The BClustLonG Package: A Dirichlet Process Mixture Model for Clustering Longitudinal Gene Expression Data

Jiehuan Sun[1], *Jose D. Herazo-Maya*[2], *Naftali Kaminski*[2], *Hongyu Zhao*[1], *and Joshua L. Warren*[1]
[1]*Department of Biostatistics, Yale School of Public Health*
[2]*Pulmonary, Critical Care and Sleep Medicine, Yale School of Medicine*

Overview

Many clustering methods have been proposed, but most of them cannot work for longitudinal gene expression data. Our newly developed method, BClustLonG, can be used to perform clustering analysis for longitudinal gene expression data. It adopts a linear-mixed effects framework to model the trajectory of genes over time, while clustering is jointly conducted based on the regression coefficients obtained from all genes. To account for the correlations among genes and alleviate the high dimensionality challenges, factor analysis models are adopted for the regression coefficients. The Dirichlet process prior distribution is utilized for the means of the regression coefficients to induce clustering (See Sun et al. (2017) for details).

This document provides a tutorial for using the **BClustLonG** package. The tutorial includes information on (1) the format of the input data and (2) how to obtain clustering results and visually show the clustering structure. As with any R package, detailed information on functions, along with their arguments and values, can be obtained in the help files.

Input data format

The analyses performed in this tutorial are based on a simulated dataset, which comes with the package. Basically, the data are generated from a mixture of two multivariate normal distributions, for which the covariance matrix satisfies the factor analysis model assumption (See simulation studies in Sun et al. (2017) for details).

The input data for **BClustLonG** has to be a list with three elements: Y (gene expression data), ID, and years (The names of the elements have to be matached exactly). Each column of Y represents one gene. The j_{th} row of "Y" represents the gene expression value for subject *ID[j]* at time *years[j]*. So, the length of the "ID", the length of the "years", and the number of rows of "Y" are the same. No missing values are allowed in the data and the variable "years" is preferably on the scale of one (e.g. if the visiting time is on the scale of years, then the unit should be in year such as 2 years instead of 730 days or 24 months), which is related to the default values for the hyperparameters.

```r
library(BClustLonG,quietly=TRUE)
data(data)
str(data)
```

```
## List of 3
##  $ Y    : num [1:250, 1:10] 1.835 2.055 2.926 0.698 1.922 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:10] "Gene1" "Gene2" "Gene3" "Gene4" ...
##  $ ID   : int [1:250] 1 1 1 1 1 2 2 2 2 2 ...
##  $ years: num [1:250] 0 0.226 0.428 0.626 0.815 ...
```

```
head(data.frame(ID=data$ID,years=data$years,data$Y),n=10)
```

```
##    ID     years     Gene1      Gene2     Gene3       Gene4    Gene5
## 1   1 0.0000000 1.8353739 -1.2239744 0.5800086  2.66940050 2.186867
## 2   1 0.2257483 2.0553505  2.5708478 0.7857208 -1.22991747 1.871690
## 3   1 0.4280467 2.9260264 -0.8934849 2.4165693 -1.19656751 3.232995
## 4   1 0.6263638 0.6980204  0.9620409 2.5101105  1.15917672 0.184191
## 5   1 0.8153780 1.9221801  0.5076534 2.3531332  0.19541270 1.411288
## 6   2 0.0000000 3.6855317  3.8642961 3.0614498 -0.08049905 1.048487
## 7   2 0.2042057 2.8629913  1.4647943 4.0888133  1.83282771 1.599615
## 8   2 0.3721783 3.6343129  0.8428815 3.0775599  2.80253545 3.888473
## 9   2 0.5972220 2.3188100  3.7502575 4.6750423  2.87213962 2.122598
## 10  2 0.7501218 2.7577214  2.9376960 3.2728889  4.73819621 3.347499
##          Gene6      Gene7     Gene8     Gene9    Gene10
## 1  -0.48769302 -0.9207095 3.1692771 0.8428223 2.5692451
## 2  -0.07973024  0.1657910 3.5902889 2.4527754 3.3578466
## 3   1.51044548  2.2546027 2.5125971 2.3194706 0.9391158
## 4   3.18819663  1.5410392 2.0678755 3.1564176 1.4713458
## 5   0.15587058  0.8633981 2.7929971 1.6936299 2.3668915
## 6   2.29308908  1.1105519 1.3806433 1.9653508 1.8167051
## 7   2.04977841  0.9646012 0.7986834 2.5034942 2.3762562
## 8   1.32009056  1.6111448 3.2174335 3.6447446 2.1229318
## 9   1.25275391  1.9878844 1.5785739 3.6134652 2.5981243
## 10  2.89166407  2.5997904 1.8923252 3.7009653 2.1786946
```

