ISSN 2279-9362

msBP: An R Package to Perform Bayesian Nonparametric Inference Using Multiscale Bernstein Polynomials Mixtures

Antonio Canale

No. 480
December 2016

msBP: An R Package to Perform Bayesian Nonparametric Inference Using Multiscale Bernstein Polynomials Mixtures

Antonio Canale
Università di Torino and Collegio Carlo Alberto

Abstract

msBP is a publicly available R package that implements a new method to perform Bayesian multiscale nonparametric inference introduced by Canale and Dunson (2016). The method, based on mixtures of multiscale beta dictionary densities, overcomes the drawback of Polya trees and inherits many of the advantages of Dirichlet process mixture models. The key idea is that an infinitely-deep binary tree is introduced, with a beta dictionary density assigned to each node of the tree. Using a multiscale stick-breaking characterization, stochastically decreasing weights are assigned to each node. The results is an infinite mixture model. The package **msBP** implements a series of basic functions to deal with this family of prior such as random densities and numbers generation, creation and manipulation of binary tree objects, and generic functions to plot and print the results. In addition, it implements the Gibbs samplers for posterior computation to perform multiscale density estimation and multiscale testing of group differences described in Canale and Dunson (2016).

Keywords: Binary trees, Density estimation, Multiscale stick-breaking, Multiscale testing.

1. Introduction

Multiscale methods have received abundant attention in the statistical literature, having several appealing characteristics that pushed their use in many applications. For example, wavelets are routinely used in signal and image processing, nonparametric regression, and density estimation (Donoho, Johnstone, Kerkyacharian, and Picard 1996). However, from the Bayesian perspective, multiscale density estimation is surprisingly understudied. Indeed, most of the approaches rely on single-scale kernel mixtures. Above the other, the Dirichlet process (DP) (Ferguson 1973, 1974) mixtures of Gaussian (Lo 1984; Escobar and West 1995) is the golden standard in many applications. An exception is represented by Polya trees (Mauldin, Sudderth, and Williams 1992; Lavine 1992a,b) that unfortunately have some unappealing characteristics. For example they tend to produce highly spiky densities even when the true density is fairly smooth and are sensible to the prior specification. This sensitivity can be overcome within a mixture approach (Hanson and Johnson 2002), but in this case there is a price to pay in terms of computation. Both the DP and Polya tree mixture models are implemented in the famous **DPpackage** (Jara, Hanson, Quintana, Mueller, and Rosner 2011), an R package that represents the de facto standard software for Bayesian nonparametric inference under a variety of settings.

Canale and Dunson (2016) recently proposed a Bayesian multiscale method that inherits some advantages of the DP mixture and avoids the key disadvantages of Polya trees. The key idea lies in introducing an infinitely-deep binary tree, with a beta dictionary density assigned to each node of the tree. Using a multiscale stick-breaking (Sethuraman 1994) characterization, the authors define a stochastically decreasing sequence of weights assigned to each node of the tree. This formulation implies that within a level of the tree, the densities are equivalent to Bernstein polynomials (Petrone 1999a,b).The DP-like characteristics derives from the formulation of a multiscale generalization of the stick-breaking process, which can be exploited to build an efficient slice sampling algorithm. The same multiscale stick-breaking process has also been used by Wang, Canale, and Dunson (2014) to learn the joint density in massive dimensional settings, using geometric multiresolution analysis to estimate the dictionary densities over the binary tree at a first stage.

The R package **msBP** implements the multiscale stick-breaking process, and its applications to density estimation and to testing of group differences as discussed in Canale and Dunson (2016), and a series of basic R functions to deal with this family of nonparametric prior such as random densities and numbers generation, creation and manipulation of binary trees, and generic functions to plot and print the results. The package's core is written in C++ be means of specific `bintree` data class and it is called from R via the `.C` function.

The rest of the paper is organized as follow: in the next section we outline the theoretical framework with particular emphasis on the multiscale stick-breaking process. Section 3 describes the main features of the C++ implementation while Section 4 is concerned with demonstrating the main features of the package.

2. A multiscale prior for densities

2.1. Basic formulation

Let $x \in \mathcal{X} \subset \mathbb{R}$, be a random variable, g be an unknown density and $x \sim g$. Under a Bayesian perspective g_0 is assumed to be a prior guess for g, with G_0 and G_0^{-1} the corresponding cumulative distribution function (CDF) and inverse CDF, respectively. A prior $g \sim \Pi$ centered on g_0 can be introduced through a prior for the density f of $y = G_0(x) \in (0,1)$. The CDFs F and G corresponding to the densities f and g, respectively, have the following relationship

$$G(x) = F\{G_0(x)\}, x \in \mathcal{X}, \quad F(y) = G\{G_0^{-1}(y)\}, y \in (0,1). \tag{1}$$

The density f is assumed to have the following structure:

$$f(y) = \sum_{s=0}^{\infty} \sum_{h=1}^{2^s} \pi_{s,h} \text{Be}(y; h, 2^s - h + 1), \tag{2}$$

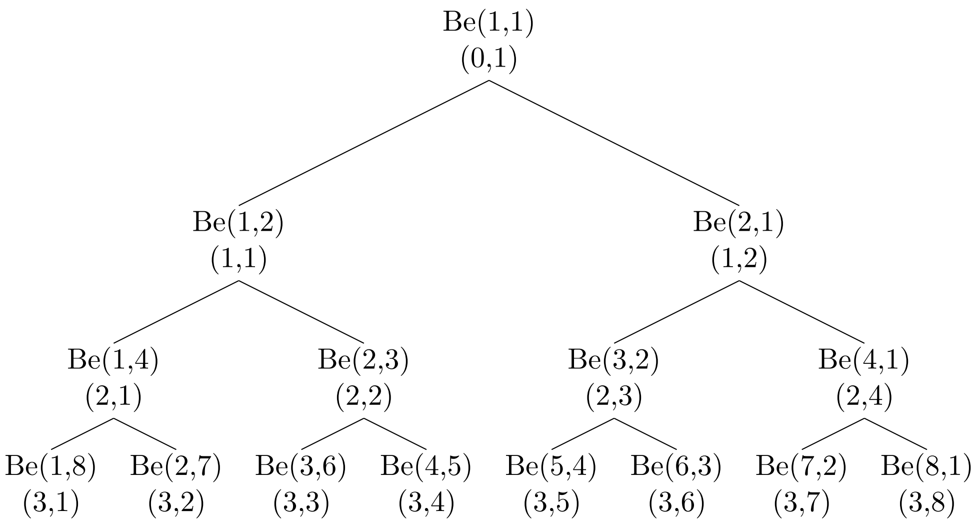


Figure 1: Binary tree with beta kernels at each node (s, h), where s is the scale level and h is the index within the scale.

where Be$(a,\, b)$ denotes the beta density with mean $a/(a+b)$. The sequence of random weights $\{\pi_{s,h}\}$ are constructed via the multiscale stick-breaking process described below. We will refer to the latter construction as multiscale Bernstein polynomial (msBP) model.

