Package ‘PINSPlus’

May 31, 2020

Encoding UTF-8
Type Package
Title Clustering Algorithm for Data Integration and Disease Subtyping
Version 2.0.4
Date 2020-05-30
Author Hung Nguyen, Bang Tran, Duc Tran and Tin Nguyen
Maintainer Hung Nguyen <hungnp@nevada.unr.edu>
Description Provides a robust approach for omics data integration and disease subtyping. PIN-SPlus is fast and supports the analysis of large datasets with hundreds of thousands of samples and features. The software automatically determines the optimal number of clusters and then partitions the samples in a way such that the results are robust against noise and data perturbation (Nguyen et.al. (2019) <DOI: 10.1093/bioinformatics/bty1049>, Nguyen et.al. (2017)<DOI: 10.1101/gr.215129.116>).
License LGPL
Depends R (>= 2.10)
Imports foreach, entropy, doParallel, matrixStats, Rcpp, RcppParallel, FNN, cluster, irlba, mclust
RoxygenNote 7.1.0
Suggests knitr, rmarkdown, survival
LinkingTo Rcpp, RcppArmadillo, RcppParallel
VignetteBuilder knitr
NeedsCompilation yes
Repository CRAN
Date/Publication 2020-05-31 01:30:03 UTC

R topics documented:

PINSPlus-package ............................................................. 2
AML2004 ................................................................. 2
KIRC ........................................................................... 3
PerturbationClustering ...................................................... 4
SubtypingOmicsData .......................................................... 9
This package implements clustering algorithms proposed by Nguyen et al. (2017, 2019). Perturbation Clustering for data INtegration and disease Subtyping (PINS) is an approach for integration of data and classification of diseases into various subtypes. PINS+ provides algorithms supporting both single data type clustering and multi-omics data type. PINSPlus is an improved version of PINS by allowing users to customize the based clustering algorithm and perturbation methods. Furthermore, PINSPlus is fast and supports the analysis of large datasets with millions of samples and features.

Details

PINS+ provides PerturbationClustering and SubtypingOmicsData functions for single data type clustering and multi-omics data type clustering. PINS makes use of different clustering algorithms such as kmeans and pam to perform clustering actions. The principle of PINS is to find the optimum number of clusters and location of each sample in the clusters based on perturbation methods such as noise or subsampling. PINS+ allows users to pass their own clustering algorithm and perturbation method.

References


See Also

PerturbationClustering, SubtypingOmicsData

AML2004 Acute myelogenous leukemia dataset

Description

Acute myelogenous leukemia dataset

Format

A list containing properties:
KIRC

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gene</td>
<td>data.frame</td>
<td>mRNA expression data</td>
</tr>
<tr>
<td>Group</td>
<td>data.frame</td>
<td>Data frame indicating the cluster to which each sample is allocated</td>
</tr>
</tbody>
</table>

**Source**

http://portals.broadinstitute.org/cgi-bin/cancer/publications/view/89

**References**


---

**KIRC**

*Kidney renal clear cell carcinoma dataset*

**Description**

The Cancer Genome Atlas Kidney Renal Clear Cell Carcinoma (TCGA-KIRC) data collection is part of a larger effort to build a research community focused on connecting cancer phenotypes to genotypes by providing clinical images matched to subjects from The Cancer Genome Atlas (TCGA). Clinical, genetic, and pathological data resides in the Genomic Data Commons (GDC) Data Portal while the radiological data is stored on The Cancer Imaging Archive (TCIA).

This embed version of KIRC in PINPlus package is the reduced version of KIRC using Principle Component Analysis.

**Format**

A list containing properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GE</td>
<td>data.frame</td>
<td>mRNA expression data</td>
</tr>
<tr>
<td>ME</td>
<td>data.frame</td>
<td>DNA Methylation data</td>
</tr>
<tr>
<td>MI</td>
<td>data.frame</td>
<td>miRNA expression data</td>
</tr>
<tr>
<td>survival</td>
<td>data.frame</td>
<td>Clinical survival data</td>
</tr>
</tbody>
</table>

**Source**

https://wiki.cancerimagingarchive.net/display/Public/TCGA-KIRC

**References**

Akin, O., Elnajjar, P., Heller, M., Jarosz, R., Erickson, B. J., Kirk, S., ... Filippini, J. (2016). Radiology Data from The Cancer Genome Atlas Kidney Renal Clear Cell Carcinoma [TCGA-KIRC]
PerturbationClustering