Run BClustLonG

Basic usage of BClustLonG

After the input data (*data*) is prepared in the right format, it is ready to run *BClustLonG* function for clustering. As *BClustLonG* is a Bayesian approach and relies on MCMC for inference, the number of iterations and the number of samples kept for posterior inference need to be specified. For the sake of time, the MCMC is run for 500 iterations (thinning by 2, so 1000 iterations in total) and the first 100 samples are discarded as burn-ins in the code below. In practice, the number of iterations and the number of samples kept for posterior inference should be set to ensure the convergence of the algorithm. Note that *BClustLonG* does not need the pre-specified number of clusters and can infer the number of clusters from the data.

In the code below, *BClustLonG* returns the membership indicators of all subjects from all iterations and *calSim* is used to calculate the posterior similarity matrix among the subjects. Based on the posterior similarity matrix, *maxpear* is used to obtain the clustering results. In this simple simulated dataset, the true cluster structure is recovered.

```
## run BClustLonG and get the clustering results ##
res = BClustLonG(data, iter=500, thin=2,savePara=FALSE, infoVar="both",factor=TRUE)
mat = calSim(t(res$e.mat[,101:500])) ## calculate the posterior similarity matrix
clust = maxpear(mat)$cl ## see maxpear for more details.
```

```
clust
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
## [36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Obtain clustering results with desired number of clusters

Although *BClustLonG* can automatically determine the number of clusters and corresponding cluster structure, it is possible to utilize the outputs of *BClustLonG* to produce clustering structure for a given number of clusters (this feature could be useful in practice, since researchers might have some ideas on the number of clusters that are clinically meaningful). Specifically, provided the posterior similarity matrix, Hierarchical Clustering method (*hclust*) can be used to produce clustering structure for a given number of clusters. The following code shows how to achieve this, where the desired number of clusters is 4.

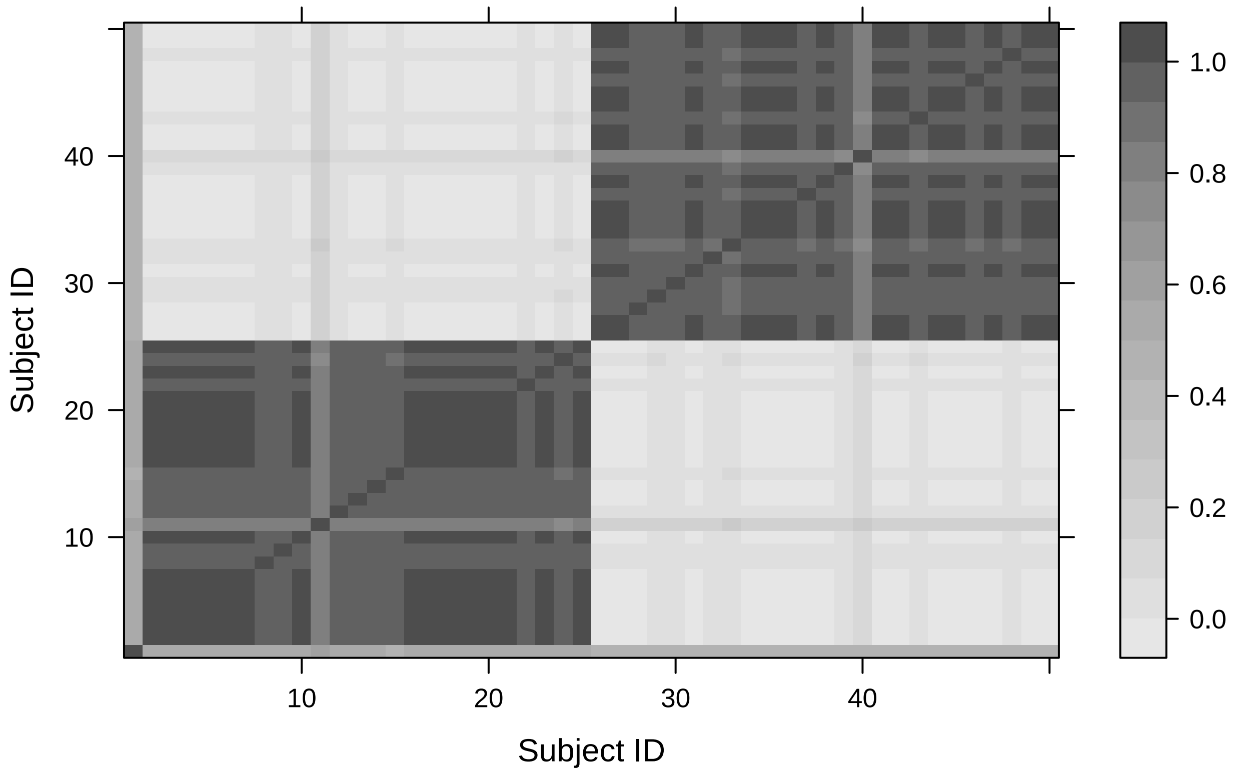
```r
## using Hierarchical Clustering method to obtain the clustering results ##
CL = cutree(hclust(as.dist(1-mat)),k=4)
CL
```

