dirichletprocess: An R Package for Fitting Complex Bayesian Nonparametric Models

Gordon J. Ross
School of Mathematics
University of Edinburgh
gordon.ross@ed.ac.uk

Dean Markwick
Department of Statistical Science
University College London
dean.markwick.15@ucl.ac.uk

Abstract

The **dirichletprocess** package provides software for creating flexible Dirichlet processes objects in R. Users can perform nonparametric Bayesian analysis using Dirichlet processes without the need to program their own inference algorithms. Instead, the user can utilise our pre-built models or specify their own models whilst allowing the **dirichletprocess** package to handle the Markov chain Monte Carlo sampling. Our Dirichlet process objects can act as building blocks for a variety of statistical models including and not limited to: density estimation, clustering and prior distributions in hierarchical models.

Keywords: dirichlet process, bayesian inference, nonparametric statistics, R.

1. Introduction

Many applications of statistical models require parametric specifications of the probability distributions of interest – that observations are generated by Normal distributions, that group-level fixed effects can be modeled by Gamma distributions, and so on. However in some situations there will be insufficient prior information and data to justify such parametric assumptions. In this case, nonparametric methods allow a more flexible and robust specification of distributions, so their essential features can be 'learned from the data' rather than specified in advance.

While frequentist nonparametrics has a long history, Bayesian nonparametrics was a relatively dormant field until the mid 1990s. Although much of the theory of nonparametric priors had been worked out in previous decades (Ferguson 1973), computational issues prevented their widespread adoption. This changed with the development of posterior simulation methods such as Metropolis-Hastings (Hastings 1970) and the Gibbs sampler (Geman and Geman 1984) which were first applied to the task of nonparametric density estimation using Dirichlet process (DP) mixtures in a seminal paper by Escobar and West (1995). This kick started research into Bayesian nonparametrics, which has now become one of the most popular research areas in the statistics and machine learning literature. While there are now several widely used models within the field of Bayesian nonparametrics including the Gaussian process, Beta process and Polya trees, the Dirichlet process mixture model (DPMM) has been the most popular due to its wide applicability and elegant computational structure.

At the most basic level, the DPMM can be viewed as an infinite dimensional mixture model

which represents an unknown density $f(y)$ as:

$$f(y) = \int k(y \mid \theta) p(\theta \mid G) \mathrm{d}\theta, \quad G \sim \mathrm{DP}(\alpha, G_0),$$

where $k(\cdot \mid \theta)$ denotes the mixture kernel, and the mixing distribution G is assigned a non-parametric Dirichlet process prior with a base measure G_0 and concentration parameter α. In the most widely used DPMM, the mixture kernel is taken to be Gaussian so that $\theta = (\mu, \sigma^2)$ and $k(y \mid \theta) = N(y \mid \mu, \sigma^2)$ with a conjugate Normal Inverse-Gamma specification for G_0. The infinite dimensional nature of such a model makes it capable of approximating any continuous distribution to an arbitrary degree of accuracy.

The use of DPMMs is not restricted to simply estimating the density of observed data. Instead, the DPMM can be used at any level in a hierarchical model where it is considered necessary to represent a density nonparametrically due to a lack of knowledge about its parametric form. For example, consider a (multilevel) random effects model where there are J groups of observations, with observations in each group j following a Gaussian distribution with a group-specific mean μ_j and a common variance σ^2. To share information across groups, the means μ_j are assumed to be exchangeable and assigned a prior $p(\mu_j)$. If $y_{i,j}$ denotes the i^{th} observation in group j, then the model is:

$$\begin{aligned} y_{i,j} &\sim N(y_{i,j} \mid \mu_j, \sigma^2), \\ \mu_j &\sim p(\mu_j \mid \gamma), \end{aligned}$$

where γ is the (hyper-)parameters of the prior. This model is an example of **partial pooling**, where the inference for each mean μ_j is based on the means of each of the other $J - 1$ groups, allowing information to be shared across groups. As a biased estimation technique, it is related to methods such as the Lasso estimator, and usually results in more accurate estimation than estimating each group mean separately. However, completing the model specification requires choosing a form for the prior distribution of group means $p(\mu_j \mid \gamma)$, which is made more difficult since the group means may not be observable with a high degree of accuracy, particularly when the number of observations is small. In this case, using a nonparametric DPMM specification for $p(\mu_j \mid \gamma)$ would avoid the risks of potentially specifying an inappropriate parametric form.

Typically when working with DPMMs, the posterior distributions are analytically intractable, so inference instead usually involved computational simulation. A variety of simulation algorithms based around Gibbs sampling and Metropolis-Hastings have been developed to draw samples from DPMM posterior distributions, which can then be used for inference. As such, the widespread adoption of DPMMs has to some extent been held back by the level of statistical and programming literacy required to implement use them. The purpose of the **dirichletprocess** package is to provide a unified implementation of these simulation algorithms for a very wide class of DP mixture models which makes it easy to incorporate DPMMs into hierarchical models.

The design philosophy of the **dirichletprocess** package is the polar opposite of the existing **DPpackage** R package which also provides an implementation of DPMMs. The purpose of **DPpackage** is to give a fast implementation of several common tasks where DPs are used, such as density estimation using Gaussian mixtures, and a variety of specific regression models. While the **DPpackage** package is very useful in these situations, it cannot be used for any

applications of DPs which do not fall into one of the pre-specified tasks incorporated in the package, or use mixture kernels or base measures other than those provided.

In contrast, the purpose of the **dirichletprocess** package is not to automate a pre-specified range of tasks, but instead to represent DPMMs as objects in R so that they can be used as building blocks inside user-specified models. The target audience is hence users who are working with (possibly hierarchical) models which uses a DPMM at some stage, and who hence require a DPMM implementation which can be used as a part of a more general model estimation scheme. As such, the number of tasks which can be achieved using the **dirichletprocess** is quite large, although the trade-off is that the functions in this package will be slower than those in **DPpackage** when it comes to the specific models which it implements.

Key features of the **dirichletprocess** package include:

- An implementation of DP mixture models for various types of mixture kernel including the Gaussian, Beta, Multivariate Normal and Weibull.
- Implementation of DP posterior sampling algorithms in both the conjugate and non-conjugate cases.
- A object-based interface which allows the user to work directly with DP objects in R so that they can be incorporated into hierarchical models.

1.1. A Technical Note

The ultimate purpose of this package is to represent Dirichlet process mixture models as objects in R, so that they can be manipulated and used as building blocks. At the time of writing, R currently features three separate object systems (S3, S4 and RC) designed to allow object-orientated programming. This package uses S3. There are two motivations for this design choice which in our opinion outweigh any advantages that come from using any of other of the R object systems.

1. Speed. While R is an excellent programming language which makes carrying out high level statistical analysis easy, its slow speed remains a bottleneck particularly in tasks such as Gibbs Sampling and Monte Carlo Markov Chain (MCMC) sampling which are inherently sequential and cannot be vectorized. While the base R system is already slow, the S4 and Reference Class (RC) object systems systems suffer from further performance hits since they must search for the correct method for each function evaluation (Wickham 2014). S3 suffers a similar slowdown but to a lesser extent. The price we pay in speed we make up for in code comprehension and ease of development.
2. Ease-of-use. A key feature of this package is that users can themselves specify new DP mixture types if the package does not implement the precise specification they desire. The object systems S4 and RC are geared towards intermediary/advanced R programmers and can be intimidating to novices. The design philosophy of this package allows users to override the behaviour of DP objects and create new mixture types without needing to learn the intricacies of any particular object system. The chosen representation where DP objects are simple S3 structures does not require the user to learn anything about the more obscure intricacies of R objects, and instead they can focus purely on writing the R functions to implement the DP models.

Current alternatives for nonparamteric inference include Stan, PyMC3 and Edward. However, whilst all three packages are much more general than the **dirichletprocess** offerings, they do not offer ease of customisation that **dirichletprocess** does. Firstly, Stan (Carpenter, Bob, Gelman, Andrew, Hoffman, Matt, Lee, Daniel, Goodrich, Ben, Betancourt, Michael, Brubaker, Michael A, Guo, Jiqiang, Li, Peter, and Riddell, Allen 2016), does not allow you to specify discrete parameters in models. As Dirichlet process models require cluster labels which are inherently discrete parameters you are unable to build Dirichlet process models directly in Stan. For both the Python libraries Edward and PyMC3, examples exist of building Dirichlet process models. However, these are built on top of TensorFlow and Theano (Tran, Kucukelbir, Dieng, Rudolph, Liang, and Blei 2016; Salvatier, Wiecki, and Fonnesbeck 2016), therefore, being able to build Dirichlet process objects into statistical work flows would require learning these external libraries. Instead our package **dirichletprocess** is written natively in R and abstracts the difficulties away, allowing users to write Dirichlet process models in R code and not worry about computational details.

2. Background information

This section provides background information about the Dirichlet Process, and includes the key mathematical properties around which the sampling algorithms in the **dirichletprocess** package are based. It can be skipped by those who simply want to know how to use the package, which is discussed in Section 3.

It is commonly required to learn the unknown probability distribution F which generated the observations $y_1, \ldots, y_n$. In parametric Bayesian inference, F is assumed to belong to a known family of distributions (such as the Normal or Exponential) with an associated parameter vector θ of finite length and which much be estimated. This leads to the model:

$$\begin{aligned} y_i &\sim F(y_i \mid \theta), \\ \theta &\sim p(\theta \mid \gamma), \end{aligned}$$

where $p(\theta \mid \gamma)$ denotes the prior distribution and γ are the prior parameters. The task of inference then involves finding an appropriate value for θ, which is equivalent to choosing which member of the specified family of distributions gives the best fit to the data.

In practice however, it may not be clear how to choose an appropriate parametric family of distributions for F. If the wrong family is chosen, then conclusions based on the estimated model may be highly misleading. For example, if it is assumed that F has a Normal distribution with unknown parameters $\theta = (\mu, \sigma^2)$ when in fact the true F is heavy-tailed, this can lead to severe underestimation of the probability of extreme events occurring (Coles 2001).

This problem can be avoided by using a nonparametric prior specification which puts positive prior mass on the whole space of probability densities rather than on a subspace spanned by the finite-length parameter vector θ. This allows the estimated F to adapt to the data, rather than being restricted to a particular family of distributions such as the Normal or Exponential. The Dirichlet process (DP) is one of the most widely used Bayesian nonparametric priors, due to its flexibility and computational simplicity. Our aim in this section is not to give a full treatment of the Dirichlet processes, and a reader unfamiliar with them should refer to a standard reference such as Escobar and West (1995). Instead we will only focus on describing the properties of the DP that are directly relevant to their implementation in the

dirichletprocess package.

The basic DP model has the form:

$$\begin{aligned} y_i &\sim F, \\ F &\sim \text{DP}(\alpha, G_0), \end{aligned}$$

where G_0 is known as the **base measure** and encapsulates any prior knowledge that might be known about F. Specifically, it can be shown that $\mathbb{E}[F \mid G_0, \alpha] = G_0$. The concentration parameter α specifies the prior variance and controls the relative contribution that the prior and data make to the posterior, as the following result shows.

Key Property 1: The DP is a conjugate prior in the following sense: if $y_1, \ldots, y_n \sim F$ and $F \sim \text{DP}(\alpha, G_0)$, then:

$$F \mid y_1, \ldots, y_n \sim \text{DP}\left(\alpha + n, \frac{\alpha G_0 + \sum_{i=1}^{n} \delta_{y_i}}{\alpha + n}\right),$$

where δ_{y_i} denotes a point-mass at y_i. In other words, the posterior distribution of F is a weighted sum of the base measure G_0 and the empirical distribution of the data, with the weighting controlled by α.

The DP is a prior distribution over the space of probability distributions. As such, samples from a DP are probability distributions. The stick-breaking representation first introduced by Sethuraman (1994) shows what such samples look like:

Key Property 2: Suppose that $F \sim \text{DP}(\alpha, H)$ is a random probability distribution sampled from a DP prior. Then with probability 1, F can be written as:

$$F = \sum_{k=1}^{\infty} w_k \delta_{\phi_k}, \quad \phi_k \sim G_0$$

where

$$w_k = z_k \prod_{i=1}^{k-1} (1 - z_i), \quad z_i \sim \text{Beta}(1, \alpha).$$

In other words, random probability distributions can be sampled from a DP by first drawing a collection of samples z_i from a Beta distribution, transforming these to produce the weights $\{w_i\}$, and then drawing the associated atoms from G_0. Note that in order for F to be a true from a DP, an infinite number of such weights and atoms must be drawn. However in practice, the above summation can be truncated with only a finite number N of draws, while still providing a very good approximation.

