Package ‘Matrix.utils’

February 27, 2020

Title Data.frame-Like Operations on Sparse and Dense Matrix Objects

Version 0.9.8

Author Craig Varrichio <canthony427@gmail.com>

Maintainer Craig Varrichio <canthony427@gmail.com>

Description Implements data manipulation methods such as cast, aggregate, and merge/join for Matrix and matrix-like objects.

Depends R (>= 3.0.0), Matrix

Imports grr, methods, stats

Suggests testthat

License GPL-3

URL https://github.com/cvarrichio/Matrix.utils

LazyData true

RoxygenNote 7.0.2

NeedsCompilation no

Repository CRAN

Date/Publication 2020-02-26 23:40:06 UTC

R topics documented:

aggregate.Matrix *Compute summary statistics of a Matrix*

Description

Similar to `aggregate`. Splits the matrix into groups as specified by groupings, which can be one or more variables. Aggregation function will be applied to all columns in data, or as specified in formula. Warning: groupings will be made dense if it is sparse, though data will not.

Usage

```
## S3 method for class 'Matrix'
aggregate(x, groupings = NULL, form = NULL, fun = "sum", ...)
```

Arguments

`x`	a `Matrix` or matrix-like object
`groupings`	an object coercible to a group of factors defining the groups
`form`	`formula`
`fun`	character string specifying the name of aggregation function to be applied to all columns in data. Currently "sum", "count", and "mean" are supported.
`...`	arguments to be passed to or from methods. Currently ignored

Details

`aggregate.Matrix` uses its own implementations of functions and should be passed a string in the `fun` argument.

Value

A sparse `Matrix`. The rownames correspond to the values of the groupings or the interactions of groupings joined by a `_`.

There is an attribute `crosswalk` that includes the groupings as a data frame. This is necessary because it is not possible to include character or data frame groupings in a sparse Matrix. If needed, one can `cbind(attr(x,"crosswalk"),x)` to combine the groupings and the aggregates.

See Also

`summarise`

`summarise`

`aggregate`

Examples

```
skus<-Matrix(as.matrix(data.frame(
   orderNum=sample(1000,10000,TRUE),
   sku=sample(1000,10000,TRUE),
   amount=runif(10000))),sparse=TRUE)
#Calculate sums for each sku
a<-aggregate.Matrix(skus[,'amount'],skus[,'sku',drop=FALSE],fun='sum')
#Calculate counts for each sku
b<-aggregate.Matrix(skus[,'amount'],skus[,'sku',drop=FALSE],fun='count')
#Calculate mean for each sku
c<-aggregate.Matrix(skus[,'amount'],skus[,'sku',drop=FALSE],fun='mean')

m<-rsparsematrix(1000000,100,.001)
labels<-as.factor(sample(1e4,1e6,TRUE))
b<-aggregate.Matrix(m,labels)

## Not run:
orders<-data.frame(orderNum=as.factor(sample(1e6, 1e7, TRUE)),
   sku=as.factor(sample(1e3, 1e7, TRUE)),
   customer=as.factor(sample(1e4,1e7,TRUE)),
   state = sample(letters, 1e7, TRUE), amount=runif(1e7))
system.time(d<-aggregate.Matrix(orders[,'amount',drop=FALSE],orders$orderNum))
system.time(e<-aggregate.Matrix(orders[,'amount',drop=FALSE],orders[,c('customer','state')]))

## End(Not run)
```

`dMcast` *Casts or pivots a long* `data frame` *into a wide sparse matrix*

Description

Similar in function to `dcast`, but produces a sparse `Matrix` as an output. Sparse matrices are beneficial for this application because such outputs are often very wide and sparse. Conceptually similar to a `pivot` operation.

