Package ‘StatMeasures’

March 27, 2015

Type Package
Title Easy Data Manipulation, Data Quality and Statistical Checks
Version 1.0
Date 2015-03-24
Author Akash Jain
Maintainer Akash Jain <akashjain02.05@gmail.com>
Description Offers useful functions to perform day-to-day data manipulation
operations, data quality checks and post modelling statistical checks.
One can effortlessly change class of a number of variables to factor,
remove duplicate observations from the data, create deciles of a
variable, perform data quality checks for continuous (integer or numeric),
categorical (factor) and date variables, and compute goodness of fit
measures such as auc for statistical models. The functions are consistent
for objects of class ‘data.frame’ and ‘data.table’, which is an enhanced
‘data.frame’ implemented in the package ‘data.table’.
Depends R (>= 3.1.3)
Imports data.table (>= 1.9.4)
License GPL-2
NeedsCompilation no
Repository CRAN
Date/Publication 2015-03-27 23:46:22

R topics documented:

accuracy ................................................................. 2
actvspred .............................................................. 3
auc ............................................................. 4
contents ............................................................. 5
decile .............................................................. 6
dqcategorical ....................................................... 7
dqcontinuous ....................................................... 8
dqdate .................................................. 9
Description

Takes in actual binary response, predicted probabilities and cutoff value, and returns confusion matrix and overall accuracy.

Usage

accuracy(y, yhat, cutoff)

Arguments

y actual binary response variable
yhat predicted probabilities corresponding to the actual binary response
cutoff threshold value in the range 0 to 1

Details

When we predict a binary response, first thing that we want to check is accuracy of the model for a particular cutoff value. This function does just that and provides confusion matrix (numbers and percentage) and overall accuracy. Overall accuracy is calculated as (TP + TN)/(P + N).

The output is a list from which the individual elements can be picked as shown in the example.

Value

a three element list: confusion matrix as a table, confusion matrix (percentages) as a table and overall accuracy value

Author(s)

Akash Jain
actvspred

Comparison of actual and predicted linear response

Description

Takes in actual, predicted linear response and quantile value, and returns average actual and predicted response in each quantile

Usage

actvspred(y, yhat, n)

Arguments

y  actual linear response
yhat  predicted linear response
n  quantiles to be created for comparison

Details

actvspred function divides the data into n (given as input by the user) quantiles and computes mean of y and yhat for each quantile. All NA's in n, y and yhat are removed for calculation.

The function also plots a line chart of average actual response and average predicted response over n quantiles. This plot can be used to visualize how close both the lines are.

Value

a data.frame with average actual and predicted response in each quantile

Author(s)

Akash Jain

See Also

ks, auc, iv, splitdata

Examples

# A 'data.frame' with y and yhat
df <- data.frame(y = c(1, 0, 1, 0),
                 yhat = c(0.86, 0.23, 0.65, 0.92, 0.37))

# Accuracy tables and overall accuracy figures
ltAccuracy <- accuracy(y = df[, 'y'], yhat = df[, 'yhat'], cutoff = 0.7)
accuracyNumber <- ltAccuracy$accuracyNum
accuracyPercentage <- ltAccuracy$accuracyPer
overallAccuracy <- ltAccuracy$overallAcc

actvspred

Description

Takes in actual, predicted linear response and quantile value, and returns average actual and predicted response in each quantile

Usage

actvspred(y, yhat, n)

Arguments

y  actual linear response
yhat  predicted linear response
n  quantiles to be created for comparison

Details

actvspred function divides the data into n (given as input by the user) quantiles and computes mean of y and yhat for each quantile. All NA's in n, y and yhat are removed for calculation.

The function also plots a line chart of average actual response and average predicted response over n quantiles. This plot can be used to visualize how close both the lines are.

