ClusterZI - Vignette

Introduction

This vignette demonstrates how to implement semi-parametric clustering methods on microbiome data, which is assumed to follow the Zero-Inflated Dirichlet-Multinomial distribution as explained in the main manuscript, [“Main Manuscript”]. It begins with brief instructions on how to install the packages for implementing the proposed model. Next, the document provides guidance on simulating the data used in the main manuscript. Finally, it covers the implementation of the model on both simulated and application data.

Installation

To install the package, we run the command below:

```r
devtools::install_github("skorsu/ClusterZI")
```

Simulation Data

In this section, we demonstrate how to obtain the simulated data used in the main manuscript. After installing and loading the `ClusterZI` package into the R environment, we use the `sim_clusDat()` function to generate two-cluster Zero-Inflated Dirichlet-Multinomial distributed data.

```r
### Simulate the data
simDat <- sim_clusDat(n = 100, Jnoise = 150, Jsignal = 50, pZero = 0.35,
                      ZSumNoise = 12500, ZSumSignal = 2500, seed = 1)
```

To use the `sim_clusDat()` function, we need to specify several parameters: the number of observations (n), the number of noise taxa (taxa that do not contain information about the clustering) (J_{noise}), the number of signal taxa (taxa that differentiate between clusters) (J_{signal}), the proportion of zeros expected in the simulated dataset (pZero), the sequencing depth for both noise and signal taxa (ZSumNoise and ZSumSignal, respectively), and the random seed (seed). Note that the `sim_clusDat()` function splits the observations into two clusters evenly. The result from the `sim_clusDat()` function is a list consisting of two elements: the simulated OTU table (dat) and the cluster allocation for each observation (c). For example, in this case, we have simulated 100 observations with 200 taxa, 50 of which are considered signal taxa. In this simulated dataset, we expect that around 35% of the counts are zero. The assumed sequencing depth for each observation is 15,000, with 12,500 for the noise taxa and 2,500 for the signal taxa. We save the simulated data in the object named `simDat`. To access the simulated OTU table, we use `simDat$dat`, while `simDat$c` gives us the cluster allocation.

Note that there are six other default arguments, which are listed below:

- shuffle: Determines whether the order of observations should be shuffled. The default is TRUE. If set to FALSE, observations from the same cluster will be grouped together.

- caseSignal: This is the complexity index, ranging from 1 to 5, where 1 is the most complex and 5 is the least complex. The default is 3.
- aPhi, bPhi, aLambda, bLambda: These parameters control the characteristics of each cluster.

The limitation of the `sim_clusDat()` function is that it is suitable for simulating data from only two clusters. However, we can manually extend this function to generate more than two clusters. For example, if we want to generate 200 observations with four different clusters, where the cluster sizes are 60, 60, 40, and 40, respectively, we can use the code below.

```r
### Extend the code for simulating 4 clusters.
#### Generate the first 120 observations, 60 from each cluster,
simDat_1 <- sim_clusDat(n = 120, Jnoise = 150, Jsignal = 50, pZero = 0.35,
                        ZSumNoise = 12500, ZSumSignal = 2500, seed = 1, shuffle = FALSE)
#### Generate the other 80 observations, 40 from each cluster,
simDat_2 <- sim_clusDat(n = 80, Jnoise = 150, Jsignal = 50, pZero = 0.35,
                        ZSumNoise = 12500, ZSumSignal = 2500, seed = 1, shuffle = FALSE)

#### Rearrange the order of the taxa to differentiate among these 4 clusters
simDat_4clus <- rbind(simDat_1$dat,
                      cbind(simDat_2$dat[1:40, 151:200], simDat_2$dat[1:40, -(151:200)]),
                      cbind(simDat_2$dat[41:80, 1:50], simDat_2$dat[1:40, 151:200],
                            simDat_2$dat[1:40, 51:150]))

#### Optional: Shuffle the order of the observations
set.seed(1)
index <- sample(1:200)
simDat_4clus <- simDat_4clus[index, ]
simDat_4clus_c <- c(simDat_1$c, simDat_2$c + 2)[index]
```

Implementing the Model

In this section, we walk through how to implement the discrete sparse finite mixture model on the simulated data, along with the possible posterior inference on the parameters of interest. The primary function used for implementing the discrete sparse finite zero-inflated Dirichlet-Multinomial model is `ZIDM_dSDMMM()`.

```r
resultMod <- ZIDM_dSDMMM(dat = simDat$dat, iter = 1500, Kmax = 10, nxi_split = 10,
                         theta = 1, s2 = 0.1, s2MH = 1e-3, MHadapt = 500,
                         thin = 1, seed = 1)
```

