Introduction to Iterative Clustering Analysis Using iterClust

Hongxu Ding and Andrea Califano
Department of Systems Biology, Columbia University, New York, USA

April 25, 2023

Contents

1 Introduction

In a scenario where populations A, B1, B2 exist, pronounce differences between A and B may mask subtle differences between B1 and B2. To solve this problem, so that heterogeneity can be better detected, clustering analysis needs to be performed iteratively, so that, for example, in iteration 1, A and B are separated and in iteration 2, B1 and B2 are separated . The `iterClust()` function in *iterClust* package provides an statistical framework for performing such iterative clustering analysis, which can be used to, for instance discover cell populations using single cells RNA-Seq profiles, clustering clinically-related patient gene expression profiles and solve general clustering problems.

1.1 General Work Flow

`iterClust()` organizes user-defined functions and parameters as follows:
ith Iteration Start =>
`featureSelect` (feature selection) =>
`minFeatureSize` (confirm enough features are selected) =>
`clustHetero` (confirm heterogeneity) =>
`coreClust` (generate several clustering schemes, only for heterogenous clusters) =>
`clustEval` (pick the optimal clustering scheme) =>
`minClustSize` (remove clusters with few observations) =>
`obsEval` (evaluate how each observations are clustered) =>
`obsOutlier` (remove poorly clustered observations) =>
results in Internal Variables (IV) =>
ith Iteration End

1.2 Internal Variables (IV)

`iterClust()` has the following IVs which can be used in user-defined functions:

`cluster`, a list with two elements, named cluster and feature, which are also list object, organized by round of iterations, containing names of observations for each clusters in this specific iteration, and features used to split clusters in previous iterations thereby produce the current clusters organized as lists, respectively.

`depth`, an integer specifying current round of iteration.

1.3 Installation

iterClust depends on *SummarizedExperiment* and *Biobase*. Running examples in *iterClust* requires *tsne*, *cluster*, *ConsensusClusterPlus* and *bcellViper*. To install *iterClust*, from bioconductor

```
if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
BiocManager::install("iterClust")
```

1.4 Citing

2 Data Preparation

We applied `iterClust()` to a B-cell expression dataset included in *bcellViper*. We load the two librarues first, followed by load and filter expression matrix and phenotype annotation.

```
> library(iterClust)
> library(bcellViper)
> data(bcellViper)
> exp <- exprs(dset)
> pheno <- as.character(dset@phenoData@data$description)
> exp <- exp[, pheno %in% names(table(pheno))[table(pheno) > 5]]
> pheno <- pheno[pheno %in% names(table(pheno))[table(pheno) > 5]]
> dim(exp)

[1] 6249  161

> table(pheno)

pheno
 B-CLL     BL   DLCL    HCL    PEL pB-CLL  pDLCL    pFL   pMCL
    16     23     53     13      9     18     15      6      8
```

3 Define functions

We define functions needed for `iterClust()`, as well as load package *cluster* that these functions needed.

```
> library(cluster)
```

In every iterations, all genes in the dataset were used for clustering analysis.

```
> featureSelect <- function(dset, iteration, feature) return(rownames(dset
```

In every iterations, the core function for clustering is `pam()` in package cluster. We searched through 2 to 5 clusters to find the optimal result.

```
> coreClust <- function(dset, iteration){
+   dist <- as.dist(1 - cor(dset))
+   range=seq(2, (ncol(dset)-1), by = 1)
+   clust <- vector("list", length(range))
+   for (i in 1:length(range)) clust[[i]] <- pam(dist, range[i])$clusterin
+   return(clust)}
```

In every iterations, the core function for evaluating different clustering schemes is `silhouette()` in package *cluster*. We considered clustering schemes with the highest average silhouette score as the optimal scheme. clust is the output for function `clustfun()`.

```
> clustEval <- function(dset, iteration, clust){
+   dist <- as.dist(1 - cor(dset))
+   clustEval <- vector("numeric", length(clust))
+   for (i in 1:length(clust)){
+       clustEval[i] <- mean(silhouette(clust[[i]], dist)[, "sil_width"])}
+   return(clustEval)}
```