By combining Key Properties 1 and 2, we can sample a DP from its posterior distribution $F \mid y_1, \ldots, y_n$ as follows:

Key Property 3: If $y_1, \ldots, y_n \sim F$ and $F \sim \text{DP}(\alpha, G_0)$ then we can draw a (truncated) sample probability distribution from the posterior $F \mid y_1, \ldots, y_n$ as follows:

$$F = \sum_{k=1}^{N} w_k \delta_{\phi_k}, \quad \phi_k \sim \frac{\alpha G_0 + \sum_{i=1}^{n} \delta_{y_i}}{\alpha + n},$$

where

$$w_k = z_k \prod_{i=1}^{k-1} (1 - z_i), \quad z_i \sim \text{Beta}(1, \alpha + n).$$

and N is a truncation parameter.

2.1. Dirichlet Process Mixtures

The stick breaking representation in Key Property 2 above shows that probability distributions sampled from a DP are discrete with probability 1. As such, the DP is not an appropriate prior for F when F is continuous. Therefore, it is usual to adopt the following mixture specification instead, which is known as the Dirichlet process mixture model (DPMM):

$$\begin{aligned} y_i &\sim k(y_i \mid \theta_i), \\ \theta_i &\sim F, \\ F &\sim \text{DP}(\alpha, G_0). \end{aligned} \tag{1}$$

In other words, F has a DP prior as before, but rather than the data y_i being drawn from F, it is instead the mixture parameters θ which are draws from F. These θ values then act as the parameters of a parametric kernel function $k(\cdot)$, which is usually continuous. The most commonly used example is the Gaussian mixture model where $\theta_i = (\mu_i, \sigma_i^2)$ so that $k(y_i \mid \theta_i) = N(y_i \mid \mu_i, \sigma_i^2)$.

The key point here is that since F is discrete, two independent draws θ_i and θ_j from F can have identical values with a non-zero probability. As such, the DPMM can be seen as sorting the data into clusters, corresponding to the mixture components. The above model can hence be written equivalently as the following mixture model, which is infinite dimensional can hence be viewed as a generalization of the finite mixture models commonly used in nonparametric statistics:

$$\begin{aligned} y_i &\sim G, \\ G &= \int k(y_i \mid \theta) F(\theta) \mathrm{d}\theta, \\ F &\sim \text{DP}(\alpha, G_0). \end{aligned}$$

When the DPMM is used in practice, interest may focus on several different parameters of the posterior distribution. In some cases, the primary object of interest will be all n of the θ_i parameters from Equation 1 which are associated with the n observations. This is particularly the case in clustering applications, where the goal is to assign similar observations to the same cluster (i.e. to identical values of θ). However in other situations it will be the distribution F which is of primary interest. The **dirichletprocess** package returns posterior samples of all these quantities, so that the user can decide which are most relevant.

Posterior inference in the **dirichletprocess** package is based around the Chinese Restaurant Process sampler (Neal 2000). This is a Gibbs-style algorithm based on the DPMM representation in Equation 1 above, and draws samples of $\theta_1, \ldots, \theta_n$ from their posterior with the distribution F integrated out.

Key Property 4: Let θ_{-i} denote the set of θ values with θ_i excluded, i.e. $\theta_{-i} = (\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_n)$. Then the posterior distribution for θ_i conditional on the other model parameters is:

$$p(\theta_i \mid \theta_{-i}, y_{1:n}, \alpha, G_0) = \sum_{j \neq i} q_{i,j}\delta(\theta_j) + r_i H_i,$$

$$q_{i,j} = bk(y_i, \theta_j), \quad r_i = b\alpha \int k(y_i, \theta)\mathrm{d}G_0(\theta)$$

where b is set such that $\sum_{j \neq i} q_{i,j} + r_i = 1$ and H_i is the posterior distribution of θ based off of the prior base measure G_0.

Based on this result, Gibbs sampling is used to repeatedly draw each value of θ_i in turn from its posterior distribution, with all other variables held constant. An important distinction needs to be made between the **conjugate** case where the G_0 base measure is the conjugate prior for θ with respect to the kernel $k(\cdot)$, and the **non-conjugate** case there is not a closed form for the posterior distribution. In the conjugate case, the integral in Key Property 4 can be computed analytically and the resulting distribution is simply the predictive distribution. In this case, the θ_i values can be sampled directly from their true posterior distribution.

In the non-conjugate case things are slightly more difficult, and the integral in Key Property 4 cannot be evaluated. As such, numerical techniques must be used instead, which will typically result in slower computation. The **dirichletprocess** package handles the non-conjugate case by using "Algorithm 8" from (Neal 2000), which is one of the most widely used techniques for performing this sampling.

In both the conjugate and non-conjugate cases, the Gibbs sampling is conceptually similar, with the new values of θ_i being proposed sequentially from their respective posterior distributions. However in practice, this can result in poor mixing of the Gibbs sampler. One approach to speed up convergence is to add in additional update of the θ_i values at the end of the sampling. For each cluster and its associated data points we update the cluster parameter using the posterior distribution

$$p(\theta_i \mid y_j) = \prod_{j=i} k(y_j \mid \theta)G_0, \tag{2}$$

for a conjugate base measure, this posterior distribution is tractable and thus can be sampled directly. For a non-conjugate G_0, a posterior sample is achieved using the Metropolis-Hastings algorithm (Hastings 1970).

For simple density estimation and non-hierarchical predictive tasks, having a posterior sample of $\theta_{1:n}$ will be sufficient for inference, and the distribution F is not of any intrinsic interest. However when the DP is used as part of a hierarchical model such as in the regression and point process examples we discuss later, it is also necessary to have samples from the posterior distribution of F. These can be obtained using the following property:

Key Property 5: Given the model from Eq. (1) let $\theta_1, \ldots, \theta_n$ be a sample from the posterior $p(\theta_{1:n} \mid y_{1:n}, \alpha, G_0)$ drawn using the CRP sampler. Then, $p(F \mid \theta_{1:n}, y_{1:n}, \alpha, G_0) = p(F \mid \theta_{1:n}, \alpha, G_0)$ is conditionally independent of $y_{1:n}$. As such, $\theta_{1:n}$ can be considered as an i.i.d sample from F, and so F can be sampled from its posterior distribution using Key Property 3 above:

$$F = \sum_{i=1}^{N} w_i \delta_{\theta_i}, \quad w_i \sim \text{Beta}(\alpha + n, 1), \quad \theta_i \sim \frac{\alpha G_0 + \sum_{i=1}^{n} \delta_{\theta_i}}{\alpha + n}$$

where N is again a truncation parameter.

2.2. Hyperparameter Inference

In the above discussion, we assumed that the concentration parameter α and base measure G_0 were constant. However in practice, better results can often be obtained if they are also learned from the data. This can all be easily performed using routines from this package.

Inferring the Concentration Parameter

Following West (1992) we use a Gamma(a, b) prior for α. The corresponding posterior distribution depends only on the number of unique values of $\theta_{1:n}$. More specifically, given the model in Equation (1) let $\theta_1, \ldots, \theta_n$ denote a sample from the posterior $p(\theta_{1:n} \mid y_{1:n}, \alpha, G_0)$. Suppose that there are k unique values in this sample. Then a sample from $p(\alpha \mid \theta_{1:n}, y_{1:n}, G_0)$ can be obtained as follows:

- Simulate a random number z from a Beta$(\alpha + 1, n)$ distribution
- Define $\tilde{\pi}_1 = a + k + 1$ and $\tilde{\pi}_2 = n(b - \log(z))$, then define $\pi = \tilde{\pi}_1 / (\tilde{\pi}_1 + \tilde{\pi}_2) \in [0, 1]$
- With probability π, draw a value of α from a Gamma$(a + k, b - \log(z))$ distribution, and with probability $(1 - \pi)$ draw it from a Gamma$(a + k - 1, b - \log(z))$ distribution instead.

When fitting a DP the value of α can be easily inferred and sampled by default in the **dirichletprocess** package.

Inferring the Base Measure

The base measure G_0 can be parameterised with values γ which themselves are also random and from some distribution $p(\gamma)$. By placing a prior distribution on the parameters of the base measure this allows for the DP to adapt to the data and ensure that the fitting algorithms converge to the stationary distributions quicker

$$\begin{aligned} \phi_i \mid \gamma &\sim G_0, \\ \gamma &\sim p(\gamma), \\ \gamma \mid \phi_i &\sim H, \end{aligned}$$

where H is the posterior distribution. If we chose our $p(\gamma)$ such that it is conjugate to the posterior, we can directly sample form the posterior distribution, otherwise, a Metropolis-Hastings step is include in the computation.

2.3. Implemented Mixture Models

As we will discuss, one of the strengths of the **dirichletprocess** package is that it allows users to specify DPMs using whichever choices of the kernel k and base measure G_0 they please. However for ease of use, we have implemented certain choices of k and G_0 directly in the package, which includes routines for resampling all hyperparameters.

Gaussian Mixture Model

The Gaussian distribution is the most commonly used mixture model. In this case, $\theta = (\mu, \sigma^2)$ for the mean and variance. The kernel is:

$$k(y \mid \theta) = N(y_i \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(y-\mu)^2}{2\sigma^2}\right).$$

The conjugate prior for θ is the Normal-Gamma distribution, with parameters $\gamma = (\mu_0, k_0, \alpha_0, \beta_0)$

$$G_0(\theta \mid \gamma) = N\left(\mu \mid \mu_0, \frac{\sigma^2}{k_0}\right) \text{Inv-Gamma}\left(\sigma^2 \mid \alpha_0, \beta_0\right).$$

the default setting of the parameters is $\mu_0 = 0, \sigma_0^2 = 1, \alpha_0 = 1, \beta_0 = 1$. We recommenced rescaling our data y such that its mean is 0 and standard deviation is 1. This leads to the default parameterisation of G_0 being uninformative.

Since this prior is conjugate, the predictive distribution for a new observation $\tilde{y}$ can be found analytically, and is a location/scale Student-t distribution:

$$p(\tilde{y} \mid \gamma) = \int k(\tilde{y} \mid \theta) p(\theta \mid G_0) d\theta = \frac{1}{\tilde{\sigma}} \text{Student-t}\left(\frac{\tilde{y} - \tilde{\mu}}{\tilde{\sigma}} \mid \tilde{v}\right),$$

where $\tilde{v} = 2\alpha_0, \quad \tilde{\mu} = \mu_0, \quad \tilde{\sigma} = \sqrt{\frac{\beta_0(k_0+1)}{\alpha_0 k_0}}$.

Finally the posterior distribution is also a Normal Inverse Gamma distribution due to the conjugacy of the prior

$$\begin{aligned}
p(\theta \mid y, \gamma) &= N(\mu \mid \mu_n, \frac{\sigma^2}{k_0 + n}) \text{Inv-Gamma}(\sigma^2 \mid \alpha_n, \beta_n), \\
\mu_n &= \frac{\kappa_0 \mu_0 + n\overline{y}}{k_0 + n}, \\
\alpha_n &= \alpha_0 + \frac{n}{2}, \\
\beta_n &= \beta_0 + \frac{1}{2}\sum_{i=1}^{n}(y_i - \overline{y})^2 + \frac{\kappa_0 n(\overline{y} - \mu_0)^2}{2(\kappa_0 + n)}.
\end{aligned}$$

Multivariate Gaussian Mixture Model - Conjugate

The multivariate Gaussian mixture model is the most widely used nonparametric modeling approach for multivariate data. It is also heavily used in clustering applications (Maceachern

and Müller 1998). For the unknown parameters we have $\theta = (\boldsymbol{\mu}, \Lambda)$ for d dimensional data $\boldsymbol{\mu}$ is a column vector of length d and Λ is a $d \times d$ dimensional matrix

$$k(y_i \mid \theta) = \frac{\mid \Lambda \mid^{\frac{1}{2}}}{2\pi^{-\frac{d}{2}}} \exp\left(-\frac{1}{2}(y_i - \boldsymbol{\mu})^\top \Lambda (y_i - \boldsymbol{\mu})\right).$$

For the prior choice we us a multivariate normal distribution for $\boldsymbol{\mu}$ and Wishart distribution for Λ

$$G_0(\boldsymbol{\mu}, \Lambda \mid \boldsymbol{\mu_0}, \kappa_0, \nu_0, T_0) = N(\boldsymbol{\mu} \mid \boldsymbol{\mu_0}, (\kappa_0 \Lambda)^{-1}) \mathrm{Wi}_{\nu_0}(\Lambda \mid T_0),$$

where $\boldsymbol{\mu_0}$ is the mean vector of the prior, κ_0, ν_0 are single values and T is a matrix. The default prior parameters are set as $\boldsymbol{\mu} = \mathbf{0}, T = \mathbf{I}, \kappa_0 = d, \nu_0 = d$.

