Package ‘birch’

February 15, 2013

Type Package

Depends R (>= 2.10), ellipse

Suggests MASS

Title Dealing with very large datasets using BIRCH

Version 1.2-3

Date 2012-05-03

Author Lysiane Charest, Justin Harrington, Matias Salibian-Barrera

Maintainer Lysiane Charest <lysiane.charest@hec.ca>

Description birch is an implementation of the algorithms described in Zhang et al (1997), and provides functions for creating CF-trees, along with algorithms for dealing with some combinatorial problems, such as covMcd and ltsReg. It is very well suited for dealing with very large data sets, and does not require that the data can fit in physical memory.

License GPL

Repository CRAN

Date/Publication 2012-05-04 05:24:11

NeedsCompilation yes

R topics documented:

`birch-package` *Working with very large data sets using BIRCH.*

Description

The functions in this package are designed for working with very large data sets by pre-processing the data set with an algorithm called BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies), which transforms the data set into compact, locally similar subclusters, each with summary statistics attached (called clustering features). Then, instead of using the full data set, these summary statistics can be used.

This approach is most advangeous in two situations: when the data cannot be loaded into memory due to its size; and/or when some form of combinatorial optimization is required and the size of the solution space makes finding global maximums/minimums difficult.

A complete explanation of this package is given in *Harrington and Salibian-Barrera (2008)*, and discussion of the underlying algorithms can be found in *Harrington and Salibian-Barrera (2010)*.

Documentation for developers can be found in the doc directory of this package. Also, the source code contains dOxygen tags for further information.

Details

The main function is

`birch`	takes a data set (an R object, text file, etc), and creates a birch object

Various generic methods are present, including

```
print
summary
plot
```

Methods for estimating the correlation matrix are also available:

```
spearman
kendall
```

Finally, some combinatorial-style problems have been implemented. These include:

covMcd.birch	Minimum Covariance Determinant (robust estimator for location and dispersion)
lts.birch	Least Trimmed Squares (robust regression estimator)
rlga.birch	Robust Linear Grouping Analysis (robust clustering about hyperplanes)
kmeans.birch	k-means

Author(s)

Lysiane Charest <lysiane.charest@hec.ca>, Justin Harrington <harringt@stat.ubc.ca> and Matias Salibian-Barrera <matias@stat.ubc.ca>

References

Harrington, J. and Salibian-Barrera, M. (2010), “Finding Approximate Solutions to Combinatorial Problems with Very Large Datasets using BIRCH”, *Computational Statistics and Data Analysis* 54, 655-667.

Harrington, J. and Salibian-Barrera, M. (2008), “birch: Working with very large data sets”, working paper.

See Also

birch, covMcd.birch, lts.birch, rlga.birch, plot.birch, kendall, spearman.

---

birch *Create a birch object*

Description

This function creates a birch object using the algorithm BIRCH.

Usage

```
birch(x, radius, compact=radius, keeptree=FALSE, columns = NULL, ...)
birch.addToTree(x, birchObject, updateDIM = TRUE, ...)
birch.getTree(birchObject)
birch.killTree(birchObject)
```

Arguments

x
: a numeric matrix of at least two columns, a file name or a connection that is compatible with read.table.

radius
: the closeness criterion

compact
: the compactness criterion

keeptree
: A Boolean, whether to keep the CF tree in memory.

`columns` the choice of variables to use if `x` is a file name or a connection (default is all variables)

`...` Arguments to be passed to `read.table` for loading a file or connection

`updateDIM` Update the dimension of the object? Defaults to TRUE (which is desirable!).

`birchObject` The output from birch.

Details

This function creates a CF-Tree not unlike that described in *Zhang et al. (1997)*, and used in *Harrington and Salibian-Barrera (2010).* A complete explanation of this package is given in *Harrington and Salibian-Barrera (2008).*

A full tree structure is used, as is the splitting of nodes (as described in the original article). However, the ‘Merging Refinement’ as described on page 149 is not currently implemented. The automatic rebuilding based on page size is not implemented.