Value

a data.frame with average actual and predicted response in each quantile

Author(s)

Akash Jain
See Also

mape, splitdata

Examples

# A 'data.frame' with y and yhat
def <- data.frame(y = c(1, 2, 3, 6, 8, 10, 15),
yhat = c(1.2, 2.5, 3.3, 6.9, 9.3, 6.5, 12.3))

# Get actual vs predicted table
ACTVSPRED <- actvspred(y = df[, 'y'], yhat = df[, 'yhat'], n = 5)

auc

Area under curve of predicted binary response

Description

Takes in actual binary response and predicted probabilities, and returns auc value

Usage

auc(y, yhat)

Arguments

y actual binary response

yhat predicted probabilities corresponding to the actual binary response

Details

Area under the receiver operating characteristic (ROC) curve is the most sought after criteria for judging how good model predictions are.

auc function calculates the true positive rates (TPR) and false positive rates (FPR) for each cutoff from 0.01 to 1 and calculates the area using trapezoidal approximation. A ROC curve is also generated.

Value

area under the ROC curve

Author(s)

Akash Jain

See Also

accuracy, ks, iv, gini, splitdata
## Examples

```r
# A 'data.frame' with y and yhat
df <- data.frame(y = c(1, 0, 1, 1, 0, 0, 1, 0, 1, 0),
                 yhat = c(0.86, 0.23, 0.65, 0.92, 0.37, 0.45, 0.72, 0.19, 0.92, 0.50))

# AUC figure
AUC <- auc(y = df[, 'y'], yhat = df[, 'yhat'])
```

## Description

Takes in a data and returns summary of the data

## Usage

```r
contents(data)
```

## Arguments

- **data**: a data.frame or data.table

## Details

This function helps when one wants to get a quick snapshot of the data such as class, distinct values, missing values and sample value of the variables.

It works for both 'data.frame' and 'data.table' but the output will be a 'data.frame' only.

## Value

A data.frame that contains variable, class, distinct values, missing values, percentage of missing value and sample value

## Author(s)

Akash Jain

## See Also

dqcontinuous, dqcategorical, dqdate
**Examples**

```r
# A data frame
df <- data.frame(x = c(1, 2, 3, 4, NA),
                 y = c('a', 'b', 'c', 'd', 'e'),
                 z = c(1, 1, 0, 0, 1))

# Summary of the data
dfContents <- contents(data = df)
```

**decile**  
*Create deciles of a variable*

**Description**

Takes in a vector, and returns the deciles

**Usage**

```r
decile(vector, decreasing = FALSE)
```

**Arguments**

- `vector` an integer or numeric vector
- `decreasing` a logical input, which if set to FALSE puts smallest values in decile 1 and if set to TRUE puts smallest values in decile 10; FALSE is default

**Details**

decile is a convinient function to get integer deciles of an integer or numeric vector. By default, the smallest values are placed in the smallest decile. Sometimes one may want to put smallest values in the biggest decile, and for that the user can set the decreasing argument to TRUE; by default it is FALSE.

**Value**

an integer vector of decile values

**Author(s)**

Akash Jain

**See Also**

pentile, outliers, imputemiss
Examples

# Scores vector
scores <- c(1, 4, 7, 10, 15, 21, 25, 27, 32, 35, 
           49, 60, 75, 23, 45, 86, 26, 38, 34, 67)

# Create deciles based on the values of the vector
decileScores <- decile(vector = scores)
decileScores <- decile(vector = scores, decreasing = TRUE)

dqncategorical Data quality check of categorical variables

Description
Takes in a data, and returns summary of categorical variables

Usage
dqncategorical(data)

Arguments
data a data.frame or data.table

Details
While trying to understand a data, it is important to know the distribution of categorical variables. dqncategorical produces an output which answers a couple of questions regarding such variables - how many distinct categories does the variable have, what are the categories, what is the frequency of each of them and the percentage frequency.

But first, it is critical to identify categorical variables in the data. They may be integer, numeric or character. All such variables should be converted to factor; one may use factorise function in this package to do this task easily.