Perturbation clustering

Description

Perform subtyping using one type of high-dimensional data

Usage

PerturbationClustering(
data, 
kMin = 2, 
kMax = 5, 
verbose = T, 
ncore = 1, 
clusteringMethod = "kmeans", 
clusteringFunction = NULL, 
clusteringOptions = NULL, 
perturbMethod = "noise", 
perturbFunction = NULL, 
perturbOptions = NULL, 
PCAFunction = NULL, 
iterMin = 20, 
iterMax = 200, 
madMin = 0.001, 
msdMin = 1e-06
)

Arguments

data Input matrix. The rows represent items while the columns represent features.
kMin The minimum number of clusters. Default value is 2.
kMax The maximum number of clusters. The algorithm runs from k = 2 to k = kMax. Default value is 5.
verbose Boolean value indicating the algorithm to run with or without logging. Default value is TRUE.
ncore Number of cores that the algorithm should use. Default value is 1.
clusteringMethod The name of built-in clustering algorithm that PerturbationClustering will use. Currently supported algorithm are kmeans, pam and hclust. Default value is "kmeans".
clusteringFunction The clustering algorithm function that will be used instead of built-in algorithms.
PerturbationClustering

clusteringOptions
A list of parameter will be passed to the clustering algorithm in clusteringMethod.

perturbMethod
The name of built-in perturbation method that PerturbationClustering will use, currently supported methods are noise and subsampling. Default value is "noise".

perturbFunction
The perturbation method function that will be used instead of built-in ones.

perturbOptions
A list of parameter will be passed to the perturbation method in perturbMethod.

PCAFunction
The customized PCA function that user can manually define.

iterMin
The minimum number of iterations. Default value is 20.

iterMax
The maximum number of iterations. Default value is 200.

madMin
The minimum of Mean Absolute Deviation of AUC of Connectivity matrix for each k. Default value is 1e-03.

msdMin
The minimum of Mean Square Deviation of AUC of Connectivity matrix for each k. Default value is 1e-06.

Details


PerturbationClustering takes input as a numerical matrix or data frame of items as rows and features as columns. It uses a clustering algorithm as the based algorithm. Current built-in algorithms that users can use directly are kmeans, pam and hclust. The default parameters for built-in kmeans are nstart = 20 and iter.max = 1000. Users can change the parameters of built-in clustering algorithm by passing the value into clusteringOptions.

PerturbationClustering also allows users to pass their own clustering algorithm instead of using built-in ones by using clusteringFunction parameter. Once clusteringFunction is specified, clusteringMethod will be skipped. The value of clusteringFunction must be a function that takes two arguments: data and k, where data is a numeric matrix or data frame containing data that need to be clustered, and k is the number of clusters. clusteringFunction must return a vector of labels indicating the cluster to which each sample is allocated.

PerturbationClustering uses a perturbation method to perturb clustering input data. There are two built-in methods are noise and subsampling that users can use directly by passing to perturbMethod parameter. Users can change the default value of built-in perturbation methods by passing new value into perturbOptions:

1. noise perturbation method takes two arguments: noise and noisePercent. The default values are noise = NULL and noisePercent = "median". If noise is specified, noisePercent will be skipped.
2. subsampling perturbation method takes one argument percent which has default value of 80.

Users can also use their own perturbation methods by passing them into perturbFunction. Once perturbFunction is specified, perturbMethod will be skipped. The value of perturbFunction must be a function that takes one argument data - a numeric matrix or data frame containing data that need to be perturbed. perturbFunction must return an object list which is as follows:
1. data: the perturbed data
2. ConnectivityMatrixHandler: a function that takes three arguments: connectivityMatrix - the connectivity matrix generated after clustering returned data, iter - the current iteration and k - the number of cluster. This function must return a compatible connectivity matrix with the original connectivity matrix. This function aims to correct the connectivityMatrix if needed and returns the corrected version of it.
3. MergeConnectivityMatrices: a function that takes four arguments: oldMatrix, newMatrix, k and iter. The oldMatrix and newMatrix are two connectivity matrices that need to be merged, k is the cluster number and iter is the current number of iteration. This function must returns a connectivity matrix that is merged from oldMatrix and newMatrix

Value

PerturbationClustering returns a list with at least the following components:

- k: The optimal number of clusters
- cluster: A vector of labels indicating the cluster to which each sample is allocated
- origS: A list of original connectivity matrices
- pertS: A list of perturbed connectivity matrices

