Package 'AssocBin'

May 6, 2026

Version 1.1-2

Encoding UTF-8

Title Measuring Association with Recursive Binning

Description
An iterative implementation of a recursive binary partitioning algorithm to measure pairwise dependence with a modular design that allows user specification of the splitting logic and stop criteria. Helper functions provide suggested versions of both and support visualization and the computation of summary statistics on final binnings. For a thorough discussion and demonstration of the algorithm, see Salahub and Oldford (2025) <doi:10.1002/sam.70042>.

Maintainer Chris Salahub <chris.salahub@uwaterloo.ca>

Depends R (>= 4.5.0)

Suggests knitr, rmarkdown

License GPL (>= 3)

NeedsCompilation no

Repository CRAN

VignetteBuilder knitr

RoxygenNote 7.3.2

Author Chris Salahub [aut, cre] (ORCID:
<https://orcid.org/0000-0003-3770-6798>),
Wayne Oldford [aut]

Date/Publication 2025-09-02 22:30:13 UTC

Contents

`binChi` *Statistics for bins*

Description

These functions compute statistics based on observed and expected counts for a list of bins.

Usage

```
binChi(bins, agg = sum)

binMI(bins, agg = sum)

binAbsDif(bins, agg = sum)
```

Arguments

`bins`	a list of bins, each a list with elements ‘x‘, ‘y‘, ‘depth‘, ‘bnds‘ (list with elements ‘x‘ and ‘y‘), ‘expn‘, ‘n‘
`agg`	function which is aggregates the individual statistics computed over each bin

Details

Binstatistics

Three functions are provided by default, ‘binChi‘ computes the chi-squared statistic by taking the squared difference between observed and expected counts and dividing this by the expected counts. ‘binMi‘ computes the mutual information for each bin using the observed and expected counts. Finally, ‘binAbsDif‘ computes the absolute difference between observed and expected counts. Each function first computes a value on every bin independently and stores all these values in memory before using the function provided in the optional argument ‘agg‘ to aggregate these values.

Value

A list with elements ‘residuals‘, ‘stat‘, and ‘nbins‘ reporting the individual statistic values (possibly transformed), the aggegrated statistic value, and the number of bins in ‘bins‘

Functions

- `binChi()`: Chi-squared statistic
- `binMI()`: Mutual information
- `binAbsDif()`: Absolute difference between observed and expected

Author(s)

Chris Salahub

Examples

```
binList1 <- list(list(x = c(1,2), y = c(3,1), depth = 1, n = 2,
                      expn = 2),
                 list(x = c(3,4), y = c(2,4), depth = 1, n = 2,
                      expn = 2))
binList2 <- list(list(x = c(1,2), y = c(3,1), depth = 6, n = 2,
                      expn = 4),
                 list(x = c(), y = c(), depth = 1, n = 0, expn = 1))
binChi(binList1)
binChi(binList2)
binMI(binList1)
binMI(binList2)
binAbsDif(binList2)
```

`binner` *Many split recursive binning*

Description

‘binner‘ is an iterative implementation of a recursive binary partitioning algorithm which accepts the splitting and stopping functions that guide partitioning as arguments.

Usage

```
binner(x, y, stopper, splitter, init = halfSplit, dropPoints = FALSE)
```

Arguments

x	numeric vector of the first variable to be binned
y	numeric vector of the second variable to be binned
stopper	function which accepts a list with elements ‘x‘, ‘y‘, ‘bnds‘, ‘expn‘, and ‘n‘ and returns a logical indicating whether a split should occur for the bin defined by that list
splitter	function which accepts a list of lists with elements ‘x‘, ‘y‘, ‘bnds‘, ‘expn‘, and ‘n‘ and returns a list where each element is a list of two corresponding to a split of the bin at that position in the original list
init	function like ‘splitter‘ applied to the first bin
dropPoints	logical; should points be dropped from final bins?

Details

‘binner‘ creates a two-dimensional histogram of the sample space of ‘x‘ and ‘y‘ by recursively splitting partitions of the data using ‘splitter‘ until ‘stopper‘ indicates that all partitions are not to be split. An optional argument ‘init‘ gives the function applied to the first bin containing all points to initialize the binning algorithm.

Value

A list of lists each with elements ‘x‘, ‘y‘, ‘bnds‘, ‘expn‘, ‘n‘, and ‘stopped‘.

Author(s)

Chris Salahub

Examples

```
## necessary set up
crits <- makeCriteria(depth >= 4, n < 10, expn <= 5)
stopFn <- function(bns) stopper(bns, crits)
spltFn <- function(bn) maxScoreSplit(bn, chiScores)
## generate data
x <- sample(1:100)
y <- sample(1:100)
## run binner
bins <- binner(x, y, stopper = stopFn, splitter = spltFn)
```

catBinner *Binning of categorical variable pairs*

Description

'catBinner' converts the cross-tabulation of two categorical variables into bins which work with all of the functionality on bins built into 'AssocBin'.