This prior choice is conjugate to the posterior, therefore the posterior distribution can be expressed analytically

$$\begin{aligned}
p(\theta \mid y_i) &= N(\boldsymbol{\mu} \mid \boldsymbol{\mu_n}, (\kappa_n \Lambda_n)^{-1}) \mathrm{Wi}(\Lambda \mid \nu_n, T_n), \\
\boldsymbol{\mu_n} &= \frac{\kappa_0 \mu_0 + n\overline{\mathbf{y}}}{k + n}, \\
\kappa_n &= \kappa_0 + n, \\
\nu_n &= \nu_0 + n, \\
T_n &= T_0 + \sum_{i=1}^{n} (y_i - \overline{\mathbf{y}})(y_i - \overline{\mathbf{y}})^\top + \frac{\kappa_0 n}{\kappa_0 + n}(\mu_0 - \overline{\mathbf{y}})(\mu_0 - \overline{\mathbf{y}})^\top.
\end{aligned}$$

Again, as this is a conjugate mixture we can write the predictive function for some new data $\tilde{y}$

$$p(\tilde{y} \mid \mathbf{y}) = \frac{1}{\pi^{\frac{nd}{2}}} \frac{\Gamma_d(\frac{\nu_n}{2})}{\Gamma_d(\frac{\nu_0}{2})} \frac{\mid T_0 \mid^{\frac{\nu_0}{2}}}{\mid T_n \mid^{\frac{\nu_n}{2}}} \left(\frac{\kappa_0}{\kappa_n}\right)^{\frac{d}{2}}.$$

Multivariate Gaussian Mixture Model - Semi-Conjugate

In the semi-conjugate case, the base measures for each parameter are specified independently

$$G_0(\boldsymbol{\mu}, \Sigma) = N(\boldsymbol{\mu} \mid \boldsymbol{\mu_0}, \Sigma_0) \mathrm{Wi}^{-1}_{nu_0}(\boldsymbol{\Phi_0}).$$

Therefore, sampling from the posterior is a two step process

$$\begin{aligned}
\Sigma \mid \boldsymbol{\mu}, \nu_0, \Phi_0 &\sim \mathrm{Wi}^{-1}_{\nu_n}(\Phi_n), \\
\nu_n &= \nu_0 + n, \\
\Phi_n &= \Phi_0 + \sum_{i=1}^{n} (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T, \\
\boldsymbol{\mu} \mid \Sigma, \boldsymbol{\mu_0}, \Sigma_0 &\sim N(\boldsymbol{\mu_n}, \Sigma_n), \\
\Sigma_n &= \left(\Sigma_0^{-1} + n\Sigma^{-1}\right)^{-1} \\
\boldsymbol{\mu_n} &= \Sigma_n \left(\Sigma_0^{-1} \boldsymbol{\mu_0} + n\Sigma^{-1} \overline{\boldsymbol{y}}\right)
\end{aligned}$$

using the conditional probabilities, each parameter can be sampled using the previous sample. This allows us to use Algorithm 8 and treat the model as a non-conjugate mixture model.

Beta Mixture Model

Dirichlet process mixtures of Beta distributions have been considered by Kottas (2006a) for the nonparametric estimation of continuous distributions that are defined on a bounded interval, $[0, T]$. For ease of interpretation, we follow their parameterisation of the Beta distribution in terms of its mean and standard deviation, In this case, $\theta = (\mu, \nu)$ with a known parameter T. The mixture kernel is:

$$k(y_i \mid \theta) = \text{Beta}(y_i \mid \mu, \nu, T) = \frac{y_i^{\frac{\mu\nu}{T}-1}(T - y_i)^{\nu(1-\frac{\mu}{T})-1}}{\text{B}(\frac{\mu\nu}{T}, \nu(1-\frac{\mu}{T}))T^{\nu-1}}.$$

There is no conjugate prior for the mixture kernel. Instead, the **dirichletprocess** package uses the (non-conjugate) prior from Kottas (2006a) where

$$G_0(\mu, \nu \mid T, \alpha_0, \beta_0) = U(\mu \mid [0, T])\text{Inv-Gamma}(\nu \mid \alpha_0, \beta_0).$$

To sample from the posterior distribution we use the Metropolis-Hastings algorithm with the parameters $\alpha_0 = 2, \beta_0 = 8$ as the default. However, there is also the ability to place a prior distribution on β_0 and update the prior parameter with each iteration. For this we use a default prior distribution of

$$\beta_0 \sim \text{Gamma}(a, b),$$

with $a = 1, b = 0.125$ by default.

Weibull Mixture Model

The Weibull distribution has strictly positive support it is mainly used for positive only data modeling. Furthermore, it is ubiquitously used in survival analysis. Mixture of Weibull distributions have been considered by Kottas (2006a) for a variety of survival applications. The parameters of the Weibull distribution are the shape a and scale b

$$k(y_i \mid \theta) = \text{Weibull}(y_i \mid a, b) = \frac{a}{b} y_i^{a-1} \exp\left(-\frac{y_i^a}{b}\right), \tag{3}$$

where $\theta = (a, b)$.

We use a non-conjugate Uniform Inverse Gamma distribution for the unknown parameters

$$G_0(a, b \mid \phi, \alpha, \beta) = U(a \mid 0, \phi)\text{Inv-Gamma}(b \mid \alpha, \beta), \tag{4}$$

by default ϕ, α and β do not have assigned values. Instead, we place priors on ϕ and β and update with each fitting procedure, α remains fixed. For ϕ we use a Pareto distribution as our prior as this is conjugate to the Uniform distribution.

$$\begin{aligned} a_i &\sim U(0, \phi), \\ \phi &\sim \text{Pareto}(x_m, k), \\ \phi \mid a_i &\sim \text{Pareto}(\max\{a_i, x_m\}, k + n), \end{aligned}$$

by default $x_m = 6, k = 2$ which is an infinite variance prior distribution.

As b is from a Inverse Gamma distribution with fixed shape α it has a conjugate prior which is the Gamma distribution.

$$\begin{aligned} b_i &\sim \text{Inv-Gamma}(\alpha, \beta), \\ \beta &\sim \text{Gamma}(\alpha_0, \beta_0), \\ \beta \mid b &\sim \text{Gamma}\left(\alpha_0 + n\alpha, \beta_0 + \sum_{i=1}^{n} \frac{1}{b_i}\right), \end{aligned}$$

with α fixed by the user and $\alpha_0 = 1, \beta_0 = 0.5$ by default. This prior on β with $\alpha_0 = 1$ is an conventional distribution with mean $\frac{1}{\beta_0}$ which allows for the user to decide how disperse the prior needs to be. As this is a non-conjugate model we must use a Metropolis-Hastings step to sample from the posterior distribution.

3. Package Overview

The **dirichletprocess** package contains implementations of a variety of Dirichlet process mixture models for nonparametric Bayesian analysis. Unlike several other R packages, the emphasis is less on providing a set of functions which completely automate routine tasks (e.g. density estimation or linear regression) and more on providing an abstract data type representation of Dirichlet process objects which allow them to be used as building blocks within hierarchical models.

To illustrate how the package is meant to be used, and how it differs from other R packages, consider the task of density estimation. Suppose we wish to estimate the density of some data stored in the variable `y` using a Dirichlet process mixture of Gaussian distributions. This is done as follows:

```
R> y <- rt(200, 3) + 2 #generate sample data
R> dp <- DirichletProcessGaussian(y)
R> dp <- Fit(dp, 1000)
```

The function `DirichletProcessGaussian` is the creator function for a mixture model of univariate Gaussians. This creates the object `dp`. We then use the function `Fit` on this object to infer the cluster parameters, which uses the Chinese Restaurant Sample algorithm described in Section 2.1. With each iteration, the assigned cluster label to each datapoint is updated, then the resulting cluster parameters are updated before finally updating the concentration parameter α. Using the `Fit` function the details of the sampling are removed from the user and this provides an 'out-of-the-box' method to easily fit a Dirichlet process to data. Only a specification of the type of mixture model is needed - in this case a Gaussian mixture. The returned object `dp` from the `Fit` function contains the following information: `dp$clusterParameterChains` stores the MCMC samples of the cluster parameters, `dp$weightsChain` stores the associate weights, and `dp$alphaChain` stores the samples of the concentration parameter α. These posterior samples can then be used for inference based on what the user is trying to accomplish.

The **dirichletprocess** package currently provides the following features:

- Implementations of Dirichlet process mixture models using Gaussian (both univariate and multivariate), Beta, and Weibull mixture kernels.

- Implementation of various schemes for re-sampling model parameters.
- Access to samples from the Dirichlet process in both marginal form, as well as in (truncated) stick-breaking form
- A flexible way for the user to add new Dirichlet process models which are not currently implemented in the package, and yet still use the re-sampling functions from the package. To illustrate this, Section 4.2 shows how simple it is to create a Dirichlet process mixture model with Poisson and Gamma distribution kernels, even though this is not implemented in the package.
- An ability to plot the likelihood, posterior and credible intervals of a Dirichlet process using `plot`.

All of the above features will be demonstrated in the following examples.

Nonparametric Density Estimation

The most simple application of DPMMs is to nonparametrically estimate the distribution of independent and identically distributed observations $y_1, \ldots, y_n$, where:

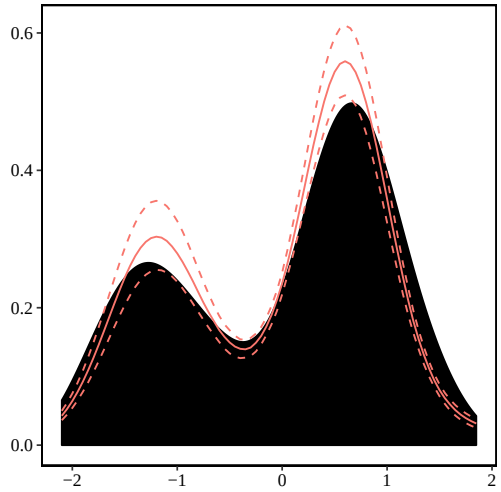
$$\begin{aligned} y_i &\sim F, \\ F &= \sum_{i=1}^{n} \pi_i k(y_i \mid \theta_i), \end{aligned}$$

where k is some density function parameterised by θ_i and n is some unknown amount of clusters (i.e. F has been specifically nonparametrically as a mixture model). The most widely used specification is the Gaussian mixture kernel with a Normal-Inverse Gamma base measure G_0, which is described more fully in Section 2.3.

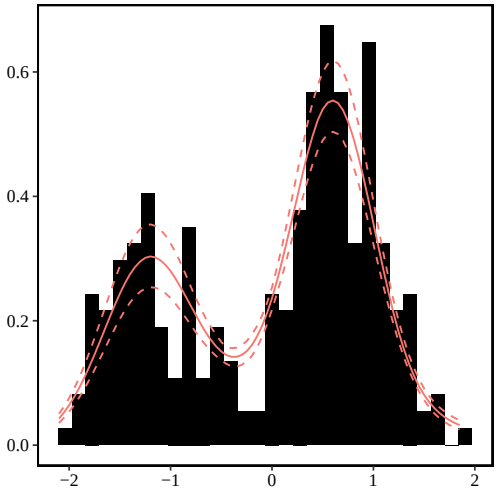
As an example we use the waiting times between eruptions of the Old Faithful volcano. This dataset is available within R and called `faithful`. We transform the waiting times to be be zero mean and and unit standard deviation and proceed to fit a DPMM with the default settings. We then model the waiting times as a mixture of Normal distributions

$$\begin{aligned} y_i &\sim F, \\ F &= \sum_{i=1}^{n} \pi_i k(y_i \mid \theta_i) \quad \theta_i = \left\{\mu_i, \sigma_i^2\right\}, \\ \theta_i &\sim G, \\ G &\sim \mathrm{DP}(\alpha, G_0), \end{aligned}$$

where $k(y_i \mid \theta)$ is the standard normal probability density and G_0 is the base measure as in Section 2.3.



(a) The estimated density of the data is plotted with the DPMM posterior mean and credible intervals overlaid in red.



(b) Instead of a density estimate, a histogram is plotted for the data.

