k = 1 \tag{11}$$

to a univariate sample $\mathbf{x} = (x_1, \ldots, x_n)$. This is the classic finite normal mixture problem and has an enormous literature (e.g., McLachlan and Peel 2000). A variational Bayes algorithm for fitting (11) is described in section 2.2.5 of the article by Ormerod and Wand (2010) and a description of an appropriate Bayesian model is given there. We describe Infer.NET fitting of the same model here, but with the addition of choosing the number of components K. Following section 10.2.4 of the book by Bishop (2006), we choose K to maximize

$$\log \underline{p}(\mathbf{x}; q) + \log(K!),$$

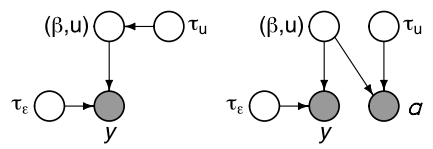

Figure 5. Directed acyclic graph representations of the original Gaussian linear mixed model (left panel) and a modification for implementation in Infer.NET. In the modified model the $(\boldsymbol{\beta}, \mathbf{u})$ node is Gaussian, with mean set to zero and covariance set to a very large multiple of the identity matrix. The shaded nodes correspond to observed data vectors. The $\mathbf{a}$ node is set to be a vector of zeroes.

where $\log \underline{p}(\mathbf{x}; q)$ is the variational Bayes approximation to the marginal log-likelihood. The $\log(K!)$ term accounts for the $K!$ configurations of (w_k, μ_k, σ_k^2) that give rise to the same normal mixture density function.

Infer.NET can compute $\log \underline{p}(\mathbf{x}; q)$ by creating a mixture of the current model with an empty model. The learned mixing weight is then the marginal log-likelihood. Further details on this trick are given in the Infer.NET user guide, where the term *model evidence* is used for $\underline{p}(\mathbf{x}; q)$. We first need to set up an auxiliary Bernoulli variable as follows:

```
Variable<bool> auxML =
  Variable.Bernoulli(0.5).Named("auxML");
```

The code for the normal mixture fitting is then enclosed with

```
IfBlock model = Variable.If(auxML);
```

and

```
model.CloseBlock();
```

The $\log \underline{p}(\mathbf{x}; q)$ is then obtained from

```
double marginalLogLikelihood =
  engine.Infer<Bernoulli>(auxML).LogOdds;
```

Figure 6 shows the results of this Infer.NET-based approach to fitting a finite normal mixture to the data on eruption durations of a geyser which are available in R via the `MASS` package (Venables and Ripley 2011), in the data frame titled `geyser`.

Note that $K = 3$ maximizes $\log \underline{p}(\mathbf{x}; q) + \log(K!)$, as shown in the upper panel of Figure 6. The lower panel shows the $K = 3$ Infer.NET fit. Also shown are 95% pointwise credible sets based on Monte Carlo samples of size 10,000 from the approximate posterior distributions.

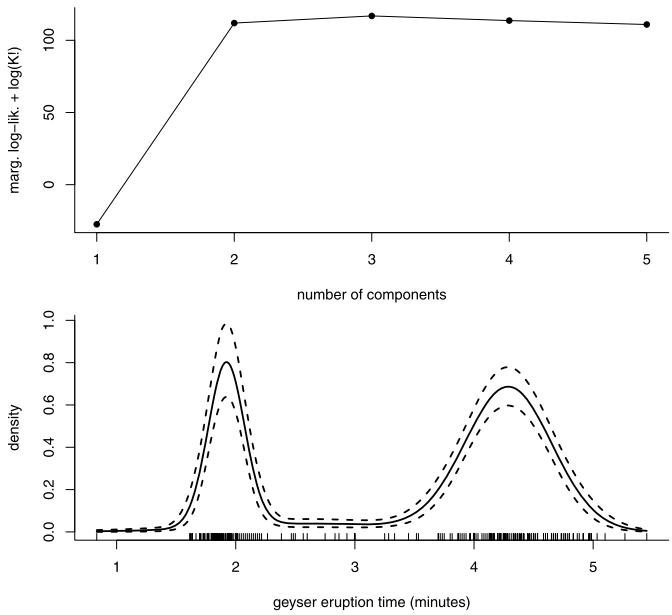

Figure 6. Upper panel: $\log \underline{p}(\mathbf{x}; q) + \log(K!)$ versus K, where K is the number of components in the normal mixture fit to the transformed geyser duration data. Lower panel: fitted normal mixture density for $K = 3$ (the K that maximizes the criterion of the upper panel plot). The dashed curves correspond to pointwise 95% credible sets.