To build a multiscale stick-breaking process, an infinite sequence of scales $s = 0, 1, \ldots, \infty$ labelling the levels of an infinite-deep binary tree is introduced. At each scale s there will be 2^s different nodes. A cartoon of the binary tree is reported in Figure 1. To describe a stochastic path from the root node to the leaves, at each scale s and node h within the scale, the following independent random variables are introduced:

$$S_{s,h} \sim \text{Be}(1, a), \quad R_{s,h} \sim \text{Be}(b, b), \tag{3}$$

corresponding to the probability of stopping at node (s, h) and taking the right path after node (s, h) conditionally on not stopping, respectively. This formulation generalizes the stick-breaking process representation of Sethuraman (1994). Each time the stick is broken, it is then randomly divided in two parts (one for the probability of going right, the remainder for the probability of going left) before the next break. Hence, similarly to Sethuraman (1994), the infinite sequence of weights can be defined as

$$\pi_{s,h} = S_{s,h} \prod_{r<s} (1 - S_{r,g_{shr}}) T_{shr} \tag{4}$$

where $g_{shr} = \lceil h/2^{s-r} \rceil$ is the node traveled through at scale r on the way to node h at scale s, $T_{shr} = R_{r,g_{shr}}$ if $(r+1, g_{shr+1})$ is the right daughter of node (r, g_{shr}), and $T_{shr} = 1 - R_{r,g_{shr}}$ if $(r + 1, g_{shr+1})$ is the left daughter of (r, g_{shr}). Note that the general (s, h) node is related to the Be$(h, 2^s - h + 1)$ density.

The above construction leads to a meaningful sequence of weights, i.e., $\sum_{s=0}^{\infty} \sum_{h=1}^{2^s} \pi_{s,h} = 1$ almost surely for any $a, b > 0$ as proved in Lemma 1 of Canale and Dunson (2016). An appealing aspect of this formulation is that it produces a multiscale clustering of the individuals. In particular, two individuals having similar observations may have the same cluster allocation up to some scale s, but are not clustered together on finer scales.

2.2. Bayesian multiscale inference on groups differences

A promising feature of this multiscale stick-breaking process is its ease of generalization to more complex settings than mere density estimation. For example, the sequence of random variables defined in Equation 3 can be generalized to include predictors or other forms of dependence, (e.g., spatial or temporal). Motivated by epigenetic data, Canale and Dunson (2016) modified model (2)–(3), to perform Bayesian multiscale inference on groups difference. Let y_i be a bounded (between zero and one) outcome for subject i with $y_i \sim f_{d_i}$ and $d_i \in \{0, 1\}$. The label d_i denotes a subject's group (e.g., cases/controls, drug/placebo). Using the msBP representation, the hypothesis $f_0 = f_1$ is true if the groups share the same weights over the binary tree. If $f_0 \neq f_1$, we may have the same weights on the dictionary elements up to a given scale, so that the densities are equivalent up to that scale but not at finer scales. Thus, one can also test for $H_0^s : f_0^s = f_1^s$, i.e., no differences between the two groups at scale s. Clearly H_0^0 is true with probability one, and thus a further modification of (3) consists to set $S_{0,1} = 0$.

The subjects surviving up to scale s can stop or progress to the next scale. Let $\mathcal{N}^s$ denote these actions, with $\mathcal{N}^s_{(d)}$ denoting the actions in group d. Conditionally on $\mathcal{N}^s$ the posterior probability of H_0 being true at scale s can be written as

$$\mathrm{pr}(H_0^s|\mathcal{N}^s) = \frac{P_0^s \mathrm{pr}(\mathcal{N}^s|H_0^s)}{P_0^s \mathrm{pr}(\mathcal{N}^s|H_0^s) + (1 - P_0^s)\mathrm{pr}(\mathcal{N}^s|H_1^s)}, \tag{5}$$

where P_0^s is our prior guess for the null being true at scale s and $\mathrm{pr}(\mathcal{N}^s|H_0^s)$ is the probability of the possible actions if H_0 is true up to scale s. To compute the latter, we can use

$$\begin{aligned}\mathrm{pr}(\mathcal{N}^s|H_0^s) &= \int_{\mathcal{T}} \mathrm{pr}(\mathcal{N}^s|\mathcal{T})\mathrm{pr}(\mathcal{T}|a,b)d\mathcal{T} \\ &= \left\{\frac{\Gamma(a+1)}{\Gamma(a)}\frac{\Gamma(2b)}{\Gamma(b)^2}\right\}^{2^s} \int_{\mathcal{T}} \prod_{h=1}^{2^s} S_{s,h}^{n_{s,h}}(1-S_{s,h})^{\hat{a}_{s,h}-1} R_{s,h}^{\hat{b}_{s,h}-1}(1-R_{s,h})^{\hat{c}_{s,h}-1} d\mathcal{T} \\ &= \left\{\frac{\Gamma(a+1)\Gamma(2b)}{\Gamma(a)\Gamma(b)^2}\right\}^{2^s} \prod_{h=1}^{2^s} \frac{\Gamma(1+n_{s,h})\Gamma(\hat{a})}{\Gamma(a+v_{s,h}+1)} \frac{\Gamma(\hat{b})\Gamma(\hat{c})}{\Gamma(2b+v_{s,h}-n_{s,h})},\end{aligned} \tag{6}$$

where $\hat{a}_{s,h} = a + v_{s,h} - n_{s,h}$, $\hat{b}_{s,h} = b + r_{s,h}$, and $\hat{c}_{s,h} = b + v_{s,h} - n_{s,h} - r_{s,h}$, and $v_{s,h}$ is the number of subjects passing through node (s,h), $n_{s,h}$ is the number of subjects stopping at node (s,h), and $r_{s,h}$ is the number of subjects that continue to the right after passing through node (s,h). Similarly