Usage

```
catBinner(x, y, dropPoints = FALSE)
```

Arguments

x	factor vector for the first categorical variable
y	factor vector for the second categorical variable
dropPoints	logical; should points be dropped from final bins?

Details

As both variables are already categorical, 'catBinner' performs no splits and does not merge any categories by default.

Value

A list of lists each with elements 'x', 'y', 'bnds', 'expn', 'n', and 'stopped'.

Author(s)

Chris Salahub

chiScores *Scoring functions*

Description

These functions define scores to evaluate candidate splits along a single margin within a partition.

Usage

```
chiScores(bounds, nbelow, n)

miScores(bounds, nbelow, n)

randScores(bounds, nbelow, n)
```

Arguments

bounds	numeric vector giving candidate split bounds in increasing order
nbelow	integer vector giving the number of points below each candidate split
n	the total number of points in the bin to be split

Details

Scorings

Each of these functions accepts ‘bounds‘, an ordered numeric vector containing the candidate splits within a bin and the bin bounds all in increasing order, and ‘nbelow‘ which gives the count of points below each split. ‘n‘ is used to determine the number of points above the split.

This implementation choice was made because AssocBin only considers splits on observed points. It can be proven that, for any convex scoring function, the internal maximum will occur at an observed point. This choice therefore limits the computational search required to identify and split at the optimal coordinate.

Value

A vector of scores.

Functions

- `chiScores()`: A chi-squared statistic score
- `miScores()`: A mutual information score
- `randScores()`: A random score for random splitting

Author(s)

Chris Salahub

Examples

```
vals <- c(2, 5, 12, 16, 19)
chiScores(vals, 1:3, 3)
## same for the miScores
miScores(vals, 1:3, 3)
## random scoring produces different output every time
randScores(vals, 1:3, 3)
randScores(vals, 1:3, 3)
```

depDisplay *Generate a departure display*

Description

This is a generic function which generates a departure display to show the dependence between pairs of variables for several common data structures.

Usage

```
depDisplay(x, y, ..., pair, quants)

## Default S3 method:
depDisplay(x, y, ..., quants, border)

## S3 method for class 'data.frame'
depDisplay(x, ..., pair, quants, border)

## S3 method for class 'DepSearch'
depDisplay(x, ..., pair, quants, border)
```

Arguments

x	a ‘data.frame‘, ‘DepSearch‘ object, or a vector
y	an optional vector, only used if ‘x‘ is a vector
...	additional arguments to pass to plot
pair	the pair of variables to display when ‘x‘ is a ‘data.frame‘ or an ‘DepSearch‘. If ‘x‘ is a ‘data.frame‘, pair can be specified in three ways: as a string with format "<y>:<z>", as a character vector of length two, or as a numeric vector of length two specifying the pair of variables to bin. If ‘x‘ is an ‘DepSearch‘, pair must be either a number or a string of the format "<y>:<z>" specifying which binned pair of ‘x‘ to display.
quants	list of two named vectors ‘x‘ and ‘y‘ providing the quantiles to display on the corresponding axis in the case it is a continuous variable. Defaults to the five number summary.
border	string providing the colour of bin borders to draw, NA suppresses borders

Details

depDisplay

‘depDisplay‘ is a wrapper of the ‘plotBinning‘ function with defaults set to be informative for most investigations.

Value

Invisibly returns the binning obtained and generates a departure display of the pairwise dependence.

Methods (by class)

- `depDisplay(default)`: Default depDisplay method
- `depDisplay(data.frame)`: data.frame method for depDisplay
- `depDisplay(DepSearch)`: DepSearch method for depDisplay

Author(s)

Chris Salahub

Examples

```
x <- rnorm(100)
y <- factor(abs(round(x*2)))
depDisplay(x, y)

## on the iris data
data(iris)
firstPair <- depDisplay(iris, pair = c(1,2))
## another way
firstPair2 <- depDisplay(iris, pair = c("Sepal.Length", "Sepal.Width"))
## a final way
firstPair2 <- depDisplay(iris, pair = "Sepal.Length:Sepal.Width")
```

`DepSearch` *Test pairwise variable independence*

Description

This is a high-level function which accepts a data set, stop criteria, and split functions for continuous variables and then applies a chi-square test for independence to bins generated by recursively binning the ranks of continuous variables or implied by the combinations of levels of categorical variables.