3. TWO ADVANCED EXAMPLES

We now describe two advanced examples which illustrate the capabilities of Infer.NET for more challenging data analyses. The first concerns multidimensional classification. The second involves fitting an additive model with low-rank smoothing splines.

3.1 Classification via Multivariate Finite Mixtures

This example involves classification of glass samples into one of two types based on their refractive index and chemical content. The training data are from the data frame `Glass` in the R package mlbench (Leisch and Dimitriadou 2009). The full dataset involves seven types of glass and nine predictor variables. The first two types (Type 1 and Type 2) comprise 68% of the data with training set sizes of 70 and 76, respectively. We restrict attention to these two classes and use the first seven predictors to build a classifier. These predictors are: refractive index (RI) and content measures of sodium (Na), magnesium (Mg), aluminum (Al), silicon (Si), potassium (K), and calcium (Ca). We fit three-mixture multivariate normal density functions to each sample:

$$f(\mathbf{x}) = \sum_{k=1}^{3} w_k (2\pi)^{-7/2} |\mathbf{T}_k|^{1/2} \times \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \mathbf{T}_k (\mathbf{x} - \boldsymbol{\mu}_k)\right\},$$

where $w_1, w_2, w_3 > 0$ and $w_1 + w_2 + w_3 = 1$. For each of $1 \le k \le 3$, the $\boldsymbol{\mu}_k$ and 7×1 vectors and the $\mathbf{T}_k$ are symmetric positive definite 7×7 matrices. As priors we used

$$(w_1, w_2, w_3) \sim \text{Dirichlet}(\alpha, \alpha, \alpha),$$
$$\boldsymbol{\mu}_k \overset{\text{ind.}}{\sim} N(\mathbf{0}, \sigma_\mu^2 \mathbf{I}), \quad \text{and}$$
$$\mathbf{T}_k \overset{\text{ind.}}{\sim} \text{Wishart}(a, \mathbf{B}).$$

The Dirichlet and Inverse-Wishart notation is such that the respective density functions take the form $p(w_1, w_2, w_3) \propto (w_1 w_2 w_3)^{\alpha - 1}$ for $w_1, w_2, w_3 > 0$ and $w_1 + w_2 + w_3 = 1$, and

$$p(\mathbf{T}_k) \propto |\mathbf{T}_k|^{(a-d-1)/2} \exp\left\{-\frac{1}{2}\text{tr}(\mathbf{B}\mathbf{T}_k)\right\},$$
$$a > 0, \mathbf{T}_k, \mathbf{B} \text{ both positive definite}$$

with $d = 7$. The hyperparameters were set at

$$\alpha = 0.001, \qquad a = 100, \qquad \text{and} \qquad \mathbf{B} = 0.01\mathbf{I}_7.$$

The Infer.NET code for fitting these to the glass data is similar to that used for fitting the univariate normal mixture models to the geyser data, as described in Section 2.4. One difference is the use of Wishart distributions, rather than Gamma distributions, in the precision matrix prior specification:

```
Range indexK =
  new Range(K).Named("indexK");
VariableArray<PositiveDefiniteMatrix>
  Tau = Variable.Array<PositiveDefiniteMatrix>
  (indexK).Named("Tau");
```

