Package ‘insurancerating’

October 28, 2020

Type Package

Title Analytic Insurance Rating Techniques

Version 0.6.3

Maintainer Martin Haringa <mtharinga@gmail.com>

Description Methods for insurance rating. It helps actuaries to implement GLMs within all relevant steps needed to construct a risk premium from raw data. It provides a data driven strategy for the construction of insurance tariff classes. This strategy is based on the work by Antonio and Valdez (2012) <doi:10.1007/s10182-011-0152-7>. It also provides recipes on how to easily perform one-way, or univariate, analyses on an insurance portfolio. In addition it adds functionality to include reference categories in the levels of the coefficients in the output of a generalized linear regression analysis.

License GPL (>= 2)

URL https://github.com/mharinga/insurancerating,
https://mharinga.github.io/insurancerating/

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Imports ciTools, classInt, data.table, DHARMa, dplyr, evtree, ggplot2, insight, lubridate, magrittr, mgcv, patchwork, stringr, tidyr

Depends R (>= 3.3)

Suggests knitr, rmarkdown, scales, testthat

NeedsCompilation no

Author Martin Haringa [aut, cre]

Repository CRAN

Date/Publication 2020-10-28 14:30:03 UTC
## R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_prediction</td>
<td>2</td>
</tr>
<tr>
<td>autoplot.bootstrap_rmse</td>
<td>3</td>
</tr>
<tr>
<td>autoplot.check_residuals</td>
<td>4</td>
</tr>
<tr>
<td>autoplot.constructtariffclasses</td>
<td>4</td>
</tr>
<tr>
<td>autoplot.fitgam</td>
<td>6</td>
</tr>
<tr>
<td>autoplot.riskfactor</td>
<td>7</td>
</tr>
<tr>
<td>autoplot.univariate</td>
<td>8</td>
</tr>
<tr>
<td>biggest_reference</td>
<td>10</td>
</tr>
<tr>
<td>bootstrap_rmse</td>
<td>11</td>
</tr>
<tr>
<td>check_overdispersion</td>
<td>12</td>
</tr>
<tr>
<td>check_residuals</td>
<td>13</td>
</tr>
<tr>
<td>construct_tariff_classes</td>
<td>14</td>
</tr>
<tr>
<td>fisher</td>
<td>16</td>
</tr>
<tr>
<td>fit_gam</td>
<td>17</td>
</tr>
<tr>
<td>histbin</td>
<td>19</td>
</tr>
<tr>
<td>model_performance</td>
<td>20</td>
</tr>
<tr>
<td>MTPL</td>
<td>21</td>
</tr>
<tr>
<td>MTPL2</td>
<td>22</td>
</tr>
<tr>
<td>period_to_months</td>
<td>22</td>
</tr>
<tr>
<td>rating_factors</td>
<td>23</td>
</tr>
<tr>
<td>rating_factors1</td>
<td>24</td>
</tr>
<tr>
<td>reduce</td>
<td>25</td>
</tr>
<tr>
<td>rmse</td>
<td>27</td>
</tr>
<tr>
<td>summary.reduce</td>
<td>28</td>
</tr>
<tr>
<td>univariate</td>
<td>29</td>
</tr>
</tbody>
</table>

### Description

Add model predictions and confidence bounds to a data frame.

### Usage

```r
add_prediction(data, ..., var = NULL, conf_int = FALSE, alpha = 0.1)
```

### Arguments

- **data**: a data frame of new data.
- **...**: one or more objects of class `glm`.
- **var**: the name of the output column(s), defaults to `NULL`.
conf_int determines whether confidence intervals will be shown. Defaults to conf_int = FALSE.

alpha a real number between 0 and 1. Controls the confidence level of the interval estimates (defaults to 0.10, representing 90 percent confidence interval).