The argument `keeptree` allows for the tree to be kept in memory after the initial processing has been completed. This allows for additional information to be added at a latter stage with `birch.addToTree`, without needing to process the whole data set a second time. However, it should be noted that the CF data (the summary statistics of each subcluster, as used by the subsequent algorithms) is not returned , and so this command should be followed by `birch.getTree` command in order to ensure that the correct information is used. In other words,

```
## Create the tree
myobject <- birch(x, 0.1, keeptree=TRUE)

## myobject has no information - let’s get some
myobject <- birch.getTree(myobject)

## add some data
birch.addToTree(y, mybirch)

## myobject is now out of date! Until
mybirch <- birch.getData(mybirch)
```

A birch object without summary information is referred to as a “birch skeleton”. Most algorithms will check for the presence of summary data, and request the information if required, but it is better for the user to request the data directly.

The birch object produced by the algorithm has a number of compatible generic methods, as well as clustering and robustness algorithms. These are given in the ‘See Also’ section.

The selection of the parameters for forming the birch object is left up to the user, though some guidance is given in *Harrington and Salibian-Barrera (2010).* One consideration is the purpose - for example, if simple summary statistics are required (means, covariances etc), then a large compactness and radius can be selected, as granularity is not required. In the case of clustering and robust methods when a refinement step is being done after the birch algorithm is completed, then once again the radius and compactness can be larger. If there is no refinement step, however, then the selection of these parameters is much more important.

Value

For birch and birch.getTree, a list for each subcluster containing

N	the number of observations in the subcluster
sumXi	the linear sum of the observations in the subcluster
sumXisq	the sum of squares of the observations in the subcluster
members	a list, each element containing a observation index (membership vector) for the subcluster

Both of birch.addToTree and birch.killTree return NULL.

Note

In this implementation, a limit of matrices with 30 columns has been set. It is the intention that this limit will be eventually removed in future versions of the package. If the user desires, this limit can be increased in the file 'll.h'. Other than the matrix column limit, there is no practical limit (excluding those imposed by the operating system and R) for how many observations can be placed in a birch object. In particular, since we retain the observation numbers belonging to each subcluster, the maximum length of a vector containing integers will determine how large the birch tree can get.

Some information for developers is provided in the doc directory of this package.

Author(s)

Justin Harrington <harringt@stat.ubc.ca> and Matias Salibian-Barrera <matias@stat.ubc.ca>

References

Harrington, J. and Salibian-Barrera, M. (2010), "Finding Approximate Solutions to Combinatorial Problems with Very Large Datasets using BIRCH", *Computational Statistics and Data Analysis* 54, 655-667.

Harrington, J. and Salibian-Barrera, M. (2008), "birch: Working with very large data sets", working paper.

Zhang, T. and Ramakrishnan, R. and Livny, M. (1997) "BIRCH: A New Data Clustering Algorithm and Its Applications", *Data Mining and Knowledge Discovery* **1**, 141–182.

See Also

lts.birch, rlga.birch, covMcd.birch, kmeans.birch, lga.birch, and plot.birch which also contains the other generic methods

Examples

```
## an example
## Create a data set
library(MASS)
set.seed(1234)
x <- mvrnorm(1e5, mu=rep(0,5), Sigma=diag(1,5))
```

```
x <- rbind(x, mvrnorm(1e5, mu=rep(10,5), Sigma=diag(0.1,5)+0.9))

## Create birch object
birchObj <- birch(x, 5, keeptree = TRUE)

## To load directly from a file or connection
## Not run: birchObj <- birch("myfile.csv", 1, sep=",", header=TRUE)
## Not run: birchObj <- birch("http://www.dot.com/myfile.csv", 1, sep=",",
header=TRUE)
## End(Not run)

## Leaving a tree in memory
birchObj <- birch(x, 5, keeptree=TRUE)
birch.addToTree(x, birchObj)
birchObj <- birch.getTree(birchObj)
## And don't forget to killTree...
#birch.killTree(birchObj)
```

birchObj *A birch object for demonstration purposes*

Description

A birch object created from simulated data for the purposes of demonstration

Usage

```
data(birchObj)
```

Format

A birch object with 318 subclusters. The object contains a list of

- `sumXi` a numeric matrix containing the column sum of the observations in each subcluster.
- `sumXisq` a numeric array containing the sum-of-squares of the observations in each subcluster.
- `N` a numeric vector containing the number of observations in each subcluster.
- `members` a list, each element of which contains a vector of the observations numbers present in that subcluster.