References


See Also

- kmeans, pam

Examples

# Load the dataset AML2004
data(AML2004)
data <- as.matrix(AML2004$Gene)

# Perform the clustering
result <- PerturbationClustering(data = data)

# Plot the result
condition = seq(unique(AML2004$Group[, 2]))
names(condition) <- unique(AML2004$Group[, 2])
plot(
  prcomp(data)$x,
  col = result$cluster,
  pch = condition[AML2004$Group[, 2]],
main = "AML2004"
)
legend(
  "bottomright",
  legend = paste("Cluster ", sort(unique(result$cluster)), sep = ""),
  fill = sort(unique(result$cluster))
)
legend("bottomleft", legend = names(condition), pch = condition)

# Change kmeans parameters
result <- PerturbationClustering(
  data = data,
  clusteringMethod = "kmeans",
  clusteringOptions = list(
    iter.max = 500,
    nstart = 50
  )
)

# Change to use pam
result <- PerturbationClustering(data = data, clusteringMethod = "pam")

# Change to use hclust
result <- PerturbationClustering(data = data, clusteringMethod = "hclust")

# Pass a user-defined clustering algorithm
result <- PerturbationClustering(data = data, clusteringFunction = function(data, k){
  # this function must return a vector of cluster
  kmeans(x = data, centers = k, nstart = k*10, iter.max = 2000)$cluster
})

# Use noise as the perturb method
result <- PerturbationClustering(data = data,
  perturbMethod = "noise",
  perturbOptions = list(noise = 0.3))

# or
result <- PerturbationClustering(data = data,
  perturbMethod = "noise",
  perturbOptions = list(noisePercent = 10))

# Change to use subsampling
result <- PerturbationClustering(data = data,
  perturbMethod = "subsampling",
  perturbOptions = list(percent = 90))

# Users can pass their own perturb method
result <- PerturbationClustering(data = data, perturbFunction = function(data){
  rowNum <- nrow(data)
  colNum <- ncol(data)
  epsilon <-
    matrix(
      data = rnorm(rowNum * colNum, mean = 0, sd = 1.234),
      nrow = rowNum,
      ncol = colNum
    )
  colNum <- ncol(data)
  epsilon <-
    matrix(
      data = rnorm(rowNum * colNum, mean = 0, sd = 1.234),
      nrow = rowNum,
PerturbationClustering

ncol = colNum

list(
    data = data + epsilon,
    ConnectivityMatrixHandler = function(connectivityMatrix, ...) {
        connectivityMatrix
    },
    MergeConnectivityMatrices = function(oldMatrix, newMatrix, iter, ...) {
        return((oldMatrix*(iter-1) + newMatrix)/iter)
    }
)

# Clustering on simulation data
# Load necessary library

if (!require("mclust")) install.packages("mclust")
library(mclust)
library(irlba)

# Generate a simulated data matrix with the size of 50,000 x 5,000
sampleNum <- 50000 # Number of samples
geneNum <- 5000 # Number of genes
subtypeNum <- 3 # Number of subtypes

# Generate expression matrix
exprs <- matrix(rnorm(sampleNum*geneNum, 0, 1), nrow = sampleNum, ncol = geneNum)
rownames(exprs) <- paste0("S", 1:sampleNum) # Assign unique names for samples

# Generate subtypes
group <- sort(rep(1:subtypeNum, sampleNum/subtypeNum + 1)[1:sampleNum])
names(group) <- rownames(exprs)

# Make subtypes separate
for (i in 1:subtypeNum) {
    exprs[[group == i, 1:100 + 100*(i-1)]] <- exprs[[group == i, 1:100 + 100*(i-1)]] + 2
}

# Plot the data
library(irlba)
exprs.pca <- irlba::prcomp_irlba(exprs, n = 2)$x
plot(exprs.pca, main = "PCA")

# Run PINSPlus clustering:
set.seed(1)
t1 <- Sys.time()
result <- PerturbationClustering(data = exprs.pca, ncore = 1)
t2 <- Sys.time()

# Print out the running time:
```r
# Print out the number of clusters:
result$k

# Get cluster assignment
subtype <- result$cluster

# Here we assess the clustering accuracy using Adjusted Rand Index (ARI).
# ARI takes values from -1 to 1 where 0 stands for a random clustering and 1
# stands for a perfect partition result.
if (!require("mclust")) install.packages("mclust")
library(mclust)
ari <- mclust::adjustedRandIndex(subtype, group)

# Plot the cluster assignments
colors <- as.numeric(as.character(factor(subtype)))
plot(exprs.pca, col = colors, main = "Cluster assignments for simulation data")
legend("topright", legend = paste("ARI:", ari))

legend("bottomright", fill = unique(colors),

legend = paste("Group ",
leveles(factor(subtype)), ": ",
table(subtype)[levels(factor(subtype))], sep = " - ")
```