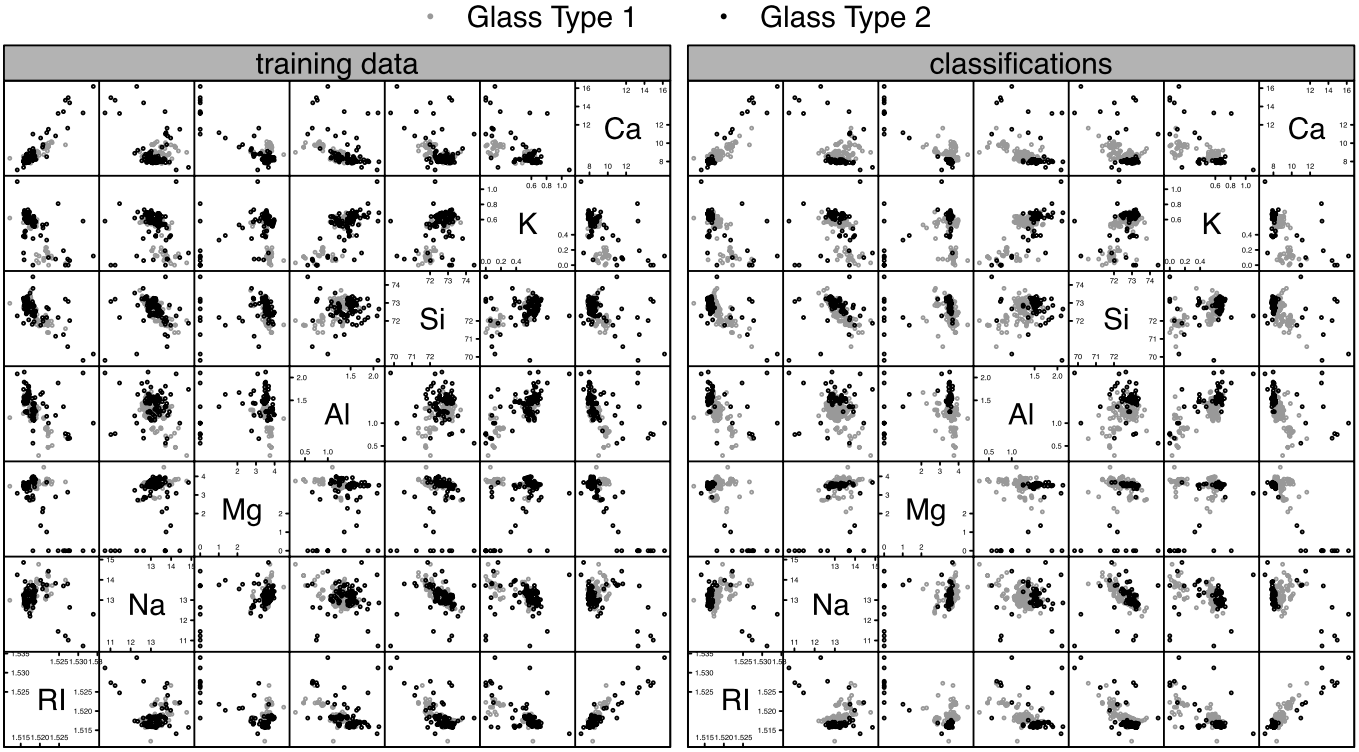

Figure 7. Left: pairwise scatterplots for the training data in the glass classification example described in the text. Right: the classifications of the training data based on Infer.NET fitting of multivariate normal mixture densities to each of the Glass Type 1 and Glass Type 2 samples.

```
Tau[indexK] =
  Variable.WishartFromShapeAndScale(aVal,
  PositiveDefiniteMatrix.IdentityScaledBy(d,
  Bfac)).ForEach(indexK);
```

Figure 7 shows the training data and their classifications according to the Infer.NET normal mixture fits. The training error was 25.3%. An estimate of the test error, obtained using 5-fold cross-validation, is 24.6% with a standard deviation of 9%.

3.2 Normal Additive Model

A three-predictor Normal additive model is

$$y_i = \beta_0 + f_1(x_{1i}) + f_2(x_{2i}) + f_3(x_{3i}) + \varepsilon_i,$$
$$\varepsilon_i \overset{\text{ind.}}{\sim} N(0, \sigma_\varepsilon^2), \tag{12}$$

where, for $1 \le i \le n$, the y_i are measurements on a continuous response variable and (x_{1i}, x_{2i}, x_{3i}) are triples containing measurements on three continuous predictor variables. We will model each of the f_j using low-rank smoothing splines with mixed model representation:

$$f_j(x) = \beta_j x + \sum_{k=1}^{K_j} u_k z_k^{(j)}(x), \qquad u_k \overset{\text{ind.}}{\sim} N(0, \sigma_{uj}^2),$$

where $z_k^{(j)}$ is a set of canonically transformed cubic B-spline basis functions on an interval containing the x_{ji}. Details on computation of the $z_k^{(j)}$ were given by Wand and Ormerod (2008). This model for the f_j corresponds to the type of function estimation performed by the R function `smooth.spline()`. The Bayesian model that we fit in Infer.NET is then

$$\mathbf{y}|\boldsymbol{\beta}, \mathbf{u}, \tau_\varepsilon, \tau_{u1}, \tau_{u2}, \tau_{u3} \sim N(\mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{u}, \tau_\varepsilon^{-1}\mathbf{I}),$$
$$\mathbf{u}|\tau_{u1}, \tau_{u2}, \tau_{u3} \sim N\left(\mathbf{0}, \begin{bmatrix} \tau_{u1}^{-1}\mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \tau_{u2}^{-1}\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \tau_{u3}^{-1}\mathbf{I} \end{bmatrix}\right),$$
$$\boldsymbol{\beta} \sim N(\mathbf{0}, \sigma_\beta^2\mathbf{I}), \qquad \tau_\varepsilon \sim \text{Gamma}(A_\varepsilon, B_\varepsilon),$$
$$\tau_{uj} \sim \text{Gamma}(A_{uj}, B_{uj}), \qquad j = 1, 2, 3,$$

where

$$\mathbf{X} = [\,1 \quad x_{1i} \quad x_{2i} \quad x_{3i}\,]_{1\le i\le n} \qquad \text{and}$$
$$\mathbf{Z} = \left[z_k^{(1)}(x_{1i}) \middle| z_k^{(2)}(x_{2i}) \middle| z_k^{(3)}(x_{3i})\right]_{1\le i\le n}.$$
$$\underset{1\le k\le K_1}{} \quad \underset{1\le k\le K_2}{} \quad \underset{1\le k\le K_3}{}$$