Details

This birch object was created from a simulated data set of two multivariate normals. Its primary use is for demonstrating functions in the help files. The sequence of commands for creating the object was as follows:

```
library(MASS) ## for mvrnorm
library(birch)
```

```
    ## Create data set
    set.seed(1234)
    x <- mvrnorm(1.5e5, mu=rep(0,5), Sigma=diag(1,5))
    x <- rbind(x, mvrnorm(1e5, mu=rep(10,5),
                          Sigma=diag(0.1,5)+0.9))

    ## Create birch object (without tree), and save it
    birchObj <- birch(x, 5)
    save(birchObj, file="birchObj.rda")
```

References

Harrington, J. and Salibian-Barrera, M. (2010), “Finding Approximate Solutions to Combinatorial Problems with Very Large Datasets using BIRCH”, *Computational Statistics and Data Analysis* 54, 655-667.

Harrington, J. and Salibian-Barrera, M. (2008), “birch: Working with very large data sets”, working paper.

Examples

```
data(birchObj)
```

---

covMcd.birch *Finding the Minimum Covariance Determinant using BIRCH*

Description

A function that uses a birch object to find an approximate solution to the MCD problem. The goal is to find a subset of size $alpha \times n$ with the smallest determinant of sample covariance.

Usage

```
covMcd.birch(birchObject, alpha=0.5, nsamp=100)
covMcdBirch.refinement(covOut, x, alpha=0.5)
```

Arguments

birchObject	an object created by the function birch.
alpha	numeric parameter controlling the size of the subsets over which the determinant is minimized, i.e., alpha*n observations are used for computing the determinant. Allowed values are between 0.5 and 1 and the default is 0.5.
nsamp	number of subsets used for initial estimates
covOut	the output from covMcd.birch
x	a data set on which to perform a set of concentration steps.

Details

The algorithm is similar to `covMcd` from the robustbase package as described in *Rousseeuw and Van Driessen (1999)*, except it uses a birch object instead. The advantage of this approach is that it does not require the full data set to be held in memory and the solution space is smaller. Further details can be found in *Harrington and Salibian-Barrera (2010)* and *Harrington and Salibian-Barrera (2008)*.

If further accuracy is desired, then an additional "refinement" step can be done, which involves using the birch solution as an initial estimate for one set of concentration steps, this time using the whole data set (rather than the birch object). However, if birch has been used because the whole data set cannot fit in memory, then this extra step is not an option.

A summary method is available for the output of this command.

Value

For `covMcd.birch`, returns a list containing:

`zbar`	estimate of location
`Sz`	estimate of covariance
`Det`	the MCD
`best`	A list containing a vector of which subclusters make up the clustering (sub) and a vector with the underlying observations that make up the clusters (obs)

Note

In order for this algorithm to produce meaningful results, the number of subclusters in the birch object should number in the hundreds, and even better, thousands.

Author(s)

Justin Harrington <`harringt@stat.ubc.ca`> and Matias Salibian-Barrera <`matias@stat.ubc.ca`>

References

Harrington, J. and Salibian-Barrera, M. (2010), "Finding Approximate Solutions to Combinatorial Problems with Very Large Datasets using BIRCH", *Computational Statistics and Data Analysis* 54, 655-667 .

Harrington, J. and Salibian-Barrera, M. (2008), "birch: Working with very large data sets", working paper.

Rousseeuw, P.J. and Van Driessen, K. (1999) "A Fast Algorithm for the Minimum Covariance Determinant Estimator", *Technometrics* **41**, 212–223.

See Also

`birch`, and the original algorithm `covMcd`

Examples

```
data(birchObj)
covOut <- covMcd.birch(birchObj, 0.5)
summary(covOut)

## If the original data set was available
## Not run: refOut <- covMcdBirch.refinement(covOut, x, 0.5)
```

`dist.birch` *Create a dissimilarity matrix from a birch object*

Description

This function computes and returns the distance matrix computed by passing the centers of the subclusters in the birch object to `dist`.