This function requires us to specify the data to which we will apply the model (dat), the number of MCMC iterations (iter), the number of components (Kmax), the sparsity concentration parameter (theta), the variance of the cluster concentration parameter (s2), the variance for the adaptive Metropolis-Hasting when updating the cluster concentration parameter (s2MH), the number of MCMC iterations before using the adaptive proposal (MHadapt), thinning (thin), and the random seed (seed). Note that the `ZIDM_dSDMMM()` function initializes all observations in the same cluster. In this example, we specify that we will apply the model to the simulated data we previously generated (`simDat$dat`). We let the model run for 15,000 iterations, where the first 500 iterations use a non-adaptive proposal for the adaptive Metropolis-Hasting (AMH). We set the variance of the cluster concentration parameter to 0.1, and the variance of the adaptive Metropolis-Hasting to 1×10^{-3}. We limit the clusters to no more than 10 clusters. Besides, if we propose to split the cluster space, the proposed cluster concentration parameters are obtained by using the cluster concentration parameters corresponding to the original cluster, while 10 of the random taxa have different cluster concentration parameters.

The result from the `ZIDM_dSDMMM()` function is a list object. To obtain the final cluster assignment, we use the `finalCLUS()` function. We need to specify the number of iterations to consider as a burn-in period. We utilize the `salso()` function from the `salso` package with the VI loss function (Dahl, Johnson, and Müller 2022) to determine the final cluster assignment.

```
### Obtain a vector of the final cluster assignment.
clusResult <- finalCLUS(resultMod, burn_in = 500, seed = 1)
```

We can obtain the Adjusted Rand Index (ARI) to measure the performance of the resulting cluster compared to the actual cluster assignment (if applicable) by using the `ariCLUS()` function.

```
### Calculate the ARI
ariCLUS(clusResult, simDat$c)
#> 1
```

Posterior Inference

We then proceed with the posterior inference on the parameters of interested, which is the number of active cluster via MCMC iterations. We use `uniqueCLUS()` to obtain the number of active cluster in each MCMC chain. We demonstrate the number of active cluster by using the line plot, as shown in Figure 2.

```
### Obtain a vector of the number of active cluster for each MCMC iteration.
clusMCMC <- uniqueCLUS(resultMod)
```

Application Data

In this section, we discuss how to apply the proposed model to the application data, mentioned in the main Manuscript, which are the HIV dataset, collected by Noguera-Julian et al. (2016) and can be accessed from the `selbal` package (Rivera-Pinto et al. 2018), and EDD dataset, provided by Singh et al. (2015) and make it publicly available by Duvallet et al. (2017). In this vignette, we use the HIV dataset for demonstrating how to apply the proposed model on the real-world dataset. We first load the data into the R environment. The first column is the observation ID, where the second and third column are the metadata corresponding to each observation. Note that for the EDD dataset, only the second column is the metadata.

```
### Import the dataset
data("selbalHIV")
otuHIV <- selbalHIV[, -(1:3)] ### Obtain the OTU table
metaHIV <- selbalHIV[, 1:3] ### Obtain the metadata
```

We then apply the proposed model to the HIV dataset, running it for 25,000 MCMC iterations without thinning. We decide to use the adaptive proposal for the AMH after the first 2,500 iterations. The variance of the cluster concentration parameter is set to 1, and the variance of the adaptive Metropolis-Hastings is set to 1×10^{-3}. We limit the model to a maximum of 20 clusters. Additionally, if a split in the cluster space is proposed, the new cluster concentration parameters are derived from those of the original cluster, with 5 randomly selected taxa having different concentration parameters.