We fit this model to variables in the Ozone data frame in the R package mlbench (Leisch and Dimitriadou 2009). The response variable is daily maximum ozone level and the predictor variables are the inversion base height (feet), pressure gradient to the town of Daggett (mm Hg), and inversion base temperature (degrees Fahrenheit) at Los Angeles International Airport. All variables were transformed to the unit interval for Infer.NET fitting and the hyperparameters were fixed at $\sigma_\beta^2 = 10^8$, $A_\varepsilon = B_\varepsilon = A_{uj} = B_{uj} = 0.01$.

The Infer.NET code is similar to that used for the random intercept model analysis of Section 2.3. The only difference is the

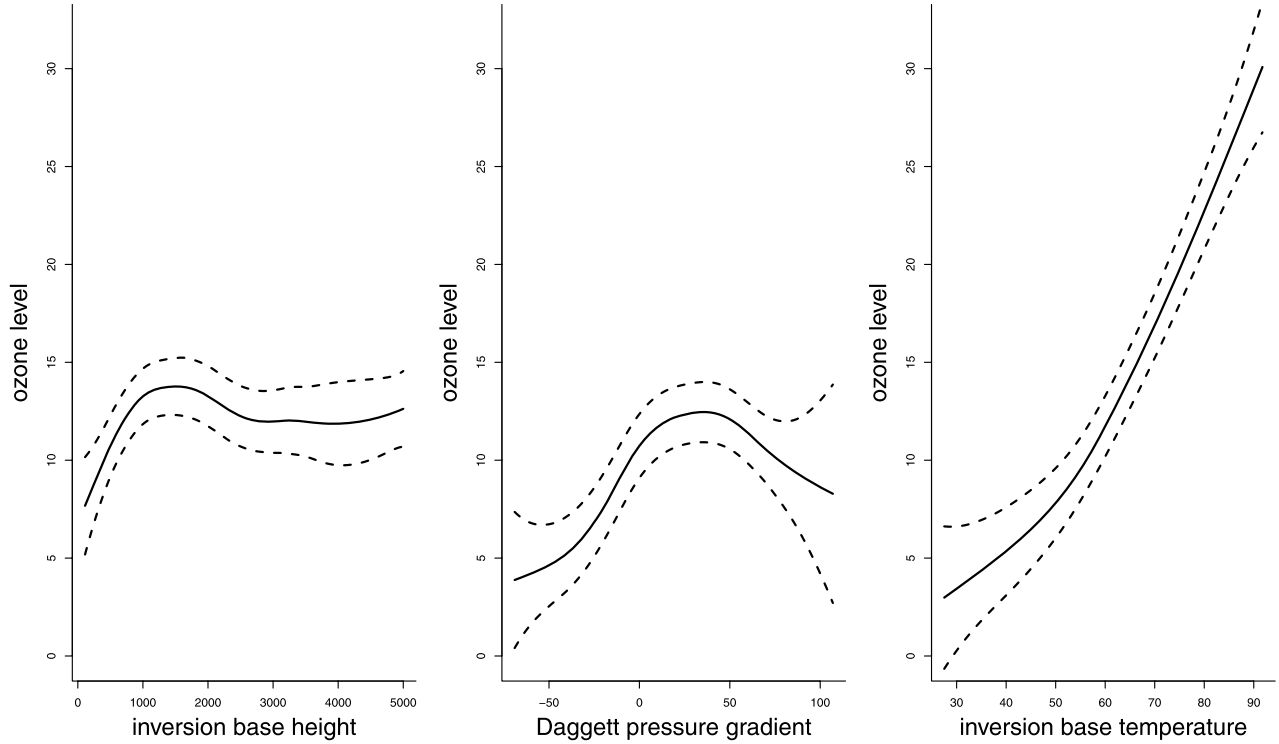

Figure 8. Variational Bayes additive model fits as produced by Infer.NET for California ozone data. The dashed curves correspond to pointwise 95% credible sets.

presence of three random effect dispersion parameters (i.e., τ_{u1}, τ_{u2}, and τ_{u3}) rather than one. Note that the trick encompassed in model (10) also needs to be employed for the current example.

The fitted curves are shown in Figure 8 and show interesting nonlinear effects. Similar fits are obtained using standard additive model software such as the `gam()` function in the mgcv package (Wood 2010).