Usage

```
dMcast(
  data,
  formula,
  fun.aggregate = "sum",
  value.var = NULL,
  as.factors = FALSE,
  factor.nas = TRUE,
  drop.unused.levels = TRUE
)
```

Arguments

`data`	a data frame
`formula`	casting `formula`, see details for specifics.
`fun.aggregate`	name of aggregation function. Defaults to 'sum'
`value.var`	name of column that stores values to be aggregated numerics
`as.factors`	if TRUE, treat all columns as factors, including
`factor.nas`	if TRUE, treat factors with NAs as new levels. Otherwise, rows with NAs will receive zeroes in all columns for that factor
`drop.unused.levels`	should factors have unused levels dropped? Defaults to TRUE, in contrast to `model.matrix`

Details

Casting formulas are slightly different than those in `dcast` and follow the conventions of `model.matrix`. See `formula` for details. Briefly, the left hand side of the ~ will be used as the grouping criteria. This can either be a single variable, or a group of variables linked using `:`. The right hand side specifies what the columns will be. Unlike `dcast`, using the `+` operator will append the values for each variable as additional columns. This is useful for things such as one-hot encoding. Using `:` will combine the columns as interactions.

Value

a sparse `Matrix`

See Also

`cast`

`dcast`

Examples

```
#Classic air quality example
melt<-function(data,idColumns)
{
  cols<-setdiff(colnames(data),idColumns)
 results<-lapply(cols,function (x) cbind(data[,idColumns],variable=x,value=as.numeric(data[,x])))
  results<-Reduce(rbind,results)
}
names(airquality) <- tolower(names(airquality))
aqm <- melt(airquality, idColumns=c("month", "day"))
dMcast(aqm, month:day ~variable,fun.aggregate = 'mean',value.var='value')
dMcast(aqm, month ~ variable, fun.aggregate = 'mean',value.var='value')

#One hot encoding
#Preserving numerics
dMcast(warpbreaks,~.)
#Pivoting numerics as well
```

```
dMcast(warpbreaks,~.,as.factors=TRUE)

## Not run:
orders<-data.frame(orderNum=as.factor(sample(1e6, 1e7, TRUE)),
   sku=as.factor(sample(1e3, 1e7, TRUE)),
   customer=as.factor(sample(1e4,1e7,TRUE)),
   state = sample(letters, 1e7, TRUE),
   amount=runif(1e7))
# For simple aggregations resulting in small tables, dcast.data.table (and
   reshape2) will be faster
system.time(a<-dcast.data.table(as.data.table(orders),sku~state,sum,
   value.var = 'amount')) # .5 seconds
system.time(b<-reshape2::dcast(orders,sku~state,sum,
   value.var = 'amount')) # 2.61 seconds
system.time(c<-dMcast(orders,sku~state,
   value.var = 'amount')) # 8.66 seconds

# However, this situation changes as the result set becomes larger
system.time(a<-dcast.data.table(as.data.table(orders),customer~sku,sum,
   value.var = 'amount')) # 4.4 seconds
system.time(b<-reshape2::dcast(orders,customer~sku,sum,
   value.var = 'amount')) # 34.7 seconds
 system.time(c<-dMcast(orders,customer~sku,
   value.var = 'amount')) # 14.55 seconds

# More complicated:
system.time(a<-dcast.data.table(as.data.table(orders),customer~sku+state,sum,
   value.var = 'amount')) # 16.96 seconds, object size = 2084 Mb
system.time(b<-reshape2::dcast(orders,customer~sku+state,sum,
   value.var = 'amount')) # Does not return
system.time(c<-dMcast(orders,customer~sku:state,
   value.var = 'amount')) # 21.53 seconds, object size = 116.1 Mb

system.time(a<-dcast.data.table(as.data.table(orders),orderNum~sku,sum,
   value.var = 'amount')) # Does not return
system.time(c<-dMcast(orders,orderNum~sku,
   value.var = 'amount')) # 24.83 seconds, object size = 175Mb

system.time(c<-dMcast(orders,sku:state~customer,
   value.var = 'amount')) # 17.97 seconds, object size = 175Mb


## End(Not run)
```

`Matrix.utils` *Data.frame-Like Operations on Sparse and Dense* `Matrix` *Objects*

Description

Implements data manipulation methods such as cast, aggregate, and merge/join for `Matrix` and matrix-like objects.