The function identifies all the factor variables and produces an output for each of them and returns a consolidated summary. It works for both 'data.frame' and 'data.table' but the output summary is a 'data.frame' only.

Value
a data.frame which contains the variable, category index, category, category frequency and percentage frequency of all factor variables

Author(s)
Akash Jain
See Also

dqcontinuous, dqdate, contents

Examples

# A 'data.frame'

df <- data.frame(phone = c('IP', 'SN', 'HO', 'IP', 'SN', 'IP', 'HO', 'SN', 'IP', 'SN'),
  colour = c('black', 'blue', 'green', 'blue', 'black', 'silver', 'black', 'white', 'black', 'green'))

# Factorise categorical variables

df <- factorise(data = df, colNames = c('phone', 'colour'))

# Generate a data quality report of continuous variables

summaryCategorical <- dqcategorical(data = df)

dqcontinuous  Data quality check of continuous variables

Description

Takes in a data, and returns summary of continuous variables

Usage

dqcontinuous(data)

Arguments

data  a data.frame or data.table

Details

It is of utmost importance to know the distribution of continuous variables in the data. dqcontinuous produces an output which tells - continuous variable, non-missing values, missing values, percentage missing, minimum, average, maximum, standard deviation, variance, common percentiles from 1 to 99, and number of outliers for each continuous variable.

The function tags all integer and numeric variables as continuous, and produces output for them; if you think there are some variables which are integer or numeric in the data but they don’t represent a continuous variable, change their type to an appropriate class.

dqcontinuous uses the same criteria to identify outliers as the one used for box plots. All values that are greater than 75th percentile value + 1.5 times the inter quartile range or lesser than 25th percentile value - 1.5 times the inter quartile range, are tagged as outliers.

This function works for both 'data.frame' and 'data.table' but returns a 'data.frame' only.
Value

A data.frame which contains the non-missing values, missing values, percentage of missing values, minimum, mean, maximum, standard deviation, variance, percentiles and count of outliers of all integer and numeric variables.

Author(s)

Akash Jain

See Also

dqcontinuous, dqdate, contents

Examples

```r
# A 'data.frame'
df <- data.frame(x = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
                 y = c(22, NA, 66, 12, 78, 34, 590, 97, 56, 37))

# Generate a data quality report of continuous variables
summaryContinuous <- dqcontinuous(data = df)
```

---

dqdate

Data quality check of date variables

Description

Takes in a data, and returns summary of date variables.

Usage

dqdate(data)

Arguments

data a data.frame or data.table

Details

dqdate produces summary of all date variables in the data. The function identifies all variables as date if they are of class 'Date' or 'IDate'.

Generally the dates are imported in R as character. They must be converted to an appropriate date format and then the function should be used.

The summary includes variable, non-missing values, missing values, minimum and maximum of the date variables. Input data can be a 'data.frame' or 'data.table' but the output summary will be a 'data.frame' only.
Value

a data.frame which contains the variable, non-missing values, missing values, minimum and maximum of all date variables

Author(s)

Akash Jain

See Also

dqcontinuous, dqcategorical, contents

Examples

# A 'data.frame'
                    temperature = c(26, 32, 35, 7, 14))

   # Convert character date to date format
   df[, 'date'] <- as.Date(df[, 'date'])

   # Generate a data quality report of date variables
   summaryDate <- dqdate(data = df)

factorise

Change the class of variables to factor

Description

Takes in data and colNames, and returns the data with all variables mentioned in colNames converted to factor

Usage

factorise(data, colNames)

Arguments

data a data.frame or data.table
colNames a character vector of variable names to be converted to factor

Details

We often face the task of converting a bunch of variables to factor. This function is particularly useful in such a situation. Just specify the data and variable names and all of them will be converted to factor.