$$\begin{aligned}\mathrm{pr}(\mathcal{N}^s|H_1^s) =& \mathrm{pr}(\mathcal{N}^s_{(0)}|H_1^s) \times \mathrm{pr}(\mathcal{N}^s_{(1)}|H_1^s) \\ &= \left\{\frac{\Gamma(a+1)\Gamma(2b)}{\Gamma(a)\Gamma(b)^2}\right\}^{2^{2s}} \prod_{h=1}^{2^s} \frac{\Gamma(1+n^{(0)}_{s,h})\Gamma(\hat{a}^{(0)})}{\Gamma(a+v^{(0)}_{s,h}+1)} \frac{\Gamma(\hat{b}^{(0)})\Gamma(\hat{c}^{(0)})}{\Gamma(2b+v^{(0)}_{s,h}-n^{(0)}_{s,h})} \times \\ & \prod_{h=1}^{2^s} \frac{\Gamma(1+n^{(1)}_{s,h})\Gamma(\hat{a}^{(1)})}{\Gamma(a+v^{(1)}_{s,h}+1)} \frac{\Gamma(\hat{b}^{(1)})\Gamma(\hat{c}^{(1)})}{\Gamma(2b+v^{(1)}_{s,h}-n^{(1)}_{s,h})},\end{aligned} \tag{7}$$

where $v^{(d)}_{s,h}$ is the number of subjects passing through node (s,h) in group d, $n^{(d)}_{s,h}$ is the number of subjects stopping at node (s,h) in group d, and $r^{(d)}_{s,h}$ is the number of subjects that continue to the right after passing through node (s,h) in group d, with $d = 0, 1$. The global null will be the cumulative product of Equation 5 for each scale.

Motivated by a DNA methylation arrays application, Canale and Dunson (2016) generalized the latter formulation in the case in which $y_i = (y_{i1}, \dots, y_{ip})^T$. To deal with p-dimensional arrays, a prior for P_0^s is assumed so to borrow informations across sites and to learn the joint null probability P_0^s. This feature is not yet implemented in the **msBP** package.

3. The C++ implementation

All the main functions of the **msBP** package are written in C++ and most of them rely on the `bintree` data structure, i.e.,

```
struct bintree
{
        double data;
        struct bintree *left;
        struct bintree *right;
};
```

The `bintree` structure is composed of a root (or parent node), each of which stores data and the two links to the leaves (or daughters nodes). Clearly each leaf connects to two other leaves and it is the beginning of a new, nested, binary tree. A binary tree is a well known data structure with appealing characteristics in computer science. For example, it is possible to easily access and insert data into a binary tree using search and insert functions recursively called on successive leaves. This data structure will be used to store the random variables $S_{s,h}$ and $R_{s,h}$, the weights in the mixtures, and other sample statistics. Basic functions to handle the `bintree` data structure, such as create a tree, write and extract the data on a given node of a tree and so forth, have also been implemented. Among them, the following functions

```
void tree2array(struct bintree *node, double *array, ...)
void array2tree(double *a, int maxScale, struct bintree *node)
```

allow for the conversion of a binary tree structure into an array and vice versa, and have been written to allow the input-output communication of R and C++ via the `.C` function. The first two arguments of the `tree2array` function are the pointers to the binary tree and to the array in which to write the values of the tree. Note that the array needs to be initialized before the use of `tree2array` and needs to have at least length $2^s - 1$, where s is the maximum depth of the tree. The `tree2array` function writes the array as described in Figure 2. The arguments of `array2tree`, instead, are the pointer to the array, an integer denoting the maximum scale of the binary tree, and the pointer to the binary tree structure to populate. In this latter case the binary tree structure need only to be initialized and the function takes care of growing the tree up to the desired depth.

In addition to the basic functions early described, the **msBP** package features more complex functions. However, most of them are then wrapped into R scripts and define the working functions of the package itself. Thus we do not further describe them here.

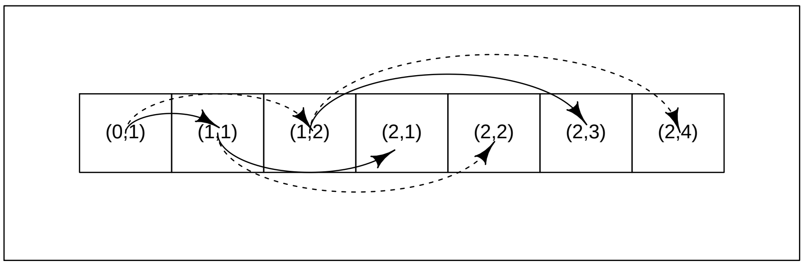


Figure 2: Behavior of the `tree2array` function. Arrows denote the branch of the original binary tree, with continuous line for the right daughter and dashed line for the left daughter. The number inside the array cells represent the original tree indexes.

4. Usage of the msBP package

The main functions of **msBP** are `msBP.Gibbs`, which allows to perform nonparametric density estimation using Gibbs sampler, and `msBP.test` which allows to perform Bayesian multiscale testing of group differences. In this section of the article we provide examples of how to use these functions. In the first subsections, basic and generic functions to handle the multiscale prior, to sample from a msBP process, and to plot the results, are first described. Then, in the second and the third subsections, `msBP.Gibbs` and `msBP.test` are extensively discussed.

4.1. Basic and generic functions

The **msBP** package introduces two new R object class. The first is the `binaryTree` class. An object of the `binaryTree` class represents a finite-depth binary tree. It consists of a list containing `T` and `max.s`, the binary tree itself and an integer denoting its depth, respectively. Specifically, `T` is a list where each element is a vector containing the values of the nodes at a given scale. A binary tree of depth 3 containing the integers from 1 to 15 can be obtained with

```
R> tree <- structure(list( T = list(1, c(2, 3), c(4, 5, 6, 7),
+          c(8, 9, 10, 11, 12, 13, 14, 15)), max.s = 3),
+          class = "binaryTree")
```

The tree structure can be converted into a vector using the `tree2vec` function

```
R> x <- tree2vec(tree)
```

while the `vec2tree(x)` populate a binary tree with the values contained into the vector `x`. The latter function is ideally constructed for vectors of length $2^n - 1$, where $n \in N$. However if the length $l \neq 2^n - 1$ for any n, the function creates a tree up to scale $\lceil \log_2(\lfloor l/2 \rfloor + 1) \rceil$

with the last leaves populated with NA. This object class will be largely used by other higher level functions, since the approach described in Section 2, deals with several binary trees such as Equations 3, 4 and so forth. General plot function is available for the binaryTree object class. The results of plot(tree, ...) is a cartoon of a binary tree with the root node at the top. As additional arguments, the function features: value, size, and white. If value=TRUE the numerical values of each node appears inside the node (up to the number of digits specified by precision); if size=TRUE the size of the nodes are proportional to their values; if white=TRUE the background color of the nodes is white, otherwise it is in color scale (default gray.colors). Figure 3 shows the output of some combinations.