In every iterations, clusters with average silhouette score greater than 0.15 were considered as heterogenous and further splitted.

```
> clustHetero <- function(clustEval, iteration){
+     return(clustEval > 0*iteration+0.15)}
```

In every iterations, the core function for evaluating each observation is `silhouette()` in package cluster. clust is the output for function `clustfun()`.

```
> obsEval <- function(dset, clust, iteration){
+   dist <- as.dist(1 - cor(dset))
+   obsEval <- vector("numeric", length(clust))
+   return(silhouette(clust, dist)[, "sil_width"])}
```

In every iterations, observations with silhouette score smaller than -1 were considered as outlier observations.

```
> obsOutlier <- function(obsEval, iteration) return(obsEval < 0*iteration-
```

4 Run iterClust

`iterClust()` was run with the above defined functions. Then we showed how the results of `iterClust()` are organized.

```
> c <- iterClust(exp, maxIter=3, minFeatureSize=100, minClustSize=5)
> names(c)

[1] "cluster"   "feature"   "clustEval" "obsEval"

> names(c$cluster)

[1] "Iter1" "Iter2"

> names(c$cluster$Iter1)

[1] "Cluster1" "Cluster2" "Cluster3" "Cluster4" "Cluster5"
```

```
> c$cluster$Iter1$Cluster1

 [1] "GSM44075" "GSM44078" "GSM44080" "GSM44081" "GSM44082" "GSM44083"
 [7] "GSM44084" "GSM44088" "GSM44089" "GSM44091" "GSM44092" "GSM44094"
[13] "GSM44095" "GSM44246" "GSM44247" "GSM44248" "GSM44249" "GSM44250"
[19] "GSM44251" "GSM44252" "GSM44261" "GSM44264" "GSM44265" "GSM44266"
[25] "GSM44267" "GSM44268" "GSM44269" "GSM44076" "GSM44077" "GSM44079"
[31] "GSM44090" "GSM44093" "GSM44192" "GSM44244" "GSM44245" "GSM44253"
[37] "GSM44254" "GSM44255" "GSM44256" "GSM44257" "GSM44258" "GSM44259"
[43] "GSM44291" "GSM44292"

> names(c$feature)

[1] "Iter1" "Iter2"

> names(c$feature$Iter1)

[1] "OriginalDataset"

> names(c$feature$Iter2)

[1] "Cluster1inIter1" "Cluster2inIter1" "Cluster3inIter1" "Cluster4inIter1"
[5] "Cluster5inIter1"

> c$feature$Iter2$Cluster1inIter1[1:10]

 [1] "ADA"     "CDH2"    "MED6"    "NR2E3"   "ACOT8"   "ABI1"    "GNPDA1"
 [8] "TANK"    "HGC6.3"  "C1orf68"
```

5 Compare iterClust, PAM and Consensus Clustering

In this section, we compared the performance of `iterClust()` with another clustering framework `ConsensusClusterPlus()` as well as their underlying clustering algorithm `pam()`.

```
> library(ConsensusClusterPlus)
> set.seed(1)
> consensusClust = ConsensusClusterPlus(exp, maxK = 10,
+                                       reps = 100, clusterAlg = "pam",
+                                       distance = "pearson", plot = FALSE)
> ICL <- calcICL(consensusClust, plot = FALSE)
> ICL <- sapply(2:10, function(k, ICL){
+   s <- ICL$clusterConsensus[grep(k, ICL$clusterConsensus[, "k"]),
+                             "clusterConsensus"]
+   mean(s[is.finite(s)])}, ICL=ICL)
```

We first projected the data on 2D-tSNE space for later visualization purpose.

```
> library(tsne)
> dist <- as.dist(1 - cor(exp))
> set.seed(1)
> tsne <- tsne(dist, perplexity = 20, max_iter = 500)
```