We have also used Infer.NET to handle generalized response extensions of (12), known as *generalized additive models*. For example, binary response models can be handled by combining the concepts of Sections 2.2 and 3.2.

4. HOW TO USE INFER.NET

The core of Infer.NET is a suite of computer programs compatible with the .NET family of languages, so can be accessed in a variety of ways. In the preparation of this article we have used Infer.NET in two different ways. We now provide brief descriptions.

The first way involves the graphical user interface known as Visual Studio 2008. This facilitates the development of C♯ scripts with calls to various Infer.NET classes. A major advantage of Visual Studio 2008 is its debugging feature, which allows for the efficient development of large and complex scripts. Other features include the ability to generate directed acyclic graphs for models defined by the Infer.NET code and easy access to help pages. The Infer.NET website has further details on this approach.

There is, however, the option to simply write C♯ in a preferred text editor and run the code via DOS commands and tailor-made scripts. We found this approach to be very useful for running the examples in Sections 2.2 and 3.2 since we also used R for preprocessing of the data and postprocessing of the Infer.NET output. This second approach allowed us to conduct each Infer.NET-based analysis with a single R script and corresponding C♯ script.

Further details on the use of Infer.NET are given in an Appendix. There we provide a complete description on Infer.NET of the simple linear regression model (1), (2) under product density restriction (5).

5. COMPARISON WITH BUGS

Of particular interest to statistical analysts is how Infer.NET compares with BUGS. Such comparison is challenging because of factors such as deciding when a particular algorithm has converged and differences in personal tastes with regard to user interface and syntax. Nevertheless, we briefly make some comparative points concerning accuracy, computation time, versatility, and ease of use.

Infer.NET is inherently inaccurate since it relies on deterministic approximation methods. The first two panels of Figure 1 show that the posterior density functions produced by Infer.NET can be overly narrow for stringent product density restrictions. BUGS, on the other hand, will produce highly accurate posterior density functions if the MCMC sample sizes are sufficiently large. Comparisons with MCMC results show Infer.NET to be quite good for the regression examples in Sections 2 and 3, provided that the regression coefficients are treated as a block.

Table 2. Computation times in both Infer.NET and BUGS for each of the examples in Sections 2 and 3 on the first author's laptop computer (Windows XP Service Pack 3; 2.67 GHz processor, 2.17 GBytes of random access memory). The times for Infer.NET involve 100 variational Bayes or expectation propagation iterations. The times for BUGS involve MCMC sample sizes of 10,000.

Example from Sections 2 and 3	Comput. time for Infer.NET (100 VB/EP iterations)	Comput. time for BUGS (MCMC samp. size = 10,000)
Simple linear regression	4 seconds	1 second
Simple logistic regression	2 seconds	3 seconds
Simple probit regression	3 seconds	3 seconds
Random intercept model	3 seconds	1 second
Normal mixture model	17 seconds	28 seconds
Multivariate classification	7 seconds	1.4 minutes
Normal additive model	8 seconds	3.8 minutes

Table 2 gives some indication of the relative computing times for the examples of Sections 2 and 3. Making these comparisons completely fair is a tall order, since the point at which convergence occurs for underlying approximation algorithms is contentious and problem dependent. Instead, we just report elapsed computing times with the number of variational Bayes or expectation propagation iterations set to 100 and the MCMC sample sizes set at 10,000, which was sufficient for convergence in these particular examples. All examples were run on the first author's laptop computer (Windows XP Service Pack 3; 2.67 GHz processor, 2.17 GBytes of random access memory). Table 2 reveals that Infer.NET offers considerable speed-ups, compared with BUGS. This is particularly the case for the two advanced examples of Section 3, where the computing times are reduced from minutes to seconds when going from BUGS to Infer.NET.

BUGS is considerably more versatile than the current version of Infer.NET. BUGS supports many more distributional forms for the nodes in the directed acyclic graph architecture that underpin each computational framework. Moreover, BUGS is able to handle virtually any directed acyclic graph involving these distributions and this allows it to handle a wide range of complications, such those arising in missing data and measurement error models. Infer.NET is more restrictive in this regard, as was seen in Section 2.3 where the random intercept model (7) had to be replaced by an approximately equivalent model to allow fitting. While one example, it demonstrates Infer.NET's limitations with regard to versatility. Note, however, that Infer.NET supports jagged arrays, provides marginal likelihood approximations, and allows model comparison via `if` statements.

