Citrus

Mengjie Chen

September 17, 2017

Citrus version: 0.99

Please cite:

Contents

1 Introduction of package *Citrus*

Citrus is a R/c++ package that is specialized in the analysis of RNA sequencing data from single cell studies. The current release features `scPLS`, a data denoising method that can control for confounding effects by borrowing information from both control and target genes. scPLS can be used to remove unwanted variation, such as technical effect or cell cycle effect.

2 Installation

Citrus relies on the following R packages: *Rcpp*, *RcppArmadillo*, *genlasso* and *gplots*. All can be directlt installed from CRAN as following:

```
install.packages("Rcpp")
install.packages("RcppArmadillo")
install.packages("genlasso")
install.packages("gplots")
```

Citrus can be installed directly from github.

```
install.packages("devtools")
library(devtools)
install_github("ChenMengjie/Citrus")
```

3 Removing unknown confounding effects using `scPLS`

Like any other genomic sequencing experiment, scRNAseq studies are influenced by many factors that can introduce unwanted variation in the sequencing data and confound the down-stream analysis. Due to low capture efficiency and low amount of input material, such unwanted variation are exacerbated in scRNAseq experiments. As a result, adjusting for confounding factors and normalizing scRNAseq data is crucial for accurate estimation of gene expression levels and successful down-stream analysis. Yet some confounding effects are due to observable batches and can be adjusted for by including batch labels and technician ids as covariates. However, many confounding factors are hidden and are difficult or even impossible to measure. Common hidden confounding factors include various technical artifacts during library preparation and sequencing, and unwanted biological confounders such as cell cycle status. These hidden confounding factors can cause systematic bias, are notoriously difficult control for, and are the focus of the present study.

To mitigate the influence of hidden confounding factors, we develop scPLS, a latent factor based model. The genes are divided into two sets: a control set of genes that are used to infer the confounding factors and a target set of genes that are of primary interest. The confounding factors inferred from the control set are used to remove unwanted variation in the target genes for subsequent downstream analysis. For example, most scRNAseq studies add ERCC spike-in controls during the PCR amplification and sequencing steps. The spike-in controls can be used to capture the hidden confounding technical noise associated with the experimental procedures for normalizing genes of primary interest. Similarly, most scRNAseq studies include a set of control genes that are known to have varying expression levels across cell cycles. These cell cycle genes can be used to capture the unmeasured cell cycle status of each cell, which is further used to normalize target genes. Compared to existing methods, our approach taking into account the information from both control and target genes. In addition, our method can model other systematic biological variation and heterogeneity, which are often observed in the target genes. By incorporating such systematic heterogeneity, we can further improve the estimation of the confounding factors and the removal of unwanted variation. We have shown the effectiveness of our approach in our manuscript. This method is implemented in function `scPLS()`.

3.1 Model and input

We model the control and target gene sets jointly by the following scPLS model

$$\begin{aligned} \mathbf{x_i} &\sim MVN(\mathbf{\Lambda}_x^T \mathbf{z_i}, \boldsymbol{\psi}_x), && (1) \\ \mathbf{y_i} &\sim MVN(\mathbf{\Lambda}_y^T \mathbf{z_i} + \mathbf{\Lambda}_u^T \mathbf{u_i}, \boldsymbol{\psi}_y), && (2) \end{aligned}$$

where for i-th individual cell, $\mathbf{x_i}$ is a q-vector of expression level for q control genes; $\mathbf{y_i}$ is a p-vector of expression level for p target genes; $\mathbf{z_i}$ is a k_z-vector of confounding factors that influence both control and target genes ($k_z < q$); the confounding effects are represented by the q by k_z loading matrix $\mathbf{\Lambda}_x$ for the controls and the p by k_1 loading matrix $\mathbf{\Lambda}_y$ for the target genes; $\mathbf{u_i}$ is a k_u-vector of biological factors unique to target genes ($k_u < p$); the biological factors could represent some underlying pathways or transcription factors that affect multiple genes; $\mathbf{\Lambda}_u$ is a p by k_u loading matrix; $\boldsymbol{\psi}_x$ is the diagonal covariance matrix for q-dimensional idiosyncratic error for x_i; $\boldsymbol{\psi}_y$ is the diagonal covariance matrix for p-dimensional idiosyncratic error for $\mathbf{y_i}$; MVN denotes the multivariate normal distribution.