Then we compared `iterClust()`, `pam()` and `ConsensusClusterPlus()`.

```
> par(mfrow = c(1, 2))
> for (j in 1:length(c$cluster)){
+     COL <- structure(rep(1, ncol(exp)), names = colnames(exp))
+     for (i in 1:length(c$cluster[[j]])) COL[c$cluster[[j]][[i]]] <- i+1
+     plot(tsne[, 1], tsne[, 2], cex = 0, cex.lab = 1.5,
+          xlab = "Dim1", ylab = "Dim2",
+          main = paste("iterClust, iter=", j, sep = ""))
+     text(tsne[, 1], tsne[, 2], labels = pheno, cex = 0.5, col = COL)
+     legend("topleft", legend = "Outliers", fill = 1, bty = "n")}
```

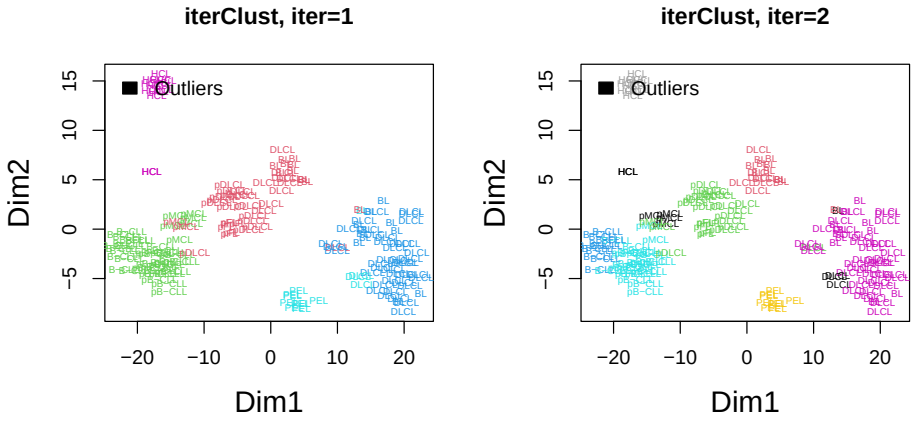

Figure 1: Result of `iterClust()`

```
> par(mfrow = c(1, 2))
> for (j in 1:length(c$cluster)){
+     plot(tsne[, 1], tsne[, 2], cex = 0, cex.lab = 1.5,
+          xlab = "Dim1", ylab = "Dim2",
+          main = paste("PAM, k=", length(c$cluster[[j]]), sep = ""))
+     text(tsne[, 1], tsne[, 2], labels = pheno, cex = 0.5,
+          col = pam(dist, k = length(c$cluster[[j]]))$clustering)}
```