`merge.Matrix` *Merges two Matrices or matrix-like objects*

Description

Implementation of `merge` for `Matrix`. By explicitly calling `merge.Matrix` it will also work for `matrix`, for `data.frame`, and `vector` objects as a much faster alternative to the built-in `merge`.

Usage

```
## S3 method for class 'Matrix'
merge(
  x,
  y,
  by.x,
  by.y,
  all.x = TRUE,
  all.y = TRUE,
  out.class = class(x)[1],
  fill.x = ifelse(is(x, "sparseMatrix"), FALSE, NA),
  fill.y = fill.x,
  ...
)

join.Matrix(
  x,
  y,
  by.x,
  by.y,
  all.x = TRUE,
  all.y = TRUE,
  out.class = class(x)[1],
  fill.x = ifelse(is(x, "sparseMatrix"), FALSE, NA),
  fill.y = fill.x,
  ...
)
```

Arguments

`x, y`	`Matrix` or matrix-like object
`by.x`	vector indicating the names to match from `Matrix` `x`
`by.y`	vector indicating the names to match from `Matrix` `y`
`all.x`	logical; if `TRUE`, then each value in `x` will be included even if it has no matching values in `y`
`all.y`	logical; if `TRUE`, then each value in `y` will be included even if it has no matching values in `x`

`out.class`	the class of the output object. Defaults to the class of x. Note that some output classes are not possible due to R coercion capabilities, such as converting a character matrix to a Matrix.
`fill.x, fill.y`	the value to put in merged columns where there is no match. Defaults to 0/FALSE for sparse matrices in order to preserve sparsity, NA for all other classes
`...`	arguments to be passed to or from methods. Currently ignored

Details

#' `all.x/all.y` correspond to the four types of database joins in the following way:

left `all.x=TRUE`, `all.y=FALSE`

right `all.x=FALSE`, `all.y=TRUE`

inner `all.x=FALSE`, `all.y=FALSE`

full `all.x=TRUE`, `all.y=TRUE`

Note that `NA` values will match other `NA` values.