Usage

```
DepSearch(
  data,
  stopCriteria,
  catCon = uniRIntSplit,
  conCon = rIntSplit,
  ptype = c("simple", "conservative", "gamma", "fitted"),
  dropPoints = FALSE
)
```

Arguments

data	'data.frame' or object coercible to a 'data.frame'
stopCriteria	output of 'makeCriteria' providing criteria used to stop binning to be passed to binning functions
catCon	splitting function to apply to pairs of one cateogorical and one continuous variable
conCon	splitting function to apply to pairs of continuous variables
ptype	one of 'simple', 'conservative', 'gamma', or 'fitted': the type of p-values to compute for continuous pairs and pairs of mixed type. 'Conservative' assumes a chi-square distribution for the statistic with highly conservative degrees of freedom based on continuous uniform margins that do not account for the constraints introduced by the ranks. 'Simple' assumes a chi-square distribution but uses contingency-table inspired degrees of freedom which can be slightly anti-conservative in the case of continuous pairs but work well for continuous/categorical comparisons. 'Gamma' assumes a gamma distribution on the resulting statistics with parameters determined by empirical investigation. 'Fitted' mixes the gamma approach and the chi-squared approach these by applying 'gamma' to continuous-categorical comparisons and a least squares fitted version of the simple approximation to continuous-continuous comparisons with parameters determined by empirical study. For all categorical-categorical comparisons the contingency table degrees of freedom are used in a chi-square distribution.
dropPoints	logical; should returned bins contain points?

Details

'DepSearch' is a wrapper function which organizes and executes pairwise binning to test independence between all variable pairs in 'data'. While splitting logic of any sort is supported for continuous margins through the use of the 'catCon' and 'conCon' arguments, the default settings apply rRandom recursive binning, which proceeds for a single pair in three basic steps.

First, the types of the two pairs are identified and rank transformations are applied. If one or both are continuous, the continuous variables are transformed to their ranks. Categorical, logical, and ordinal variables are not transformed.

Second, the ranks of the continuous margins are partitioned by edges added at random positions recursively. For the case of dual continuous variables, the edge at each recursive step is added on a randomly selected margin. If one variable is not continuous, then only the continuous margin is recursively split.

Finally, the resulting partition is evaluated using a chi-square test. For non-continuous variables, this is the classic contingency table test. For continuous variables, expected counts for each cell of the partition are determined based on the area of the cell. The degrees of freedom for the case of a continuous margin are motivated by the contingency table case verified by empirical investigations. Alternatively, several other options are provided to allow a user to select the degrees of freedom approximation they prefer.

This procedure produces a p-value for every pairwise test, placing all pairwise measures on a comparable scale to each other. By placing edges randomly, the method avoids any systematic bias

against particular patterns while still remaining powerful in the detection of function and non-function dependencies of any type.

The output of ‘DepSearch‘ is a list, the first element of which is a list of lists, each of which records the details of the binning of a particular pair of variables.

Value

A ‘DepSearch‘ object, with slots ‘data‘, ‘types‘, ‘pairs‘, ‘binnings‘, ‘residuals‘, ‘statistics‘, ‘K‘, ‘logps‘, and ‘pvalues‘ that stores the results of using recursive binning with the specified splitting logic to test independence on a data set. ‘data‘ gives the name of the data object in the global environment which was split, ‘types‘ is a character vector giving the data types of each pair, ‘pairs‘ is a character vector of the variable names of each pair, ‘binnings‘ is a list of lists where each list is the binning fir to the corresponding pair by the recursive binning algorithm, ‘residuals‘ is list of numeric vectors giving the residual for each bin of each pairwise binning, ‘statistics‘ is a numeric vector giving the chi-squared statistic for each binning, ‘K‘ is a numeric vector giving the number of bins in each binning, ‘logps‘ gives the natural logarithm of the statistic’s p-value, and finally ‘pvalues‘ is a numeric vector of p-values for ‘statistics‘ based on the specified p-value computation, which defaults to ’simple’. Internally, the p-values are computed on the log scale to better distinguish between strongly dependent pairs and the ‘pvalues‘ returned are computed by calling ‘exp(logps)‘. The order of all returned values is by increasing ‘logps‘.

Author(s)

Chris Salahub

Examples

```
## load the iris data set
data(iris)
## evaluate dependence in the iris data
iris_binnings <- DepSearch(iris)
## plot top departure displays
plot(iris_binnings)
## summarize reults
summary(iris_binnings)
```

`depthFill` *Encoding bin features to bin colour fills*

Description

These functions all accept a list of bins and return a vector of colours of the same length that encode some feature of the bins. importanceFill is a special case which adjusts the residuals obtained by the binChi function by the variance of each bin to obtain a better normal approximation and then only shades those bins which are greater than 2 standard deviations from the mean with a color ramp that fully saturates for any bins which are greater than a 0.001 standard normal quantile with a Bonferroni correction applied to account for the number of bins.