Last, we comment on ease of use. We believe that typical members of the statistical community will, at first, find Infer.NET more challenging to use in comparison with BUGS. For example, the model specification syntax of BUGS is closer to the mathematical expressions used in Bayesian model formulation. In addition, BUGS now interfaces quite nicely with the popular R computing environment—courtesy of the packages BRugs (Ligges et al. 2009) and `R2WinBUGS` (Sturtz, Ligges, and Gelman 2005; Sturtz et al. 2008). On the other hand, Infer.NET has an advantage in terms of code development and debugging, due to its compatibility with Visual Studio 2008.

6. SUMMARY

We believe that software engines for fast approximate inference, using deterministic methods such as variational Bayes and expectation propagation, will play a big role in statistical analyses of the future. They have the potential to handle the large and ever-growing datasets and models in the current era of rapid technological change. The emergence of Infer.NET, with its script basis and carefully designed syntax, is an important early development in this direction.

Our examples have demonstrated that Infer.NET can be used, effectively, for both basic and advanced statistical analyses and offers significant speed gains over MCMC-based computing frameworks such as BUGS. However, potential users of Infer.NET should be aware that only a moderate proportion of statistical models in current popular use can be handled using Infer.NET. In addition, Infer.NET can produce inaccurate results if, for example, the product restriction for variational Bayes is too stringent.

We hope that this article will lead to useful discourse on the confluence between Infer.NET and statistical analyses.

APPENDIX: FULL DESCRIPTION OF AN INFER.NET ANALYSIS

Here we provide a full description of the first Infer.NET simple linear regression analysis of Section 2.1.

The Mitsubishi age/price data are listed in Table A.1. Let $(\text{age}_i, \text{price}_i)$ denote the ith age/price pair, $1 \le i \le 39$, and define

$$\begin{aligned} x_i &= \{\text{age}_i - \min(\text{age})\}/\{\max(\text{age}) - \min(\text{age})\}, \\ y_i &= \{\text{price}_i - \min(\text{price})\}/\{\max(\text{price}) - \min(\text{price})\}, \end{aligned} \tag{A.1}$$

where, for example, min(age) is the minimum among the age values. For scale-invariance reasons, we work with these unit interval-transformed data in the Bayesian analysis. The first example in Section 2 involves variational Bayesian inference for the model

$$\begin{aligned} y_i|\beta_0, \beta_1, \tau &\overset{\text{ind.}}{\sim} N(\beta_0 + \beta_1 x_i, \tau^{-1}), \qquad 1 \le i \le n, \\ \beta_0, \beta_1 &\overset{\text{ind.}}{\sim} N(0, 10^8), \qquad \tau \sim \text{Gamma}(0.01, 0.01), \end{aligned} \tag{A.2}$$

with the product restriction

$$p(\beta_0, \beta_1, \tau|\mathbf{y}) \approx q_{\beta_0}(\beta_0) q_{\beta_1}(\beta_1) q_\tau(\tau). \tag{A.3}$$

Our first step is to prepare six text files, which we call `sigsqBeta.txt`, `A.txt`, `B.txt`, `x.txt`, `y.txt`, and

Table A.1. Data on the age (years) and price (Australian dollars) of 39 Mitsubishi cars (source: Smith 1998).

Age	Price	Age	Price	Age	Price	Age	Price	Age	Price
13	2950	9	4500	15	2000	11	4500	14	3250
14	2300	14	2800	11	3400	15	1600	10	2500
14	3900	13	1990	7	8999	13	3900	14	2400
12	2800	12	3500	13	4000	10	4200	11	3990
9	5000	9	5100	13	2950	9	6500	13	4600
15	2999	10	3900	10	3250	9	3500	14	450
10	3950	15	2900	9	3950	15	2999	10	4700
14	2995	11	4950	6	4600	14	2600		

`nIterVB.txt`. The first three of these files contain the single numbers 100,000,000, 0.01, and 0.01, respectively, corresponding to the three hyperparameter values in (A.2). The file `x.txt` contains each of the 39 x_i values in a single column. The file `y.txt` has similar storage of the y_i values. The last file, `nIterVB.txt`, contains a single positive integer that specifies the number of variational Bayes iterations. The analysis presented in Section 2.1 uses 100 iterations. We use R to set up these files.

The next step is to run a C♯ script, which we call `simpLinReg1IN.cs`. This script is contained in the supplemental materials that accompany the online version of this article. As mentioned in Section 4, `simpLinReg1IN.cs` can be run from Visual Studio 2008 within the Microsoft Windows operating system. It can also be run from DOS, although quite a bit of setting up is required for this approach.