The second object class implemented in the **msBP** package is the msBPTree class. An object of the class msBPTree is a list of 5 elements that represent a random draw from a msBP(a, b) process. The first two elements are the trees of the stopping and descending-to-the-right probabilities, described by Equation 3. Both are object of the class binaryTree with the same max.s. The third and fourth argument are the hyperparameters of the msBP prior, namely a and b. The last value is an integer with the maximum depth of both trees. To simulate a random density from a msBP(a, b) prior truncated at scale 3, the msBP.rtree function can be used as

```
R> set.seed(17012014)
R> draw <- msBP.rtree(a = 5, b = 1, max.s = 3)
```

Note that the last scale have $S_{s,h} = 1$. The induced trees of probabilities, calculated by means of (4) can be obtained with the msBP.compute.prob function as

```
R> weights <- msBP.compute.prob(draw)
```

and the results can be plotted using plot, as it is an object of the class binaryTree. An additional argument root=FALSE let $S_{0,1} = 0$. This can be used, for example, in the settings of Section 2.2. The induced random density can be drawn on a finite grid of length n.points of its domain with the function msBP.pdf, i.e.,

```
R> density <- msBP.pdf(weights, n.points = 100)
R> plot(density$dens ~ density$x, xlab = "x", ylab = "Density", ty = 'l')
```

Given a random density from a msBP(a, b) process, it is also possible to generate a sample of size n from that density. To this end we use the Algorithm 1, implemented in C++ and wrapped into R via the msBP.rsample function. The msBP.rsample needs two parameters only: the sample size n and an object of the msBPTree class.

4.2. Density estimation

Posterior density estimation under the msBP setup relies on Markov chain Monte Carlo (MCMC) sampling algorithm. We briefly recall the algorithm proposed by Canale and Dunson (2016) in what follows. It basically consists of two steps: (i) multiscale cluster allocation conditionally on the current values of the parameters $\{\pi_{s,h}\}$, and (ii) update of the probabilities parameters in the mixture, conditionally on the cluster allocation. Such structure is typical of mixture models in which a first data augmentation allocates the observation to a

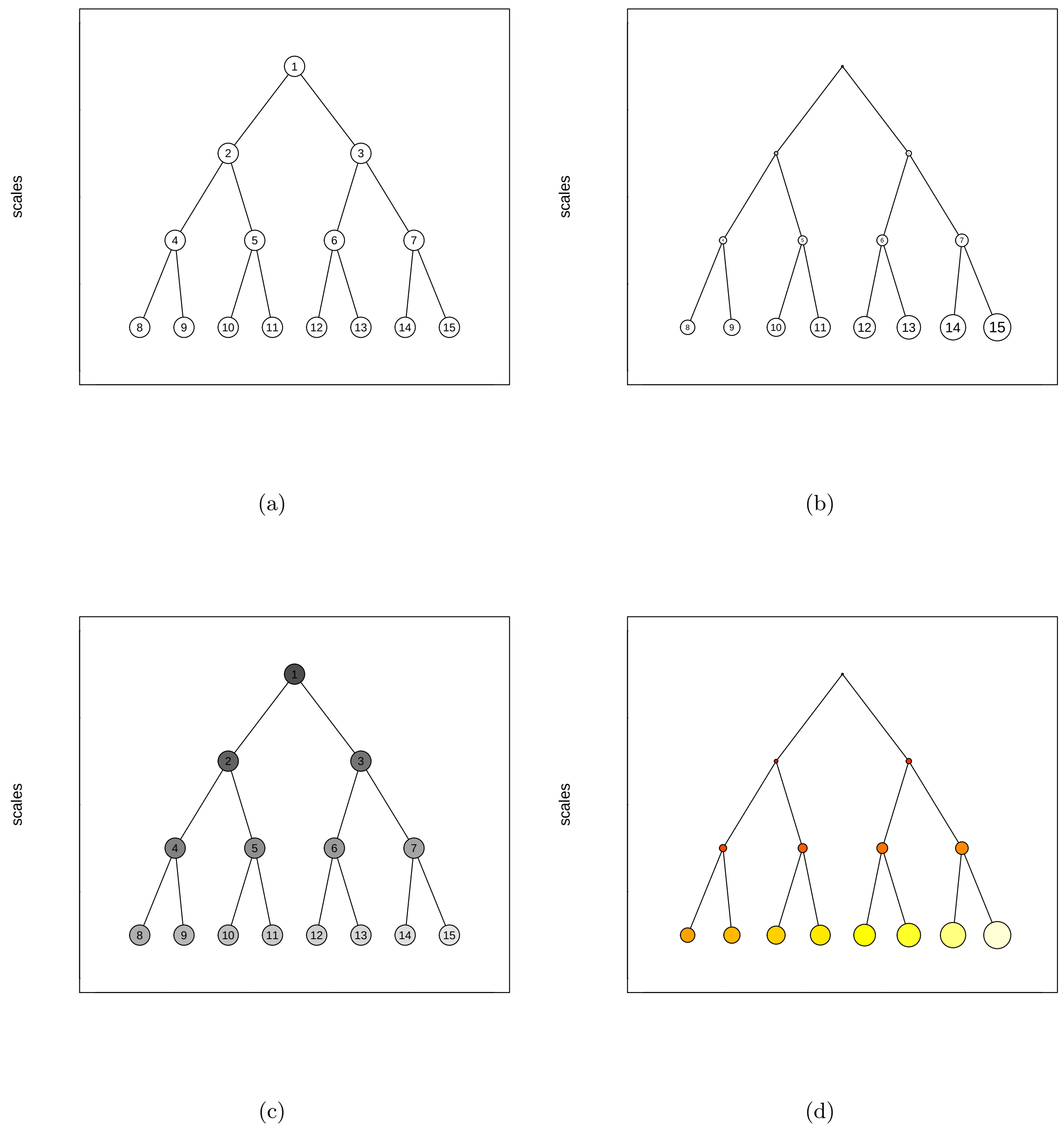


Figure 3: Output of the `plot(tree)` function with (a) default parameters values, (b) `size = TRUE`, (c) `white = FALSE`, and (d) `white = FALSE, size = TRUE, value = FALSE, col.grid = heat.colors(15)`.