In the above model, x_i and y_i represent input data, k_1 and k_2 are prespeficied number of latent factors, $\mathbf{\Lambda}_x$, $\mathbf{\Lambda}_y$, $\mathbf{\Lambda}_u$, $\mathbf{z_i}$ and $\mathbf{u_i}$ are parameters to be estimated and will be the output of our algorithm.

3.2 Removing technical effect using spike-ins

We use the cell cycle data published in Buettner et al as an example to illustrate the usage of `scPLS()` function. We selected top 500 non cell cyle genes, 100 cell cycle genes and 20 spike-ins with largest variance across samples.

```
library(Citrus)
data(CellCycleData)
table(CellCycleData$Label)

##
##    CellCycle NonCellCycle       Spikein
##          100          500            20
```

To remove technical effect, `scPLS()` takes inputs of a $n \times q$ matrix of target gene expression, a $n \times p$ matrix of spike-in, a pre-specified number of technical factors k_1 and a pre-specified number of structured biological factors k_2. The numbers of factors k_1 and k_2 can be determined by model selection criteria later.

```
Control <- CellCycleData$Expression[, CellCycleData$Label ==
    "Spikein"]
```

```
Target <- CellCycleData$Expression[, CellCycleData$Label != "Spikein"]
system.time(res <- scPLS(Target, Control, k1 = 2, k2 = 5, iter = 100,
    method = "EM", Chunk = FALSE, center = TRUE))
```

```
##    user  system elapsed
##   17.32    5.58   22.82
```

The default algorithm is the “EM-in-Chunks” algorithm, which will randomly devide genes into different chunks, average the estimates from chunks and thus accelarate the computation. The default chunk size is 1000. We recommend to use the “EM-in-Chunks” algorithm only for large gene sets. When `Chunk = FALSE`, the “Naive EM” algorithm will be used. In addition, `scPLS()` will center the expression of each gene to mean of 0 unless setting `center = FALSE`.

```
summary(res)
```

```
##                 Length Class      Mode
## Factor            3000 -none-     numeric
## Loading            910 -none-     numeric
## Likelihood           1 -none-     numeric
## Z                  364 -none-     numeric
## lambdaY           1200 -none-     numeric
## lambdaX             40 -none-     numeric
## psi_y              600 -none-     numeric
## Method               1 -none-     character
## Adjusted        109200 -none-     numeric
## VarianceSummary      6 data.frame list
```

`scPLS()` outputs the estimates for confounding factor matrix $\mathbf{\Lambda}_y$ (`lambdaY`), confounding loading matrix $\mathbf{Z}$ (`Z`), strutured factor matrix $\mathbf{\Lambda}_u$ (`Factor`), strutured factor loading matrix $\mathbf{U}$ in 2 (`Loading`). The likelihood for 2 is saved as `Likelihood`, which can be used for model comparison later. The expression matrix after adjusting for coufounding effect is stored in `Adjusted`, which is calculated by $Y - \Lambda_y\mathbf{Z}$.

```
head(res$VarianceSummary)
```

```
##                     Sample SampleConfounding SampleStructure
## ENSMUSG00000000127   1.04             0.002           0.088
## ENSMUSG00000000149   1.13             0.017           0.109
## ENSMUSG00000000276   1.19             0.009           0.061
## ENSMUSG00000000308   1.12             0.084           0.074
## ENSMUSG00000000325   1.19             0.004           0.187
## ENSMUSG00000000409   1.14             0.004           0.048
##                     Model ModelConfounding ModelStructure
## ENSMUSG00000000127  1.04            0.002          0.089
## ENSMUSG00000000149  1.12            0.018          0.112
## ENSMUSG00000000276  1.18            0.009          0.068
## ENSMUSG00000000308  1.11            0.090          0.076
## ENSMUSG00000000325  1.18            0.004          0.188
## ENSMUSG00000000409  1.14            0.004          0.050
```

