Package 'PINSPlus'

June 4, 2019

Type Package

Title Clustering Algorithm for Data Integration and Disease Subtyping

Version 2.0.0

Date 2019-06-04

Author Hung Nguyen, Sangam Shrestha and Tin Nguyen

Maintainer Hung Nguyen <hungnp@nevada.unr.edu>

Description Provides a robust approach for omics data integration and disease subtyping. 'PINSPlus' supports both single and multiple data types. The software automatically determines the optimal number of clusters and then partitions the samples in a way such that the results are robust to noise and data perturbation. 'PINSPlus' is fast and it supports parallel computing on Windows, Linux, and Mac OS.

License LGPL

Depends R (>= 2.10), cluster, entropy, parallel, pbmcapply, doParallel

Imports foreach

RoxygenNote 6.1.1

NeedsCompilation no

Suggests knitr, rmarkdown, survival

VignetteBuilder knitr

Repository CRAN

Date/Publication 2019-06-04 15:00:10 UTC

R topics documented:

PINSPlus-package *Perturbation Clustering for data INtegration and disease Subtyping*

Description

This package implements clustering algorithms proposed by Nguyen, et al (2017). Perturbation Clustering for data INtegration and disease Subtyping (PINS) is an approach for integraton 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 extended version of PINS by allowing users to customize the based clustering algorithm and perturbation methods. Furthermore, PINSPlus fastens PINS by supporting parallel processing and introducing efficient stopping criteria.

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

T Nguyen, R Tagett, D Diaz, S Draghici. A novel method for data integration and disease subtyping. Genome Research, 27(12):2025-2039, 2017.

See Also

`PerturbationClustering`, `SubtypingOmicsData`

AML2004 *Acute myelogenous leukemia dataset*

Description

Acute myelogenous leukemia dataset

Format

A list containing properties:

Name	Type	Description
Gene	data.frame	mRNA expression data
Group	data.frame	Data frame indicating the cluster to which each sample is allocated

Source

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

References

Brunet, J. P., Tamayo, P., Golub, T. R., & Mesirov, J. P. (2004). Metagenes and molecular pattern discovery using matrix factorization. Proceedings of the national academy of sciences, 101(12), 4164-4169.

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:

Name	Type	Description
GE	data.frame	mRNA expression data
ME	data.frame	DNA Methylation data
MI	data.frame	miRNA expression data
survival	data.frame	Clinical survival data

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] collection. The Cancer Imaging Archive. http://doi.org/10.7937/K9/TCIA.2016.V6PBVTDR

`PerturbationClustering` *Perturbation clustering*

Description

Perform subtyping using one type of high-dimensional data

Usage

```
PerturbationClustering(data, kMax = 5, verbose = T, ncore = 2,
  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 or data frame. The rows represent items while the columns represent features.
`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 `2`.
`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.
`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 function to reduce dimension of input `data` before clustering. By default `prcomp` will be used with `rank.=min(nrow(data), 200)`. If declared, this function receives `data` as the parameter and must return a numeric matrix.
`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 implements the Perturbation Clustering algorithm of Nguyen, et al (2017). It aims to determine the optimum cluster number and location of each sample in the clusters in an unsupervised analysis.

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

1. T Nguyen, R Tagett, D Diaz, S Draghici. A novel method for data integration and disease subtyping. Genome Research, 27(12):2025-2039, 2017.

2. T. Nguyen, "Horizontal and vertical integration of bio-molecular data", PhD thesis, Wayne State University, 2017.

See Also

kmeans, pam

Examples

```
# Load the dataset AML2004
data(AML2004)

# Perform the clustering
result <- PerturbationClustering(data = AML2004$Gene)

# Plot the result
condition = seq(unique(AML2004$Group[, 2]))
names(condition) <- unique(AML2004$Group[, 2])
plot(
    prcomp(AML2004$Gene)$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 = AML2004$Gene,
    clusteringMethod = "kmeans",
    clusteringOptions = list(
        iter.max = 500,
        nstart = 50
    )
)
```

```
# Change to use pam
result <- PerturbationClustering(data = AML2004$Gene, clusteringMethod = "pam")

# Change to use hclust
result <- PerturbationClustering(data = AML2004$Gene, clusteringMethod = "hclust")

# Pass a user-defined clustering algorithm
result <- PerturbationClustering(data = AML2004$Gene, clusteringFunction = function(data, k){
    # this function must return a vector of cluster
    kmeans(x = data, centers = k, nstart = k*10, iter.max = 2000)$cluster
})

# Use noise as the perturb method
result <- PerturbationClustering(data = AML2004$Gene,
                                 perturbMethod = "noise",
                                 perturbOptions = list(noise = 0.3))
# or
result <- PerturbationClustering(data = AML2004$Gene,
                                 perturbMethod = "noise",
                                 perturbOptions = list(noisePercent = 10))

# Change to use subsampling
result <- PerturbationClustering(data = AML2004$Gene,
                                 perturbMethod = "subsampling",
                                 perturbOptions = list(percent = 90))

# Users can pass their own perturb method
result <- PerturbationClustering(data = AML2004$Gene, perturbFunction = function(data){
   rowNum <- nrow(data)
   colNum <- ncol(data)
   epsilon <-
       matrix(
           data = rnorm(rowNum * colNum, mean = 0, sd = 1.234),
           nrow = rowNum,
           ncol = colNum
       )

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

SubtypingOmicsData *Subtyping multi-omics data*

Description

Perform subtyping using multiple types of data

Usage

```
SubtypingOmicsData(dataList, kMax = 5, agreementCutoff = 0.5,
  verbose = T, ...)
```

Arguments

dataList	a list of data matrices or data frames. 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.
kMax	the maximum number of clusters. Default value is 5.
agreementCutoff	agreement threshold to be considered consistent. Default value is 0.5.
verbose	set it to TRUE of 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 Perturbaion clustering algorithm of Nguyen, et al (2017). 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

1. T Nguyen, R Tagett, D Diaz, S Draghici. A novel method for data integration and disease subtyping. Genome Research, 27(12):2025-2039, 2017.

2. T. Nguyen, "Horizontal and vertical integration of bio-molecular data", PhD thesis, Wayne State University, 2017.

See Also

PerturbationClustering

Examples

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

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

# Change Pertubation 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 = ""
    )
)
legend(
    "bottomleft",
    fill = colors,
    legend = paste(
        "Group ",
        levels(factor(cluster2)),": ", table(cluster2)[levels(factor(cluster2))],
        sep =""
    )
)
```

Index