Algorithm 1 Generating a random sample from a random density having an msBP prior

```
for $i = 1, \dots, n$ do
  loop = TRUE;
  $s_i = 0$, $h_i = 1$;
  while loop do
    let loop = FALSE with probability $S_{s,h}$.
    if loop then
      with probability $R_{s_i,h_i}$, let $h_i = 2h_i$
      with probability $1 - R_{s_i,h_i}$, let $h_i = 2h_i - 1$
    end if
  end while
  generate $y_i \sim \mathrm{Be}(h_i, 2^{s_i} - h_i + 1)$.
end for
```

mixture component and conditionally on such allocation the parameters of each component are updated (Bush and MacEachern 1996; MacEachern and Müller 1998; Ishwaran and James 2001).

Suppose that s_i and h_i are the scale and the node within scale labels for subject i. Conditionally on the binary tree of weights $\{\pi_{s,h}\}$, the posterior probability of subject i belonging to node (s, h) is simply

$$\mathrm{pr}(s_i = s, h_i = h | y_i, \pi_{s,h}) \propto \pi_{s,h} \mathrm{Be}(y; h, 2^s - h + 1).$$

Now rewrite model (2) as

$$f(y) = \sum_{s=0}^{\infty} \pi_s \sum_{h=1}^{2^s} \bar{\pi}_{s,h} \mathrm{Be}(y; h, 2^s - h + 1),$$

where $\pi_s = \sum_{h=1}^{2^s} \pi_{s,h}$, and $\bar{\pi}_{s,h} = \pi_{s,h}/\pi_s$. The cluster allocation uses a modification of the slice sampler of Kalli, Griffin, and Walker (2011) and is reported in Algorithm 2.

Algorithm 2 Multiscale cluster posterior allocation for ith subject

```
for each scale $s$ do
  calculate $\pi_s = \sum_{h=1}^{2^s} \pi_{s,h}$:
end for
simulate $u_i | y_i, s_i \sim U(0, \pi_{s_i})$;
for each scale $s$ do
  if $\pi_s > u_i$ then
    for $h = 1, \dots 2^s$ do
      compute $\bar{\pi}_{s,h} = \pi_{s,h}/\pi_s$
    end for
    compute $\mathrm{pr}(s_i = s | u_i, y_i) \propto \sum_{h=1}^{2^s} \bar{\pi}_{s,h} \mathrm{Be}(y_i; h, 2^s - h + 1)$
  else
    $\mathrm{pr}(s_i = s | u_i, y_i) = 0$;
  end if
end for
sample $s_i$ with probability $\mathrm{pr}(s_i = s | u_i, y_i) \propto \mathbb{1}(s : \pi_s > u_i) \sum_{h=1}^{2^s} \bar{\pi}_{s,h} \mathrm{Be}(y_i; h, 2^s - h + 1)$;
sample $h_i$ with probability $\mathrm{pr}(h_i = h | y_i, s_i) \propto \bar{\pi}_{s_i,h} \mathrm{Be}(y_i; h, 2^{s_i} - h + 1)$;
```

Algorithm 2 is implemented in the `msBP.postCluster` function. It requires two arguments: the sample of observations `y` and a binary tree of weights `weights`. The function makes a call to the `postCluster` `C++` subroutine. The output of the function is a matrix with `length(y)` rows and two columns of cluster labels: the first denoting the scale and the second denoting the node within a given scale. The same `C++` subroutine called by `msBP.postCluster` is called at each iteration of the MCMC sampler as described below.

Conditionally on the cluster allocations, the stopping and descending-right probabilities can be updated from their full conditional posteriors, namely:

$$S_{s,h} \sim \text{Be}(1 + n_{s,h}, a + v_{s,h} - n_{s,h}), \quad R_{s,h} \sim \text{Be}(b + r_{s,h}, b + v_{s,h} - n_{s,h} - r_{s,h}). \tag{8}$$

Calculation of $v_{s,h}$ and $r_{s,h}$ can be performed via the `msBP.nrvTrees` function, a wrapper calling the `allTree` `C++` subroutine. The input of `msBP.nrvTrees` is the output of `msBP.postCluster`, i.e., a matrix with 2 columns and a number of rows equal to the sample size. The output of `msBP.nrvTrees` is a list containing tree objects of the class `binaryTree`.

The main function implemented in the **msBP** package is the `msBP.Gibbs` function, performing the actual MCMC simulation from the posterior. The function basically iterates between cluster allocation, using the `postCluster` `C++` subroutine and parameter updating, calculating first the elements $n_{s,h}$, $r_{s,h}$, and $v_{s,h}$ by means of the `allTree` `C++` subroutine, and then using Equation 8. The Markov chains sampling is written in `C++` but additional `R` language is used to initialize the function.

To describe the use of `msBP.Gibbs`, we will now walk the reader through the entire process of density estimation under the msBP setup. We will start showing how to elicit prior informations, how to run the sampler, and how to analyze the output of the analysis. We do this using the famous Galaxy dataset (Roeder 1990). The dataset consists on the velocity of 82 galaxies. The histogram of the speeds reveals that the data are clearly multimodal. This feature supports the Big Bang theory, as the different modes of density can be though as clusters of galaxies moving at different speed.

```
R> data("galaxy")
R> galaxy <- data.frame(galaxy)
R> x <- galaxy$speed / 1000
```

We start by discussing prior elicitation. In Section 2.1 we assumed that g_0 is our a prior guess for g, the density of x and we want to center our msBP process in such prior. We discussed that if G_0 and G_0^{-1} are the corresponding CDF and inverse CDF, we can first transform the data with $y = G_0(x) \in (0, 1)$, and then estimate $f \sim \text{msBP}(a, b)$. It can be showed (see Canale and Dunson 2016) that the expectation $E\{F(A)\} = \lambda(A)$, where $\lambda(A)$ is the Lebesgue measure over the set A and $F(A) = \int_A f$. Since th prior for f is centered the uniform, the prior on g is automatically centered in g_0. To allow this from a practical viewpoint we can use the parameter `g0` of the `msBP.Gibbs` function. The package features four different prior guess for g_0: `g0=c("uniform", "normal", "gamma","empirical")` for uniform, normal, gamma and, following and empirical Bayes approach, the empirical kernel density estimate. For `"normal"` and `"gamma"`, the parameters can be fixed using `g0_par` a vector of size two corresponding to mean and standard deviation of the normal, or shape and rate parameters for the gamma, respectively. As default choice the function implements the `"empirical"` specification.