Usage

```
depthFill(bins, colrng = c("white", "firebrick"))

residualFill(
  bins,
  resFun = binChi,
  maxRes,
  colrng = c("steelblue", "white", "firebrick"),
  breaks = NA,
  nbr = NA
)

importanceFill(
  bins,
  nbr = NA,
  breaks = NA,
  colrng = c("steelblue", "white", "firebrick")
)
```

Arguments

bins	list of bins to be visualized
colrng	hue range to be passed to ‘colorRampPalette‘ to generate the final hue scale
resFun	function which returns a result with a name element ‘residuals‘ that is a numeric vector of the same length as ‘bins‘
maxRes	numeric maximum value of the residuals to maintain the correct origin and scale the saturation correctly, taken to be the maximum observed residual if not provided
breaks	numeric vector of breakpoints to control hues, defaults to breakpoints that indicate Pearson residuals outside the asymptotic 95 percent confidence interval around zero under the null
nbr	number of breakpoints for automatic breakpoint generation if ‘breaks‘ is not provided

Details

Shadings

depthFill and residualFill do as indicated: mapping the bin depths and residual colours to saturations applied to the bins.

Value

A vector of colours the same length as ‘bins‘.

Functions

- `depthFill()`: Fill by depth
- `residualFill()`: Fill by residual values
- `importanceFill()`: Fill by variance-adjusted chi residuals

Author(s)

Chris Salahub

Examples

```
bin <- makeBin(x = 1:10, y = sample(1:10))
bin2 <- halfSplit(bin, "x")
bin3 <- unlist(lapply(bin2, maxScoreSplit,
                      scorer = chiScores, minExp = 2),
               recursive = FALSE)
plotBinning(bin3, fill = depthFill(bin3)) # all the same depth
plotBinning(bin3, fill = residualFill(bin3)) # chi resids
```

`halfCutTie` *Halve continuously to break ties*

Description

This function halves a bin based on the midpoint of the bounds along whichever margin produces the larger score.

Usage

```
halfCutTie(bin, xscore, yscore, wider, squarify = FALSE)
```

Arguments

`bin`	a bin to be split with elements ‘x‘, ‘y‘, ‘depth‘, ‘bnds‘ (list with elements ‘x‘ and ‘y‘), ‘expn‘, ‘n‘
`xscore`	numeric value giving the score for all splits along x
`yscore`	numeric value giving the score for all splits along y
`wider`	logical; is the bin wider than it is tall?
`squarify`	logical value, should we force splitting on the longer side regardless of scores?

Details

The goal of this function is to break ties within bin splitting in a way which prevents very small or lopsided bins from forming, a common problem with the ‘halfSplit‘ function

Value

A list of two bins resulting from the split of ‘bin‘ in half along the margin corresponding to the larger score.

Author(s)

Chris Salahub

Examples

```
bin <- makeBin(x = 1:10, y = sample(1:10))
halfCutTie(bin, 1, 2, wider = FALSE) # splits on y
halfCutTie(bin, 2, 1, wider = FALSE) # splits on x
halfCutTie(bin, 1, 1, wider = FALSE) # ties are random
```

---

halfSplit *Halve at an observed point*

Description

This function halves a bin under the restriction that splits can only occur at observation coordinates.

Usage

```
halfSplit(bin, margin = sample(c("x", "y"), 1))
```

Arguments

bin	a bin to be split with elements ‘x‘, ‘y‘, ‘depth‘, ‘bnds‘ (list with elements ‘x‘ and ‘y‘), ‘expn‘, ‘n‘
margin	string, one of ‘x‘ or ‘y‘

Details

Given a bin and a margin, this function splits the bin so half the points are above the new split point and half are below.

Value

A list of two bins resulting from the split of ‘bin‘ in half along the specified margin

Author(s)

Chris Salahub

Examples

```
bin <- list(x = 1:10, y = sample(1:10),
            bnds = list(x = c(0, 10), y = c(0, 10)),
            expn = 10, n = 10, depth = 0)
halfSplit(bin)
halfSplit(bin, margin = "y")
```

`heart` *Heart Disease Diagnosis Data*

Description

This data (adapted from the UCI Machine Learning Repository at https://archive.ics.uci.edu/) presents a single data frame reporting heart disease diagnosis results for patients from studies carried out by Andras Janosi at the Hungarian Institute of Cardiology; William Steinbrunn and Matthias Pfisterer at the University Hospitals of Zurich and Basel; and two separate studies by Robert Detrano carried out at the Cleveland Clinic Foundation and Long Beach V.A. Medical Center. The data contains measurements of 15 variables collected on 920 participants:

age Age in years

sex Sex

cp Reported chest pain type: typical angina, non-typical angina, non-angina, or no pain

trestbps Resting blood pressure (mmHg on admission to hospital)

chol Serum cholesterol in mg/dl

fbs Indicator of fasting blood sugar >120 mg/dl

restecg Resting electrocardiographic results: normal, indicating ventricular hypertrophy, or displaying ST-T wave abnormality

thalach Maximum measured heart rate

exang Indicator of exercise induced angina

oldpeak ST wave depression induced by exercise relative to rest

slope The slope of the ST segment during peak exercise

ca Number of major blood vessels coloured by fluoroscopy

thal Type of heart defect

num Diagnosis of heart disease. Values greater than one indicate heart disease of different sorts while a value of zero indicates no heart disease

study The study where the participant's data was collected

Usage

```
data(heart)
```

Format

A matrix with 920 rows and 15 columns, with each row reporting measurements for a participant in one of the heart disease studies.