We will now explain each of the commands in `simpLinReg1IN.cs`. The first set of commands load all required C♯ and Infer.NET libraries:

```
using System;
  using System.Collections.Generic;
using System.Text; using System.IO;
using System.Reflection;
  using MicrosoftResearch.Infer;
using MicrosoftResearch.Infer.Distributions;
using MicrosoftResearch.Infer.Maths;
using MicrosoftResearch.Infer.Models;
```

Next, we specify the full path name of the directory containing the above-mentioned input files. Such a specification is machine-dependent, but takes a form such as

```
string dir =
  "C:\\research\\myInferNETproject\\";
```

The hyperparameters and number of variational Bayes iterations are specified using the commands

```
double sigsqBeta =
  new DataTable(dir+
  "sigsqBeta.txt").DoubleNumeric;
double A = new DataTable(dir+
  "A.txt").DoubleNumeric;
double B = new DataTable(dir+
  "B.txt").DoubleNumeric;
int nIterVB = new DataTable(dir+
  "nIterVB.txt").IntNumeric;
```

The (x_i, y_i) data are read in using the commands

```
DataTable xTable = new DataTable(dir+"x.txt");
DataTable yTable = new DataTable(dir+"y.txt");
```

These commands involve the C♯ function named `DataTable`. This function, which we wrote ourselves, facilitates the input of general rectangular arrays of numbers when in a text file. The code for `DataTable` is part of the online supplemental materials for this article. The model specification code uses the sample size $n = 39$. The script plucks this value off `xTable` via the command

```
int n = xTable.numRow;
```

Specification of prior distributions is next. This is achieved using the commands in code chunk (3) in Section 2.1. The script then uses the code chunk (4) to specify the likelihood. Note that the likelihood specification uses arrays named `x` and `y`. The observed values of these arrays are specified to be those stored in `xTable` and `yTable` via the next set of commands:

```
x.ObservedValue = xTable.arrayForIN;
y.ObservedValue = yTable.arrayForIN;
```

Note that `arrayForIN` is a member of the class `xTable` and stores the data as an array to match the type of `x.Observed Value`.

Infer.NET has three inference engine options: expectation propagation, Gibbs sampling, and variational Bayes. The last of these is named *variational message passing* in Infer.NET. The following commands specify use of this inference engine and the number of iterations:

```
InferenceEngine engine =
  new InferenceEngine();
engine.Algorithm =
  new VariationalMessagePassing();
engine.NumberOfIterations =
  nIterVB;
```

For model (A.2) with product restriction (A.3) it may be shown (e.g., Bishop 2006, chap. 10) that the approximate posterior density function of (β_0, β_1, τ) is a product of two univariate Normal density functions (for β_0 and β_1) and a Gamma density function (for τ). Let $\mu_{q(\beta_0)}$ and $\sigma^2_{q(\beta_0)}$ be the mean and variance for the approximate posterior for β_0.

Define $\mu_{q(\beta_1)}$ and $\sigma^2_{q(\beta_1)}$ analogously. Then the following commands write the fitted values of these parameters to files named `mu.q.beta0.txt`, `sigsq.q.beta0.txt`, `mu.q.beta1.txt`, and `sigsq.q.beta1.txt`:

```
SaveData.SaveTable(
  engine.Infer<Gaussian>(beta0).GetMean(),
  dir+"mu.q.beta0.txt");
SaveData.SaveTable(
  engine.Infer<Gaussian>(beta0).GetVariance(),
  dir+"sigsq.q.beta0.txt");
SaveData.SaveTable(
  engine.Infer<Gaussian>(beta1).GetMean(),
  dir+"mu.q.beta1.txt");
SaveData.SaveTable(
  engine.Infer<Gaussian>(beta1).GetVariance(),
  dir+"sigsq.q.beta1.txt");
```

These commands involve the C♯ function named `SaveTable`, which is a method in the class `SaveData`. We wrote `SaveTable` to facilitate writing the output from Infer.NET to a text file. Let $\alpha_{q(\tau)}$ and $\lambda_{q(\tau)}$ denote the shape and scale parameter for the approximate posterior density function for τ. Then the following command writes $\alpha_{q(\tau)}$, $\lambda_{q(\tau)}$, and $\alpha_{q(\tau)} \times \lambda_{q(\tau)}$ (the posterior mean) to the file `parms.q.sigsq.txt`:

```
SaveData.SaveTable(engine.Infer<Gamma>(tau),
  dir+"parms.q.sigsq.txt");
```

Summary plots, such as those shown in Figure 1, can be made by reading the posterior density parameters into a graphical computing environment such as that provided by R. However, in view of (A.1), the parameters need to be back-transformed prior to plotting so that the posterior densities correspond to the original data.

SUPPLEMENTARY MATERIALS

C♯ (Infer.NET) code: We have prepared a ZIP archive file containing a collection of C♯ scripts and support code. The C♯ scripts call Infer.NET classes to run each of the examples in Sections 2 and 3. The file README.txt describes the contents of the ZIP archive. (WangWandExamples.zip)

[Received August 2010. Revised March 2011.]