Then one has to specify the values of a and b, the hyperparameters of the msBP prior. The hyperparameter a controls the decline in probabilities over scales. Let $S^{(i)}$ denotes the scale of ith observation. It can be showed that $E(S^{(i)}) = a$ which means that for small a, high probability is placed on coarse scales, leading to smoother densities and as a increases, finer scale densities will be weighted more, leading to spiker realizations. Additional hyperpriors for a and b can be assumed. Clearly, this will lead to have additional sampling steps in the Gibbs sampling algorithm. In the `msBP.Gibbs` function this can be achieved letting `hyper$hyperpriors$a=TRUE` and `hyper$hyperpriors$b=TRUE`, respectively. Specifically the algorithm implements $a \sim Ga(\beta, \gamma)$ and $b \sim Ga(\delta, \lambda)$. This lead to the following conditional posterior distributions:

$$a|- \sim \text{Ga}\left(\beta + 2^{s'+1} - 1, \gamma - \sum_{s=0}^{s'}\sum_{h=1}^{2^s} \log(1 - S_{s,h})\right), \tag{9}$$

and

$$\text{pr}(b|-) \propto \frac{b^{\delta-1}}{B(b,b)^{2^{s'+1}-1}} \exp\left\{ b \left(\sum_{s=0}^{s'}\sum_{h=1}^{2^s} \log\{R_{s,h}(1 - R_{s,h})\} - \lambda \right) \right\}, \tag{10}$$

where s' is the maximum occupied scale and $B(p, q)$ is the Beta function. To sample from the conditional posterior distribution of b, a Griddy-Gibbs approach over the grid defined by `hyper$hyperpar$gridB` is used. See Ritter and Tanner (1992). For sake of illustration, let us assume $a \sim Ga(50, 5)$, and $b \sim Ga(10, 1)$, with the prior for b evaluate only on a grid from 0 to 20 of length 30, i.e.,

```
R> hyper <- list(hyperprior = list(a = TRUE, b = TRUE),
+          hyperpar = list(beta = 50, gamma = 5, delta = 10, lambda = 1,
+          gridB = seq(0, 20, length = 30)))
```

The number of iterations to perform in the MCMC can be set via the function parameter `mcmc`, a list including `nb`, the number of burn-in iterations, `nrep` the total number of iterations (including `nb`), and `ndisplay` the multiple of iterations to be displayed on screen while the C++ routine is running:

```
R> mcmc <- list(nrep = 10000, nb = 5000, ndisplay = 1000)
```

To obtain a posterior estimate of the density, the `grid` parameter need to be fixed. It consists of a list giving the parameters for plotting the posterior mean density over a finite grid of points. It must include `low` and `upp` giving the lower and upper bound respectively of the grid and `n.points`, an integer giving the number of evaluation points.

```
R> grid <- list(n.points = 150, low = 5, upp = 38)
```

Additional parameters to be set are `maxS` and `printing`. The former is an upper bound for the binary trees involved in the MCMC, and the latter is a control parameter. If `printing = TRUE` the C++ prints on standard output what is doing every `ndisplay` iterations. With the following expression we run our MCMC simulation:

```
R> set.seed(2)
R> fit.msbp <- msBP.Gibbs(x, a = 10, b = 10, g0 = "empirical", mcmc = mcmc,
+                     hyper = hyper, printing = 0, maxS = 4, grid = grid)
```

The function output is a list containing three objects:

- `density`: a list containing `postMeanDens`, the posterior mean density estimate evaluated over `xDens` and the related lower and upper pointwise 95% credible bands (`postLowDens` and `postUppDens`).
- `mcmc`: a list containing the MCMC chains: `dens` is a matrix (`nrep-nb`) $\times$ `n.grid`, `a` and `b` are vectors containing the MCMC replicates for the two parameters (if `hyperprior$a` or `hyperprior$b` are set as `TRUE`), `scale` is a matrix where each column is a MCMC sample of the total mass for each scale, `R` and `S`, are matrices where each column in the `tree2vec` form of the corresponding trees, `weights` is a matrix where each column is the `tree2vec` form of the corresponding tree of weights, `s` and `h` are matrices where each column is the MCMC chain for the node labels for a subject.
- `postmean`: a list containing posterior means over the MCMC samples of a, b, and of all binary trees

To plot the posterior mean density and the related 95% credible bands (see Figure 4) it is sufficient to write

```
R> plot(fit.msbp$density$postMeanDens ~ fit.msbp$density$xDens, ty = "l")
```

while, to plot the posterior mean binary tree of weights use simply

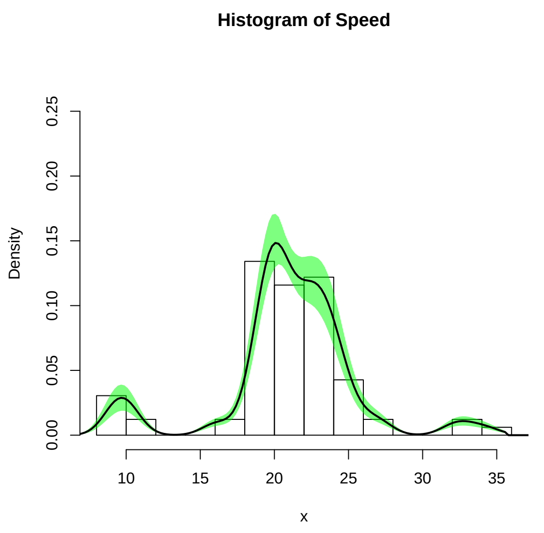
```
R> plot(fit.msbp$postmean$weights, size = TRUE, white = FALSE,
+         value = FALSE, main = "Posterior tree of weights")
```

To assess the convergence of the MCMC, one can have visual inspections of the traceplots of the chains for some parameter of interest. For example, if hyperpriors on the msBP prior parameters have been assumed, one can monitor the convergence of the chains for a and b with

