Package ‘clustering.sc.dp’

February 10, 2023

Type Package

Title Optimal Distance-Based Clustering for Multidimensional Data with Sequential Constraint

Version 1.1

Date 2023-02-03

Author Tibor Szkaliczki [aut, cre],
J. Song [ctb]

Maintainer Tibor Szkaliczki <szkaliczki.tibor@sztaki.hu>

Depends R (>= 2.10.0)

Description A dynamic programming algorithm for optimal clustering multidimensional data with sequential constraint. The algorithm minimizes the sum of squares of within-cluster distances. The sequential constraint allows only subsequent items of the input data to form a cluster. The sequential constraint is typically required in clustering data streams or items with time stamps such as video frames, GPS signals of a vehicle, movement data of a person, e-pen data, etc. The algorithm represents an extension of 'Ckmeans.1d.dp' to multiple dimensional spaces. Similarly to the one-dimensional case, the algorithm guarantees optimality and repeatability of clustering. Method clustering.sc.dp() can find the optimal clustering if the number of clusters is known. Otherwise, methods findwithinss.sc.dp() and backtracking.sc.dp() can be used. See Szkaliczki, T. (2016) ``clustering.sc.dp: Optimal Clustering with Sequential Constraint by Using Dynamic Programming'' <doi:10.32614/RJ-2016-022> for more information.

License LGPL (>= 3)

NeedsCompilation yes

Repository CRAN

Date/Publication 2023-02-10 09:30:11 UTC

R topics documented:

`backtracking.sc.dp` *Backtracking Clustering for a Specific Cluster Number*

Description

Creates clustering for `k` number of clusters by using the backtrack data produced by `findwithinss.sc.dp()`.

Usage

```
backtracking.sc.dp(x, k, backtrack)
```

Arguments

`x`	a multi-dimensional array containing input data to be clustered
`k`	the number of clusters
`backtrack`	the backtrack data

Details

If the number of clusters is unknown `findwithinss.sc.dp()` followed by `backtracking.sc.dp()` can be used for performing clustering. If only subsequent elements of the input data may form a cluster method `findwithinss.sc.dp()` calculates the exact minimum of the sum of squares of within-cluster distances (*withinss*) from each element to its corresponding cluster centre (mean) for different cluster numbers. The user may analyse the *withinss* in order to select the proper number of clusters. In this case, it is enough to run method `backtracking.sc.dp()` only once. Another option is to run `findwithinss.sc.dp()` once, repeat the `backtracking.sc.dp()` step for a range of potential cluster numbers and then the user may evaluate the optimal solutions created for different number of clusters. This requires much less time than repeating the whole clustering algorithm for the different cluster numbers.

Value

An object of class '`clustering.sc.dp`' which has a print method and is a list with components:

`cluster`	A vector of integers (`1:k`) indicating the cluster to which each point is allocated.
`centers`	A matrix whose rows represent cluster centres.
`withinss`	The within-cluster sum of squares for each cluster.
`size`	The number of points in each cluster.

Author(s)

Tibor Szkaliczki <`szkaliczki.tibor@sztaki.hu`>

See Also

`findwithinss.sc.dp`, `clustering.sc.dp`

Examples

```
# Example: clustering data generated from a random walk with small withinss
x<-matrix(, nrow = 100, ncol = 2)
x[1,]<-c(0,0)
for(i in 2:100) {
  x[i,1]<-x[i-1,1] + rnorm(1,0,0.1)
  x[i,2]<-x[i-1,2] + rnorm(1,0,0.1)
}
k<-10
r<-findwithinss.sc.dp(x,k)

# select the first cluster number where withinss drops below a threshold
thres <- 5.0
k_th <- 1;
while(r$twithinss[k_th] > thres & k_th < k) {
    k_th <- k_th + 1
}

# backtrack
result<-backtracking.sc.dp(x,k_th, r$backtrack)
plot(x, type = 'b', col = result$cluster)
points(result$centers, pch = 24, bg = (1:k_th))
```

`clustering.sc.dp` *Optimal Clustering Multidimensional Data with Sequential Constraint by Dynamic Programming*

Description

Perform optimal clustering on multidimensional data with sequential constraint (i.e. only subsequent elements of the input may form a cluster).

Usage

```
clustering.sc.dp(x, k)
```

Arguments

`x`	a multi-dimensional array containing input data to be clustered
`k`	the number of clusters

Details

The ’`clustering.sc.dp`’ algorithm (Szkaliczki, 2016) groups multidimensional data given by `x` into `k` clusters with sequential constraint by dynamic programming. It generalises the one-dimensional ’`Ckmeans.1d.dp`’ algorithm (Wang and Song, 2011) to multidimensional data. If only subsequent elements of the input data may form a cluster the algorithm guarantees the optimality of clustering – the sum of squares of within-cluster distances (*withinss*) from each element to its corresponding cluster centre (mean) is always the minimum. The sequential constraint is typically required in clustering datastreams or items with time stamps such as video frames, GPS signals of vehicles or movement data of persons etc. The run time of the algorithm is $O($ k d n^2) where k, d and n gives the number of clusters, the dimensions of the elements and the number of elements, respectively.