`VarianceSummary` stores total sample variance and sample technical variance as well as total variance and technical variance estimated by the fitted model for each gene. We can further quantify the proportion of variance explained by different sources as the following. For each gene, the expression variance is partitioned into three components: a component that is explained by technical factors, a component that is explained by structured biological factors, and the residual error variance. Genes are evenly divided into ten quantiles based on the sample variance.

```
Var <- res$VarianceSummary
ConProp <- round(Var["ModelConfounding"]/Var["Model"], 3)
StructuredProp <- round(Var["ModelStructure"]/Var["Model"], 3)
PropTable <- data.frame(ConProp, StructuredProp, 1 - ConProp -
    StructuredProp)
colnames(PropTable) <- c("Tech", "Structured", "Residue")
QuantileTable <- QuantileSummary(PropTable, quantiles = seq(0.1,
    1, by = 0.1), rankingby = unlist(Var["Sample"]))
head(QuantileTable)
```

```
##     Tech Structured Residue Quantile
## 1 0.0317      0.139   0.830       10
## 2 0.0347      0.121   0.845       20
## 3 0.0285      0.107   0.864       30
## 4 0.0285      0.127   0.844       40
## 5 0.0353      0.114   0.851       50
## 6 0.0200      0.124   0.856       60
```

We can use barplot to visualize the result as in Fig 1.

```
library(reshape2)
library(ggplot2)
library(easyGgplot2)

df <- melt(QuantileTable, id.vars = "Quantile")
colnames(df)[3] <- "Proportion"
colnames(df)[2] <- "Variance"

ff <- ggplot2.barplot(data = df, xName = "Quantile", yName = "Proportion",
    groupName = "Variance", groupColors = c("#66B2FF", "#FFB266",
        "#FFAAD4"), position = position_stack(), backgroundColor = "white",
    color = "black", xtitle = "Quantile (%)", ytitle = "Proportion",
    mainTitle = "", removePanelGrid = TRUE, removePanelBorder = TRUE,
    axisLine = c(0.5, "solid", "black"), ylim = c(0, 1.05), legendPosition = "top",
    legendTextFont = c(10, "bold", "black"))
```

```
ff
```

3.3 Removing cell cycle effect

On the same dataset, we can futher remove the cell cycle effect by treating cell cycle genes as control and non cell cycle genes as target.

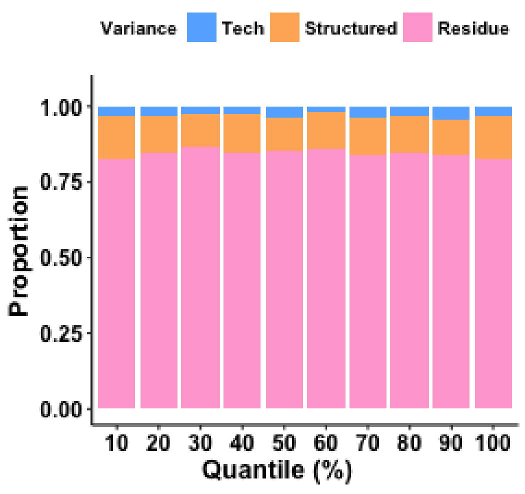

Figure 1: Decomposition of variance into three components. Genes are evenly divided into ten quantiles based on the sample variance.

```
Expression <- res$Adjusted
Label <- CellCycleData$Label[CellCycleData$Label != "Spikein"]
Control <- Expression[, Label == "CellCycle"]
Target <- Expression[, Label == "NonCellCycle"]
system.time(res2 <- scPLS(Target, Control, k1 = 2, k2 = 5, iter = 100,
    method = "EM", Chunk = FALSE, center = TRUE))

##    user  system elapsed
##   19.31    6.07   25.27
```

Similarly we can quantify the proportion of variance explained by cell cycle process.

```
Var <- res2$VarianceSummary
ConProp <- round(Var["ModelConfounding"]/Var["Model"], 3)
StructuredProp <- round(Var["ModelStructure"]/Var["Model"], 3)
PropTable <- data.frame(ConProp, StructuredProp, 1 - ConProp -
    StructuredProp)
colnames(PropTable) <- c("CellCycle", "Structured", "Residue")
median(PropTable$CellCycle)

## [1] 0.0685
```

```
QuantileTable <- QuantileSummary(PropTable, quantiles = seq(0.1,
    1, by = 0.1), rankingby = unlist(Var["Sample"]))
df <- melt(QuantileTable, id.vars = "Quantile")
colnames(df)[3] <- "Proportion"
colnames(df)[2] <- "Variance"

ff <- ggplot2.barplot(data = df, xName = "Quantile", yName = "Proportion",
    groupName = "Variance", groupColors = c("#66B2FF", "#FFB266",
```