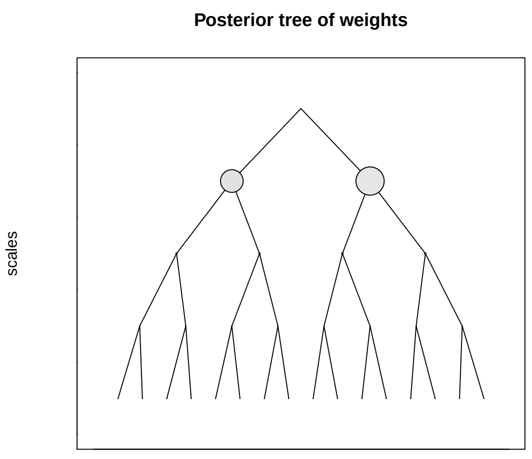
```
R> par(mfrow=c(2,1))
R> plot(fit.msbp$mcmc$a, type = 'l', main  ="Traceplot for a", ylab = "")
R> plot(fit.msbp$mcmc$b, type = 'l', main = "Traceplot for b", ylab = "")
```

and test for convergence using, for example, the Geweke (1992) diagnostics implemented in the **coda** package, i.e.,

```
R> library("coda")
R> geweke.diag(fit.msbp$mcmc$a)
```



(a)



(b)

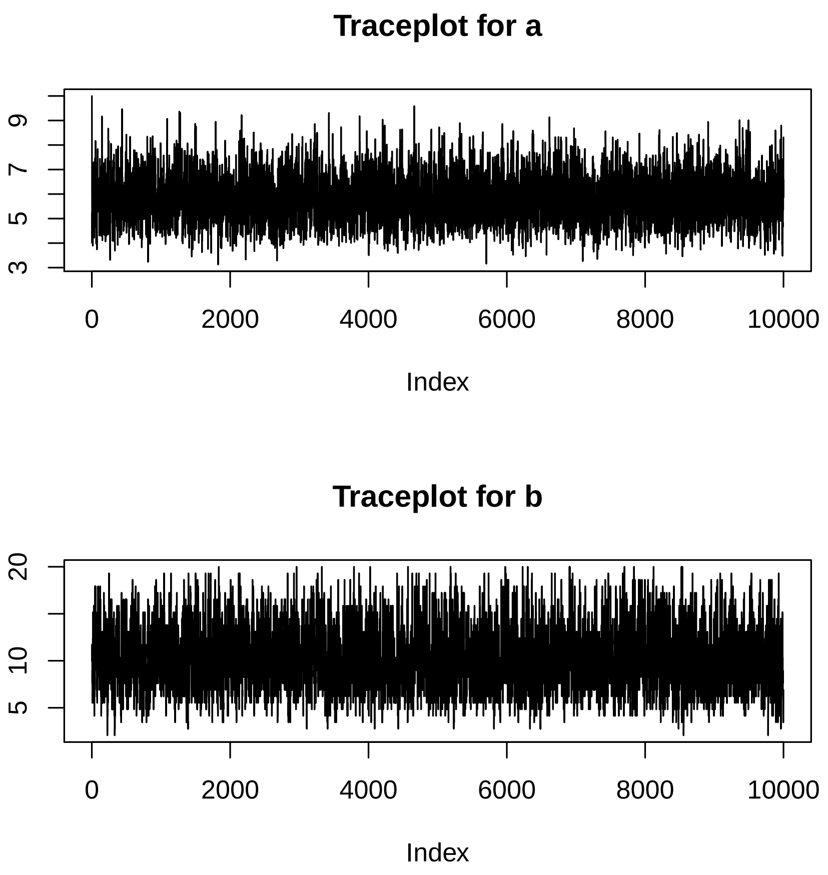
Figure 4: Posterior mean density (a) and tree of weights (b).

```
Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

  var1
0.4721

> geweke.diag(fit.msbp$mcmc$b)

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

  var1
0.1882
```

4.3. Inference in group differences

The function `msBP.test` performs multiscale hypothesis testing of difference in the distribution of two groups. It exploits the closed form expression for the conditional posterior probability for H_0^s in Equation 5. However, since it cannot be directly used due to the dependence on the unknown multiscale clustering structure, the function relies on a Gibbs sampling algorithm. Again, the algorithm is made of two steps: multiscale cluster allocation, and update of the tree of weights. For node h at scale s, let $\pi_{s,h}^{(0)}$ denote the weight under H_0^s and $\pi_{s,h}^{(1,d)}$ for $d = 0, 1$ denote the group-specific weights under H_1^s. The allocation of subject i, at each iteration, will be made via `msBP.postCluster` using the following set of weights:

$$\pi_{s,h}^{(d_i)} = P(H_0^s|\mathcal{N}_{(0)}^s, \mathcal{N}_{(1)}^s)\pi_{s,h}^{(0)} + \{1 - P(H_0^s|\mathcal{N}_{(0)}^s, \mathcal{N}_{(1)}^s)\}\pi_{s,h}^{(1,d_i)}. \tag{11}$$



(a)

Figure 5: Posterior draw for a and b.

Then, at a given iteration the quantities in Equations 6–7 can be calculated explicitly, and used to update the stopping and descending probabilities.

We describe the parameters and the behavior of the function via the Indian Liver dataset, available at the UCI Machine Learning repository (Bache and Lichman 2013). This data set contains data on 580 subjects of which 413 liver patients and 167 non-liver patients. Subjects with liver problems typically register higher levels of bilirubin in their blood and thus we want to test if there is a difference in the distribution of the relative direct bilirubin, calculated as the ratio of the direct bilirubin over the total bilirubin. An histogram of the raw data is reported in Figure 6, Panel (a).