Usage

```
dist.birch(birchObject, ...)
```

Arguments

`birchObject`	An object created by the function `birch`.
`...`	Further arguments passed to `dist`.

Details

This function is a wrapper for the `dist` function, for application on the centers of birch objects. It is provided as a convenient mechanism for provided estimates of initial centers for the `kmeans.birch` function. An example of this is provided in the section below.

Value

An object of the class “dist”. See `dist` for more details.

Author(s)

Justin Harrington <`harringt@stat.ubc.ca`> and Matias Salibian-Barrera <`matias@stat.ubc.ca`>

References

Harrington, J. and Salibian-Barrera, M. (2010), “Finding Approximate Solutions to Combinatorial Problems with Very Large Datasets using BIRCH”, *Computational Statistics and Data Analysis* 54, 655-667.

Harrington, J. and Salibian-Barrera, M. (2008), “birch: Working with very large data sets”, working paper.

See Also

birch, dist, kmeans.birch

Examples

```
## Load a demo birch Object
data(birchObj)

## Calculate the distances, perform hclust, and then
## find out which clustering corresponds to 2 groups
bDist <- dist.birch(birchObj)
hc <- hclust(bDist)
clusters <- cutree(hc, 2)


## Calculate centers based on these clusters, and cluster with them
centers <- rbind(summary(birchObj[clusters == 1,])$mean,
summary(birchObj[clusters == 2,])$mean)
kOut <- kmeans.birch(birchObj, centers)
```

generic methods   *Generic methods for birch objects*

Description

A description of the generic methods that are compatible with birch objects.

Usage

```
## S3 method for class 'birch'
print(x, ...)
## S3 method for class 'birch'
summary(object, ...)
## S3 method for class 'birch'
x[i, j]
## S3 method for class 'birch'
dim(x)
## S3 method for class 'birch'
length(x)
## S3 method for class 'birch'
cbind(scalar, x)
## S3 method for class 'birch'
plot(x, centers=FALSE, ...)
## S3 method for class 'birch'
pairs(x, centers=FALSE, ...)
## S3 method for class 'birch'
points(x, ...)
## S3 method for class 'birch'
lines(x, col, ...)
```

Arguments

`x, object`	A birch object
`i, j`	Indices specifying which subclusters and dimensions to extract.
`scalar`	A scalar to add as a column to the birch object.
`centers`	A Boolean. If TRUE, then just plots the centers of the subclusters. See the details for more information.
`col`	Color argument for the `lines` method.
`...`	Pass additional information to the respective methods.

Details

Most of these methods do not require further explanation, as their behaviour is fairly standard. The plot and pairs methods, however, do something a little different.

While they accept all the usual arguments as plot (the arguments are simply passed on to e.g. `plot.xy`), instead of simply producing points/lines of the values, which is not possible with a birch object, these methods produce ellipses for each subcluster, based on its covariance structure, and making the assumption that the underlying data is multivariate normal. The ellipses are then formed (using the ellipse package) at the 95% confidence level.

However, if there are a lot of subclusters, then this plot gets "messy", and so it is possible to plot just the centers of each subcluster by setting the `centers` argument to true. The `points` and `lines` methods add the centers and ellipses respectively to the existing plot.

Value

The `dim` method returns a vector of length three containing the total number of observations in the tree, the number of columns of the data set, and the number of subclusters. The `length` method just returns the number of subclusters. Note that both of these commands operate on the object given as an argument, and do not check if the tree has been updated (e.g. by adding data with the `birch.addToTree` command).

The `summary` method returns the mean and covariance of the whole birch object - equivalently (and, as it turns out, identically) the whole underlying data set.

The '[i,j]' method selects the i-th subcluster and the j-th column of the birch object, and has functionality similar to that of the usual indexing of matrices. Similarly, the `cbind` method effectively inserts a column containing a scalar (recycled to the appropriate length) in front of the data set. See note for a caveat with these methods.

The `print`, `plot`, `pairs`, `points` and `lines` methods return nothing.

Note

Care should be taken when using the indexing, as

```
birch(z, 1)[,1:2]
```

will not produce the same result as

```
birch(z[,1:2], 1)
```

Similarly, there are no guarantees for `cbind`. Moreover, the use of indexing necessitates that the birch object was made with keeptree = TRUE.