Examples

```
orders<-Matrix(as.matrix(data.frame(orderNum=1:1000,
 customer=sample(100,1000,TRUE))))
 cancelledOrders<-Matrix(as.matrix(data.frame(orderNum=sample(1000,100),
 cancelled=1)))
skus<-Matrix(as.matrix(data.frame(orderNum=sample(1000,10000,TRUE),
sku=sample(1000,10000,TRUE), amount=runif(10000))))
a<-merge(orders,cancelledOrders,orders[,'orderNum'],cancelledOrders[,'orderNum'])
b<-merge(orders,cancelledOrders,orders[,'orderNum'],cancelledOrders[,'orderNum'],all.x=FALSE)
c<-merge(orders,skus,orders[,'orderNum'],skus[,'orderNum'])

#The above Matrices could be converted to matrices or data.frames and handled in other methods.
#However, this is not possible in the sparse case, which can be handled by this function:
sm<-cbind2(1:200000,rsparsematrix(200000,10000,density=.0001))
sm2<-cbind2(sample(1:200000,50000,TRUE),rsparsematrix(200000,10,density=.01))
sm3<-merge.Matrix(sm,sm2,by.x=sm[,1],by.y=sm2[,1])

 ## Not run:
#merge.Matrix can also handle many other data types, such as data frames, and is generally fast.
orders<-data.frame(orderNum=as.character(sample(1e5, 1e6, TRUE)),
   sku=sample(1e3, 1e6, TRUE),
   customer=sample(1e4,1e6,TRUE),stringsAsFactors=FALSE)
cancelledOrders<-data.frame(orderNum=as.character(sample(1e5,1e4)),
   cancelled=1,stringsAsFactors=FALSE)
system.time(a<-merge.Matrix(orders,cancelledOrders,orders[,'orderNum'],
   cancelledOrders[,'orderNum']))
system.time(b<-merge.data.frame(orders,cancelledOrders,all.x = TRUE,all.y=TRUE))
system.time(c<-dplyr::full_join(orders,cancelledOrders))
system.time({require(data.table);
d<-merge(data.table(orders),data.table(cancelledOrders),
   by='orderNum',all=TRUE,allow.cartesian=TRUE)})

orders<-data.frame(orderNum=sample(1e5, 1e6, TRUE), sku=sample(1e3, 1e6,
TRUE), customer=sample(1e4,1e6,TRUE),stringsAsFactors=FALSE)
cancelledOrders<-data.frame(orderNum=sample(1e5,1e4),cancelled=1,stringsAsFactors=FALSE)
system.time(b<-merge.Matrix(orders,cancelledOrders,orders[,'orderNum'],
cancelledOrders[,'orderNum']))
system.time(e<-dplyr::full_join(orders,cancelledOrders))
system.time({require(data.table);
 d<-merge(data.table(orders),data.table(cancelledOrders),
 by='orderNum',all=TRUE,allow.cartesian=TRUE)})

#In certain cases, merge.Matrix can be much faster than alternatives.
one<-as.character(1:1000000)
two<-as.character(sample(1:1000000,1e5,TRUE))
system.time(b<-merge.Matrix(one,two,one,two))
system.time(c<-dplyr::full_join(data.frame(key=one),data.frame(key=two)))
system.time({require(data.table);
 d<-merge(data.table(data.frame(key=one)),data.table(data.frame(key=two)),
 by='key',all=TRUE,allow.cartesian=TRUE)})

## End(Not run)
```

`rBind.fill` *Combine matrixes by row, fill in missing columns*

Description

rbinds a list of Matrix or matrix like objects, filling in missing columns.

Usage

```
rBind.fill(x, ..., fill = NULL, out.class = class(rbind(x, x))[1])
```

Arguments

`x, ...`	Objects to combine. If the first argument is a list and `..` is unpopulated, the objects in that list will be combined.
`fill`	value with which to fill unmatched columns
`out.class`	the class of the output object. Defaults to the class of x. Note that some output classes are not possible due to R coercion capabilities, such as converting a character matrix to a Matrix.

Details

Similar to `rbind.fill.matrix`, but works for `Matrix` as well as all other R objects. It is completely agnostic to class, and will produce an object of the class of the first input (or of class `matrix` if the first object is one dimensional).

The implementation is recursive, so it can handle an arbitrary number of inputs, albeit inefficiently for large numbers of inputs.

This method is still experimental, but should work in most cases. If the data sets consist solely of data frames, `rbind.fill` is preferred.

Value

a single object of the same class as the first input, or of class `matrix` if the first object is one dimensional

See Also

`rbind.fill`

`rbind.fill.matrix`

Examples

```
df1 = data.frame(x = c(1,2,3), y = c(4,5,6))
rownames(df1) = c("a", "b", "c")
df2 = data.frame(x = c(7,8), z = c(9,10))
rownames(df2) = c("a", "d")
rBind.fill(df1,df2,fill=NA)
rBind.fill(as(df1,'Matrix'),df2,fill=0)
rBind.fill(as.matrix(df1),as(df2,'Matrix'),c(1,2),fill=0)
rBind.fill(c(1,2,3),list(4,5,6,7))
rBind.fill(df1,c(1,2,3,4))

m<-rsparsematrix(1000000,100,.001)
m2<-m
colnames(m)<-1:100
colnames(m2)<-3:102
system.time(b<-rBind.fill(m,m2))
```

Index