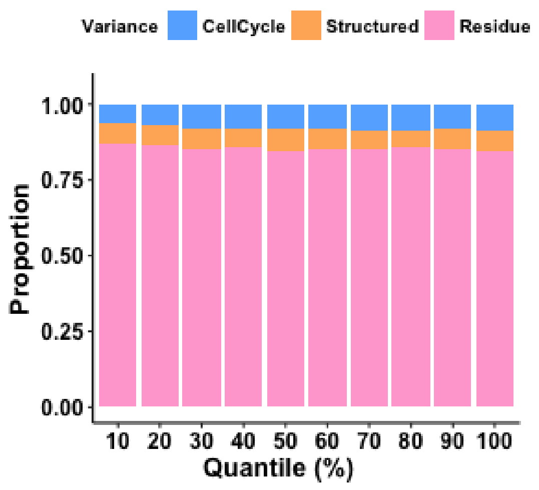

Figure 2: Decomposition of variance into three components. Genes are evenly divided into ten quantiles based on the sample variance.

```
        "#FFAAD4"), position = position_stack(), backgroundColor = "white",
    color = "black", xtitle = "Quantile (%)", ytitle = "Proportion",
    mainTitle = "", removePanelGrid = TRUE, removePanelBorder = TRUE,
    axisLine = c(0.5, "solid", "black"), ylim = c(0, 1.05), legendPosition = "top",
    legendTextFont = c(10, "bold", "black"))
```

```
ff
```

We can compare the expression of non cell cycle genes in orginal dataset and after removing both technical effect and cell cycle effect.

```
celltypes <- gsub("_[^<>]*", "", rownames(CellCycleData$Expression))
pcs <- prcomp(CellCycleData$Expression[, CellCycleData$Label ==
    "NonCellCycle"], center = TRUE)
PC1 <- pcs$x[, 1]
PC2 <- pcs$x[, 2]
df <- data.frame(PC1, PC2, "raw", celltypes)
colnames(df) <- c("PC1", "PC2", "Method", "Type")

ff1 <- ggplot2.scatterplot(data = df, xName = "PC1", yName = "PC2",
    groupName = "Type", size = 5, backgroundColor = "white",
    groupColors = c("#66B2FF", "#FFB266", "#FFAAD4"), xtitle = "PC1",
    ytitle = "PC2", mainTitle = "Raw", removePanelGrid = TRUE,
    removePanelBorder = TRUE, setShapeByGroupName = TRUE, showLegend = FALSE,
    mainTitleFont = c(25, "bold", "black"), xtitleFont = c(25,
        "bold", "black"), ytitleFont = c(25, "bold", "black"),
    xTickLabelFont = c(25, "bold", "black"), yTickLabelFont = c(25,
        "bold", "black"))
```