Care should also be taken when using the affectation when object birch was built with keeptree = TRUE, as

```
 newBirch <- birchObj
```

will only create a new variable (newBirch) pointing to the same object as birchObj does. To create a copy of a birch object, use

```
 newBirch <- birchObj[]
```

Author(s)

Lysiane Charest <`lysiane.charest@hec.ca`>, Justin Harrington <`harringt@stat.ubc.ca`> and Matias Salibian-Barrera <`matias@stat.ubc.ca`>

References

Harrington, J. and Salibian-Barrera, M. (2010), “Finding Approximate Solutions to Combinatorial Problems with Very Large Datasets using BIRCH”, *Computational Statistics and Data Analysis* 54, 655-667.

Harrington, J. and Salibian-Barrera, M. (2008), “birch: Working with very large data sets”, working paper.

See Also

`birch`

Examples

```
## Load a demo birch Object
data(birchObj)

dim(birchObj)
newObj <- cbind(1, birchObj)
dim(newObj)

dim(birchObj[-1, -2])
```

kendall *Estimating Kendall's matrix of correlation using BIRCH*

Description

This function builds an estimate of the matrix of Kendall's correlation from a birch object or from a dataset. In the latter case, it uses the birch algorithm to pre-process the data.

The Kendall's tau is used as the correlation measure.

Usage

```
kendall(x, radius, compact = radius, columns = NULL,  estimate , ... )
```

Arguments

`x`	a birch object, a numeric matrix of at least two columns, a file name or a connection that is compatible with `read.table`.
`radius`	the closeness criterion (unused if x is a birch object)
`compact`	the compactness criterion (unused if x is a birch object)
`columns`	the choice of variables to use if x is a file name or a connection (default is all variables)
`estimate`	the choice of corrections to use for the estimation ("Student", "Woodbury" or "Stuart")
`...`	arguments to be passed to `read.table` for loading a file or connection

Details

There are two versions of the function. When x is a matrix, a file name or a connection, the first version is used. When x is a birch object, the second version is used.

First version: For each pair of variables, one birch object is created using the radius and compactness criteria. Estimation is based on the summary statistics resulting from the usage of birch algorithm that regroups data in compact clusters. The advantage of this approach is that it does not require the full data set to be held in memory. Further details can be found in *Charest (2012).* The result of the algorithm is a correlation matrix with estimated correlations for each pairs of variables.

Second version: For each pair of variables, the correlation is estimated based on their own summary statistics.

Estimates:

"Student", "student", '1' Tau-b

"Woodbury", "woodbury", '2' Basic formula without correction (Kendall (1945), Woodbury (1945))

"Stuart", "stuart", '3' Correction to extend the boundaries to {-1,+1} (Stuart (1953))

Value

Returns a list with components:

`nClus` matrix showing the number of clusters used for the estimation of each coefficient

`estimate[,,i]` one matrix of correlation estimate for each selected correction

Author(s)

Lysiane Charest <`lysiane.charest@hec.ca`>

References

Charest (2012)

See Also

`birch`, `spearman`

Examples

```
## Create a data set
library(MASS)
set.seed(1234)
x <- mvrnorm(1e5, mu=rep(0,5), Sigma=diag(1,5))
x <- rbind(x, mvrnorm(1e5, mu=rep(10,5), Sigma=diag(0.1,5)+0.9))

##Create a csv for the example
write.csv(x,"csvExample.csv", row.names=FALSE)

## Create a birch object
birchObj <- birch(x, radius = 5, keeptree = TRUE)
birchObj <- birch.getTree(birchObj)

##With x: birch object.
##In this example, although the birch object was created with V1-V5,
##we only want the estimates for V2, V4 and V5
fromBirchObj <- kendall(birchObj[,c(2,4,5)], estimate = c("student", "woodbury", "stuart"))


##With x: dataset.
##In this example, we require estimates for every variable
fromDataset <- kendall(x, radius = 5, estimate = c("student", "woodbury", "stuart"))

##With x: file name or connection
##In this example, we require estimates for variables V2, V4 and V5 only
fromFile <- kendall("csvExample.csv", 1, sep=",", header=TRUE, radius = 5, columns = c(2,4,5),
estimate = c("student", "woodbury", "stuart"))
## Not run: fromConnection <- spearman("http://www.dot.com/myfile.csv", 1, sep=",",
header=TRUE, radius = 5, columns = c(2,4,5), estimate = c("student", "woodbury", "stuart"))
## End(Not run)
```

kmeans.birch *K-means with BIRCH*

Description

Perform the k-means clustering algorithm using a birch object.