Figure 1: Old Faithful waiting times density estimation with a DPMM of Gaussians.

```
R> its <- 500
R> faithfulTransformed <- scale(faithful$waiting)
R> dp <- DirichletProcessGaussian(faithfulTransformed)
R> dp <- Fit(dp, its)
R> plot(dp)
R> plot(dp, data_method="hist")
```

The resulting fit from the model is show in Figure 1. The multi-modal nature has been successfully captured a property that a singular distribution would not find.

When plotting the resulting object, the `plot` function has the argument `data_method` which controls how the data is represented in the plot. In the default case it is a density estimation. For cases where the amount of data is limited, a density estimation may not be suitable, in which case, `data_method` should be set to `"hist"` for a histogram.

For most users the `Fit` function is sufficient for practical purposes. However, for the more advanced users who wish to alter how they fit the dirichletprocess object there are a number of functions available to help.

By default, the `Fit` function updates the cluster allocation, the cluster parameters and then the α parameter. In some rare cases, updating α every iteration can delay convergence. Instead, the user can instead choose to update α every 10 iterations.

```
R> dp <- DirichletProcessGaussian(y)
R> samples <- list()
R> for(s in seq_len(1000)){
+    dp <- ClusterComponentUpdate(dp)
+    dp <- ClusterParameterUpdate(dp)
+
+    if(s %% 10 == 0) {
+            dp <- UpdateAlpha(dp)
+    }
+    samples[[s]] <- list()
+    samples[[s]]$phi <- dp$clusterParameters
+    samples[[s]]$weights <- dp$weights
+  }
```

The function `ClusterComponentUpdate` iterates through all the data points, y_i for $i = 1, \ldots, n$, updating its cluster assignment sequentially via the Chinese Restaurant Process sampling algorithm, using Key Property 4 in Section 2.1. For each data point, it can either be assigned to an existing cluster, or form a new cluster. The probability it is assigned to an existing cluster is proportional to $n_i k(y_j \mid \theta_i)$, where n_i is the number of points already assigned to the cluster θ_i and k is the likelihood of the data point point evaluated with the cluster parameter θ_i. The probability is forms a new cluster is proportional to α, the concentration parameter. If the datapoint is selected to form a new cluster, then this new cluster parameter θ_{new} is drawn from G_0 and added to the cluster pool. Subsequent points can now also be added to this cluster. Note that when a conjugate base measure is used in the DP, this function samples directly from the conditional posteriors, while if a non-conjugate sampler is used then the sampling is carried out using Algorithm 8 from Neal (2000).

After each data point has its cluster allocation updated the function `ClusterParameterUpdate` is called and resamples each of the unique θ_j parameters (i.e. if there are m unique values/-clusters in $\theta_1, \ldots, \theta_m$ then all m are resampled). The new values of θ_j are sampled from the posterior distribution of the parameter using all the data associated to that cluster parameter as per Equation (2).

Finally, `UpdateAlpha` samples a new value of α from its posterior distribution using the method outlined in Section 2.2. By manually calling these functions the user has control over the MCMC routine without having to have specific knowledge of the required algorithms.

The key point of the **dirichletprocess** package which the above code highlights is that a) the user controls when to re-sample the DP parameters, and b) the current sample is contained in the DP object and ready for inspection at any point in the code. This allows DP objects to be used as building blocks within hierarchical models.

3.1. Density Estimation on Bounded Intervals

In some situations it will be necessary to estimate densities on bounded intervals. For example, it might be known that the observations y_i are restricted to lie within the interval $[0, 1]$. In this case, a mixture of Gaussian distributions is inappropriate, since this will assign positive probability to the whole real line. An alternative specification is a mixture of Beta distributions, since the Beta distribution only has positive mass in $[0, 1]$. The full model is the same as in the previous example but replacing k with the Beta distribution.

```
R> y <- c(rbeta(150, 1, 3), rbeta(150, 7, 3)) #generate sample data
R> dp <- DirichletProcessBeta(y, 1)
R> dp <- Fit(dp, 1000)
```

As we want to compare the resulting posterior distribution to a known distribution we must manually draw posterior samples, calculate the credible intervals and plot the results.

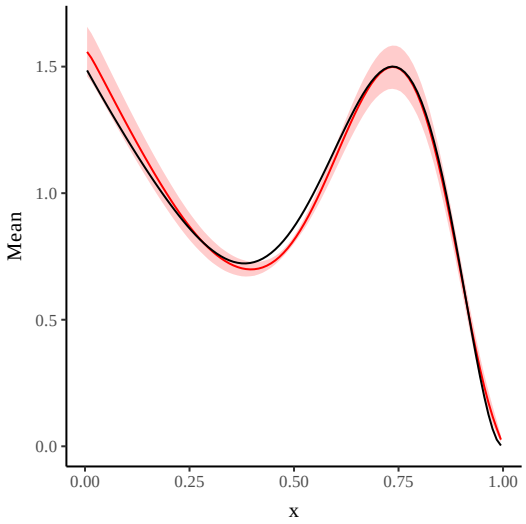


Figure 2: Estimated generating density using a Beta Dirichlet process mixture model.

```
R> posteriorFrame <- PosteriorFrame(dp, ppoints(100), ci_size = 0.05)
R> trueFrame <- data.frame(x=ppoints(100),
+                          y=0.5*dbeta(ppoints(100), 1, 3)+
+                            0.5*dbeta(ppoints(100), 7, 3))
R> ggplot() +
+    geom_ribbon(data=posteriorFrame,
+                aes(x=x, ymin=X2.5., ymax=X97.5.),
+                alpha=0.2,
+                colour=NA,
+                fill="red") +
+    geom_line(data=posteriorFrame, aes(x=x, y=Mean), colour="red") +
+    geom_line(data=trueFrame, aes(x=x, y=y))
```

Figure 2 shows the resulting posterior distribution and true density.

3.2. Cluster Analysis (Multivariate)

For any Dirichlet model, each data point y_i is assigned a cluster parameter θ_i. The collection of unique values of cluster parameters θ_i^* allows for a natural way of grouping the data and hence the Dirichlet process is an effective way of performing cluster analysis. For multidimensional data it is most common to use a mixture of multivariate normal distributions to cluster the observations into appropriate groups. In the **dirichletprocess** package, the clustering labels is available at each fitting iteration and available to the user as `dp$clusterLabels`. Examples of the use of Dirichlet processes in clustering can be found in Teh, Jordan, Beal, and Blei (2005) and Kim, Tadesse, and Vannucci (2006).

To demonstrate this we return to the `faithful` dataset which consists of two-dimensional data. In this example we also consider the length of the eruption as well as the amount of

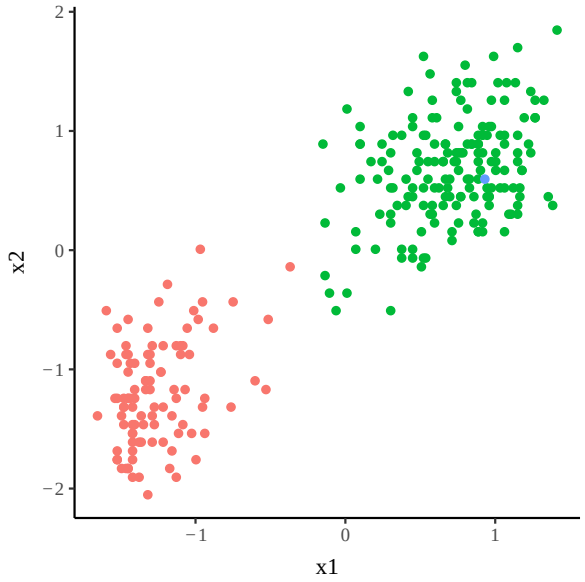


Figure 3: The colours of the points indicates that there are groups in the `faithful` dataset.

time between eruptions. The full model can be written as

$$
\begin{aligned}
y_i &\sim N(y \mid \theta_i), \\
\theta_i &= \{\boldsymbol{\mu}_i, \Sigma_i\}, \\
\theta_i &\sim G, \\
G &\sim \mathrm{DP}(\alpha, G_0),
\end{aligned}
$$

where the prior parameters of G_0 take on their default value as shown in Section 2.3. We will be using the cluster labels to indicate which group each data point belongs to.

We transform the data so that each variable is zero mean and unit standard deviation.

```
R> faithfulTrans <- scale(faithful)
```

We form the dirichletprocess object and perform 1000 MCMC samples.

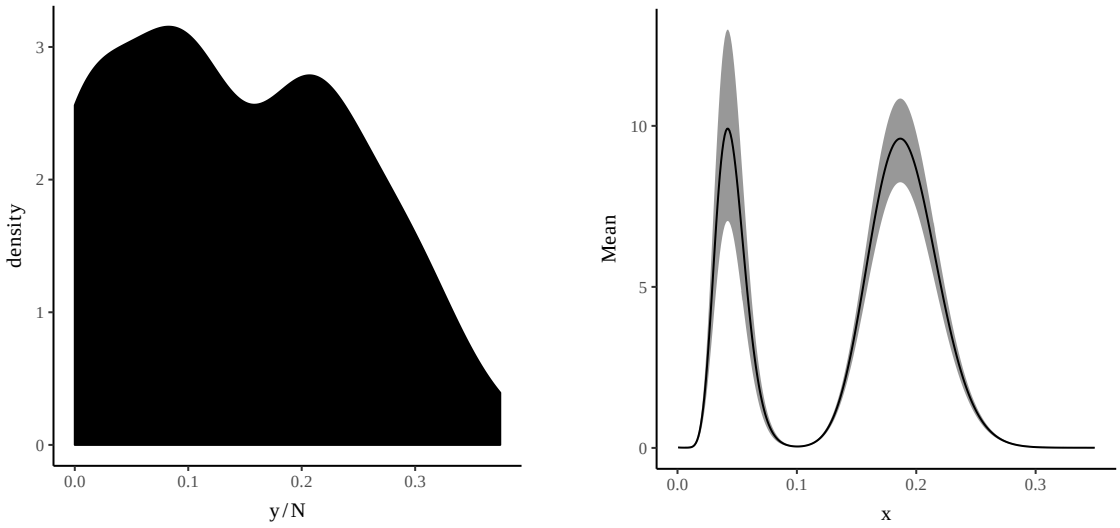
```
R> dp <-  DirichletProcessMvnormal(faithfulTrans)
R> dp <- Fit(dp, 1000)
R> plot(dp)
```

When analysing the results of the fit we are interested in the final Gibbs sample of the cluster labels. Using the cluster label to assign a colour to each datapoint we can easily see the distinct clusters that form.

Here Figure 3 shows the last iteration of the cluster labels and the colours indicate the found clusters. Whilst this example only shows two dimensions, the code is generalised to work with as many dimensions as necessary.

3.3. Modifying the Observations

In some applications of Dirichlet processes the data being used can change from iteration to iteration of the sampling algorithm. This could be because the values of the data change, or



(a) Empirical distribution of $\hat{\theta}_i$.

(b) Dirichlet prior model for $p(\theta_i)$.

Figure 4: Rat tumour risk empirical density and fitted prior distribution.

because for a full data set $\mathbf{y} = y_1, \ldots, y_n$, only subsets of the data are used at each iteration. When fitting a DP object we provide the appropriate function `ChangeObservations` to change the observations between iterations.

This function takes the new data, predicts what clusters from the previous fitting the new data belongs to and updates the clustering labels and parameters accordingly. A modified object with the new data associated to clusters and the function `Fit` is ready to be used to sample the cluster parameters and weights again.

Example: Priors in Hierarchical Models

One application of observations changing with each iteration is using a Dirichlet process as a prior for a parameter in a hierarchical model. An example of hierarchical modelling comes from Gelman, Carlin, Stern, and Rubin (2014) involving tumour risk in rats. In this example, there are 71 different experiments, and during each experiment a number of rats are inspected for tumours, with the number of tumours in each experiment being the observed data. This data is included in our packages in the **rats** variable, with the first column being the number of tumours in each experiment, and the second being the number of rats.