makeBin *Make a bin*

Description

Creating a new bin object

Usage

```
makeBin(
  x,
  y,
  bnds = list(x = range(x) - c(1, 0), y = range(y) - c(1, 0)),
  expn = length(x),
  n = length(x),
  depth = 0,
  stopped = FALSE
)
```

Arguments

x	numeric vector of observations on the first variable
y	numeric vector of observations on the second variable
bnds	list of length two with named elements ‘x‘ and ‘y‘ each a vector of length two giving respective bin boundaries
expn	expected number of points in the bin, can be non-integer
n	observed count of points in the bin
depth	number of splits from the initial bin to the bin
stopped	logical; should the bin be split further?

Details

‘makeBin‘ creates a bin list based on the arguments provided to it. Should some be missing, basic defaults ensure that the complete set of bin characteristics are created in the resulting list representing the bin object.

Value

A list with named elements matching these arguments

Author(s)

Chris Salahub

Examples

```
makeBin(x = 1:10, y = sample(1:10),
bnds = list(x = c(0,10), y = c(0, 10)), expn = 10, n = 10,
depth = 0, stopped = FALSE)
```

`makeCriteria` *Make stop crteria*

Description

Capture a sequence of logical statements and append them into a single expression.

Usage

```
makeCriteria(...)
```

Arguments

`...`	an arbitrary number of expressions which evaluate to logicals

Details

This function, along with ‘stopper‘ dictates the stop behaviour of recursive binning. It accepts an arbitrary number of arguments, each a logical statement, and appends them all into a string separated by the pipe character.

Value

A string which appends all expressions together.

Author(s)

Chris Salahub

Examples

```
makeCriteria(depth >= 5, n < 1)
```

`maxScoreSplit` *Size-restricted bivariate score maximizing splitting*

Description

Splits a bin based on the location maximizing a score function with restrictions on minimum bin size.

Usage

```
maxScoreSplit(bin, scorer, minExp = 5, squarify = FALSE)
```

Arguments

`bin`	a bin to be split with elements ‘x‘, ‘y‘, ‘depth‘, ‘bnds‘ (list with elements ‘x‘ and ‘y‘), ‘expn‘, ‘n‘
`scorer`	function which accepts a numeric vector of potential split coordinates and the bounds of ‘bin‘ and returns a numeric vector of scores for each
`minExp`	value giving the smallest expected count allowed for bin splits
`squarify`	logical value, should we force splitting on the longer side regardless of scores?

Details

This function serves as a wrapper which manages the logic of splitting bins using a score function while maintaining a minimum size and allowing forced splits along the wider edge.

Value

A list of two bins resulting from the split of ‘bin‘ along the corresponding margin at the maximum location

Author(s)

Chris Salahub

Examples

```
bin <- makeBin(x = 1:10, y = sample(1:10))
maxScoreSplit(bin, chiScores)
maxScoreSplit(bin, miScores) # pretty similar for both
maxScoreSplit(bin, randScores)
maxScoreSplit(bin, randScores) # different every time
```

`numNumFittedDf` *Computing a binning's degrees of freedom*

Description

Functions which compute the degrees of freedom of a binning for a chi-squared approximation or parameters for a gamma approximation based on empirical results.

Usage

```
numNumFittedDf(nbins)

numNumSimpleDf(nbins)

numNumGammaShape(nbins)

numNumGammaScale(nbins)

facNumSimpleDf(nbins, ncat)

facNumFittedDf(nbins, ncat)

facNumGammaShape(nbins, ncat)

facNumGammaScale(nbins, ncat)
```

Arguments

`nbins`	the number of bins resulting from recursive random binning
`ncat`	if one variable is categorical, the number of values the variable can take

Details

These exported functions are used to compute parameters needed to approximate the distribution of the chi-squared statistic computed over bins. A full discussion can be found in the accompanying paper.

Value

A numeric estimate of the paramter. In the case of degrees of freedom, this is generally not an integer.

Functions

- `numNumFittedDf()`: Dual continuous fitted df
- `numNumSimpleDf()`: Dual continuous simple df
- `numNumGammaShape()`: Dual continuous gamma shape

- `numNumGammaScale()`: Dual continuous gamma scale
- `facNumSimpleDf()`: Mixed type simple df
- `facNumFittedDf()`: Mixed type fitted df
- `facNumGammaShape()`: Mixed type gamma shape
- `facNumGammaScale()`: Mixed type gamma scale

Author(s)

Chris Salahub

`plotBinning` *Plot a binning using shaded rectangles*

Description

Use a binning and vector of fill colours to visualize the sample space of pairwise data.

