profdpm: An R Package for MAP Estimation in a Class of Conjugate Product Partition Models

Matthew S. Shotwell
Vanderbilt University

Abstract

The **profdpm** package facilitates inference at the posterior mode for a class of product partition models (PPM). Dirichlet process mixtures are currently the only available class members. Several methods are implemented to search for the maximum posterior estimate of the data partition. This article discusses the relevant theory, the R and underlying C implementation, and examples of high level functionality.

Keywords: product partition model, MAP estimate, clustering, R, C.

1. Introduction

profdpm is an extension package for the R language and environment for statistical computing (**?**). This package facilitates inference at the posterior mode in a class of conjugate product partition models (PPM) by approximating the maximum *a posteriori* data partition. The class of PPMs is motivated by an augmented formulation of the Dirichlet process mixture, which is currently the only available member of this class. The **profdpm** package consists of two model-fitting functions, `profBinary` and `profLinear`, their associated summary methods `summary.profBinary` and `summary.profLinear`, and a function (`pci`) that computes several metrics of agreement between two data partitions. However, the **profdpm** package was designed to be extensible to other types of product partition models.

The remainder of this article proceeds as follows: the relevant theory of product partition models is discussed in Section 2, examples and discussion are presented in Sections 3 and 4, and Section 5 discusses some future directions for the **profdpm** package. The underlying C methods are detailed in an appendix.

2. Product partition models

Consider the following hierarchical Bayesian model for a collection of possibly multivariate observations $\boldsymbol{y} = \{y_1, \ldots, y_n\}$:

$$\begin{aligned} y_i | z_i = k, \phi_k &\sim f(y_i|\phi_k) \\ \phi_k &\sim \pi_\phi(\phi_k) \end{aligned}$$

$$\boldsymbol{z} \sim \pi_{\boldsymbol{z}}(\boldsymbol{z}) \propto \prod_{k=1}^{r} c_k(\boldsymbol{z}),$$

where $\boldsymbol{z} = \{z_1, \ldots, z_n\}$ is a collection of cluster membership variables such that $z_i = k$ indicates that observation i is a member of cluster k. The collection $\boldsymbol{z}$ represents a partition of $\boldsymbol{y}$ into r clusters, identified by the r unique values among $\boldsymbol{z}$. The values of $\boldsymbol{z}$ are not important, as long as they are distinct. For simplicity, the positive integers are used to enumerate the distinct values. The tilde symbol '$\sim$' is read 'independently distributed as'. The function f is a probability density indexed by parameter ϕ_k. For $k = 1, \ldots, r$, ϕ_k are independently distributed according to prior density π_ϕ. The prior mass function $\pi_{\boldsymbol{z}}$ is proportional to a product of *cohesion* functions c_k, which specify the prior distribution of the data partition. The Dirichlet process mixture has a PPM representation when $c_k(\boldsymbol{z}) = \alpha\Gamma(n_k)$, where α is the scalar Dirichlet process 'precision' parameter, Γ is the gamma function, and n_k represents the number of observations assigned to the k^{th} cluster. Alternative cohesions yield other well-known process mixtures (for a partial listing, see **?**). The posterior distribution over the data partition is proportional to the product

$$p(\boldsymbol{z}|\boldsymbol{y}) \propto \prod_{k=1}^{r} c_k(\boldsymbol{z}) \int L(\phi_k|\boldsymbol{y}, \boldsymbol{z})\pi_\phi(\phi_k)d\phi_k, \tag{1}$$

where $L(\phi_k|\boldsymbol{y}, \boldsymbol{z}) = \prod_{i=1}^{n} f(y_i|\phi_k)^{I(z_i=k)}$ is the cluster-specific likelihood. Hence, the product partition model is conjugate in the sense that both prior and posterior may be written as a product of cluster-specific terms. Further details regarding the PPM are given in **?** and **?**.

A maximum *a posteriori* (MAP) estimate of the data partition variable $\boldsymbol{z}$ partitions the data into clusters. The principal difficulty with MAP estimation is the size of the partition space (see the 'Bell number', **??**). Hence, computing the MAP estimate using enumerative methods is not practical.

Profile inference about the parameter ϕ_k is conditional on an estimate of the data partition. Conditional on the estimate, $\{\phi_1, \ldots, \phi_r\}$ are independent *a posteriori*, and distributed according to

$$p(\phi_k|\boldsymbol{y}, \hat{\boldsymbol{z}}) \propto L(\phi_k|\boldsymbol{y}, \hat{\boldsymbol{z}})\pi_\phi(\phi_k).$$

For PPMs implemented in the **profdpm** framework, the likelihood and prior over ϕ_k are selected to be conjugate in ϕ_k. Hence, the integral in Equation 1 has simplified form.

The **profdpm** package computes a MAP estimate for two types of PPMs, corresponding to the R functions `profLinear` and `profBinary`. Both functions accept model formulae, and return an instance of the class `profLinear` or `profBinary` respectively. Instances of these classes consist of a data partition estimate and other information related to profile inference.

2.1. Product partition of linear models

The `profLinear` function fits a product partition of conjugate normal linear models:

$$\begin{aligned} y_i|\boldsymbol{x}_i, z_i = k, \boldsymbol{\mu}_k, \tau_k &\sim N(\boldsymbol{x}_i'\boldsymbol{\mu}_k, \tau_k) \\ \boldsymbol{\mu}_k, \tau_k &\sim N_q G(\boldsymbol{m}_0, s_0 I_q, a_0/2, 2/b_0) \\ \boldsymbol{z} &\sim \pi_{\boldsymbol{z}}(\boldsymbol{z}) \propto \prod_{k=1}^{r} \alpha\Gamma(n_k), \end{aligned} \tag{2}$$