A naive approach would model each experiment as a Binomial draw with unknown θ_i and known N_i. A Beta distribution is the conjugate prior for the Binomial distribution and would be used as the prior on θ:

$$\begin{aligned} y_i \mid \theta_i, N_i &\sim \text{Binomial}(N_i, \theta_i), \\ \theta_i &\sim \text{Beta}(\alpha, \beta). \end{aligned}$$

However, Figure 4a shows the empirical distribution of $\hat{\theta}_i = \frac{y_i}{N_i}$. This distribution shows hints of bimodality, something that a single Beta distribution cannot capture and hence the prior choice of $p(\theta_i)$ is dubious. An alternative procedure is to instead use a nonparametric prior

on the $\theta_i's$. Since these parameters are constrained to lie between 0 and 1, one choice might be a Dirichlet process mixture of Beta distributions. This leads to the following model

$$\begin{aligned} y_i \mid \theta_i, N_i \sim & \text{Binomial}(N_i, \theta_i), \\ \theta_i \sim & \text{Beta}(\alpha_i, \beta_j), \\ \alpha_i, \beta_i \sim & F, \\ F \sim & DP(\alpha, G_0), \end{aligned}$$

where α and G_0 follow the default implementations of the **dirichletprocess** package. We then implement this model using the **dirichletprocess** functions as follows:

```
R> numSamples = 200
R> thetaDirichlet <- matrix(nrow=numSamples, ncol=nrow(rats))
R> dpobj <- DirichletProcessBeta(rats$y/rats$N,
+                                maxY=1,
+                                g0Priors = c(2, 150),
+                                mhStep=c(0.25, 0.25),
+                                hyperPriorParameters = c(1, 1/150))
R> dpobj <- Fit(dpobj, 10)
R> clusters <- dpobj$clusterParameters
R> a <- clusters[[1]] * clusters[[2]]
R> b <- (1 - clusters[[1]]) * clusters[[2]]
R> for(i in seq_len(numSamples)){
+
+    posteriorA <- a[dpobj$clusterLabels] + rats$y
+    posteriorB <- b[dpobj$clusterLabels] + rats$N - rats$y
+    thetaDirichlet[i, ] <- rbeta(nrow(rats), posteriorA, posteriorB)
+
+    dpobj <- ChangeObservations(dpobj, thetaDirichlet[i, ])
+    dpobj <- Fit(dpobj, 5)
+    clusters <- dpobj$clusterParameters
+
+    a <- clusters[[1]] * clusters[[2]]
+    b <- (1 - clusters[[1]]) * clusters[[2]]
+  }
```

Note the reason why the observations are changing is because the DP mixture model is applied to the θ_i parameters, which are resampled (and hence have different values) during each MCMC iteration.

```
R> ggplot(rats, aes(x=y/N)) +
+    geom_density(fill="black") #Plot the emperical distribution
R> posteriorFrame <- PosteriorFrame(dpobj, ppoints(1000))
R> ggplot() +
+    geom_ribbon(data=posteriorFrame,
+                aes(x=x, ymin=X5.,ymax=X95.),
+                alpha=0.2) +
+    geom_line(data=posteriorFrame, aes(x=x, y=Mean)) +
+    xlim(c(0, 0.35)) #Plot the resulting prior distribution
R>
```

Plotting the resulting estimation in Figure 4b reveals that the DP is a more suitable prior than the Beta distribution. This confirms what we saw from the empirical distribution that the data is bi-modal.

3.4. Hierarchical Dirichlet process

A hierarchical Dirichlet process Teh *et al.* (2005) can be used for grouped data. Each individual dataset is modeled using a separate Dirichlet process but where the base measure itself is

also a Dirichlet process. Mathematically this can be expressed as

$$\begin{aligned} y_{ij} \sim & F(\theta_{ij}), \\ \theta_{ij} \sim & G_j, \\ G_j \sim & \text{DP}(\alpha_j, G_0), \\ G_0 \sim & \text{DP}(\gamma, H), \end{aligned}$$

for each dataset $j = 1, \ldots, n$ with data $y_{1j}, \ldots y_{Nj}$ there is a separate Dirichlet process generating the required parameters θ_{ij}. Using the stick breaking construction, we can express G_0 as an infinite sum (Key Property 2), the same procedure can be applied to the G_j measures

$$\begin{aligned} G_j &= \sum_{i=1}^{\infty} \pi_{jk} \delta_{\phi_k}, \quad \phi_k \sim H, \\ \pi'_{jk} &= \text{Beta}\left(\alpha_j \beta_k, \alpha\left(1 - \sum_{l=1}^{k} \beta_l\right)\right), \quad \pi_{jk} = \pi'_{jk} \prod_{l=1}^{k-1} (1 - \pi'_{jl}), \\ \beta'_k &\sim \text{Beta}\left(1, \gamma\right), \quad \beta_k = \beta'_k \prod_{l=1}^{k-1} (1 - \beta'_l), \end{aligned} \tag{5}$$

we call H the global distribution and each G_j the local distribution. For further details see Teh *et al.* (2005).

To fit a hierarchical Dirichlet process, Algorithm 8 from Neal is used as detailed in Section 2.1. Each datapoint y_{ij} is further assigned as label k_{ij} which indicates which global parameter ϕ_k it is assigned. Then for each global parameter, we can update its value by drawing from its posterior distribution, using all the data available across the data sets

$$\phi_k \mid x_{ij} = h(\phi_k) \prod_{k_{ij}=k} f(x_{ij} \mid \phi_k), \tag{6}$$

where h is the density of the global distribution H and f is the density of the mixing distribution F. From these updated parameters a new G_j can be drawn using for each j using Key Property 5.

For a hierarchical DP model each individual concentration parameter α_j can be inferred using the usual algorithm as per Section 2.2.1 without modification for each individual dataset. For the top level concentration parameter γ, the number of unique cluster parameters across all the individual G_j's is used for n in the sampling of γ.

In this example we create two synthetic data sets and fit a Hierarchical Dirichlet process to demonstrate the use of such a tool. In this case we are fitting a Beta Dirichlet mixture model and therefore simulate from two Beta distributions

$$\begin{aligned} y_1 \sim & \text{Beta}(0.25, 5) + \text{Beta}(0.75, 6), \\ y_2 \sim & \text{Beta}(0.25, 5) + \text{Beta}(0.4, 10), \end{aligned}$$

where there is a common group of parameters between the two datasets. First we simulate the two data sets.

```
R> mu <- c(0.25, 0.75, 0.4)
R> tau <- c(5, 6, 10)
R> a <- mu * tau
R> b <- (1 - mu) * tau
R> y1 <- c(rbeta(500, a[1], b[1]), rbeta(500, a[2], b[2]))
R> y2 <- c(rbeta(500, a[1], b[1]), rbeta(500, a[3], b[3]))
```

We then use the appropriate constructor functions to build a hierarchical Dirichlet object with uninformative priors for the global base distribution and fit for 5000 iterations.

```
R> dplist <- DirichletProcessHierarchicalBeta(list(y1, y2),
+                                              maxY=1,
+                                              hyperPriorParameters = c(1, 0.01),
+                                              mhStepSize = c(0.1, 0.1),
+                                              gammaPriors = c(2, 4),
+                                              alphaPriors = c(2, 4))
R> dplist <- Fit(dplist, 500)
```

The creator function `DirichletProcessHierarchicalBeta` returns a list of `dirichletprocess` objects for each dataset (in this case two objects), a vector containing the global stick breaking weights, a list of the global parameters and the variable containing the global concentration parameter γ. The function `Fit` updates the cluster allocations locally of each `dirichletprocess` object using Algorithm 8 from Neal (2000), then the local concentration parameter α_j is updated. The global cluster parameters are then updated using all the data pooled from the individual datasets by drawing from Eq. (6). Using these parameters, a new sample of G_0 is taken from which the individual G_j's are also drawn using the above procedure in Equation 5.

```
R> xGrid <- ppoints(100)
R> postDraws <- lapply(dplist$indDP,
+                      function(x) {
+                        replicate(1000, PosteriorFunction(x)(xGrid))
+                      }
+                      )
R> postMeans <- lapply(postDraws, rowMeans)
R> postQuantiles <- lapply(postDraws,
+                           function(x) {
+                             apply(x, 1, quantile, probs=c(0.025, 0.975))
+                           }
+                           )
R> postFrame <- do.call(rbind,
+                        lapply(seq_along(postMeans),
+                               function(i) data.frame(Mean=postMeans[[i]],
+                                                      t(postQuantiles[[i]]),
+                                                      x=xGrid, ID=i)
+                               )
+                        )
R> trueFrame1 <- data.frame(y=0.5*dbeta(xGrid, a[1], b[1]) +
+                              0.5*dbeta(xGrid, a[2], b[2]),
+                            x=ppoints(100), ID=1)
R> trueFrame2 <- data.frame(y=0.5*dbeta(xGrid, a[1], b[1]) +
+                              0.5*dbeta(xGrid, a[3], b[3]),
+                            x=xGrid, ID=2)
R> trueFrame <- rbind(trueFrame1, trueFrame2)
R> ggplot() +
+    geom_ribbon(data=postFrame, aes(x=x, ymin=X2.5., ymax=X97.5.),
+                alpha=0.2, colour=NA, fill="red") + #credible interval
+    geom_line(data=postFrame, aes(x=x, y=Mean), colour="red") + #mean
+    geom_line(data=trueFrame, aes(x=x, y=y)) + #true density
+    facet_grid(~ID)
```

The resulting G_j's from the above example are plotted in Figure 5.

3.5. Stick Breaking Representation

The stick breaking representation of a Dirichlet process allows for easy posterior inference using Key Property 3 and 5 (Section 2). In the **dirichletprocess** package drawing from the posterior is easily achieved using both `PosteriorClusters` and `PosteriorFunction` depending on users need.

- `PosteriorClusters`: Returns the posterior clusters ϕ_k and weights w_k as a list for the user.
- `PosteriorFunction`: Draws the posterior clusters and uses the likelihood function of the mixing distribution to return a function with appropriate posterior weights and parameters. This is a sample of the measure F for models of the form in Eq. (1).

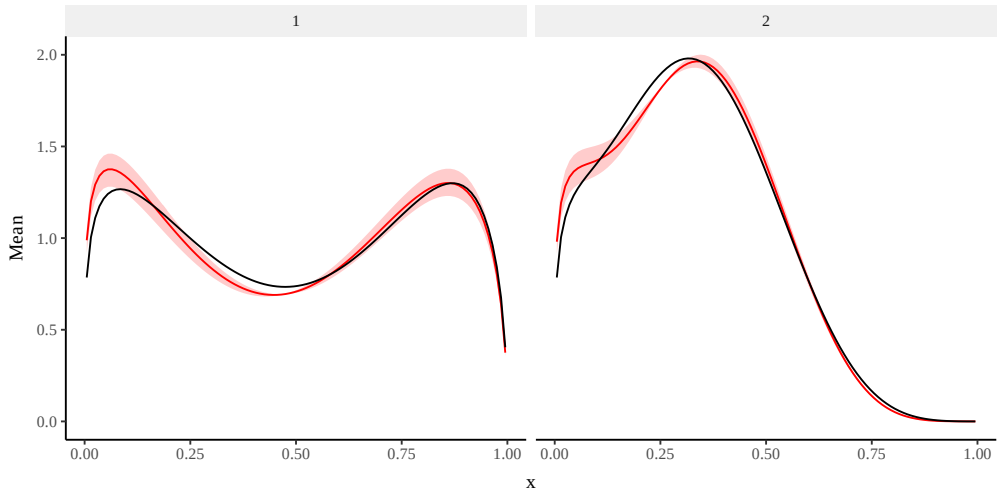


Figure 5: Hierarchical Beta Dirichlet process mixture results. The red lines indicate the true generating distributions. The black and shaded area are the posterior mean and credible intervals.

Example: Point Process Intensity Estimation

One practical application of Beta mixture models is the estimation of an inhomogeneous Poisson process intensity function, previous work has been done in this area at Taddy and Kottas (2012). A Poisson process is a collection of points in space distributed with rate λ. In the inhomogeneous case, the intensity is dependent on time and as such the number of events in some window $[0, T]$ can be written as

$$N \sim \text{Poisson}(\lambda(t)).$$

In parametric estimation, a functional form of $\lambda(t)$ would be constructed i.e. $\alpha_0 + \alpha t$ and the parameters $\{\alpha_0, \alpha\}$ would be estimated. However, the accuracy of such a method would be dependent on correctly identifying the parametric form of $\lambda(t)$. With the nonparametric methods that a DPMM provides, such assumptions can be ignored and an intensity function can be built without the need to specify a parametric form. Firstly, we assume that $\lambda(t) = \lambda_0 h(t)$ where $\int_0^T h(t)\text{d}t = 1$, i.e. the intensity rate can be decomposed into an amplitude λ_0 controlling the number of events and a density $h(t)$ controlling the distribution of the events over the window of observation $[0, T]$. To infer the value of λ_0 a conjugate Gamma prior can be used and thus the posterior distribution can be directly sampled.