Value

An object of class 'clustering.sc.dp' which has a print method and is a list with components:

cluster a vector of cluster indices assigned to each element in x. Each cluster is indexed by an integer from 1 to k

centers a matrix whose rows represent cluster centres

withinss the within-cluster sum of squares for each cluster

size a vector of the number of points in each cluster

Author(s)

Tibor Szkaliczki <szkaliczki.tibor@sztaki.hu>

References

Szkaliczki, T. (2016) "clustering.sc.dp: Optimal Clustering with Sequential Constraint by Using Dynamic Programming" <doi: 10.32614/RJ-2016-022> Wang, H. and Song, M. (2011) "Ckmeans.1d.dp: optimal k-means clustering in one dimension by dynamic programming" <doi: 10.32614/RJ-2011-015>

Examples

```
# Example: clustering data generated from a random walk
x<-matrix(, nrow = 100, ncol = 2)
x[1,]<-c(0,0)
for(i in 2:100) {
  x[i,1]<-x[i-1,1] + rnorm(1,0,0.1)
  x[i,2]<-x[i-1,2] + rnorm(1,0,0.1)
}
k<-2
result<-clustering.sc.dp(x,k)
plot(x, type = 'b', col = result$cluster)
points(result$centers, pch = 24, bg = (1:k))
```

findwithinss.sc.dp *Finding Optimal Withinss in Clustering Multidimensional Data with Sequential Constraint by Dynamic Programming*

Description

Performs the main step of clustering multidimensional data with sequential constraint by a dynamic programming approach guaranteeing optimality. It returns the minimum *withinss* for each number of clusters less than or equal to `k` and backtracking data that can be used to find quickly the optimal clustering for a specific cluster number. This function was created in order to support the case when the number of clusters is unknown in advance.

Usage

```
findwithinss.sc.dp(x, k)
```

Arguments

`x`	a multi-dimensional array containing input data to be clustered
`k`	the maximal number of clusters, the output will be generated for cluster numbers between 1 and `k`

Details

Method `clustering.sc.dp()` is split into two methods (`findwithinss.sc.dp()` and `backtracking.sc.dp()`) in order to support the case when the number of clusters is not known in advance. Method `findwithinss.sc.dp()` returns the minimal sum of squares of within-cluster distances (*withinss*) for each number of clusters less than or equal to `k` and the backtrack data which can be used to quickly determine the optimal clustering for a specific cluster number. The returned *withinss* are guaranteed to be optimal among the solutions where only subsequent items form a cluster.

The outputs of the method can be used to select the proper number of clusters. The user may analyse the *withinss* in order to select the proper number of clusters. Another option is to run `findwithinss.sc.dp()` once, repeat the `backtracking.sc.dp()` step for a range of potential cluster numbers and then the user may evaluate the optimal solutions created for different number of clusters. This requires much less time than repeating the whole clustering algorithm.

Value

A list with components:

`twithinss`	a vector of total within-cluster sums of the optimal clusterings for each number of clusters less than or equal to `k`.
`backtrack`	backtrack data used by `backtracking.sc.dp()`.

Author(s)

Tibor Szkaliczki <`szkaliczki.tibor@sztaki.hu`>

See Also

`clustering.sc.dp`, `backtracking.sc.dp`

Examples

```
# Example: clustering data generated from a random walk with small withinss
x<-matrix(, nrow = 100, ncol = 2)
x[1,]<-c(0,0)
for(i in 2:100) {
  x[i,1]<-x[i-1,1] + rnorm(1,0,0.1)
  x[i,2]<-x[i-1,2] + rnorm(1,0,0.1)
}
k<-10
r<-findwithinss.sc.dp(x,k)

# select the first cluster number where withinss drops below a threshold
thres <- 5.0
k_th <- 1;
while(r$twithinss[k_th] > thres & k_th < k) {
    k_th <- k_th + 1
}

# backtrack
result<-backtracking.sc.dp(x,k_th, r$backtrack)
plot(x, type = 'b', col = result$cluster)
points(result$centers, pch = 24, bg = (1:k_th))
```

---

print.clustering.sc.dp

Print the result returned by calling clustering.sc.dp

Description

Print the result returned by calling clustering.sc.dp

Usage

```
## S3 method for class 'clustering.sc.dp'
print(x, ...)
```

Arguments

x	object returned by calling clustering.sc.dp()
...	ignored arguments

Value

An object of class ’clustering.sc.dp’ which has a print method and is a list with components:

cluster	A vector of integers (1:k) indicating the cluster to which each point is allocated.
centers	A matrix whose rows represent cluster centres.
withinss	The within-cluster sum of squares for each cluster.
size	The number of points in each cluster.

Author(s)

Tibor Szkaliczki <szkaliczki.tibor@sztaki.hu>

Examples

```
# Example: clustering data generated from a random walk
x<-matrix(, nrow = 100, ncol = 2)
x[1,]<-c(0,0)
for(i in 2:100) {
  x[i,1]<-x[i-1,1] + rnorm(1,0,0.1)
  x[i,2]<-x[i-1,2] + rnorm(1,0,0.1)
}
result<-clustering.sc.dp(x,2)
print(result)
```

Index