```
##  [1] 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
## [36] 3 3 3 3 4 3 3 3 3 3 3 3 3 3 3
```

Visualization of the similarity matrix

The posterior similarity matrix can also be visually shown so that users can have a rough idea of how many clusters there are. As shown in the figure, it is clear that there are two major clusters.

```r
## plot similarity matrix ##
require(lattice,quietly=TRUE)
n = length(unique(data$ID))
x = rep(1:n,times=n)
y = rep(1:n,each=n)
z = as.vector(mat)
levelplot(z~x*y,col.regions=rev(gray.colors(n^2)), xlab = "Subject ID",ylab = "Subject ID")
```



Other options

BClustLonG adopts a linear-mixed effects with random intercepts and slopes to model the trajectory of genes over time. Hence, each subject has a subject-specific intercept and slope for each gene. It is possible that only the intercepts (i.e. baseline gene expression profiles) are informative of the clustering structure, in which case we may want to cluster only on the intercepts. This can be done by setting the parameter *infoVar="int"*. Sometimes, a diagonal covariance matrix instead of the factor analysis model is preferred for the intercepts and slopes and this can be done by setting the parameter *factor=TRUE* (this can speed up the algorithm and may only be used to get an initial sense of the possible clustering structure in the data).

```r
## Clustering based only on intercepts ##
res = BClustLonG(data, iter=500, thin=2,savePara=FALSE, infoVar="int",factor=TRUE)
## clustering based on intercepts and slopes ##
## assume diagonal covariance matrix for the intecepts and slopes ##
res = BClustLonG(data, iter=500, thin=2,savePara=FALSE, infoVar="both",factor=FALSE)
```

References

Sun, Jiehuan, Jose D Herazo-Maya, Naftali Kaminski, Hongyu Zhao, and Joshua L Warren. 2017. "A Dirichlet Process Mixture Model for Clustering Longitudinal Gene Expression Data." *Statistics in Medicine* 36 (22). Wiley Online Library: 3495–3506.