In this example, we will instead be estimating an intensity rate $\lambda(t)$ with each iteration and using it to update the data. The full model can be written as

$$\begin{aligned} N &\sim \text{Poisson}(\lambda(t)), \\ \lambda(t) &= \lambda_0 h(t), \\ h(t) &= \int k(t \mid \theta)\text{d}F, \\ F &\sim \text{DP}(\alpha, G_0), \end{aligned}$$

where k and G_0 are as per Section 2.3 for the Beta distribution mixture models. We sample the posterior distribution of G using Key Property 5 (Section 2) which states that a sample

of G can be drawn independently of the data using the stick breaking representation of the data and the model parameters θ.

In this toy model we simulate 500 event times using the intensity function $\lambda(t) = \sin^2 \frac{t}{50}$. Instead of passing the full data set into the Dirichlet process object, we just use a random sample of 100 of these event times.

```
R> y <- cumsum(runif(1000))
R> pdf <- function(x) sin(x/50)^2
R> accept_prob <- pdf(y)
R> pts <- sample(y, 500, prob=accept_prob)
```

We then fit the Dirichlet process, draw a posterior sample of the intensity function $\hat{\lambda}(t)$ and sample 150 new points from the full data set with probabilities proportional to $\hat{\lambda}(t)$. The Dirichlet process object is then modified with the new data and the process is repeated.

```
R> dp <- DirichletProcessBeta(sample(pts, 100), maxY = max(pts)*1.01,
+  alphaPrior = c(2, 0.01))
R> dp <- Fit(dp, 100, TRUE)
R> for(i in seq_len(2000)){
+    lambdaHat <- PosteriorFunction(dp)
+    newPts <- sample(pts, 150, prob=lambdaHat(pts))
+    newPts[is.infinite(newPts)] <- 1
+    newPts[is.na(newPts)] <- 0
+    dp <- ChangeObservations(dp, newPts)
+    dp <- Fit(dp, 2, TRUE)
+  }
```

After the fitting process has finished, we want to draw from the resulting posterior distribution again and comparing it to the true intensity function.

```
R> posteriorFrame <- PosteriorFrame(dp, seq(0, max(pts)*1.01, by=0.1))
R> trueFrame <- data.frame(y=pdf(seq(0, max(pts)*1.01, by=0.1))/238,
+                          x=seq(0, max(pts)*1.01, by=0.1))
R> ggplot() +
+    geom_ribbon(data=posteriorFrame, aes(x=x, ymin=X5., ymax=X95.),
+                alpha=0.2, fill="red", colour=NA) + #credible interval
+    geom_line(data=posteriorFrame, aes(x=x, y=Mean), colour="red") + #mean
+    geom_line(data=trueFrame, aes(x=x, y=y)) #true intensity
```

Figure 6 shows the true intensity function is being recovered even though the full dataset is never observed.

4. Advanced Features

The material in this section can largely be skipped as long as the user is getting good results from the **dirichletprocess** package using the default functions and specifications. However

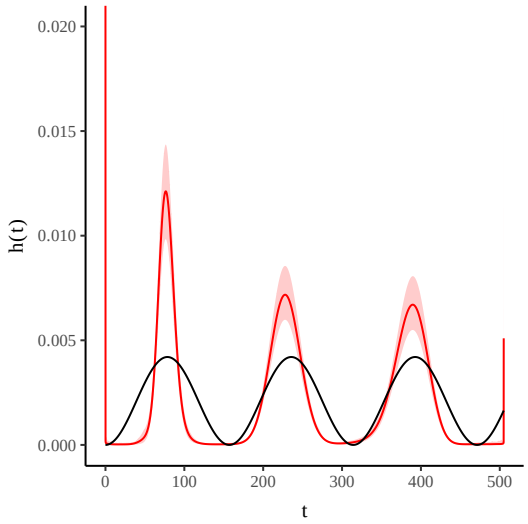


Figure 6: Estimation of the inhomogeneous Poisson process using stick breaking.

some users will require more control over the DP implementation – perhaps the default base measure hyper parameters are inadequate for a particular data set, or the sampler seeming not to converge due to bad initial default parameter values. Alternatively, the user may wish to use a mixture kernel other than the ones we have included in the package (Normal, Beta, Weibull, etc). In this case, the user will need to know what is going on under the hood so that they can change the default settings to better suited values, or otherwise modify the internal sampling procedure. We have done our best to ensure that this will usually only require changing a small number of the parameters which control the sampling behaviour, but understanding what needs to be changed (and why) requires some understanding of how the objects are constructed. Note that parts of this section do require an intermediate level of R programming knowledge.

4.1. Structure of a DP Object: The Gory Details

This package implements Dirichlet Processes as S3 object in R. All DPs have the following fields and available which are defined upon construction and do not ever change. When using the DP objects implemented in this package, these will be set by the constructor functions and so can largely be ignored for the default mixture models.

A DP object is defined by its kernel mixing distribution $k(y \mid \theta)$. Each mixing distribution has the following functions and variables associated with its class

- `Likelihood(...)`: a function which specifies the density of the mixture kernel $k(y \mid \theta)$.
- `PriorDraw(...)`: a function which returns a random sample of size n from the DP base measure G_0. This is used to define G_0.
- `g0Priors`: a list of parameters for the base measure G_0. Again, this is used to define G_0.

For a conjugate mixing distribution the posterior distribution of θ is tractable and can be

sampled from directly and the marginal distribution of the data can also be evaluated. Therefore two more functions are needed to complete the specification of a conjugate DP mixture model:

- `PosteriorDraw(...)`: a function that returns a sample of size n given data y from the posterior distribution of θ, i.e. a sample from the distribution of $p(\theta \mid y)$.
- `Predictive(...)`: a function that returns the value of the marginal distribution of the data $f(y) = \int k(y, \theta) \mathrm{d}G(\theta)$.

With these specified, the **Fit** function can be used to fit the DP, which carries out the Chinese Restaurant Sampling can be performed using the conjugate Algorithm 4 from (Neal 2000).

For a non-conjugate mixing distribution we can no longer directly sample from the posterior distribution $p(\theta \mid y)$ or calculate the marginal distribution of the data. Instead the Metropolis-Hastings algorithm is used to sample from the distribution $p(\theta \mid y)$. The Metropolis-Hastings algorithm works by generating a candidate parameter θ^{i+1} and accepting this candidate value as a sample from the posterior with probability proportional to $\frac{k(y|\theta^{i+1})p(\theta^{i+1})}{k(y|\theta^{i})p(\theta^{i})}$. Typically, the candidate parameter is distributed as $\theta_{i+1} \sim N(\theta_i, h^2)$. From this, the non-conjugate mixture model requires 2 additional functions and an extra parameter to be defined.

- `PriorDensity(...)`: a function which evaluates $p(\theta)$ which is the DP base measure G_0 for a given θ.
- `mhParameterProposal(...)`: a function that returns a candidate parameter to be evaluated for the Metropolis-Hastings algorithm.
- `mhStepSize`: h, the size of the step to make when proposing a new parameter for the Metropolis-Hastings algorithm.

With these specified, the **Fit** function can again be used to fit the DP, which carries out the Chinese Restaurant Sampling can be performed using 'Algorithm 8' (Neal 2000).

Once the appropriate mixing distribution is defined we can create a **dirichletprocess** object which contains the data, the mixing distribution object and the parameter α. Then the rest of **dirichletprocess** class functions are available.

By using the default constructor functions `DirichletProcessBeta/Gaussian/Mvnormal/Weibull` the base measure prior parameters are chosen to be non-informative, see Section for 2.3 for the specific values of the prior parameters.

4.2. Creating New Dirichlet process Mixture Types

The **dirichletprocess** package currently implements Dirichlet process mixture models using Gaussian, Beta and Weibull kernels. While these kernels should be appropriate for most applications, there will inevitably be times when a user wants to fit a DP model for a kernel which has not been implemented, or otherwise wants to do something complex with a DP which goes beyond the scope of this package. To anticipate this, we have tried to make it easy for users to construct their own mixture models, which can then automatically use the implemented algorithms for fitting a Dirichlet process.

The functions in the package are designed to work on S3 R objects, where each object represents a type of Dirichlet process mixture (e.g Gaussian or Beta). In order to create new types of Dirichlet process mixtures, the user must create a new S3 object type which encapsulates his model and ensure that its specifications correspond to those of the package. If this is done, then all the package functions for re-sampling/prediction/etc should continue work on the new DP type. This means that the package can hopefully be used for DP applications that we did not consider when writing it, while saving the user from having to write their own functions for re-sampling and fitting.

To illustrate how this works, this section will work through an extended example of how to create a new S3 type which represents a DP mixture model not implemented in the **dirichletprocess** package. We will explain how the S3 objects are constructed in detail so that the user will be able to create their own.

Conjugate Mixture

Suppose we have a particular scenario that requires a Dirichlet process mixture of Poisson distributions. The conjugate prior for the Poisson distribution is the Gamma distribution.

First, we start with the likelihood of the Poisson distribution

$$k(x \mid \theta) = \frac{\theta^x \exp(-\theta)}{x!},$$

as there is only one parameter in the Poisson distribution the parameter list θ if of length 1.

```
Likelihood.poisson <- function(mdobj, x, theta){
  return(as.numeric(dpois(x, theta[[1]])))
}
```

Note that the [[1]] part is essential, since parameters are internally represented as lists even when they only have one element.

We then write the random prior draw function which draws a value of θ from the base measure G_0. The conjugate prior to the Poisson distribution is the Gamma distribution

$$G_0 \sim \text{Gamma}(\alpha_0, \beta_0).$$

```
PriorDraw.poisson <- function(mdobj, n){
  draws <- rgamma(n, mdobj$priorParameters[1], mdobj$priorParameters[2])
  theta <- list(array(draws, dim=c(1,1,n)))
  return(theta)
 }
```

The prior parameters α_0, β_0 are stored in the mixing distribution object `mdobj`.

We then write the `PosteriorDraw` function to sample from the posterior distribution of θ. Again, as the base measure G_0 is conjugate this is a direct sample from the posterior distribution

$$\theta \mid x \sim \text{Gamma}(\alpha_0 + \sum_{i=1}^{n} x_i, \beta_0 + n),$$

using the inbuilt `rgamma` function this is trivial.

```
PosteriorDraw.poisson <- function(mdobj, x, n=1){
  priorParameters <- mdobj$priorParameters
  lambda <- rgamma(n, priorParameters[1] + sum(x), priorParameters[2] + nrow(x))
  return(list(array(lambda, dim=c(1,1,n))))
}
```

Finally the marginal distribution of the data $f(y)$ can be evaluated as it is a conjugate mixture model and translated into the appropriate R function:

```
Predictive.poisson <- function(mdobj, x){
  priorParameters <- mdobj$priorParameters
  pred <- numeric(length(x))
  for(i in seq_along(x)){
    alphaPost <- priorParameters[1] + x[i]
    betaPost <- priorParameters[2] + 1
    pred[i] <- (priorParameters[2] ^  priorParameters[1]) / gamma(priorParameters[1])
    pred[i] <- pred[i] * gamma(alphaPost) / (betaPost^alphaPost)
    pred[i] <- pred[i] * (1 / prod(factorial(x[i])))
  }
  return(pred)
}
```

With these functions written for the Poisson mixture model we now need to use the `MixingDistribution` constructor function to create a new object that can be used by the Dirichlet process constructor function, `DirichletProcessCreate`.

The constructor function `MixingDistribution` creates an object of class `distribution`, in this case 'poisson', with prior parameters $\alpha_0, \beta_0 = 1$ and that it is conjugate.

```
R> poisMd <- MixingDistribution(distribution="poisson",
+                               priorParameters = c(1, 1),
+                               conjugate="conjugate")
```

This object is now ready to be used in a **dirichletprocess** object and the appropriate sampling tasks can be carried out. To demonstrate we simulate some test data and fit a Dirichlet process with our new mixing distribution.