It works for both 'data.frame' and 'data.table', and the output is data of the same class as that of input.
Value

data of same class as input with specified variables converted to factor

Author(s)

Akash Jain

See Also

randomise, rmdupkey, rmdupobs

Examples

```r
# A 'data.frame'
df <- data.frame(x = c(1, 2, 3, 4, 5),
                 y = c('a', 'b', 'c', 'd', 'e'),
                 z = c(1, 1, 0, 0, 1))

# Change the class of variables y and z to factors
dfFac <- factorise(data = df, colNames = c('y', 'z'))
```

gini

Gini coefficient of a distribution

Description

Takes in a distribution and returns gini coefficient

Usage

gini(y)

Arguments

y

an integer or numeric distribution

Details

To compute the gini coefficient of a distribution, gini is the right function. It uses trapezoidal approximation to calculate the area of the curve. Lorenz curve is also plotted as output.

Value

gini coefficient of the distribution

Author(s)

Akash Jain
See Also

auc

Examples

```r
# Distribution
dist <- c(1, 4, 7, 15, 10)

# Gini coefficient
GINI <- gini(y = dist)
```

---

### imputemiss

**Impute missing values in a variable**

**Description**

Takes in a vector and a value, and returns the vector with missing values imputed with that value.

**Usage**

```r
imputemiss(vector, value)
```

**Arguments**

- `vector`: a vector with missing values
- `value`: the value to be used for imputation

**Details**

`imputemiss` imputes the missing (NA) values in the vector with a specified value. The function simplifies the code for imputation.

**Value**

- `vector`: of the same class as input vector with imputed missing values

**Author(s)**

Akash Jain

**See Also**

`decile`, `pentile`, `outliers`
Examples

```r
# Scores vector
scores <- c(1, 2, 3, NA, 4, NA)

# Imputed scores vector
scoresImp <- imputemiss(vector = scores, value = 5)
```

---

**iv**

Information value of an independent variable in predicting a binary response

---

**Description**

Takes in independent and dependent variable and returns IV value

**Usage**

`iv(x, y)`

**Arguments**

- `x`: an independent variable
- `y`: a binary response variable

**Details**

Information value of a variable is a significant indicator of its relevance in the prediction of a binary response variable. **iv** computes that value using the formula, \( IV = \sum (\text{Responders} - \text{Non-responders}) \times \ln(\frac{\text{Responders}}{\text{Non-responders}}) \) for each bin.

Ten bins are created for continuous variables while categories itself are used as bins for categorical independent variables.

**Value**

information value of `x`

**Author(s)**

Akash Jain

**See Also**

accuracy, auc, ks, splitdata
Examples

# A 'data.frame'
df <- data.frame(x = c('a', 'a', 'a', 'b', 'b'),
                 y = c(0, 1, 0, 1, 1))

# Information value
IV <- iv(x = df[, 'x'], y = df[, 'y'])

ks

Kolmogorov-Smirnov statistic for predicted binary response

Description

Takes in actual binary response and predicted probabilities, and returns table used for KS computation and KS value

Usage

ks(y, yhat)

Arguments

y actual binary response
yhat predicted probabilities corresponding to the actual binary response

Details

Kolmogorov-Smirnov statistic can be easily computed using ks function. It not only computes the statistic but also returns the table used for ks computation. A chart is also plotted for quick visualization of split between percentage of responders and non-responders. Deciles are used for computation.

Appropriate elements of the 2 element list (ie. ksTable or ks) can be picked using the code given in the example.

Value

a two element list: table for KS computation and KS value itself

Author(s)

Akash Jain

See Also

accuracy, auc, iv, splitdata
Examples

```r
# A 'data.frame' with y and yhat
df <- data.frame(y = c(1, 0, 1, 1, 0),
                 yhat = c(0.86, 0.23, 0.65, 0.92, 0.37))

# KS table and value
ktks <- ks(y = df[, 'y'], yhat = df[, 'yhat'])
ktstable <- ktks$ksTable
KS <- ktks$ks

mape <- function(y, yhat)
  Compute mean absolute percentage error

Example 1

Dataframe with y and yhat:
```