where y_i is a continuous scalar observation, $\boldsymbol{x}_i$ is a covariate vector of dimension q, N represents the normal distribution with mean $\boldsymbol{x}'\boldsymbol{\mu}_k$ and precision τ_k, and N_qG represents the $(q+1)$-variate normal-gamma distribution with mean $\boldsymbol{m}_0$, precision matrix $\tau s_0 I_q$, shape $a_0/2$, and scale $2/b_0$. The observation vector and design matrix are specified using the R formula mechanism. The prior parameters $\boldsymbol{m}_0$, s_0, a_0, b_0, and α may also be specified as arguments to `profLinear`. Conditional on an estimated partition, the pairs $\{(\boldsymbol{\mu}_1, \tau_1), \ldots, (\boldsymbol{\mu}_r, \tau_r)\}$ are independent *a posteriori* and distributed according to the $(q+1)$-variate normal gamma distribution with mean $\boldsymbol{m}_k$, precision matrix $\tau \boldsymbol{S}_k$, shape $a_k/2$, and scale $2/b_k$. The posterior statistics $\boldsymbol{m}_k$, $\boldsymbol{S}_k$, a_k, and b_k are returned by `profLinear` for each cluster.

2.2. Product partition of binary models

The `profBinary` function fits a product partition of conjugate binary models:

$$\begin{aligned} y_{ij}|z_i = k, \phi_{kj} &\sim B(\phi_{kj}) \\ \phi_{kj} &\sim \beta(a_0, b_0) \\ \boldsymbol{z} &\sim \pi_{\boldsymbol{z}}(\boldsymbol{z}) \propto \prod_{k=1}^{r} \alpha\Gamma(n_k), \end{aligned} \tag{3}$$

where y_{ij} for $j = 1, \ldots, q$ is the j^{th} binary observation for a subject i, B represents the Bernoulli distribution with probability ϕ_{kj}, and β represents the beta distribution with shape parameters a_0 and b_0. Conditional on an estimated partition, ϕ_{kj} are independently beta distributed *a posteriori* with shape parameters a_{kj} and b_{kj}. This PPM models multivariate binary outcomes only (*i.e.*, without covariates). Hence, the set of outcomes is specified by supplying a one-sided R formula. The `profBinary` function returns the posterior statistics a_{kj} and b_{kj} for $k = 1, \ldots, r$ and $j = 1, \ldots, q$.

2.3. Partition Estimation

The **profdpm** package utilizes four methods to approximate the MAP estimate in an iterative fashion. The first method is the agglomerative method of **?**, later used by **?** in the context of Dirichlet process mixtures. Initially, all observations are partitioned into separate clusters. Two clusters are merged at each subsequent step such that the change in marginal posterior mass is largest. This process repeats until all observations are partitioned into a single cluster. This method produces a deterministic and finite sequence of data partition estimates. The estimate with largest marginal posterior mass is selected to approximate the MAP estimate. Although the agglomerative method evaluates the marginal posterior mass function $O(n^2)$ times, this method is usually the second fastest among the implemented methods. This method does not yield arbitrarily precise approximations to the MAP estimate, but often performs well in practice.

The second method is the Polya urn Gibbs sampler of **?**, **?**, and **?**. The Gibbs sampler sequentially samples from the full conditional distributions having mass functions of the form $p(z_i|\boldsymbol{z}_{-i}, \boldsymbol{y})$, where $\boldsymbol{z}_{-i}$ is the collection of cluster membership variables with the exception of z_i. The Gibbs method produces a sample from the posterior distribution. The MAP estimate is approximated by the sampled partition that achieves the highest posterior mass. Larger samples from the posterior distribution yield increasingly precise approximations. The Gibbs

method is guaranteed to find the MAP estimate in finite time, but is prone slow mixing in the partition space, and is computationally intensive.

For the Gibbs method, the **profdpm** package stores only the most recently sampled partition and the approximate MAP estimate. Hence, there is no simple mechanism to retrieve the entire sequence of sampled partitions. However, setting the `sampler` argument to `TRUE` (for `profLinear` or `profBinary`) causes the most recently sampled partition to be returned, rather than the approximate MAP partition. Additional partitions may be sampled from the Markov chain using the `sampler`, `clust`, and `maxiter` arguments in combination. In this way, a sample from the posterior distribution may be constructed and used to characterize, for example, the posterior distribution for the number of clusters.

The third strategy is a variant of the Sequential Update and Greedy Search (SUGS) method proposed by **?**. The SUGS algorithm assigns observations to clusters sequentially. Initially, the first observation forms a singleton cluster. Subsequent observations are assigned to an existing cluster, or form a new singleton cluster by optimizing the associated posterior probability, conditional on previous cluster assignments. The entire process is repeated in random order. That is, over repeated application, the result is independent of the original data ordering. SUGS is the fastest optimization method, but may perform poorly in MAP estimation of data partitions. Indeed, SUGS was designed for nonparametric applications where the data partition is a nuisance parameter. Hence, SUGS is usually reserved for initial partitioning.

The last method is an iterative stochastic search utilizing 'explode' and 'merge' operations on the clusters of a partition (**?**). At the explode step, a randomly selected subset of observations are redistributed uniformly at random to an existing or new cluster. Each of the exploded observations are then merged with one of the existing clusters in a sequentially optimal fashion. The explode-merge method is motivated by the split-merge Metropolis Hastings algorithms of **?**, and **??**. This method utilizes a Markov chain to approximate the MAP estimate, but does not sample from the posterior distribution over the data partition. Hence, the explode-merge method avoids the complexity and computational expense of ensuring the chain is ergodic.