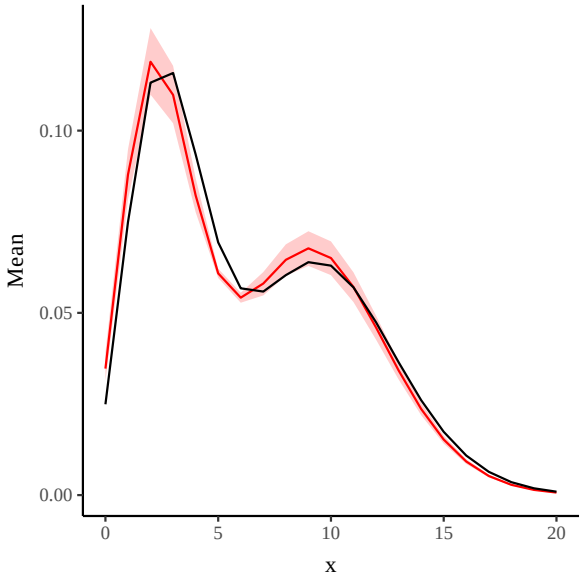


Figure 7: The true and estimated distributions from the Poisson mixture model.

```
R> y <- c(rpois(150, 3), rpois(150, 10)) #generate sample data
R> dp <- DirichletProcessCreate(y, poisMd)
R> dp <- Initialise(dp)
R> dp <- Fit(dp, 1000)
R> pf <- PosteriorFrame(dp, 0:20, 1000)
R> trueFrame <- data.frame(x= 0:20,
+                          y= 0.5*dpois(0:20, 3) + 0.5*dpois(0:20, 10))
R> ggplot() +
+      geom_ribbon(data=pf,
+                  aes(x=x, ymin=X5., ymax=X95.),
+                  colour=NA,
+                  fill="red",
+                  alpha=0.2) + #credible intervals
+      geom_line(data=pf, aes(x=x, y=Mean), colour="red") + #mean
+      geom_line(data=trueFrame, aes(x=x, y=y)) #true
R>
R>
```

As Figure 7 shows, the true generating function has been recovered. This shows how easy it is for the user to create their own mixture models using the **dirichletprocess** package.

Nonconjugate Mixture

Suppose that a particular application requires a Dirichlet process mixture of Gamma distributions. As the Gamma distribution does not have a conjugate prior distribution additional steps must be taken when creating the necessary functions.

Firstly we must write the likelihood function, as the Gamma distribution has two parameters α, β the list θ will also have two components. The density of the Gamma distribution can be

written as

$$k(y \mid \alpha, \beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} y^{\alpha-1} e^{-\beta y},$$

which can be easily translated using `dgamma` in R.

```
Likelihood.gamma <- function(mdobj, x, theta){
  return(as.numeric(dgamma(x, theta[[1]], theta[[2]])))
}
```

We now need the function to draw random parameters from the base measure G_0. For the parameters of the Gamma distribution we will be using a prior distribution of an Exponential distribution

$$\alpha \sim \text{Exp}(\alpha_0),$$
$$\beta \sim \text{Exp}(\beta_0).$$

```
PriorDraw.gamma <- function(mdobj, n=1){
  theta <- list()
  theta[[1]] = array(rexp(n, mdobj$priorParameters[1]), dim=c(1,1, n))
  theta[[2]] = array(rexp(n, mdobj$priorParameters[2]), dim=c(1,1, n))
  return(theta)
}
```

Now as we are drawing from the posterior distribution using the Metropolis-Hastings algorithm, we also need a function that calculates the prior density for a given α, β.

```
PriorDensity.gamma <- function(mdobj, theta){
  priorParameters <- mdobj$priorParameters
  thetaDensity <- dexp(theta[[1]], priorParameters[1])
  thetaDensity <- thetaDensity * dexp(theta[[2]], priorParameters[2])
  return(as.numeric(thetaDensity))
}
```

Finally, the Metropolis-Hastings algorithm also needs a function that perturbs the parameters to explore the posterior distribution. As for the Gamma distribution the parameters $\alpha, \beta > 0$ we must constrain our proposals. This is achieved by taking the absolute value of a standard normal perturbation.

$$\begin{aligned} \alpha^{i+1} &= |\alpha^i + h \cdot \eta|, \\ \eta &\sim N(0,1), \\ \beta^{i+1} &= |\beta^i + h \cdot \zeta|, \\ \zeta &\sim N(0,1). \end{aligned}$$

Again this is easy to translate into R:

```
MhParameterProposal.gamma <- function(mdobj, oldParams){
  mhStepSize <- mdobj$mhStepSize
  newParams <- oldParams
  newParams[[1]] <- abs(oldParams[[1]] + mhStepSize[1]*rnorm(1))
  newParams[[2]] <- abs(oldParams[[2]] + mhStepSize[2]*rnorm(1))
  return(newParams)
}
```

We can now construct our mixing distribution object using the constructor function `MixingDistribution`. The arguments of this function specify the prior parameters α_0, β_0 and set the scale h at which the new parameter proposals are made using the parameter `mhStepSize`.

```
gammaMd <- MixingDistribution(distribution = "gamma",
                              priorParameters = c(0.1, 0.1),
                              conjugate = "nonconjugate",
                              mhStepSize = c(0.1, 0.1))
```

The dirichletprocess object can now be created and fit to some test data. As it is a new type of mixture, it must be initialised.

```
R> y <- c(rgamma(100, 2, 4), rgamma(100, 6, 3)) #generate sample data
R> dp <- DirichletProcessCreate(y, gammaMd)
R> dp <- Initialise(dp)
R> dp <- Fit(dp, 1000)
R> pf <- PosteriorFrame(dp, ppoints(100)*6, 1000)
R> trueFrame <- data.frame(x=ppoints(100)*6,
+                          y= 0.5*dgamma(ppoints(100)*6, 2, 4) +
+                          0.5*dgamma(ppoints(100)*6, 6, 3))
R> ggplot() +
+    geom_ribbon(data=pf,
+                aes(x=x,ymin=X5.,ymax=X95.),
+                colour=NA, fill="red", alpha=0.2) +
+    geom_line(data=pf, aes(x=x, y=Mean), colour="red") +
+    geom_line(data=trueFrame, aes(x=x, y=y))
```

From Figure 8 we can see that the true distribution has been correctly identified.

4.3. Resampling the Base Measure, G_0

It is helpful that the user knows how to best set the parameters of the base measure to correctly represent the underlying data. However, whilst desirable this is not always practical. In which case **dirichletprocess** offers functionality to use hyper-prior parameters on G_0 and update them with each iteration.

For the mixing distributions that allow for re-sampling of the base measure, it is simple to include the flag `Fit(dp,...,updatePrior=TRUE)`. At each fitting iteration the base measure with variable parameters will be updated based on the current cluster parameters. For details on the exact specification of the hyper-prior distributions for each implemented mixture kernel see Section 2.3. If a user wishes to change the default prior on the hyper parameters then

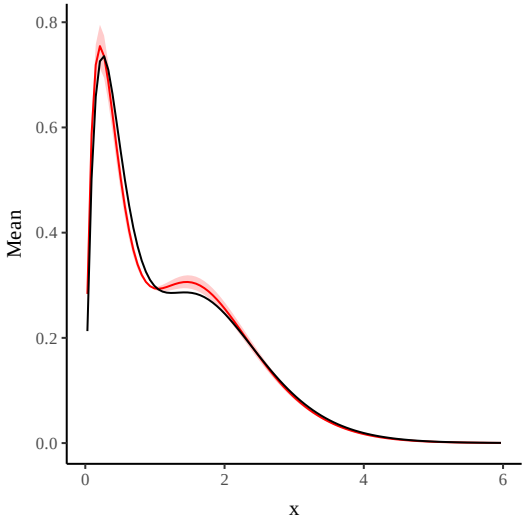


Figure 8: The results of implementing the new Gamma mixture model.

it is as simple as changing the `PriorParametersUpdate` function for the mixing distribution object.

4.4. Extended Example - Working with Censored Observations

The following example is intended to illustrate how simple the **dirichletprocess** package makes it for the user to extend Dirichlet process mixture modeling to situations which are not directly supported by the package, in a less trivial context than the above density estimation examples.

Survival analysis is an area of statistics concerned with analysing the duration of time before an event happens such as a failure in a mechanical system or a death in a biological context. We are concerned with constructing a survival function which as the name indicates shows how the probability of not experiencing the event changes with time. Survival data is often generated by observational studies which result in censoring. In the context of medical statistics, censoring occurs due to finite time periods of the studies. When analysing the effects of medical treatments patients events can be censored for a variety of reasons. This is a missing data problem as we no longer know the exact time at which an event occurred, just that it occurred before of after a specific time. Right censoring is when a patient is yet to be effected by the event after a study ends. Left censoring is when it is not known exactly when the event occurred, just that it occurred before the study started. To deal with this censored information we must adapt our likelihoods.

One approach for modeling such data non parametrically is a Dirichlet process mixture of Weibull distributions. The **dirichletprocess** package does not directly support the analysis of censored data – as stated throughout, the purpose of the package is not to provide the user with a small number of functions for solving predefined problems, but instead to make it easy to use Dirichlet process mixtures in a wide variety of contexts. As such, it is very simple for the user to extend the functionality of the package to allow for censoring.

For the data we replicate the work of Kottas (2006b) and use leukaemia remission times taken from Lawless (2011). This dataset contains two groups of censored observations and we wish

to investigate the impact two different treatments have on leukaemia remission times. We fit a censored Weibull DP model to each of the data sets and compare the survival functions. The full model can be written as

$$\begin{aligned} y_i &\sim \text{Weibull}(y_i \mid a_i, b_i), \\ a_i, b_i &\sim G, \\ G &\sim \text{DP}(\alpha, G_0), \end{aligned}$$

where the Weibull density and G_0 follow the form shown in Section 2.3.

Censored data can come in the form of two columns - the time it takes for the event to occur and an indicator variable; 1 for a right censored observation, 0 otherwise. Therefore the density of the Weibull distribution can be written as

$$\begin{aligned} k(y \mid a, b) &= \frac{a}{b} y^{a-1} \exp\left(-\frac{y^a}{b}\right) \quad \text{for uncensored,} \\ k(y \mid a, b) &= 1 - \exp\left(-\frac{y^a}{b}\right) \quad \text{for censored.} \end{aligned}$$

Therefore we must translate this likelihood into the appropriate function for the **dirichletprocess** package.

```
Likelihood.weibullcens <- function(mdobj, x, theta){
  a = theta[[1]][,,,drop=TRUE]
  b = theta[[2]][,,,drop=TRUE]

  y <- as.numeric(
    b^(-1) * a * x[,1]^(a-1) * exp(-b^(-1) * x[, 1]^a)
  )
  y_cens <- as.numeric(1 - exp(-x[,1]^a / b))

  if(nrow(x) == 1){
    if(x[,2] == 0) return(y)
    if(x[,2] == 1) return(y_cens)
  }
  else{
    y_ret <- y
    y_ret[x[, 2] == 1] <- y_cens[x[, 2]==1]
    return(y_ret)
  }
}
```

We now wish to create two different `mixingDistribution` objects for each of the data sets. Again we use the `MixingDistribution` constructor function and provide the resulting object with a custom class using the `class` function.

```
R> mdobjA <- MixingDistribution("weibullcens",
+                               c(1,2,0.5), "nonconjugate",
```

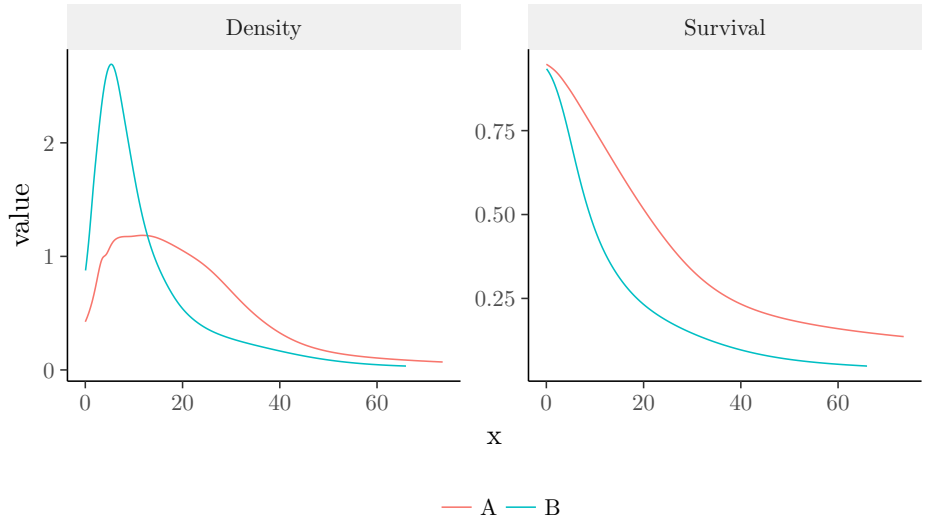


Figure 9: Point estimates for the survival and density functions of the two treatments.

```
+                                  mhStepSize=c(0.11,0.11),
+                                  hyperPriorParameters=c(2.222, 2, 1, 0.05))
R> mdobjB <- MixingDistribution("weibullcens",
+                                  c(1,2,0.5), "nonconjugate",
+                                  mhStepSize=c(0.11,0.11),
+                                  hyperPriorParameters=c(2.069, 2, 1, 0.08))
R> class(mdobjA) <- c("list", "weibullcens", "weibull", "nonconjugate")
R> class(mdobjB) <- c("list", "weibullcens", "weibull", "nonconjugate")
```

We can easily use this modified mixture model for our censored data. The sampling is then carried out as normal with no other changes needed. The default functions available for the Weibull mixture model are applied to our custom dirichletprocess object.

```
R> dpA <- DirichletProcessCreate(data_a, mdobjA, c(2, 0.9))
R> dpA <- Initialise(dpA)
R> dpB <- DirichletProcessCreate(data_b, mdobjB, c(2, 0.9))
R> dpB <- Initialise(dpB)
R> dpA <- Fit(dpA, 500, TRUE)
R> dpB <- Fit(dpB, 500, TRUE)
```

Using the fitted values we can easily extract the estimate density of the survival times using the weights and cluster parameters. The survival function is calculated as $S(y) = 1 - \exp(-\frac{y^a}{b})$. The resulting density and survival estimate is shown in Figure 9 which correctly replicate the findings of Kottas (2006b).