Usage

```
plotBinning(
  bins,
  fill,
  add = FALSE,
  factor = 0.5,
  xlab = "x",
  ylab = "y",
  showXax = FALSE,
  showYax = FALSE,
  border = "black",
  ...
)
```

Arguments

`bins`	list of lists each with a named elements ‘x‘, ‘y‘, and ‘bnds‘, the last of which is a list having named elements ‘x‘ and ‘y‘
`fill`	vector of values which can be interpreted as colours of the same length as ‘bins‘
`add`	logical, should the plot of bins be added to the current plot area?
`factor`	number between 0 and 1 giving the factor applied to jitter categorical variables
`xlab`	string, the label to be placed on the x axis
`ylab`	string, the label to be placed on the y axis
`showXax`	logical indicating whether to plot x axis markings
`showYax`	logical indicating whether to plot y axis markings
`border`	argument to be passed to ‘rect‘ internally giving the border colour
`...`	optional additional arguments to be passed to ‘plot‘, ‘points‘

Details

'plotBinning' plots each bin within a list of bins with custom shading to communicate large residuals, the depth of bins, or highlight particular bins. It automatically jitters points within categorical levels to avoid overplotting.

Value

A list of lists each with elements 'x', 'y', 'bnds', 'expn', and 'n'.

Author(s)

Chris Salahub

Examples

```
bin <- list(x = 1:10, y = sample(1:10),
            bnds = list(x = c(0, 10), y = c(0, 10)),
            expn = 10, n = 10, depth = 0)
bin2 <- halfSplit(bin, "x")
bin3 <- unlist(lapply(bin2, maxScoreSplit, scorer = chiScores),
               recursive = FALSE)
plotBinning(bin3)
```

---

rIntSplit *Random integer splitting*

Description

A function which splits a bin at a random integer conforming to limits on minimum bin size.

Usage

```
rIntSplit(bin, minExp = 5, squarify = TRUE)
```

Arguments

bin	a bin to be split with elements 'x', 'y', 'depth', 'bnds' (list with elements 'x' and 'y'), 'expn', 'n'
minExp	numeric giving the minimum expected count allowed in a bin
squarify	logical value, should we force splitting on the longer side?

Details

This function serves as a wrapper which manages the interaction of a score function, marginal splitting functions, tie breaking function, and a maximum selection function to split a bin at the observation coordinate which maximizes the score function.

Value

A list of two bins resulting from the split of ‘bin‘ along the corresponding margin at the maximum location

Author(s)

Chris Salahub

Examples

```
bin <- makeBin(x = 1:10, y = sample(1:10))
rIntSplit(bin, minExp = 2)
```

`rUnifSplit` *Random uniform splitting*

Description

Split bins randomly and uniformly

Usage

```
rUnifSplit(bin, minExp = 0, squarify = FALSE)
```

Arguments

`bin`	a bin to be split with elements ‘x‘, ‘y‘, ‘depth‘, ‘bnds‘ (list with elements ‘x‘ and ‘y‘), ‘expn‘, ‘n‘
`minExp`	numeric giving the minimum expected count allowed in a bin
`squarify`	logical value, should we force splitting on the longer side?

Details

This function samples a coordinate uniformly along a random margin and splits a bin at that coordinate. In contrast to maxScoreSplit with randScores, this can introduce splits at locations other than the points.

Value

A list of two bins resulting from the split of ‘bin‘ at a random location on a random margin

Author(s)

Chris Salahub

Examples

```
bin <- makeBin(x = 1:10, y = sample(1:10))
rUnifSplit(bin, minExp = 2)
```

`sandboxMaxSplit` *Bivariate score maximizing splitting*

Description

A function which splits a bin based on the location maximizing a score function.

Usage

```
sandboxMaxSplit(
  bin,
  scorer,
  ties = halfCutTie,
  minExp = 5,
  pickMax = which.max,
  ...
)
```

Arguments

`bin`	a bin to be split with elements ‘x‘, ‘y‘, ‘depth‘, ‘bnds‘ (list with elements ‘x‘ and ‘y‘), ‘expn‘, ‘n‘
`scorer`	function which accepts a numeric vector of potential split coordinates and the bounds of ‘bin‘ and returns a numeric vector of scores for each
`ties`	function which is called to break ties when all splits generate the same score
`minExp`	value giving the smallest expected count allowed for bin splits
`pickMax`	function which accepts a list of scores and returns the element of the largest score according to some rule
`...`	optional additional arguments to ‘scorer‘

Details

This function serves as a wrapper which manages the interaction of a score function, marginal splitting functions, tie breaking function, and a maximum selection function to split a bin at the observation coordinate which maximizes the score function.