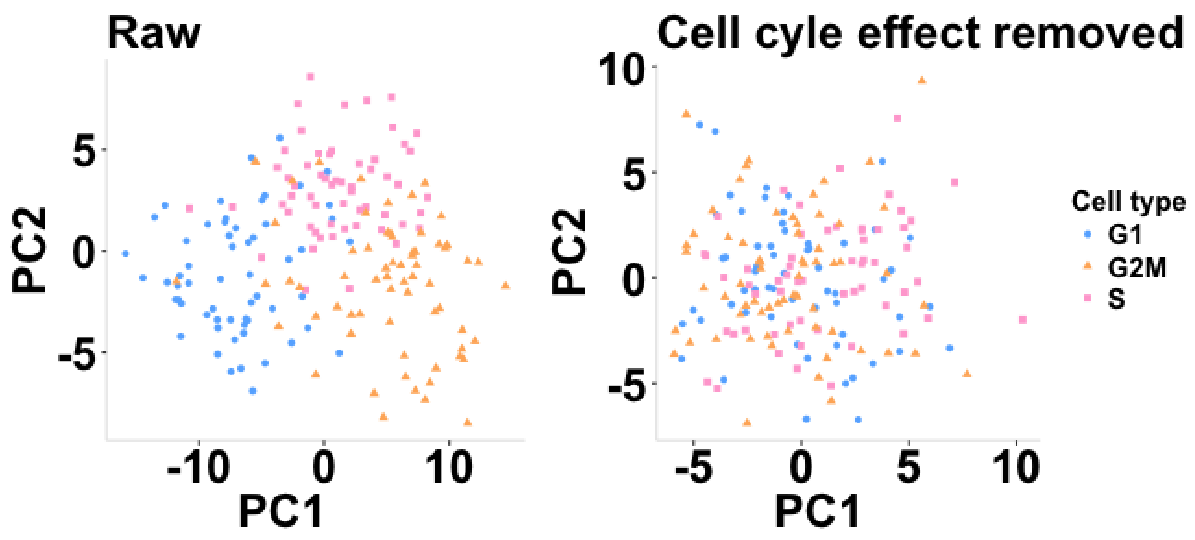

Figure 3: PCA analysis for the uncorrected data and scPLS corrected data. In the uncorrected data, there is a clear separation of cells by cell-cycle stage. Such separation of cells is no longer observed in the corrected data.

```
pcs <- prcomp(res2$Adjusted, center = TRUE)
PC1 <- pcs$x[, 1]
PC2 <- pcs$x[, 2]
df <- data.frame(PC1, PC2, "rmcellcyle", celltypes)
colnames(df) <- c("PC1", "PC2", "Method", "Type")

ff2 <- ggplot2.scatterplot(data = df, xName = "PC1", yName = "PC2",
    groupName = "Type", size = 5, backgroundColor = "white",
    groupColors = c("#66B2FF", "#FFB266", "#FFAAD4"), xtitle = "PC1",
    ytitle = "PC2", mainTitle = "Cell cyle effect removed", removePanelGrid = TRUE,
    removePanelBorder = TRUE, setShapeByGroupName = TRUE, showLegend = TRUE,
    mainTitleFont = c(25, "bold", "black"), xtitleFont = c(25,
        "bold", "black"), ytitleFont = c(25, "bold", "black"),
    xTickLabelFont = c(25, "bold", "black"), yTickLabelFont = c(25,
        "bold", "black"), legendTitle = "Cell type", legendTitleFont = c(15,
        "bold", "black"), legendTextFont = c(15, "bold", "black"))
```

```
ff1
```

```
ff2
```

3.4 Other algorithms

`scPLS()` also provides other alternative algorithms to estimate the confounding factors. Users can select different algorithms by specifying the option `method`. `PCA` algorithm implements the initializer of our EM algorithm. The latent factors are estimated from a Singular Value Decomposition. Details of the algorithm can be found in our manuscript. Other choices all impose sparsity assumptions on the structured biological factor matrix to improve interpretability. These include “EMSparse” algorithm, with penalty on sparsity of the factor matrix pre-specified, “EMSparseTraining” algorithm with penalty on sparsity learned from training samples, “EMparseNfold” algorithm with penalty on sparsity learned from N fold cross-validation, and “IBP” algorithm with the sparse factor matrix modeled by an Indian Buffet Process prior.

4 Clustering analysis using CAFE

4.1 Method

A key analytic task in scRNAseq involves classifying cells into sub-populations, which requires the development of statistical methods that can perform effective and accurate unsupervised clustering. However, standard clustering methods are not directly applicable to scRNAseq data, as these methods often fail to account for the high measurement noise and an abundance of drop-out events encountered in scRNAseq data, which are results of the low capture efficiency and low amount of input material in scRNAseq. Here, we present a Bayesian nonparametric method that is tailored for clustering analysis in scRNAseq, called CAFE. CAFE accounts for both high measurement noise and drop-out events, and is capable of automatically inferring the number of cell sub-populations from the data at hand. It incorporates the sparse factor model and the Dirichlet process normal mixture clustering model into a same joint framework, and effectively performs clustering on a low dimensional informative manifold inferred from the noisy data - thus enabling accurate clustering performance.

4.2 running CAFE

We use the cell lines data published in Pollen et al as an example to illustrate the usage of `CAFE()` function. We selected top 1000 genes with largest variance across samples, stored in data matrix `Expression`.

```
library(Citrus)
data(Pollen)
str(Expression)
```

```
##  num [1:301, 1:1000] 5.32 0 0 2.44 0 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:301] "Hi_2338_1" "Hi_2338_2" "Hi_2338_3" "Hi_2338_4" ...
##   ..$ : chr [1:1000] "AAAS" "AARSD1" "ABRACL" "ABT1" ...
```

`CAFE()` takes inputs of a $n \times p$ matrix of target gene expression and a pre-specified number of latent factors K. When `ModelZero` set to be true, drop-out events will be modeled. The default is FALSE.

```
ClusterResult <- CAFE(Expression, K = 8, iter = 10000, method = "SpikeSlab",
    ModelZero = FALSE)
```