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>yhat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.86</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.23</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.65</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0.92</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0.37</td>
</tr>
</tbody>
</table>
```

Compute MAPE:
```
ks <- ks(y = df[, 'y'], yhat = df[, 'yhat'])
ksTable <- ks$ksTable
KS <- ks$ks
mape(y = df[, 'y'], yhat = df[, 'yhat'])
```

Description

Takes in actual and predicted linear response, and returns MAPE value.

Usage

```r
mape(y, yhat)
```

Arguments

- `y`: actual linear response
- `yhat`: predicted linear response

Details

`mape` calculates the mean absolute percentage error in a predicted linear response.

Value

- mean absolute percentage error

Author(s)

Akash Jain

See Also

`actvspred`, `splitdata`

Examples

```r
# A 'data.frame' with y and yhat
df <- data.frame(y = c(1.5, 2, 3.2),
                 yhat = c(3.4, 2.2, 2.7))

# Compute mape
MAPE <- mape(y = df[, 'y'], yhat = df[, 'yhat'])
```
outliers
Identify outliers in a variable

Description
Takes in a vector, and returns count and index of outliers

Usage
outliers(vector)

Arguments
vector an integer or numeric vector

Details
The function uses the same criteria to identify outliers as the one used for box plots. All values that are greater than 75th percentile value + 1.5 times the inter quartile range or lesser than 25th percentile value - 1.5 times the inter quartile range, are tagged as outliers.
The individual elements (number of outliers and index of outliers) of the two element output list can be picked using the code given in example. The index of outliers can be used to get a vector of all outliers.

Value
a list with two elements: count and index of outliers

Author(s)
Akash Jain

See Also
decile, pentile, imputemiss

Examples
# Scores vector
scores <- c(1, 4, 7, 10, 566, 21, 25, 27, 32, 35,
           49, 60, 75, 23, 45, 86, 26, 38, 34, 223, -3)

# Identify the count of outliers and their index
ltOutliers <- outliers(vector = scores)
numOutliers <- ltOutliers$numOutliers
idxOutliers <- ltOutliers$idxOutliers
valOutliers <- scores[idxOutliers]
pentile

*Create pentiles of a variable*

**Description**

Takes in a vector, and returns a vector of pentiles

**Usage**

pentile(vector, decreasing = FALSE)

**Arguments**

- **vector**: an integer or numeric vector
- **decreasing**: a logical input, which if set to FALSE puts smallest values in pentile 1 and if set to TRUE puts smallest values in pentile 5; FALSE is default

**Details**

pentile is a convinient function to get integer pentiles of an integer or numeric vector. By default, the smallest values are placed in the smallest pentile. Sometimes one may want to put smallest values in the biggest pentile, and for that the user can set the decreasing argument to TRUE; by default it is FALSE.

**Value**

an integer vector of pentile values

**Author(s)**

Akash Jain

**See Also**

decile, outliers, imputemiss

**Examples**

```r
# Scores vector
scores <- c(1, 4, 7, 10, 15, 21, 25, 27, 32, 35, 49, 60, 75, 23, 45, 86, 26, 38, 34, 67)

# Create pentiles based on the values of the vector
pentileScores <- pentile(vector = scores)
pentileScores <- pentile(vector = scores, decreasing = TRUE)
```
randomise

*Order the rows of a data randomly*

**Description**
Takes in data and seed, and returns the data with randomly ordered observations

**Usage**
randomise(data, seed = NULL)

**Arguments**
data
- a matrix, data.frame or data.table
seed
- an integer value

**Details**
Some of the modeling algorithms pick top p percent of the observations for training the model, which could lead to skewed predictions. This function solves that problem by randomly ordering the observations so that the response variable has more or less the same distribution even if the algorithms don’t pick training observations randomly.

**Value**
data of same class as input with randomly ordered observations

**Author(s)**
Akash Jain

**See Also**
factorise, rmdupkey, rmdupobs

**Examples**
```r
# A 'data.frame'
df <- data.frame(x = c(1, 2, 3, 4, 5), y = c('a', 'b', 'c', 'd', 'e'))