Value

A list of two bins resulting from the split of ‘bin‘ along the corresponding margin at the maximum location

Author(s)

Chris Salahub

`singleBinner` *Single split recursive binning*

Description

'singleBinner' is an iterative implementation of a recursive binary partitioning algorithm which accepts the splitting and stopping functions that guide partitioning as arguments.

Usage

```
singleBinner(
  x,
  y,
  stopper,
  splitter,
  init = halfSplit,
  maxK = 5,
  dropPoints = FALSE
)
```

Arguments

`x`	numeric vector of the first variable to be binned
`y`	numeric vector of the second variable to be binned
`stopper`	function which accepts a list with elements 'x', 'y', 'bnds', 'expn', and 'n' and returns a logical indicating whether a split should occur for the bin defined by that list
`splitter`	function which accepts a list of lists with elements 'x', 'y', 'bnds', 'expn', and 'n' and returns a list where each element is a list of two corresponding to a split of the bin at that position in the original list
`init`	function like 'splitter' applied to the first bin
`maxK`	integer giving the number of bins where splitting is stopped regardless of stop criteria
`dropPoints`	logical; should points be dropped from final bins?

Details

'singleBinner' creates a two-dimensional histogram of the sample space of 'x' and 'y' by recursively splitting partitions of the data using 'splitter' until 'stopper' indicates that all partitions are not to be split. An optional argument 'init' gives the function applied to the first bin containing all points to initialize the binning algorithm. Unlike 'binner', it does this by splitting one bin at a time, and so accepts an argument to specify exactly how many bins to produce.

Value

A list of lists each with elements 'x', 'y', 'bnds', 'expn', 'n', and 'stopped'.

Author(s)

Chris Salahub

Examples

```
## necessary set up
crits <- makeCriteria(depth >= 4, n < 10, expn <= 5)
stopFn <- function(bns) stopper(bns, crits)
spltFn <- function(bn) rIntSplit(bn, minExp = 5)
## generate data
x <- sample(1:100)
y <- sample(1:100)
## run binner
bins <- singleBinner(x, y, stopper = stopFn, splitter = spltFn)
```

---

sp500pseudo *De-Garched S&P 500 returns*

Description

This data is the result of code from the 'zenplots' package to process S&P 500 consituent stock returns into uniform pseudo-observations for measuring association.

Usage

```
data(sp500pseudo)
```

Format

A matrix with 755 rows and 461 columns, the rows correspond to dates between 2007 and 2009 and the columns correspond to the different S&P 500 constituent stocks.

---

splitX *Helper functions for marginal splitting*

Description

These functions are helpers to safely split bins along X or Y.

Usage

```
splitX(bin, bd, above, below)

splitY(bin, bd, above, below)
```

Arguments

bin	a bin to be split with elements ‘x‘, ‘y‘, ‘depth‘, ‘bnds‘ (list with elements ‘x‘ and ‘y‘), ‘expn‘, ‘n‘
bd	numeric split point within the bin bounds
above	indices of ‘x‘ and ‘y‘ points in the bin above ‘bd‘
below	indices of ‘x‘ and ‘y‘ points in the bin below ‘bd‘

Details

These unexported functions have been defined primarily to clean up other code, but could be changed to obtain different core functionality.

Value

A list of two bins resulting from the split of ‘bin‘ at ‘bds‘.

Functions

- `splitX()`: Splitting on x
- `splitY()`: Splitting on y

Author(s)

Chris Salahub

`stopper` *Check bins against stop criteria*

Description

Evaluate the stop ‘criteria‘ for each bin in ‘binList‘

Usage

```
stopper(binList, criteria)
```

Arguments

binList	a list of bins, each a list which can be cast as an environment for evaluation
criteria	string of logical expressions separated by pipes to be evaluated within each bin of ‘binList‘

Details

This function makes use of R’s lexical scoping to evaluate ‘criteria‘ (a string), within each bin of ‘binList‘.

Value

A logical vector of the same length as ‘binList‘.

Author(s)

Chris Salahub

Examples

```
crits <- makeCriteria(depth >= 5, n < 1)
binList1 <- list(makeBin(x = c(1,2), y = c(3,1), depth = 1, n = 2),
                 makeBin(x = c(3,4), y = c(2,4), depth = 1, n = 2))
binList2 <- list(makeBin(x = c(1,2), y = c(3,1), depth = 6, n = 2),
                 makeBin(x = c(), y = c(), depth = 1, n = 0))
stopper(binList1, crits)
stopper(binList2, crits)
```

---

summary.DepSearch *S3 methods for ‘DepSearch‘*

Description

The ‘summary‘ and ‘plot‘ methods outlined here support the quick description of an ‘DepSearch‘ object.