We can evaluate the clustering results by visualizing the adjacency matrix.

```
celltypes <- unlist(strsplit(rownames(Expression), "_"))[seq(2,
    301 * 3, by = 3)]
```

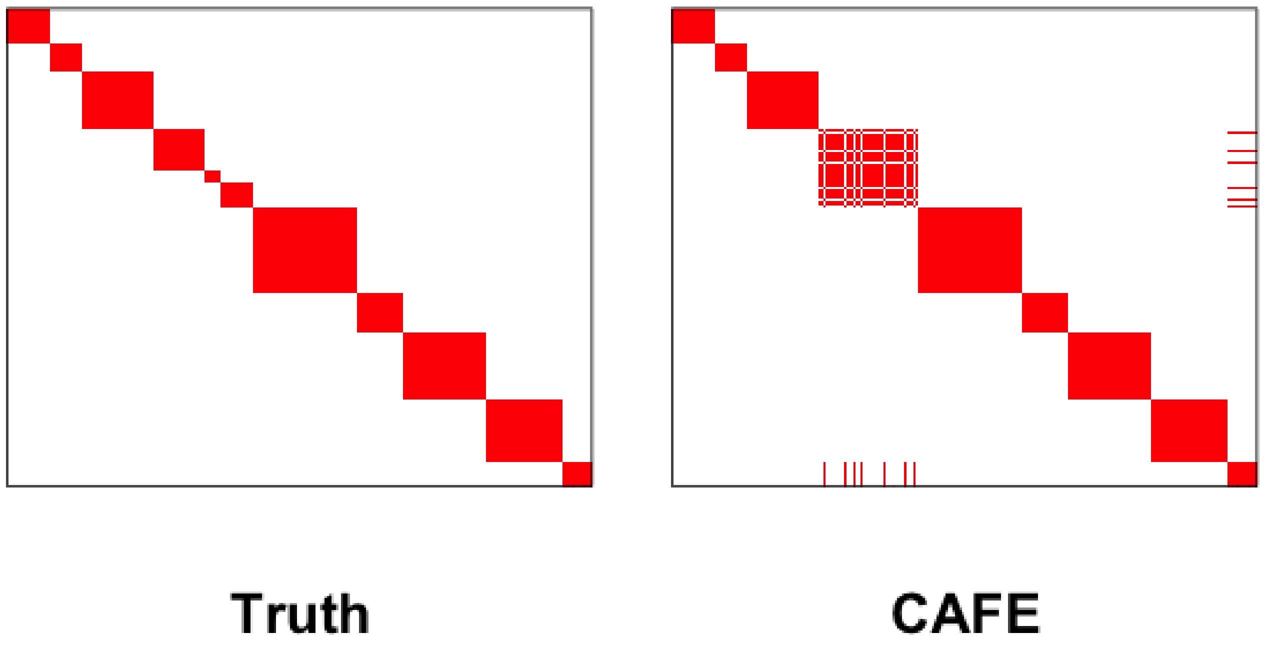

Figure 4: Visualization of the adjacency matrices.

```
neworder <- order(celltypes)
Proposed <- AdjacencyMatrix(ClusterResult$Cluster[neworder])
original <- AdjacencyMatrix(celltypes[neworder])
par(mfrow = c(1, 2), mar = c(4, 0.5, 0.5, 0.5))
plotBlock(original, supportOnly = TRUE, plotname = "Truth")
plotBlock(Proposed, supportOnly = TRUE, plotname = "CAFE")
```

```
truth <- as.numeric(as.factor(celltypes))
clusterSummary(truth, ClusterResult$Cluster)

##     RandIndex  Sensitivity  Specificity          PPV
##          1.00         0.97         0.98         0.88
##           NPV       Mutual  RandIndexAdj
##          1.00         0.45         0.91
```

`CAFE()` can output pruned clustering results. When `mergeCluster` set to be true, clusters with number of samples less than `mergeN` will be merged into clusters with closest mean.

```
ClusterResult <- CAFE(Expression, K = 10, iter = 5000, method = "SpikeSlab",
    ModelZero = FALSE, mergeCluster = TRUE, mergeN = 5)
```

The pruning can also be done on existing clustering results using function `MergeCluster`.

```
Pruned <- MergeCluster(ClusterResult, mergeN = 5)
```