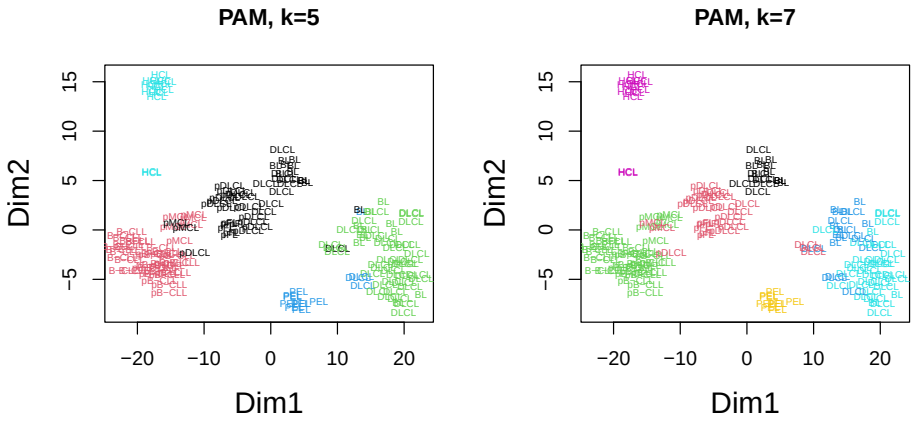

Figure 2: Result of `PAM()` with same number of clusters given by `iterClust()`

```
> par(mfrow = c(2, 2))
> plot(c(2:10), ICL, xlab = "#Clusters", ylab = "Cluster Consensus Score",
+      col = c(2, rep(1, 8)), ylim = c(0.8, 1),
+      cex.lab = 1.5, pch = 16, main = "")
> plot(tsne[, 1], tsne[, 2], cex = 0, cex.lab = 1.5,
+      xlab = "Dim1", ylab = "Dim2",main = "Consensus Clustering+PAM, k=2"
> text(tsne[, 1], tsne[, 2], labels = pheno,
+      cex = 0.5, col = consensusClust[[2]]$consensusClass)
> plot(c(2:10), ICL, xlab = "#Clusters", ylab = "Cluster Consensus Score",
+      col = c(rep(1, 5), 2, 1, 1), ylim = c(0.8, 1),
+      cex.lab = 1.5, pch = 16, main = "")
> plot(tsne[, 1], tsne[, 2], cex = 0, cex.lab = 1.5,
+      xlab = "Dim1", ylab = "Dim2",main = "Consensus Clustering+PAM, k=7"
> text(tsne[, 1], tsne[, 2], labels = pheno, cex = 0.5,
+      col = consensusClust[[7]]$consensusClass)
```

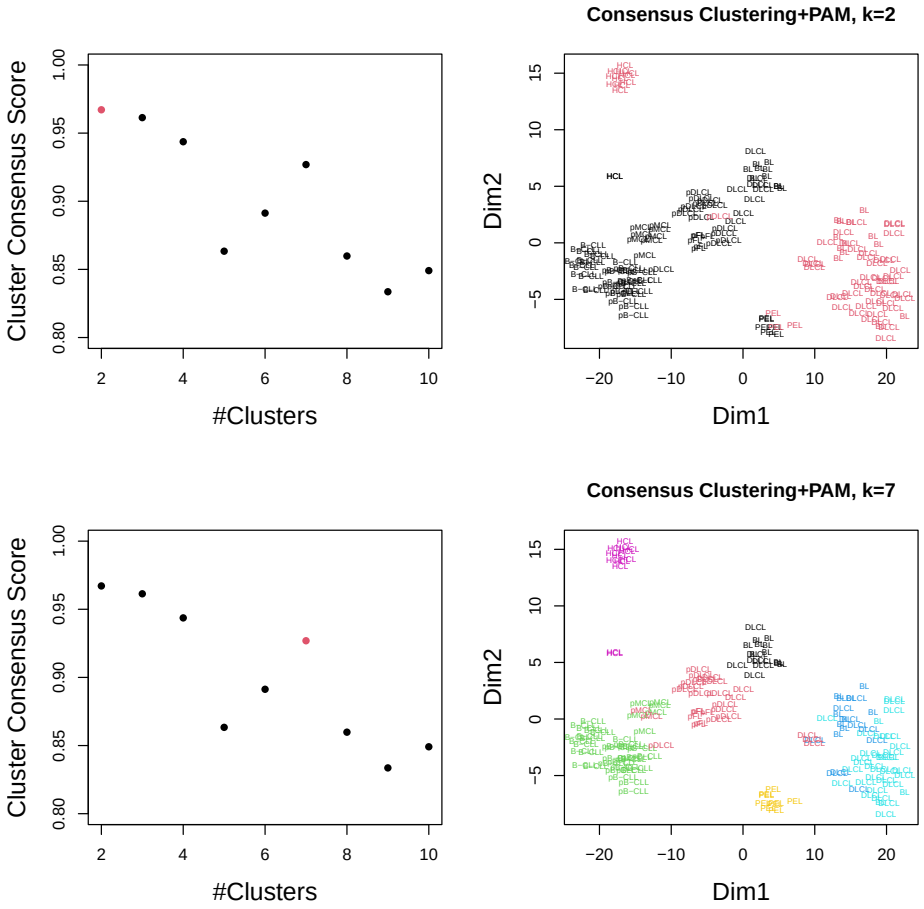

Figure 3: Result of `ConsensusClusterPlus()`

The results showed that `iterClust()` can distinguish subtle differences between purified and unpurified B-cells (pDLCL VS DLCL, B-CLL VS pB-CLL), which cannot be distinguished by `pam()` and `ConsensusClusterPlus()`. Also, `pam()` and `ConsensusClusterPlus()` falsely separated a homogenous cluster containing DLCL samples (DLCL samples are known to have subpopulations and this is one subpopulation).