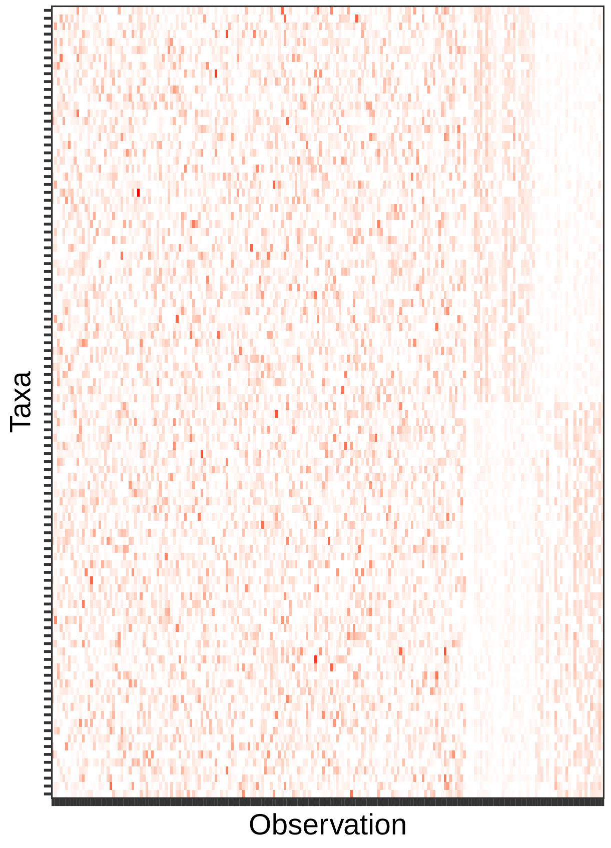
```
### Apply the model on the HIV dataset.
HIVMod <- ZIDM_dSDMMM(dat = otuHIV, iter = 25000, Kmax = 20, nxi_split = 5,
                      theta = 1, s2 = 1, s2MH = 1e-3, MHadapt = 2500,
                      thin = 1, seed = 1)
```

We obtain the final cluster assignment by using `finalCLUS()` function. We then see the ability of the model to recover the cluster pattern in the data by considering the resulting cluster with the metadata, as shown in Table 2. The result reveals that similar finding as discussed in Noguera-Julian et al. (2016).

```
### Obtain the final cluster assignment for the HIV dataset.
HIVclus <- finalCLUS(HIVMod, burn_in = 5000, seed = 1)
```

We also consider the number of active clusters across MCMC iterations by applying the `uniqueCLUS()` function. The result, illustrated in Figure 3, shows that the number of active clusters stabilizes after the first few iterations.

```
### Calculate the number of the active cluster for each MCMC iteration.
HIVMCMC <- uniqueCLUS(HIVMod)
```



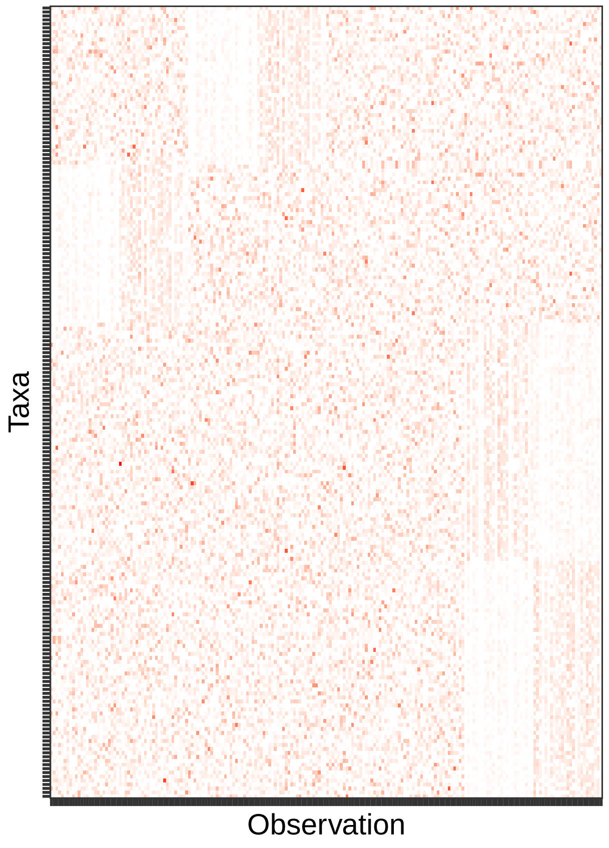


Figure 1: Heatmaps of the simulated data. Left: Data generated by the `sim_clusDat()` function, saved as `simDat`. Right: Data generated by an extended version of the `sim_clusDat()` function to create 4 clusters, saved as `simDat_4clus`.