The following pseudocode routine describes the stochastic method from iteration t to $t+1$. The return value is the MAP estimate at time $t+1$.

set $\boldsymbol{z}' = \boldsymbol{z}^{(t)}$
draw n_J from $\{1, \ldots, n\}$
draw vector J of length n_J from $\{1, \ldots, n\}$ w/o replacement
for j in J **do**
  draw z'_j from $\{1, \ldots, n\}$ (explode step)
end for
if $p(\boldsymbol{z}'|\boldsymbol{y}) > p(\boldsymbol{z}^{(t)}|\boldsymbol{y})$ **then**
  return $\boldsymbol{z}'$
end if
for j in J **do**
  set z'_j to maximize $p(z'_j|\boldsymbol{z}'_{-j}, \boldsymbol{y})$ (merge step)
end for
if $p(\boldsymbol{z}'|\boldsymbol{y}) > p(\boldsymbol{z}^{(t)}|\boldsymbol{y})$ **then**
  return $\boldsymbol{z}'$
else
  return $\boldsymbol{z}^{(t)}$

17: **end if**

The explode-merge optimization terminates when the change in $p(\boldsymbol{z}'|\boldsymbol{y})$ from one iteration to the next is less than a threshold value (*i.e.*, the `crit` argument to `profBinary` and `profLinear`). Because the partition state is discrete, improvements in the marginal posterior mass may be irregular. The **profdpm** package implements a moving-average in calculating improvement.

3. Illustration: `profLinear`

The **profdpm** package adheres to the principle of 'convention over configuration'. That is, reasonable default values are selected for model and optimization parameters, but these may also be configured. This strategy contrasts with the BUGS (**?**) language, where most modeling and sampling aspects must be configured explicitly.

3.1. Default Parameters

By default, the `profLinear` and `profBinary` functions utilize the agglomerative optimization method. This method is deterministic, relatively fast, and yields MAP estimates that are often comparable with other optimization strategies. However, the best MAP approximations are usually obtained using the Gibbs and stochastic search methods, at the expense of additional computation and other consequences of stochastic optimization (*e.g.*, estimates may not be identical over repeated optimizations). The SUGS method is recommended only for fast initial partition estimation, or in nonparametric applications of the kind considered by **?**.

All PPMs implemented in the **profdpm** framework must specify the Dirichlet process precision parameter α, which affects the prior distribution over the data partition parameter. Smaller values result in more concentrated prior mass on partitions with fewer clusters. Conversely, larger values for α concentrate prior mass on partitions with additional clusters. This effect may be summarized by computing the prior expected number of clusters. **?** gives two expressions, one approximate and one exact, for the expected number of clusters in a Dirichlet process mixture. The exact expression is given by

$$\sum_{i=1}^{n} \frac{\alpha}{\alpha + i - 1}.$$

The approximate expression is

$$\alpha \log\left(\frac{n+\alpha}{\alpha}\right).$$

When $\alpha >> 1$, the approximate expression performs well, otherwise the exact expression should be used.

PPMs may be used to detect outlying heterogeneity in otherwise homogeneous observations (**?**). The corresponding prior belief dictates that prior mass on the space of data partitions should be concentrated on the partition consisting of just one cluster. For the Dirichlet process mixture, this is enforced by selecting α to be small. The `profLinear` and `profBinary` functions both set `alpha = 1/1000` by default. Hence, for $n < 1000$ and $\alpha = 1/1000$, the expected number of clusters *a priori* is less than 1.01. Considering that 1.00 is the minimum

expected number of clusters for all α and n, taking $\alpha = 1/1000$ clearly concentrates prior mass on partitions with few clusters.

The PPM of linear models has four additional prior parameters; a_0, b_0, s_0, and $\boldsymbol{m}_0$. Because the model is selected to be conjugate, the posterior parameters may be written in simple terms:

$$\begin{aligned} \boldsymbol{S}_k &= s_0 \boldsymbol{I}_q + \boldsymbol{X}^{(k)\prime} \boldsymbol{X}^{(k)} \\ \boldsymbol{m}_k &= \boldsymbol{S}_k^{-1} (s_0 \boldsymbol{m}_0 + \boldsymbol{X}^{(k)\prime} \boldsymbol{y}^{(k)}) \\ a_k &= a_0 + n_k \\ b_k &= b_0 + \boldsymbol{y}^{(k)\prime} \boldsymbol{y}^{(k)} + s_0 \boldsymbol{m}_0' \boldsymbol{m}_0 - \boldsymbol{m}_k' \boldsymbol{S}_k \boldsymbol{m}_k, \end{aligned} \tag{4}$$

for $i = 1 \ldots n$, where $\boldsymbol{y}^{(k)}$ is the vector formed by concatenating all of the y_i in the k^{th} cluster, $\boldsymbol{X}^{(k)}$ is the matrix formed by joining as rows, the $\boldsymbol{x}_i$ of the k^{th} cluster, and n_k is the number of groups comprising the k^{th} cluster.

Equation 4 isolates the influence of prior parameters on posterior quantities. In particular, prior influence may be reduced by setting each parameter close to zero. Note that the prior (normal-gamma) distribution is improper for a_0 less than one. However, the posterior is always proper for $n \geq 1$. The `profLinear` function specifies $a_0 = 0.001$, $b_0 = 0.001$, $s_0 = 0.001$, and $\boldsymbol{m}_0 = [0.000, \ldots, 0.000]$ by default. These default values encode a heavy-tailed prior distribution over the linear coefficients, and weights the posterior distribution more heavily by the observed data.

3.2. `profLinear` Example

The following example simulates a dataset consisting of 99 longitudinal measurements on 33 units of observation, or subjects. Each subject is measured at three times, drawn uniformly and independently from the unit interval. Each of the three measurements per subject are drawn independently from the normal distribution with one of three linear mean functions of time, and with unit variance. The linear mean functions vary by intercept and slope. The longitudinal structure imposes a grouping among measurements on a single subject. Observations grouped in this way should always cluster together. A grouping structure is specified using the `group` parameter; a factor that behaves similarly to the `groups` parameter of **lattice** graphics functions. For the PPM of conjugate binary models, the grouping structure is imposed by the model formula. More specifically, grouped observations correspond to rows of the model matrix, resulting from a call to `model.matrix` on the formula passed to `profBinary`. Hence, the `profBinary` function does not have a `group` parameter in its prototype.