The `msBP.test` function, in addition to a vector of observations and to a vector of group labels, requires a prior value for, a, b, and for the probability of H_0. The choice of the hyperparameters a and b can be made using prior information. In what follows, however, the choice is done with a two-step procedure. First, the density of the pooled dataset is fitted with the `msBP.Gibbs` function assuming hyperpriors for a and b

```
R> mcmc.test = list(nrep = 8000, nb = 4000, ndisplay = 1000)
R> hyper.test = list(hyperprior = list(a = TRUE, b = TRUE),
+          hyperpar = list(beta = 5,gamma = 0.5, delta = 1, lambda = 1))
R> set.seed(17012014)
R> dens.liver <- msBP.Gibbs(liver$dirbil, a = 10, b = 10, mcmc = mcmc.test,
+          hyper = hyper.test, plot.it = TRUE, maxScale = 5)
```

then the posterior mean of a and b are used as plug-in estimates for the testing. We fix the prior probability of H_0 to 0.5 in order to equal weights the null and the alternative, and we fix the upper bound for the scales to 5. The function can be executed via:

```
R> test.fit <- msBP.test(liver$dirbil, a = dens.liver$postmean$a,
+          b=dens.liver$postmean$b, group = liver$group,
+          priorH0 = 0.5, mcmc = mcmc.test, plot.it = TRUE, maxScale = 5)
```

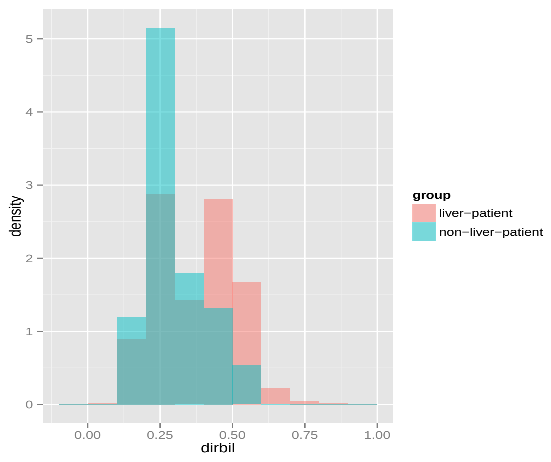
The function's output is a list containing all the MCMC replicates for $\mathrm{pr}(H_0^s|-)$ and their posterior mean. Figure 6 (b) reports the posterior mean of the global null hypothesis, in function of the scale. The differences between the two groups can be noted soon at the first scale and it is even more evident for increasing scales.

5. Conclusions

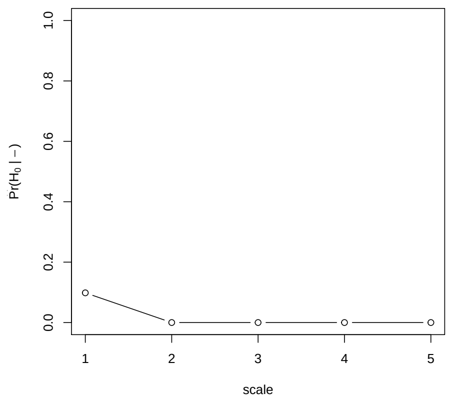
We have presented a detailed introduction to the R package **msBP**, which implements a recently introduced multiscale stick-breaking prior and allows to perform density estimation and to test for differences in the distribution of two groups. The package implements also basic and generic functions to handle the involved multiscale trees structures.

Acknowledgement

The author thanks David Dunson and Giovanna Menardi for comments on early versions of the manuscript. Comments on the package implementation by Roberto Vigo, Marco Pischedda, and Brian Ripley are gratefully acknowledged.



(a)



(b)

Figure 6: Histogram of the raw data (a) and posterior mean of H_0 in function of s for the Indian liver dataset.

References

Bache K, Lichman M (2013). "UCI Machine Learning Repository." URL `http://archive.ics.uci.edu/ml`.

Bush CA, MacEachern SN (1996). "A semiparametric Bayesian model for randomised block designs." *Biometrika*, **83**(2), 275–285.

Canale A, Dunson D (2016). "Multiscale Bernstein Polynomial for Densities." *Statistica Sinica*, **26**, 1175–1195.

Donoho D, Johnstone I, Kerkyacharian G, Picard D (1996). "Density estimation by wavelet thresholding." *Annals of Statistics*, **24**, 508–539.

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

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

Ferguson TS (1974). "Prior Distributions on Spaces of Probability Measures." *The Annals of Statistics*, **2**, 615–629.

Geweke J (1992). "Evaluating the Accuracy of Sampling-based Approaches to the Calculation of Posterior Moments." In JM Bernardo, JO Berger, AP Dawid, AFM Smith (eds.), *Bayesian Statistics 4*. Oxford: Oxford University Press.

Hanson T, Johnson WO (2002). "Modeling regression error with a mixture of Polya trees." *Journal of the American Statistical Association*, **97**, 1020–1033.

Ishwaran H, James Lancelot F (2001). "Gibbs Sampling Methods for Stick Breaking Priors." *Journal of the American Statistical Association*, **96**(453), 161–173.

Jara A, Hanson T, Quintana FA, Mueller P, Rosner GL (2011). "DPpackage: Bayesian Semi- and Nonparametric Modeling in R." *Journal of Statistical Software*, **40**(5), 1–30. ISSN 1548-7660. URL `http://www.jstatsoft.org/v40/i05`.

Kalli M, Griffin J, Walker S (2011). "Slice sampling mixture models." *Statistics and Computing*, **21**(1), 93–105. ISSN 0960-3174.

Lavine M (1992a). " Some aspects of Polya tree distributions for statistical modelling." *Annals of Statistics*, **20**, 1222–1235.

Lavine M (1992b). " More aspects of Polya tree distributions for statistical modelling." *Annals of Statistics*, **22**, 1161–1176.

Lo AY (1984). "On a Class of Bayesian Nonparametric Estimates: I. Density Estimates." *The Annals of Statistics*, **12**, 351–357.

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

Mauldin D, Sudderth WD, Williams SC (1992). "Polya trees and random distributions." *Annals of Statistics*, **20**, 1203–1203.

Petrone S (1999a). "Bayesian density estimation using Bernstein polynomials." *Canadian Journal of Statistics*, **27**, 105–126.

Petrone S (1999b). "Random Bernstein polynomials." *Scandinavian Journal of Statistics*, **26**, 373–393.

Ritter C, Tanner MA (1992). "Facilitating the Gibbs sampler: the Gibbs stopper and the griddy-Gibbs sampler." *Journal of the American Statistical Association*, **87**(419), 861–868.

Roeder K (1990). "Density estimation with confidence sets emplified by superclusters and voids in galaxies." *Journal of the American Statistical Association*, **85**, 617–624.

Sethuraman J (1994). "A Constructive Definition of Dirichlet Priors." *Statistica Sinica*, **4**, 639–650.

Wang Y, Canale A, Dunson DB (2014). "Scalable multiscale density estimation." *arXiv:1410.7692.*