Usage

```
kmeans.birch(birchObject, centers, nstart = 1)
```

Arguments

birchObject	An object created by the function birch.
centers	Either the number of clusters or a set of initial (distinct) cluster centres. If a number, a random set of (distinct) subclusters in 'birchObject' is chosen as the initial centres.
nstart	If 'centers' is a number, how many random sets should be chosen?

Details

The birch object given by 'birchObject' is clustered by the k-means method, adjusted for dealing with birch objects. The aim is to partition the subclusters into k groups such that the sum of squares of all the points in each subcluster to the assigned cluster centers is minimized.

The result should be approximately similar to that found by performing k-means on the original data set. However, this approximation depends on the "coarseness" of the underlying tree, and the size of the combinatorial problem. These aspect as discussed in detail in the references.

Value

Returns a list with components:

RSS	The total residual sum-of-squares of the clustering.
clust	A list containing a vector of which subclusters make up the clustering (sub) and a vector with the underlying observations that make up the clusters (obs)

Note

In order for this algorithm to produce meaningful results, the number of subclusters in the birch object should number in the hundreds, and even better, thousands.

Author(s)

Justin Harrington <harringt@stat.ubc.ca> and Matias Salibian-Barrera <matias@stat.ubc.ca>

References

Harrington, J. and Salibian-Barrera, M. (2010), “Finding Approximate Solutions to Combinatorial Problems with Very Large Datasets using BIRCH”, *Computational Statistics and Data Analysis* 54, 655-667 .

Harrington, J. and Salibian-Barrera, M. (2008), “birch: Working with very large data sets”, working paper.

See Also

`birch`, `dist.birch`

Examples

```
## Load a demo birch Object
data(birchObj)

## Perform k-means, specifying the number of clusters
kOut <- kmeans.birch(birchObj, 2, nstart=10)

## Perform k-means, specifying the initial cluster centers
## See dist.clust for one method of initial cluster centers
kOut <- kmeans.birch(birchObj, matrix(c(0,10), ncol=5, nrow=2))

## To plot using the birch object
plot(birchObj, col=kOut$clust$sub)

## To plot using the underlying data (if available)
## Not run: plot(x, col=kOut$clust$obs)
```

`lts.birch` *Finding the Least Trimmed Squares (LTS) regression estimate using BIRCH*

Description

This algorithm searches for the Least Trimmed Squares (LTS) solution using a BIRCH object.

Usage

```
lts.birch(birchObject, alpha=0.5, intercept = FALSE, nsamp=100)
ltsBirch.refinement(ltsOut, x, y, alpha=0.5, intercept = FALSE)
```

Arguments

`birchObject` an object created by the function `birch`. See details for information on how to specify the exploratory and response.

`alpha` numeric parameter controlling the size of the subsets over which the trimmed residuals are minimized, i.e., alpha*n observations are used when computing the trimmed residual sum of squares . Allowed values are between 0.5 and 1 and the default is 0.5.

`intercept` a Boolean - is there a intercept?

`nsamp` number of subsets used for initial estimates

`ltsOut` the output from `lts.birch`.

`x, y` a data set of explanatory and response variables on which to perform a set of concentration steps.

Details

The algorithm is very similar to the `ltsRef` function from the robustbase package from *Rousseeuw and Van Driessen (2006)*, except it uses a BIRCH object instead. A complete description is given in *Harrington and Salibian-Barrera (2010)* and *Harrington and Salibian-Barrera (2008)*

The algorithm assumes that the last column of the birch object contains the response variable, and that all the other columns are explanatories. While it is possible to select columns using the usual

```
[,j]
```

, it is recommended that the birch object be rebuilt from the underlying data set with just the explanatories and response variables selected.