The goal of the following example is to recover the simulated partition and to create simultaneous 95% credible bands for the mean within each cluster. The following R code block creates and fits the simulated dataset. To ensure that the figures and statistics presented in this document are more reproducible, the pseudo random number generator seed is fixed.

```
R> set.seed(42)
R> sim <- function(multiplier = 1) {
R+     x <- as.matrix(runif(99))
R+     a <- multiplier * c(5,0,-5)
```

```
R+     s <- multiplier * c(-10,0,10)
R+     y <- c(a[1]+s[1]*x[1:33],
R+            a[2]+s[2]*x[34:66],
R+            a[3]+s[3]*x[67:99]) + rnorm(99)
R+     group <- rep(1:33, rep(3,33))
R+     return(data.frame(x=x,y=y,gr=group))
R+ }
R> dat <- sim()
R> library("profdpm")
R> fitL <- profLinear(y ~ x, group=gr, data=dat)
```

The `profLinear` function returns an object of class `profLinear`. The `summary.profLinear` method prints and returns (invisibly) summary information regarding the estimated product partition model, including 95% credible intervals for the linear coefficients under a Laplace approximation to the profile posterior distribution.

Figure 1 presents the simulated data in a scatterplot. For each cluster identified in the estimated partition, the profile posterior mean is represented by a dashed line. Simultaneous 95% credible bands (using "Method 3" of **?**) are represented by flanking solid lines.

```
R> sfitL <- summary(fitL)
R> plot(fitL$x[,2], fitL$y, col=grey(0.9), xlab='x', ylab='y')
R> for(grp in unique(fitL$group)) {
R+     ind <- which(fitL$group==grp)
R+     ord <- order(fitL$x[ind,2])
R+     lines(fitL$x[ind,2][ord],
R+           fitL$y[ind][ord],
R+           col=grey(0.9))
R+ }
R> for(cls in 1:length(sfitL)) {
R+     # The following implements the (3rd) method of
R+     # Hanson & McMillan (2012) for simultaneous credible bands
R+
R+     # Generate coefficients from profile posterior
R+     n   <- 1e4
R+     tau <- rgamma(n, shape=fitL$a[[cls]]/2, scale=2/fitL$b[[cls]])
R+     muz <- matrix(rnorm(n*2, 0, 1),n,2)
R+     mus <- (muz / sqrt(tau)) %*% chol(solve(fitL$s[[cls]]))
R+     mu  <- outer(rep(1,n), fitL$m[[cls]]) + mus
R+
R+     # Compute Mahalanobis distances
R+     mhd <- rowSums(muz^2)
R+
R+     # Find the smallest 95% in terms of Mahalanobis distance
R+     # I.e., a 95% credible region for mu
R+     ord <- order(mhd, decreasing=TRUE)[-(1:floor(n*0.05))]
R+     mu  <- mu[ord,]
R+
```

```
R+     #Compute the 95% credible band
R+     plotx <- seq(min(dat$x), max(dat$x), length.out=200)
R+     ral  <- apply(mu, 1, function(m) m[1] + m[2] * plotx)
R+     rlo  <- apply(ral, 1, min)
R+     rhi  <- apply(ral, 1, max)
R+     rmd  <- fitL$m[[cls]][1] + fitL$m[[cls]][2] * plotx
R+
R+     lines(plotx, rmd, col=cls, lty=2)
R+     lines(plotx, rhi, col=cls)
R+     lines(plotx, rlo, col=cls)
R+ }
```

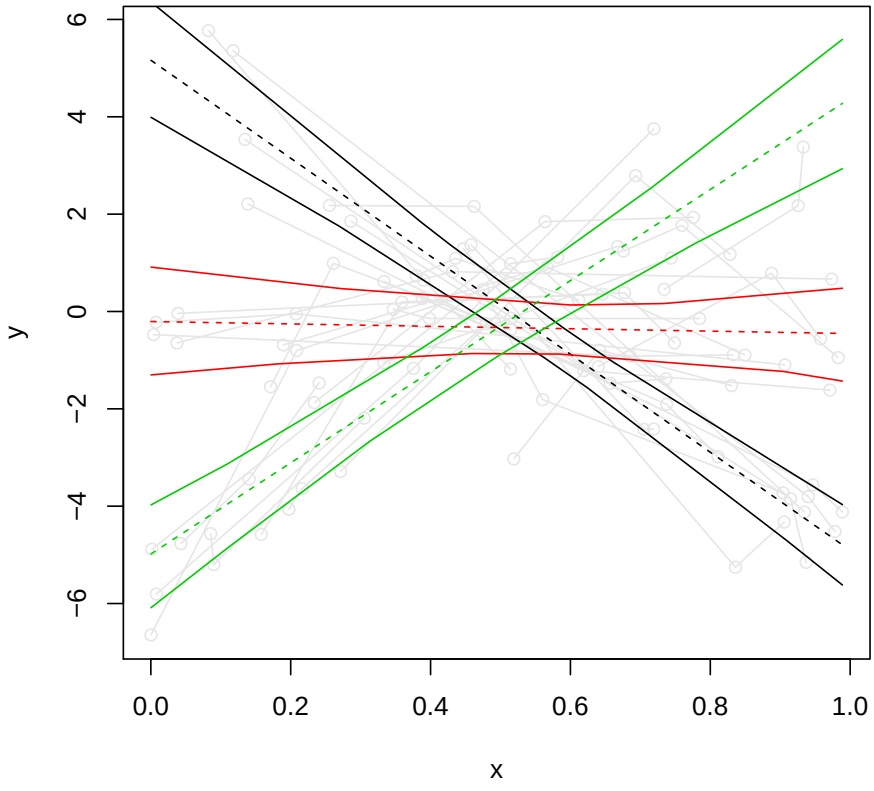

Figure 1: Scatterplot of three simulated linear subgroups. Mean estimates (dashed) and 95% credible intervals (solid) are presented, conditional on the data partition estimated using the `profLinear` function.