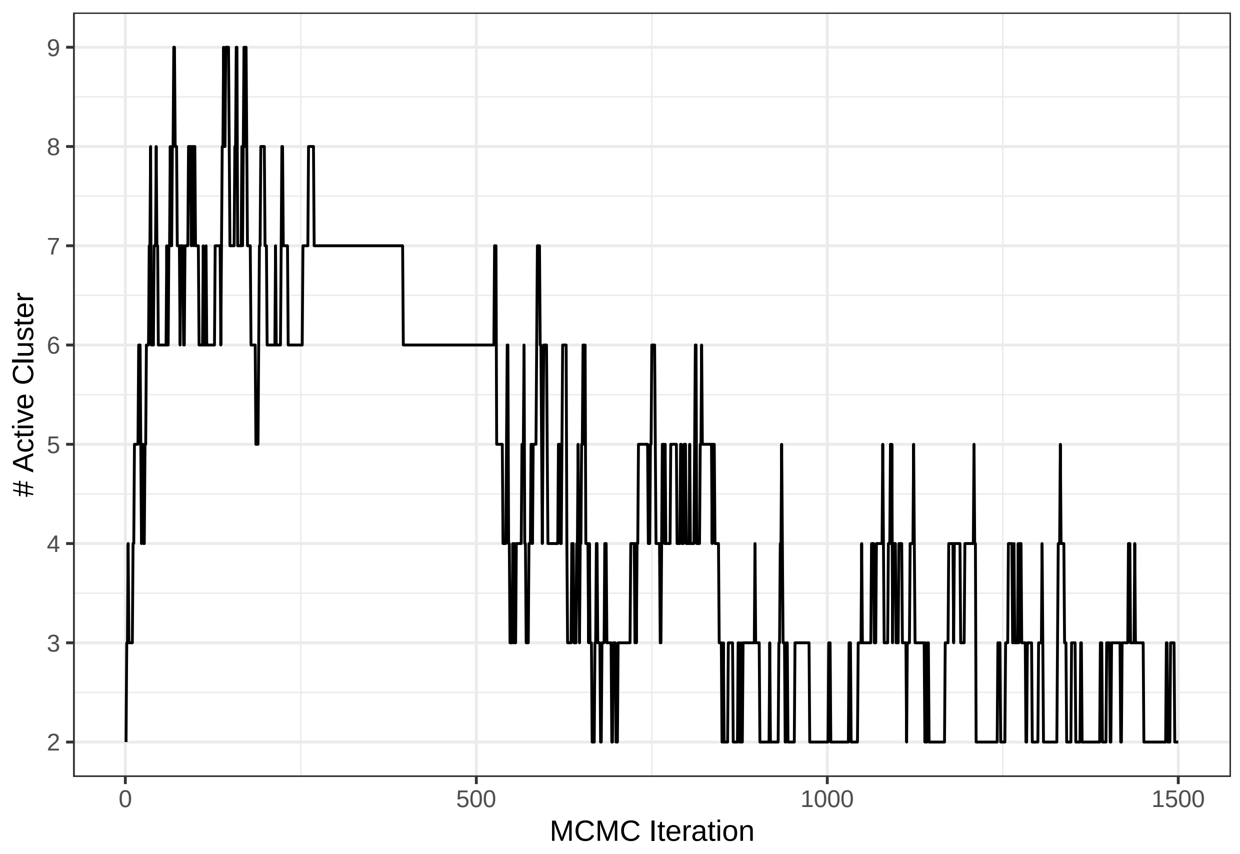


Figure 2: The line plot shows the number of active clusters for each MCMC iteration when the model is applied to the simulated data (`simDat`).

Table 1: Example of the metadata for the HIV dataset. The first column contains the observation ID, while the other two columns represent the metadata for each individual: sexual preference and HIV status, respectively.

ID	Sexual Preference	HIV Status
Sample_001	nonMSM	Pos
Sample_002	nonMSM	Pos
Sample_003	MSM	Pos
Sample_004	MSM	Neg
Sample_005	MSM	Neg
Sample_006	MSM	Neg

Table 2: Distribution of patients and sexual preferences within the resulting clusters.

HIVclus	Healthy: non-MSM	HIV: non-MSM	Healthy: MSM	HIV: MSM
1	4	51	1	13
2	0	2	8	19
3	0	2	14	41