If an intercept is required in the model, either the `intercept` argument can be set to true, or a column of ‘ones’ should be column-wise appended to the data (prior to building the birch object).

A summary method is available for the output of this command.

Value

Returns a list containing:

`best` A list containing a vector of which subclusters make up the clustering (sub) and a vector with the underlying observations that make up the clusters (obs).

`raw.coefficients`
the fitted LTS regression line.

`Resids` A list containing the sum of squared residuals for the best subset, as well as the sum of squared residuals for the whole data set (based on the LTS regression equation).

Note

In order for this algorithm to produce meaningful results, the number of subclusters in the birch object should number in the hundreds, and even better, thousands.

Author(s)

Justin Harrington <`harringt@stat.ubc.ca`> and Matias Salibian-Barrera <`matias@stat.ubc.ca`>

References

Harrington, J. and Salibian-Barrera, M. (2010), “Finding Approximate Solutions to Combinatorial Problems with Very Large Datasets using BIRCH”, *Computational Statistics and Data Analysis* 54, 655-667.

Harrington, J. and Salibian-Barrera, M. (2008), “birch: Working with very large data sets”, working paper.

Rousseeuw, P.J. and Van Driessen, K. (2006) “Computing LTS Regression for Large Data Sets”, *Data Mining and Knowledge Discovery* **12**, 29–45.

See Also

`birch`, and the original algorithm `ltsReg`

Examples

```
data(birchObj)
ltsOut <- lts.birch(birchObj, 0.5)
ltsOut2 <- lts.birch(birchObj, 0.5, intercept=TRUE)
summary(ltsOut2)

## If the original data set was available
## Not run: refOut <- ltsBirch.refinement(ltsOut2, x, y, 0.5, intercept=TRUE)
```

`rlga.birch` *Finding the Robust/Non-Robust LGA solution using BIRCH*

Description

Performs Linear Grouping Analysis (LGA) and Robust Linear Grouping Analysis (RLGA) using a BIRCH object.

Usage

```
lga.birch(birchObject, k, nsamp=100)
rlga.birch(birchObject, k, alpha=0.5, nsamp=100)
```

Arguments

`birchObject` an object created by the function `birch`.

`k` the number of clusters

`alpha` numeric parameter controlling the size of the subsets over which the orthogonal residuals are minimized, i.e., alpha*n observations are used when calculating orthogonal residuals in the hyperplane calculations. Allowed values are between 0.5 and 1 and the default is 0.5.

`nsamp` number of subsets used for initial estimates

Details

Robust Linear Grouping (*Garcia-Escudero et al, 2010*) is the robust implementation of LGA (*Van Aelst et al, 2006*), and is concerned with clustering around hyperplanes. The non-birch versions can be found in the package `lga`, which is also available on CRAN.

This algorithm is the equivalent design for use with a BIRCH object. For further details, please see *Harrington and Salibian-Barrera (2010).*

Value

Returns a list containing:

`clust`	A list containing a vector of which subclusters make up the clustering (sub) and a vector with the underlying observations that make up the clusters (obs). For the robust algorithm, a value of zero indicates it does not belong to the best h-subset.
`ROSS`	the residual sum of squares of orthogonal distances to the fitted hyperplanes based on the best data set.

Note

In order for this algorithm to produce meaningful results, the number of subclusters in the birch object should be in the hundreds, and even better, thousands.

Author(s)

Justin Harrington <`harringt@stat.ubc.ca`> and Matias Salibian-Barrera <`matias@stat.ubc.ca`>

References

Garcia-Escudero, L.A. and Gordaliza, A. and San Martin, R. and Van Aelst, S. and Zamar, R. (2007) “Robust Linear Clustering”, *Unpublished Manuscript*.

Harrington, J. and Salibian-Barrera, M. (2010), “Finding Approximate Solutions to Combinatorial Problems with Very Large Datasets using BIRCH”, *Computational Statistics and Data Analysis* 54, 655-667 .

Harrington, J. and Salibian-Barrera, M. (2008), “birch: Working with very large data sets”, working paper.

Van Aelst, S. and Wang, X. and Zamar, R.H. and Zhu, R. (2006) “Linear grouping using orthogonal regression”, *Computational Statistics \& Data Analysis* **50**, 1287–1312.