3.3. Partition Comparisons

Partition estimate comparisons are made using one of many statistics designed for this purpose, including the Rand, Fowlkes and Mallows, Jaccard, and Wallace indices (**???**). Each of these indices assume that a partition is represented by a length n vector of one or more distinct values $\boldsymbol{z} = z_1, z_2, \ldots, z_n$, where the z_i represents the cluster assignment of observation i. Hence, $z_i = z_j$ indicates that observations i and j are assigned to the same cluster. The number of clusters is determined by the number of unique values in the vector $\boldsymbol{z}$.

Consider two partitions represented by the cluster assignments $\boldsymbol{z}_a$ and $\boldsymbol{z}_b$. Let n_{11} be the number of unique i, j pairs such that $z_{ai} = z_{aj}$ *and* $z_{bi} = z_{bj}$. In words, n_{11} is the count of observation pairs where the pair are clustered together in both partitions. Let n_{00} be the count of observation pairs where the pair are assigned different clusters in both partitions. Hence, n_{11} and n_{00} are the counts of observation pairs where partitions $\boldsymbol{z}_a$ and $\boldsymbol{z}_b$ are in agreement with regard to cluster assignment. Further define n_{10} as the count of observation pairs where the pair are assigned the same cluster in partition $\boldsymbol{z}_a$ but different clusters in partition $\boldsymbol{z}_b$, and let n_{01} be the converse. Hence, n_{01} and n_{10} are the counts of observation pairs where partitions $\boldsymbol{z}_a$ and $\boldsymbol{z}_b$ are discordant. Also observe the equivalence $n(n-1)/2 = n_{11} + n_{00} + n_{10} + n_{01}$, where $n(n-1)/2$ is the number of distinct pairs of n items. Each of the indices mentioned above are a function of these counts.

The Rand index is the most intuitive. It is the proportion of concordant observation pairs:

$$\frac{n_{11} + n_{00}}{n_{11} + n_{00} + n_{01} + n_{10}}.$$

The Fowlkes and Mallows index is given by

$$\frac{n_{11}}{\sqrt{(n_{11} + n_{01})(n_{11} + n_{10})}}.$$

The Wallace indices are respectively given by

$$\frac{n_{11}}{n_{11} + n_{10}} \quad \text{and} \quad \frac{n_{11}}{n_{11} + n_{01}}.$$

The Jaccard index is given by

$$\frac{n_{11}}{n_{11} + n_{01} + n_{10}}.$$

Each of these indices take values in the closed interval $[0, 1]$, where larger values indicate greater agreement. The cited works, and the associated discussions consider the relative merits of each index.

The `pci` function computes each of the partition comparison indices. The returned value is a named vector, where the names indicate the index computed. The Rand, Fowlkes and Mallows, Wallace, and Jaccard indices are abbreviated by `R`, `FM`, `W10`, `W01`, and `J` respectively. The following demonstrates the `pci` function using the estimated and simulated partitions of the previous example.

```
R> simulatedPartL <- rep(1:3, rep(33,3))
R> estimatedPartL <- fitL$clust
R> pci(simulatedPartL, estimatedPartL)
```

```
        R        FM       W10       W01         J
0.9257885 0.8863636 0.8863636 0.8863636 0.7959184
```

The estimated partition has good agreement with the simulated partition in the example above, partly due to the large differences in slopes of the simulated linear mean functions relative to the error variance. The grouping structure among longitudinal observations reduces the partition space complexity, further promoting recovery of the simulated partition.

3.4. Sensitivity to Cluster Homogeneity

The next example assesses the sensitivity of simulated partition recovery to reduced heterogeneity in the simulated mean functions. The three sets of coefficients in the last example were $(5, -10)$, $(0, 0)$, and $(-5, 10)$. Hence, heterogeneity is reduced by multiplying each coefficient by a proportion. The following R code evaluates the Rand index between the estimated and simulated partition in a series of simulated datasets with reduced coefficient heterogeneity.

```
R> mult <- rep(seq(1, 0.01, length.out=33), 10)
R> Rand <- sapply(mult, function(mult) {
R+     dat <- sim(mult)
R+     fit <- profLinear(y ~ x, group=gr, data=dat)
R+     pci(rep(1:3, rep(33,3)), fit$clust)['R']
R+ })
R> lws <- lowess(x=mult,y=Rand)
R> plot(lws$x, lws$y, type='n',
R+      xlab='coefficient multiplier', ylab='Rand index')
R> points(mult + rnorm(length(mult),0,1/200),
R+      Rand + rnorm(length(Rand),0,1/200), col=grey(0.9))
R> lines(lws$x, lws$y)
```

Figure 2 illustrates the effect of reduced coefficient heterogeneity on the Rand index between estimated and simulated partitions. Clearly, it is increasingly difficult to recover the simulated partition when the mean functions are more alike. Limited or absent grouping information may also hinder recovery of the simulated partition.

4. Illustration: `profBinary`

4.1. Default Parameters

The PPM of multivariate binary models specifies two prior shape parameters, a_0 and b_0. For each cluster $k = 1, \ldots, r$ and binary outcome $j = 1, \ldots, q$, the associated posterior probability ϕ_{jk} has a beta distribution with shape parameters a_{kj} and b_{kj}, given by

$$\begin{aligned} a_{kj} &= a_0 + n_{kj} \\ b_{kj} &= b_0 + n_k - n_{kj}, \end{aligned} \tag{5}$$