# Change the order of the observations randomly
dfRan <- randomise(data = df)
dfRan <- randomise(data = df, seed = 150)
```
rmdupkey

Remove observations with duplicate keys from data

Description

Takes in a data and key, and returns data with duplicate observations by key removed

Usage

rmdupkey(data, by)

Arguments

data a data.frame or data.table
by a character vector of keys to be used

Details

Remove duplicate observations by key(s) is what this function does. How it is different from other functions that remove duplicates is that rmdupkey works for both 'data.frame' and 'data.table', and it also returns the duplicated observations.

Many a times we want to go back to the duplicated observations and see why that duplication occurred. One can pick the duplicated observations using the code given in example.

Value

a two element list: unique data and duplicate data

Author(s)

Akash Jain

See Also

randomise, factorise, rmdupobs

Examples

# A 'data.frame'
df <- data.frame(x = c(1, 2, 1), y = c(3, 3, 1, 3))

# Remove duplicate observations by key from data
ltDF <- rmdupkey(data = df, by = c('x'))
unqDF <- ltDF$unqData
dupDF <- ltDF$dupData
rmdupobs  Remove duplicate observations from data

Description

Takes in a data, and returns it with duplicate observations removed

Usage

rmdupobs(data)

Arguments

data a data.frame or data.table

Details

Duplicate observations are redundant and they need to be removed from the data. rmdupobs does just that; it removes the duplicated observations (the ones in which value of every variable is duplicated) and returns the data with only unique observations.

It works for both 'data.frame' and 'data.table' and returns the data with same class as that of input.

Value

a data of same class as input with only unique observations

Author(s)

Akash Jain

See Also

randomise, rmdupkey, factorise

Examples

# A 'data.frame'
df <- data.frame(x = c(1, 2, 5, 1), y = c(3, 3, 1, 3))

# Remove duplicate observations from data
dfUnq <- rmdupobs(data = df)
splitdata

Split modeling data into test and train set

Description

Takes in data, fraction (for train set) and seed, and returns train and test set

Usage

splitdata(data, fraction, seed = NULL)

Arguments

data a matrix, data.frame or data.table

fraction proportion of observations that should go in the train set

seed an integer value

Details

An essential task before doing modeling is to split the modeling data into train and test sets. splitdata is built for this task and returns a list with train and test sets, which can be picked using the code given in example.

fraction corresponds to the train dataset, while the rest of the observations go to the test dataset. If the user wants to generate the same test and train dataset everytime, he should specify a seed value.

Value

a list with two elements: train and test set

Author(s)

Akash Jain

See Also

actvspred, mape, accuracy, auc, iv, ks

Examples

# A 'data.frame'
df <- data.frame(x = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
                 y = c('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'),
                 z = c(1, 1, 0, 0, 1, 0, 0, 1, 1, 0))

# Split data into train (70%) and test (30%)
ltData <- splitdata(data = df, fraction = 0.7, seed = 123)
trainData <- ltData$train
testData <- ltData$test
Index

accuracy, 2, 4, 13, 14, 21
actvspred, 3, 15, 21
auc, 3, 4, 12–14, 21
contents, 5, 8–10
decile, 6, 12, 16, 17
dq categorical, 5, 7, 9, 10
dq continuous, 5, 8, 8, 10
dq date, 5, 8, 9, 9
factorise, 10, 18–20
gini, 4, 11
imputemiss, 6, 12, 16, 17
iv, 3, 4, 13, 14, 21
ks, 3, 4, 13, 14, 21
mape, 4, 15, 21
outliers, 6, 12, 16, 17
pentile, 6, 12, 16, 17
randomise, 11, 18, 19, 20
rmdupkey, 11, 18, 19, 20
rmdupobs, 11, 18, 19, 20
splitdata, 3, 4, 13–15, 21