See Also

`birch`, and the non-birch algorithms `lga` and `rlga`

Examples

```
library(MASS) ## for mvrnorm
library(birch)

## Create new data set (that is more applicable to RLGA and LGA
set.seed(1234)
x <- mvrnorm(1e4, mu=rep(0,2), Sigma=diag(c(0.25,1),2))
x <- rbind(x, mvrnorm(1e4, mu=rep(10,2),
                      Sigma=diag(c(5,0.5),2)))

## Create birch object, and save it
birchObj <- birch(x, 0.5)
length(birchObj)

library(birch)
rlgaOut <- rlga.birch(birchObj, k=2, 0.5)
plot(birchObj, col=rlgaOut$clust$sub+1)

lgaOut <- lga.birch(birchObj, k=2)
plot(birchObj, col=lgaOut$clust$sub)
```

spearman *Estimating Spearman's matrix of correlation using BIRCH*

Description

This function builds an estimate of the matrix of Spearman's correlation from a birch object or from a dataset. In the latter case, it uses the birch algorithm to pre-process the data.

The Spearman's rho is used as the correlation measure.

Usage

```
spearman(x, radius, compact = radius, columns = NULL,  estimate , ... )
```

Arguments

`x`	a birch object, a numeric matrix of at least two columns, a file name or a connection that is compatible with `read.table`.
`radius`	the closeness criterion (unused if x is a birch object)
`compact`	the compactness criterion (unused if x is a birch object)
`columns`	the choice of variables to use if x is a file name or a connection (default is all variables)
`estimate`	the choice of corrections to use for the estimation ("Student" and "Woodbury")
`...`	arguments to be passed to `read.table` for loading a file or connection

Details

There are two versions of the function. When x is a matrix, a file name or a connection, the first version is used. When x is a birch object, the second version is used.

First version: For each pair of variables, one birch object is created using the radius and compactness criteria. Estimation is based on the summary statistics resulting from the usage of birch algorithm that regroups data in compact clusters. The advantage of this approach is that it does not require the full data set to be held in memory. Further details can be found in *Charest (2012).*

The result of the algorithm is a correlation matrix with estimated correlations for each pair of variables.

Second version: For each pair of variables, the correlation is estimated based on their own summary statistics.

Estimates:

"Student", "student", '1' Correction using the product-moment correlation with mid-ranks (Kendall (1945), Student (1921))

"Woodbury", "woodbury", '2' Basic formula without correction (Woodbury, 1945)

Value

Returns a list with components:

`nClus`	matrix showing the number of clusters used for the estimation of each coefficient
`estimate[,,i]`	one matrix of correlation estimate for each selected correction

Author(s)

Lysiane Charest <lysiane.charest@hec.ca>

References

Charest (2012)

See Also

birch, kendall

Examples

```
## Create a data set
library(MASS)
set.seed(1234)
x <- mvrnorm(1e5, mu=rep(0,5), Sigma=diag(1,5))
x <- rbind(x, mvrnorm(1e5, mu=rep(10,5), Sigma=diag(0.1,5)+0.9))

##Create a csv for the example
write.csv(x,"csvExample.csv", row.names=FALSE)

## Create a birch object
birchObj <- birch(x, radius = 5, keeptree = TRUE)
birchObj <- birch.getTree(birchObj)

##With x: birch object.
##In this example, although the birch object was created with V1-V5,
##we only want the estimates for V2, V4 and V5
fromBirchObj <- spearman(birchObj[,c(2,4,5)], estimate = c("student", "woodbury"))


##With x: dataset.
##In this example, we require estimates for every variable
fromDataset <- spearman(x, radius = 5, estimate = c("student", "woodbury"))

##With x: file name or connection
##In this example, we require estimates for variables V2, V4 and V5 only
fromFile <- spearman("csvExample.csv", 1, sep=",", header=TRUE, radius = 5, columns = c(2,4,5),
estimate = c("student", "woodbury"))
## Not run: fromConnection <- spearman("http://www.dot.com/myfile.csv", 1, sep=",",
header=TRUE, radius = 5, columns = c(2,4,5), estimate = c("student", "woodbury"))
## End(Not run)
```

Index