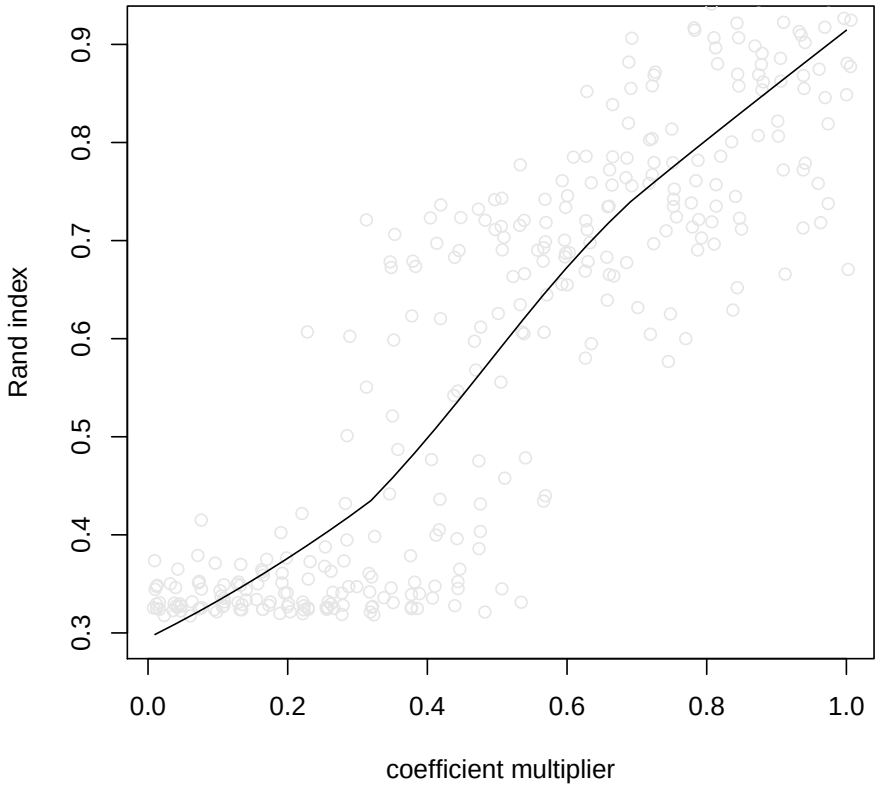

Figure 2: Scatterplot of Rand indices (between estimated and simulated partitions) versus slope multiplier. Smaller coefficient multipliers reduce the heterogeneity among the simulated clusters. Light gray points represent simulated datasets. Jitter was added to aid in visual discrimination. The black line is a LOWESS estimate of the trend in Rand index versus coefficient multiplier. For α less than 0.3, the partition estimates mostly consisted of a single cluster.

where n_k is the number of observations assigned to the k^{th} cluster, and n_{kj} is the number of observations for which $z_i = k$ and $y_{ij} = 1$. That is, n_{kj} is the count of observations in the k^{th} cluster that exhibit outcome j. Relative to the data likelihood, the prior influence on the posterior distribution may be reduced by setting a_0 and b_0 near to zero. Indeed, when a_0 and b_0 equal zero, the posterior expected probability of outcome j in cluster k is equal to the corresponding sample proportion. By default, the `profBinary` function takes a_0 and b_0 to be 1.00, which encodes a uniform prior distribution over ϕ_{jk}.

4.2. `profBinary` Example

The following R code block simulates a multiple binary outcome dataset with two latent clusters. Each row in the dataset corresponds to a single multivariate binary observation.

The `profBinary` function automatically clusters observations in a single row. Hence, a `group` argument is unnecessary, and not implemented. The outcomes used for partitioning are specified using a one-sided formula.

```
R> p <- seq(0.9,0.1,length.out=9)
R> y1 <- matrix(rbinom(999, 1, p), 111, 9, TRUE)
R> y2 <- matrix(rbinom(999, 1, rev(p)), 111, 9, TRUE)
R> dat <- as.data.frame(rbind(y1, y2))
R> fitb <- profBinary(~0+., data=dat)
```

Conditional on the estimated partition, the outcome probabilities for each cluster are independently beta distributed *a posteriori*. The parameters of the profile posterior distribution for each cluster may be accessed in the return value of `profBinary`. In a fashion similar to `summary.profLinear`, the `summary.profBinary` method computes and prints the profile posterior means and 95% credible limits of the outcome probabilities for each cluster.

4.3. Sensitivity to DPM Precision α

The value of the DPM precision parameter α has a significant effect on simulated partition recovery in the last example. To illustrate, consider repeating the analysis under a sequence of values for α. Figure 3 presents the Rand index as a function of α.

```
R> alpha <- 10^(-seq(0, 40, length.out=33))
R> Rand <- sapply(alpha, function(alpha) {
R+     fit <- profBinary(~0+., data=dat, param=list(alpha=alpha))
R+     pci(rep(1:2, rep(111,2)), fit$clust)['R']
R+ })
R> lws <- lowess(x=log(alpha),y=Rand)
R> plot(lws$x, lws$y, type='l',
R+      ylim=c(0.45,1),
R+      xlab=expression(paste('log', alpha)),
R+      ylab='Rand index')
R> points(log(alpha), Rand + rnorm(length(Rand),0,1/200),
R+      col=grey(0.9))
```

In these data, very small α is required to most closely recover the simulated partition. However, in practice, this optimal precision parameter is unknown. To overcome this, we recommend that α be fixed in some principled manner (*e.g.*, the strategy of **?**). The default value for α (1/1000) concentrates prior mass on partitions with few clusters. Even so, the corresponding partition estimate consisted of fourteen clusters in these data, twelve more than the simulated partition.

? warn that 'sensitivity to α is marked', that the Dirichlet process precision is 'a critical smoothing parameter', and that the observed data might be used to draw inferences about α. The authors treat the precision as an unknown parameter with heavily right skewed gamma prior density. Through inspection of the posterior density, they conclude that the data are informative about α. Posterior inference about α has been similarly adopted in much of the subsequent Dirichlet process literature (**????**).