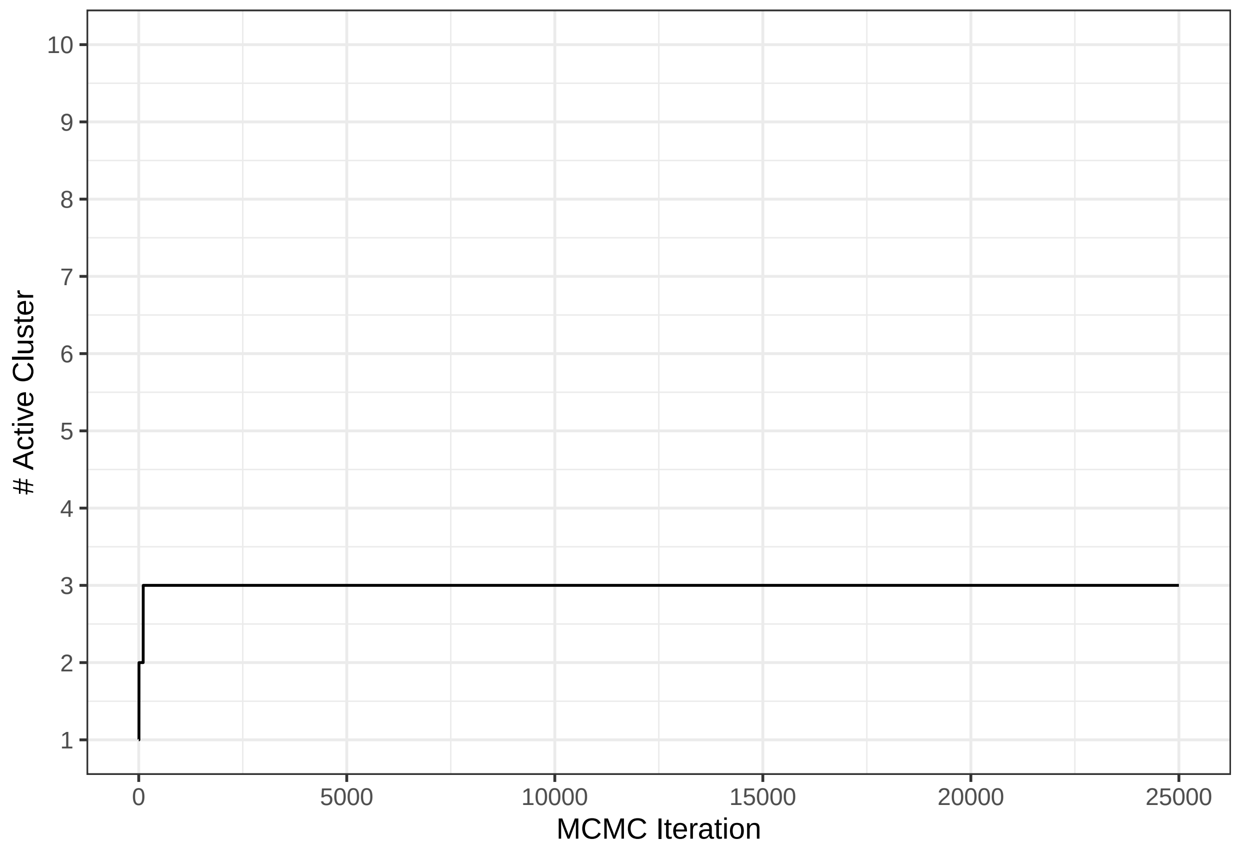


Figure 3: The line plot shows the number of active clusters for each MCMC iteration when the model is applied to the application data (`selbalHIV`).

Reference

Dahl, David B, Devin J Johnson, and Peter Müller. 2022. "Search Algorithms and Loss Functions for Bayesian Clustering." *Journal of Computational and Graphical Statistics* 31 (4): 1189–1201.

Duvallet, Claire, Sean M Gibbons, Thomas Gurry, Rafael A Irizarry, and Eric J Alm. 2017. "Meta-Analysis of Gut Microbiome Studies Identifies Disease-Specific and Shared Responses." *Nature Communications* 8 (1): 1784.

Noguera-Julian, Marc, Muntsa Rocafort, Yolanda Guillén, Javier Rivera, Maria Casadellà, Piotr Nowak, Falk Hildebrand, et al. 2016. "Gut Microbiota Linked to Sexual Preference and HIV Infection." *EBioMedicine* 5: 135–46.

Rivera-Pinto, Javier, Juan Jose Egozcue, Vera Pawlowsky-Glahn, Raul Paredes, Marc Noguera-Julian, and M Luz Calle. 2018. "Balances: A New Perspective for Microbiome Analysis." *MSystems* 3 (4): 10–1128.

Singh, Pallavi, Tracy K Teal, Terence L Marsh, James M Tiedje, Rebekah Mosci, Katherine Jernigan, Angela Zell, et al. 2015. "Intestinal Microbial Communities Associated with Acute Enteric Infections and Disease Recovery." *Microbiome* 3: 1–12.