Usage

```
## S3 method for class 'DepSearch'
summary(object, ..., adjustP = FALSE)

## S3 method for class 'DepSearch'
print(x, ...)

## S3 method for class 'DepSearch'
plot(
  x,
  ...,
  which = 1:5,
  border = "black",
  buffer = 0.01,
  dropPoints = FALSE,
  colrng = c("steelblue", "white", "firebrick"),
  nbr = NA,
  pch = "."
)
```

Arguments

object	'DepSearch' object to summarize
...	additional arguments to pass on to the method
adjustP	logical: should the p-values be adjusted for multiple testing?
x	object with class 'DepSearch'
which	indices of binnings to display from 'x', where binnings are ordered by increasing p-value
border	colour of borders to be drawn on the binnings
buffer	relative width of empty space separating categories
dropPoints	logical: should points be dropped for the plot of the binnings?
colrng	colour range to be passed to 'residualFill' for plotting
nbr	number of breaks to be passed to 'residualFill' for plotting
pch	point type passed to plot

Details

Methods

For each index in 'which', this function produces a row of three plots. The first plot is the raw data, the second plot is the ranks of the data, and the final plot is the binning contained in the 'DepSearch' object.

Value

Nothing for the plot method, while summary quietly returns a summary of 'DepSearch'

Functions

- `summary(DepSearch)`: Summary method for 'DepSearch'
- `print(DepSearch)`: Print method for 'DepSearch'
- `plot(DepSearch)`: Plot method for 'DepSearch'

Author(s)

Chris Salahub

`uniBinner` *Single margin binning*

Description

'uniBinner' is an iterative implementation of a recursive binary partitioning algorithm which accepts the splitting and stopping functions that guide partitioning as arguments and applies them to a specified margin alone.

Usage

```
uniBinner(x, y, stopper, splitter, dropPoints = FALSE, on = "y")
```

Arguments

`x`	factor vector for the the first variable
`y`	numeric vector of the second variable (to be split)
`stopper`	function which accepts a list with elements 'x', 'y', 'bnds', 'expn', and 'n' and returns a logical indicating whether a split should occur for the bin defined by that list
`splitter`	function which accepts a list of lists with elements 'x', 'y', 'bnds', 'expn', and 'n' and returns a list where each element is a list of two corresponding to a split of the bin at that position in the original list
`dropPoints`	logical; should points be dropped from final bins?
`on`	one of 'x' or 'y': the margin to split

Details

'binner' creates a one-dimensional histogram of 'y' or 'x' for each categorical value of the other by recursively splitting partitions of the data using 'splitter' until 'stopper' indicates that all partitions are not to be split.

Value

A list of lists each with elements 'x', 'y', 'bnds', 'expn', 'n', and 'stopped'.

Author(s)

Chris Salahub

uniMaxScoreSplit *Univariate score maximizing splitting*

Description

A function which splits a bin based on the location maximizing a score function.

Usage

```
uniMaxScoreSplit(bin, scorer, minExp = 5, on = "y")
```

Arguments

bin	a bin to be split with elements ‘x‘, ‘y‘, ‘depth‘, ‘bnds‘ (list with elements ‘x‘ and ‘y‘), ‘expn‘, ‘n‘
scorer	function which accepts a numeric vector of potential split coordinates and the bounds of ‘bin‘ and returns a numeric vector of scores for each
minExp	numeric giving the minimum expected count allowed in a bin
on	one of "x" or "y": the margin to split

Details

This function is the univariate version of ‘maxScoreSplit‘ and so is considerably simpler. It assumes the variable to be split is named ‘x‘ in the bin, and the other variable is to remain unsplit.

Value

A list of two bins resulting from the split of ‘bin‘ at the maximum split location along y

Author(s)

Chris Salahub

uniRIntSplit *Univariate random integer splitting*

Description

A function which splits a bin along x at a random integer conforming to limits on minimum bin size.

Usage

```
uniRIntSplit(bin, minExp = 5, on = "y")
```

Arguments

bin	a bin to be split with elements ‘x‘, ‘y‘, ‘depth‘, ‘bnds‘ (list with elements ‘x‘ and ‘y‘), ‘expn‘, ‘n‘
minExp	numeric giving the minimum expected count allowed in a bin
on	one of "x" or "y": the margin to split

Details

This function serves as a wrapper which manages the interaction of a score function, marginal splitting functions, tie breaking function, and a maximum selection function to split a bin along a single margin at the observation coordinate which maximizes the score function.

Value

A list of two bins resulting from the split of ‘bin‘ along the corresponding margin at the maximum location

Author(s)

Chris Salahub

Examples

```
bin <- makeBin(x = 1:10, y = sample(1:10))
rIntSplit(bin, minExp = 2)
```

Index