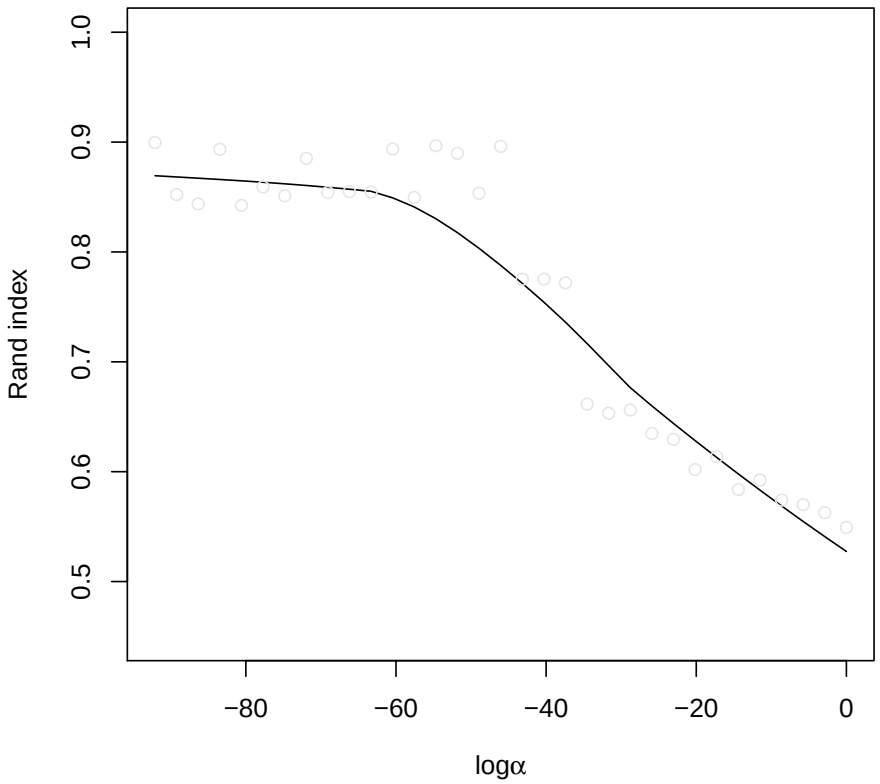

Figure 3: Scatterplot of Rand indices (estimated versus simulated partition) against DPM precision parameter α. Smaller α imposes a stronger penalty against partitions with many clusters. Light gray points represent estimated partitions. Jitter was added to aid in visual discrimination. The black line is a LOWESS estimate of the trend in Rand index versus α.

Since the DP precision is an important smoothing parameter, an alternative rationale is that allowing the data to inform our belief about α may result in too little smoothing. Hence, a fixed α is recommended. Arguments and experiments regarding fixed precision are also prevalent in the DP literature (**???**).

5. Extensions

The **profdpm** package is currently limited to product partitions of linear and binary models, and the optimization strategies described above. Additional models may be implemented as need arises. Also, in the current framework, each cluster of the partitioned data is required to have an identical parametric model, albeit with different parameters. This contrasts, for example, with the finite mixture clustering methods implemented in the **flexmix** package (**?**), which allow multiple parametric forms. This type of functionality is possible in the **profdpm**

framework by specifying a sufficiently complex `model` structure, and corresponding `move` and `logp` methods (see Appendix).

Estimating a partition is computationally intensive, and an open field of computational research. Other aspects of partition estimation are also under scrutiny. For example, work by **?** suggest that marginal likelihood optimization may overfit the observed data, yielding more clusters than necessary. The authors recommend a pseudo marginal likelihood (PML) approach, which imposes a leave-one-out cross-validation strategy to penalize extraneous clusters in partition estimates. The **profdpm** framework has potential to accommodate alternative likelihoods by implementing alternative `logp` methods.

Finally, there are plans to extend the **profdpm** framework to accommodate novel PPMs by modifying the cohesion function in the prior distribution over data partitions (see Section 2). Note that such modifications result in PPMs other than the augmented Dirichlet process mixture. **?** discuss two such alternatives, including the partition model arising from the Poisson-Dirichlet process; a two-parameter generalization of the Dirichlet process.

Acknowledgments

This research was partly funded by the following National Institutes of Health, National Institute of General Medical Sciences, and National Science Foundation funding projects: NIH 1T32GM074934, NIH R03CA137805, NSF DMS0604666, NIH P20RR017696.

Appendix: Programming strategy

The **profdpm** package is primarily implemented in C (following the C99 standard) to facilitate efficient and flexible usage of memory, enable direct interface with the BLAS and LAPACK matrix libraries, and to ease translation to environments other than R.

Product partition models are implemented as a state machine. The model state is represented using the C `struct pdpm_t`, declared in `src/profdpm.h` as follows (edited for this document):

```
typedef struct pdpm_t {
unsigned char   flags;
double          alp;     //DP precision parameter
unsigned int    ngr;     //number of groups
unsigned int    ncl;     //number of clusters
unsigned int  * vcl;     //cluster membership array
unsigned int  * gcl;     //number of groups per cluster array
double          logpval; //unnormalized log posterior value

void         (*move)( struct pdpm_t * obj,\
                 unsigned int grp, unsigned int cls );
double       (*logp)( struct pdpm_t * obj );
double       (*logponly)( struct pdpm_t * obj,\
                 unsigned int * only, unsigned int size );

void          * model;   //model specific/private data
```

```
} pdpm_t;
```

The `flags` member of `struct pdpm_t` is a bitmask for flags, for example, to indicate that convergence criteria for an optimization strategy are met. The `alp` member is the Dirichlet process precision parameter α. In the **profdpm** framework, multivariate observations, or univariate observations that should always be clustered together are called 'groups'. When no complex grouping structure is present, each observation forms a singleton group. Clusters consist of one or more groups. The `ngr` and `ncl` members of `struct pdpm_t` store the number of groups and clusters that comprise the PPM state, respectively. Each group is assigned exactly one cluster, identified by an integer stored in the `vcl` array. Hence, `vcl[4]` indexes the cluster number for group four. For groups that are not yet assigned to a cluster (*i.e.*, during initialization), the corresponding `vcl` entry may take the value defined by the C preprocessor macro `BAD_CLS`. The `gcl` array stores the number of groups assigned to each cluster. For example, `gcl[3]` indexes the number of groups assigned to cluster number three. Because each group might form a singleton cluster, `vcl` and `gcl` are both allocated to have length `ngr`. The `logpval` member stores the unnormalized log posterior mass value at the current state.