---

### Description

Perform subtyping using multiple types of data

### Usage

```r
SubtypingOmicsData(
dataList,
kMin = 2,
kMax = 5,
agreementCutoff = 0.5,
ncore = 1,
verbose = T,
...
)
```
Arguments

dataList a list of data matrices. Each matrix represents a data type where the rows are items and the columns are features. The matrices must have the same set of items.

kMin The minimum number of clusters. Default value is 2.

kMax the maximum number of clusters. Default value is 5.

agreementCutoff agreement threshold to be considered consistent. Default value is 0.5.

ncore Number of cores that the algorithm should use. Default value is 1.

verbose set it to TRUE or FALSE to get more or less details respectively.

... these arguments will be passed to PerturbationClustering algorithm. See details for more information.

Details

SubtypingOmicsData implements the Subtyping multi-omic data that are based on Perturbation clustering algorithm of Nguyen, et al (2017) and Nguyen, et al (2019). The input is a list of data matrices where each matrix represents the molecular measurements of a data type. The input matrices must have the same number of rows. SubtypingOmicsData aims to find the optimum number of subtypes and location of each sample in the clusters from integrated input data dataList through two processing stages:

1. Stage I: The algorithm first partitions each data type using the function PerturbationClustering. It then merges the connectivities across data types into similarity matrices. Both kmeans and similarity-based clustering algorithms - partitioning around medoids pam are used to partition the built similarity. The algorithm returns the partitioning that agrees the most with individual data types.

2. Stage II: The algorithm attempts to split each discovered group if there is a strong agreement between data types, or if the subtyping in Stage I is very unbalanced.

Value

SubtypingOmicsData returns a list with at least the following components:

cluster1 A vector of labels indicating the cluster to which each sample is allocated in Stage I

cluster2 A vector of labels indicating the cluster to which each sample is allocated in Stage II

dataTypeResult A list of results for individual data type. Each element of the list is the result of the PerturbationClustering for the corresponding data matrix provided in dataList.

References


See Also

PerturbationClustering

Examples

```r
# Load the kidney cancer carcinoma data
data(KIRC)

# Perform subtyping on the multi-omics data
dataList <- list(as.matrix(KIRC$GE), as.matrix(KIRC$ME), as.matrix(KIRC$MI))
names(dataList) <- c("GE", "ME", "MI")
result <- SubtypingOmicsData(dataList = dataList)

# Change Perturbation clustering algorithm's arguments
result <- SubtypingOmicsData(
  dataList = dataList,
  clusteringMethod = "kmeans",
  clusteringOptions = list(nstart = 50)
)

# Plot the Kaplan-Meier curves and calculate Cox p-value
library(survival)
cluster1 = result$cluster1; cluster2 = result$cluster2
a <- intersect(unique(cluster2), unique(cluster1))
names(a) <- intersect(unique(cluster2), unique(cluster1))
a[setdiff(unique(cluster2), unique(cluster1))] <- seq(setdiff(unique(cluster2), unique(cluster1))) + max(cluster1)
colors <- a[levels(factor(cluster2))]

coxFit <- coxph(
  Surv(time = Survival, event = Death) ~ as.factor(cluster2),
  data = KIRC$survival,
  ties = "exact"
)
mfit <- survfit(Surv(Survival, Death == 1) ~ as.factor(cluster2), data = KIRC$survival)
plot(
  mfit, col = colors,
  main = "Survival curves for KIRC, level 2",
  xlab = "Days", ylab = "Survival", lwd = 2
)
legend("bottomright",
  legend = paste("Cox p-value:",
                  round(summary(coxFit)$sctest[3], digits = 5),
                  sep = "")
)
```
```r
legend("bottomleft",
fill = colors,
legend = paste("Group ",
levels(factor(cluster2)),": ", table(cluster2)[levels(factor(cluster2))],
sep ="")
)
```
Index

*Topic datasets
  AML2004, 2
  KIRC, 3
*Topic package
  PINSPlus-package, 2

AML2004, 2
KIRC, 3
kmeans, 6
pam, 6
PerturbationClustering, 2, 4, 11
PINSPlus (PINSPlus-package), 2
PINSPlus-package, 2

SubtypingOmicsData, 2, 9