To fully understand what has happened here, it is vital to understand that the DP is defined by its base likelihood and G_0 distribution. In creating a new mixing distribution of class `weibullcens` and `weibull` we are able to include the new likelihood but retain all the previous functions of a Weibull DP mixture. This makes it trivial to define your own likelihoods based off the foundations laid in the different classes available.

4.5. Resampling Component Indexes and Parameters

When calling the `Fit` function on a DP object the component indexes and parameters are resampled following Algorithm 4 for the conjugate case and Algorithm 8 for the non-conjugate case from Neal (2000). For both types of DP mixture the two functions that do the majority of the work are `ClusterComponentUpdate` and `ClusterParameterUpdate`.

In a conjugate DPMM new component indexes and new cluster parameters are drawn directly from the predictive and posterior distributions making the algorithm very efficient. In such case, the only option available to users is to change the prior parameters of the base distribution G_0. Ensuring that the base distribution is correctly parameterised with sensible values for the underlying data will provide optimal performance for the fitting algorithm.

However, in a non-conjugate case new cluster components are proposed from the chosen prior distribution and new cluster parameters are sampled using the Metropolis-Hastings algorithm to obtain a posterior sample. By using the Metropolis-Hastings algorithm, the parameters in question are proposed using a random walk but constrained to the particular support of the parameter. For example, the parameters in a Weibull distribution are strictly positive, therefore the random walk is restricted to fall on the positive real line. An ill proposed prior distribution can severely effect the convergence of the fitting process. The parameter `mhStepSize` in the constructor function for a non-conjugate mixture controls the scale of new parameter proposals for the random walk. When creating a new DP object, the constructor function has a flag `verbose` that outputs an estimated acceptance ratio, for optimal performance of the Metropolis-Hastings algorithm this value should be around 0.234 (Gelman, Roberts, Gilks, and others 1996). Therefore the user should adjust `mhStepSize` to reach this value. As with the conjugate case, care must be taken to ensure that the base measure is well suited for the data.

Overriding Default Behaviour

For both conjugate and non-conjugate mixture models, the user can write their own `ClusterComponentUpdate` and `ClusterParameterUpdate` functions to override the default behaviour. The user can still benefit from the other S3 methods and structures implemented in **dirichletprocess** but with their custom sampling schemes.

For the non-conjugate mixture models there is a further option available to change the component index and parameter re-sampling. In Algorithm 8 of Neal (2000) each datapoint can form a new cluster with parameter drawn from the base measure, these proposals are called 'auxiliary' variables and m are drawn for each data point. By default $m = 3$. However this can be changed in the `Initialise(dp, ...,m=m)` function. Using more auxiliary variables can lead to more changes in the component indexes and greater exploration of the base measure but at the cost of computational time.

4.6. Resampling the Base Measure, G_0

It is helpful that the user knows how to best set the parameters of the base measure to correctly represent the underlying data. However, whilst desirable this is not always practical. In which case **dirichletprocess** offers functionality to use hyper-prior parameters on G_0 and update them with each iteration.

For the mixing distributions that allow for re-sampling of the base measure, it is simple to

include the flag `Fit(dp,...,updatePrior=TRUE)`. At each fitting iteration the base measure with variable parameters will be updated based on the current cluster parameters. For details on the exact specification of the hyper-prior distributions for each implemented mixture kernel see Section 2.3. If a user wishes to change the default prior on the hyper parameters then it is as simple as changing the `PriorParametersUpdate` function for the mixing distribution object.

4.7. Component Prediction

Given a fitted DP object and some new data $\hat{y}$ to predict the cluster labels of the new data we use the command `ClusterLabelPredict`. Using the appropriate algorithm for a conjugate or non-conjugate mixture model the cluster label probabilities are calculated from the new data $\hat{y}$, these probabilities are then sampled once to obtain a cluster label. It is these cluster labels that are returned with the appropriate cluster parameters.

We refer back to our example in Section 3.2 where we used a Dirichlet process to cluster the `faithful` dataset. In this example, we will withhold the last five entries of the data as the prediction set and use `ClusterLabelPredict` to estimate their cluster allocation.

```
R> faithfulTrans <- scale(faithful)
R> trainIndex <- 1:(nrow(faithfulTrans)-5)
R> dp <-  DirichletProcessMvnormal(faithfulTrans[trainIndex, ])
R> dp <- Fit(dp, 1000)
R> labelPred <- ClusterLabelPredict(dp, faithfulTrans[-trainIndex, ])
```

The function `ClusterLabelPredict` works by calculating and sampling the clustering label from the probability that each test point $\hat{y}_j$ belongs to each cluster θ_i or should form its own cluster proportional to α

$$p(i) \propto n_i k(\hat{y}_j \mid \theta_i),$$
$$p(i = \text{new}) \propto \alpha \int k(\hat{y}_j, \theta_i) \mathrm{d}G_0.$$

The function returns a list with multiple entries:

- The predicted cluster labels for the data under `labelPred$componentIndexes`.
- The cluster parameters assigned to the predicted data points in case a new cluster is predicted `labelPred$clusterParameters`.
- The new number of data points per cluster `labelPred$pointsPerCluster`.
- The total number of clusters are also returned `labelPred$numLabels` as this can change with each prediction.

We can then use the results of this function to form a dataframe and plot the results.

```
R> faithfulTrainPlot <- data.frame(faithful[trainIndex, ],
+                                  Label=dp$clusterLabels)
```

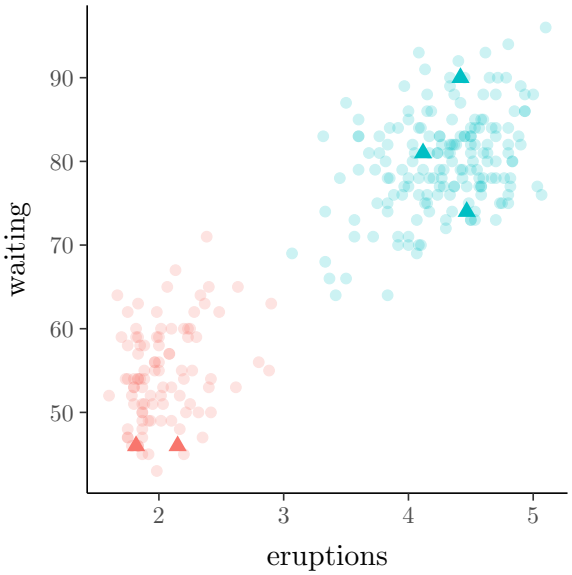


Figure 10: The predicted labels of the last 5 entries of the `faithful` dataset against the training data. The predicted values are indicated by a solid colour and triangle shapes.

```
R> faithfulTestPlot <- data.frame(faithful[-trainIndex, ],
+                                 Label=labelPred$componentIndexes)
R> ggplot() +
+      geom_point(data=faithfulTrainPlot,
+                 aes(x=eruptions,
+                     y=waiting,
+                     colour=as.factor(Label)),
+                 size=1) +
+      geom_point(data=faithfulTestPlot,
+                 aes(x=eruptions,
+                     y=waiting,
+                     colour=as.factor(Label)),
+                 shape=17, size=5) +
+    guides(colour=FALSE)
```

Figure 10 shows the test data being correctly identified with the appropriate cluster.

Computational details

The results in this paper were obtained using R 3.5.0 with the **dirichletprocess** 0.3.1.900 package. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/.

Acknowledgements

We thank Kees Mulder for his contributions. We thank Gianluca Baio and Federico Ricciardi for their helpful comments in the development of this package and vignette.

References

Carpenter, Bob, Gelman, Andrew, Hoffman, Matt, Lee, Daniel, Goodrich, Ben, Betancourt, Michael, Brubaker, Michael A, Guo, Jiqiang, Li, Peter, Riddell, Allen (2016). "Stan: A probabilistic programming language." *J Stat Softw*.

Coles S (2001). *An Introduction to Statistical Modeling of Extreme Values*. Springer Series in Statistics. Springer London, London. ISBN 978-1-84996-874-4 978-1-4471-3675-0.

Escobar MD, West M (1995). "Bayesian Density Estimation and Inference Using Mixtures." *Journal of the American Statistical Association*, **90**(430), 577–588.

Ferguson TS (1973). "A Bayesian Analysis of Some Nonparametric Problems." *The Annals of Statistics*, **1**(2), 209–230.

Gelman A, Carlin JB, Stern HS, Rubin DB (2014). *Bayesian Data Analysis*, volume 2. Chapman & Hall/CRC Boca Raton, FL, USA.

Gelman A, Roberts GO, Gilks WR, others (1996). "Efficient Metropolis jumping rules."

Geman S, Geman D (1984). "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-6**(6), 721–741.

Hastings WK (1970). "Monte Carlo Sampling Methods Using Markov Chains and Their Applications." *Biometrika*, **57**(1), 97–109.

Kim S, Tadesse MG, Vannucci M (2006). "Variable selection in clustering via Dirichlet process mixture models." *Biometrika*, **93**(4), 877–893.

Kottas A (2006a). "Dirichlet process mixtures of beta distributions, with applications to density and intensity estimation." In *Workshop on Learning with Nonparametric Bayesian Methods, 23rd International Conference on Machine Learning (ICML)*.

Kottas A (2006b). "Nonparametric Bayesian survival analysis using mixtures of Weibull distributions." *Journal of Statistical Planning and Inference*, **136**(3), 578–596.

Lawless JF (2011). *Statistical models and methods for lifetime data*, volume 362. John Wiley & Sons.

Maceachern SN, Müller P (1998). "Estimating Mixture of Dirichlet Process Models." *Journal of Computational and Graphical Statistics*, **7**(2), 223–238.

Neal RM (2000). "Markov Chain Sampling Methods for Dirichlet Process Mixture Models." *Journal of Computational and Graphical Statistics*, **9**(2), 249–265.

Salvatier J, Wiecki TV, Fonnesbeck C (2016). "Probabilistic programming in Python using PyMC3." *PeerJ Computer Science*, **2**, e55.

Sethuraman J (1994). "A constructive definition of Dirichlet priors." *Statistica sinica*, pp. 639–650.

Taddy MA, Kottas A (2012). "Mixture Modeling for Marked Poisson Processes." *Bayesian Analysis*, **7**(2), 335–362.

Teh YW, Jordan MI, Beal MJ, Blei DM (2005). "Sharing clusters among related groups: Hierarchical Dirichlet processes." In *Advances in neural information processing systems*, pp. 1385–1392.

Tran D, Kucukelbir A, Dieng AB, Rudolph M, Liang D, Blei DM (2016). "Edward: A library for probabilistic modeling, inference, and criticism." *arXiv:1610.09787 [cs, stat]*. ArXiv: 1610.09787.

West M (1992). *Hyperparameter estimation in Dirichlet process mixture models*. Duke University ISDS Discussion Paper\# 92-A03.

Wickham H (2014). *Advanced r*. CRC Press.

Affiliation:

Gordon J. Ross
School of Mathematics
University of Edinburgh
James Clerk Maxwell Building
Edinburgh, UK
E-mail: gordon.ross@ed.ac.uk
URL: http://gordonjross.co.uk/
and
Dean Markwick
Department of Statistical Science
University College London
1-19 Torrington Place
London, UK
E-mail: dean.markwick.15@ucl.ac.uk
URL: http://dm13450.github.io/