REFERENCES

Albert, J. H., and Chib, S. (1993), "Bayesian Analysis of Binary and Polychotomous Response Data," *Journal of the American Statistical Association*, 88, 669–679. [118]

Bishop, C. M. (2006), *Pattern Recognition and Machine Learning*, New York: Springer. [115,120,125]

Leisch, F., and Dimitriadou, E. (2009), "mlbench 1.1. A Collection of Artificial and Real-World Machine Learning Problems," R package, available at *http://cran.r-project.org*. [121,122]

Ligges, U., Thomas, A., Spiegelhalter, D., Best, N., Lunn, D., Rice, K., and Sturtz, S. (2009), "BRugs 0.5: OpenBUGS and Its R/S-PLUS Interface BRugs," available at *http://www.stats.ox.ac.uk/pub/RWin/src/contrib/*. [124]

Lunn, D. J., Thomas, A., Best, N., and Spiegelhalter, D. (2000), "WinBUGS—A Bayesian Modelling Framework: Concepts, Structure, and Extensibility," *Statistics and Computing*, 10, 325–337. [115]

McLachlan, G. J., and Peel, D. (2000), *Finite Mixture Models*, New York: Wiley. [120]

Minka, T. P. (2001), "Expectation Propagation for Approximate Bayesian Inference," in *Proceedings of Conference on Uncertainty in Artificial Intelligence*, Seattle: University of Washington, pp. 362–369. [115]

Minka, T., Winn, J., Guiver, J., and Knowles, D. (2010), "Infer.Net 2.4," Microsoft Research Cambridge, Cambridge, U.K., available at *http://research.microsoft.com/infernet*. [115]

Ormerod, J. T., and Wand, M. P. (2010), "Explaining Variational Approximations," *The American Statistician*, 64, 140–153. [115,120]

Pagano, M., and Gauvreau, K. (1993), *Principles of Biostatistics*, Florence, KY: Duxbury. [118]

Pinheiro, J. C., and Bates, D. M. (2000), *Mixed-Effects Models in S and S-PLUS*, New York: Springer. [119]

Pinheiro, J., Bates, D., DebRoy, S., Sarkar, D., and The R Core Team (2009), "nlme: Linear and Nonlinear Mixed Effects Models," R package version 3.1-93. [119]

R Development Core Team (2010), *R: A Language and Environment for Statistical Computing*, Vienna, Austria: R Foundation for Statistical Computing, ISBN 3-900051-07-0, available at *http://www.R-project.org*. [119]

Smith, P. (1998), *Into Statistics: A Guide to Understanding Statistical Concepts in Engineering and the Sciences* (2nd ed.), Singapore: Springer-Verlag. [117,118,125]

Spiegelhalter, D. J., Thomas, A., Best, N. G., Gilks, W. R., and Lunn, D. (2003), "BUGS: Bayesian Inference Using Gibbs Sampling," Medical Research Council Biostatistics Unit, Cambridge, U.K., available at *http://www.mrc-bsu.cam.ac.uk/bugs*. [115]

Sturtz, S., Gelman, A., Ligges, U., Gorjanc, G., and Kerman, J. (2008), "R2WinBUGS 2.1: A Package for Running WinBUGS From R," R package, available at *http://cran.r-project.org*. [124]

Sturtz, S., Ligges, U., and Gelman, A. (2005), "R2WinBUGS: A Package for Running WinBUGS From R," *Journal of Statistical Software*, 12 (3), 1–16. [124]

Venables, W. N., and Ripley, B. D. (2011), "MASS 7.3. Functions and Datasets to Support Venables and Ripley, 'Modern Applied Statistics With S' (4th edition, 2002)," R package, available at *http://cran.r-project.org*. [121]

Wand, M. P., and Ormerod, J. T. (2008), "On Semiparametric Regression With O'Sullivan Penalized Splines," *Australian and New Zealand Journal of Statistics*, 50, 179–198. [122]

Wand, M. P., and Ripley, B. D. (2010), "KernSmooth 2.23. Functions for Kernel Smoothing Corresponding to the Book: Wand, M. P. and Jones, M. C. (1995), *Kernel Smoothing*," R package, available at *http://cran.r-project.org*. [117]

Winn, J., and Bishop, C. M. (2005), "Variational Message Passing," *Journal of Machine Learning Research*, 6, 661–694. [115]

Wood, S. N. (2010), "mgcv 1.5. Routines for Generalized Additive Models and Other Generalized Ridge Regression With Multiple Smoothing Parameter Selection," R package, available at *http://cran.r-project.org*. [123]