The **profdpm** package implements four methods for partition estimation (see Section 2). Each utilizes a Markov-like process. That is, a model state is initialized and updated sequentially, where the updated state depends on the previous state. Each of the four methods decompose into a series of just two simple operations on the model state: (1) reassigning a single observation from one cluster to another, and (2) computing the marginal posterior mass at the current partition estimate.

For example, consider the SUGS optimization strategy. At iteration t, group $t+1$ is either assigned to an existing cluster, or assigned to a new cluster. In order to evaluate the alternatives, group $t+1$ is reassigned to each potential cluster in sequence. The marginal posterior mass is evaluated after each reassignment. Finally, group $t+1$ is reassigned to the cluster corresponding to the largest posterior mass. Hence, each iteration of the SUGS algorithm decomposes into a series of reassignments and posterior mass calculations. No other operations on the model state are necessary.

Manipulating the model state with simple operations is strategic. By implementing the simple operations efficiently, estimation algorithms may be written generically, and with emphasis on simplicity. Generic application of the estimation routines requires implementing the two simple operations for each specific PPM type (*e.g.*, two for product partitions of linear models, and two for multivariate binary models).

The `move` and `logp` members of `struct pdpm_t` are pointers to C functions that implement the reassignment and posterior mass calculation routines. The `move` member should point to a function that takes three arguments; a pointer to the PPM state structure (`obj`), a group number (`grp`), and a cluster number (`cls`). That is, in model state `obj`, the group with number `grp` is reassigned to the cluster with number `cls`. The function pointed-to by `move` should not return a value, but should update the `ncl`, `vcl`, and `gcl` members to reflect the reassignment operation. The `logp` member should point to a function that takes the PPM state structure as an argument, and returns the unnormalized log posterior mass, evaluated at the current model state. The `logp` method may or may not update the `logpval` member automatically.

The marginal posterior mass function in conjugate PPMs is a product of cluster specific terms. Hence, moving a single group from one cluster to another affects two terms at most. The

posterior mass calculation may be optimized by computing only the terms associated with clusters modified since the previous calculation. This optimization is implemented using an optional third C function pointer; `logponly`. The function pointed-to by `logponly` should accept three arguments; a pointer to the PPM state structure (`obj`), an array of cluster numbers (`only`), the array's length (`size`), and return the partial log posterior mass associated with the clusters referenced in the `only` array. In particular, the returned value need only satisfy that the algebraic difference in the return values of two identical calls to `obj->logponly` represents the change in log posterior mass associated with model state changes in the period between calls. For example, the following C code updates the log posterior mass associated with a reassignment operation, where `obj` is a pointer to an initialized structure of type `struct pdpm_t`:

```
...
cls_from_to[0] = obj->vcl[grp];
cls_from_to[1] = cls_to;
obj->logpval -= obj->logponly(obj, cls_from_to, 2);
obj->move(obj, grp, cls_to);
obj->logpval += obj->logponly(obj, cls_from_to, 2);
...
```

For model states with many clusters, updating the log posterior mass in this way is often more efficient than recomputing all cluster-specific terms of the log posterior mass function. Initializing the `logponly` pointer is optional. When `logponly` is left uninitialized (`NULL`), estimation routines utilize the `logp` pointer instead.

Finally, the `model` pointer may be used to store data related to a specific type of PPM. For example, in a PPM of linear models, `model` points to a structure that holds the response vector, design matrix, and additional parameter values. The functions pointed-to by `move` and `logp` may access the structure pointed-to by `model`. As a rule, routines that operate generically on the model state, including the four optimization methods should not dereference the `model` pointer.

The structure pointed-to by `model` may be simple or complex, depending on the resources of the target platform and the available programming effort. A simple model state might store only the original data arrays and other required parameters. When additional memory is available, intermediate values might be cached for more efficient computations. For example, linear models with covariate vector $\boldsymbol{x}_i$ often involve the outer product $\boldsymbol{x}_i\boldsymbol{x}_i'$. A computationally efficient method would compute and store this quantity once, rather than recompute the quantity on demand. Furthermore, since $\boldsymbol{x}_i\boldsymbol{x}_i'$ is symmetric, only the upper or lower triangular portion requires storage. The PPM of linear models implemented in the **profdpm** package utilizes both optimizations.

The **profdpm** package leaves the programmer to balance the antagonistic relationship between memory usage and computational efficiency by offering a uniform API to partition estimation methods. The C function with prototype `pdpm_t * pdpm_init(unsigned int ngr)` is a convenience function that allocates memory (using `R_alloc`), partially initializes, and returns a pointer to an instance of `struct pdpm_t`. The argument `ngr` specifies the number of groups that comprise the data. However, the structure returned by `pdpm_init` is not fully initialized. The `flags`, `alp`, `model` and function pointer members (except `logponly`) of

`struct pdpm_t` must be initialized explicitly. Once fully initialized, a pointer to the model state structure is passed to an optimization routine. For example, the function with prototype `void method_fast(pdpm_t *)` implements the modified SUGS optimization method. When `method_fast` returns, the structure pointed-to by its argument will have been modified (optimized) according to the SUGS algorithm. An interface between R and new PPMs implemented using the **profdpm** C API must also be written.

Affiliation:

Matthew S. Shotwell
Assistant Professor
Department of Biostatistics
Vanderbilt University
1161 21st Ave. South, Nashville, TN
E-mail: matt.shotwell@Vanderbilt.edu
URL: http://biostat.mc.vanderbilt.edu/wiki/Main/MattShotwell