Value
data.frame

Examples

mod1 <- glm(nclaims ~ age_policyholder, data = MTPL, offset = log(exposure), family = poisson())
add_prediction(MTPL, mod1)

# Include confidence bounds
add_prediction(MTPL, mod1, conf_int = TRUE)

autoplot.bootstrap_rmse

Automatically create a ggplot for objects obtained from bootstrap_rmse()

Description
Takes an object produced by bootstrap_rmse(), and plots the simulated RMSE

Usage

## S3 method for class 'bootstrap_rmse'
autoplot(object, ...)

Arguments

object bootstrap_rmse object produced by bootstrap_rmse()
...
other plotting parameters to affect the plot

Value

a ggplot object

Author(s)

Martin Haringa
Description

Takes an object produced by `check_residuals()`, and produces a uniform quantile-quantile plot.

Usage

```r
## S3 method for class 'check_residuals'
autoplot(object, show_message = TRUE, ...)
```

Arguments

- `object`: check_residuals object produced by `check_residuals()`
- `show_message`: show output from test (defaults to TRUE)
- `...`: other plotting parameters to affect the plot

Value

a ggplot object

Author(s)

Martin Haringa

Description

Takes an object produced by `construct_tariff_classes()`, and plots the fitted GAM. In addition the constructed tariff classes are shown.
Usage

```r
## S3 method for class 'constructtariffclasses'
autoplot(
  object,
  conf_int = FALSE,
  color_gam = "steelblue",
  show_observations = FALSE,
  color_splits = "grey50",
  size_points = 1,
  color_points = "black",
  rotate_labels = FALSE,
  remove_outliers = NULL,
  ...
)
```

Arguments

- `object`: `constructtariffclasses` object produced by `construct_tariff_classes`.
- `conf_int`: determines whether 95% confidence intervals will be plotted. The default is `conf_int = FALSE`.
- `color_gam`: a color can be specified either by name (e.g., "red") or by hexadecimal code (e.g., "#FF1234") (default is "steelblue").
- `show_observations`: add observed frequency/severity points for each level of the variable for which tariff classes are constructed.
- `color_splits`: change the color of the splits in the graph ("grey50" is default).
- `size_points`: size for points (1 is default).
- `color_points`: change the color of the points in the graph ("black" is default).
- `rotate_labels`: rotate x-labels 45 degrees (this might be helpful for overlapping x-labels).
- `remove_outliers`: do not show observations above this number in the plot. This might be helpful for outliers.
- `...`: other plotting parameters to affect the plot.

Value

- a `ggplot` object

Author(s)

- Martin Haringa

Examples

```r
## Not run:
library(ggplot2)
```
library(dplyr)
fit_gam(MTPL, nclaims = nclaims, x = age_policyholder, exposure = exposure) %>%
  construct_tariff_classes(.) %>%
  autoplot(. , show_observations = TRUE)

## End(Not run)

autoplot.fitgam    Automatically create a ggplot for objects obtained from fit_gam()

Description

Takes an object produced by fit_gam(), and plots the fitted GAM.

Usage

## S3 method for class 'fitgam'
autoplot(
  object,
  conf_int = FALSE,
  color_gam = "steelblue",
  show_observations = FALSE,
  x_stepsize = NULL,
  size_points = 1,
  color_points = "black",
  rotate_labels = FALSE,
  remove_outliers = NULL,
  ...
)

Arguments

object       fitgam object produced by fit_gam()
conf_int     determines whether 95% confidence intervals will be plotted. The default is
             conf_int = FALSE
color_gam    a color can be specified either by name (e.g.: "red") or by hexadecimal code
             (e.g.: "#FF1234") (default is "steelblue")
show_observations  add observed frequency/severity points for each level of the variable for which
                   tariff classes are constructed
x_stepsize   set step size for labels horizontal axis
size_points  size for points (1 is default)
color_points change the color of the points in the graph ("black" is default)
rotate_labels rotate x-labels 45 degrees (this might be helpful for overlapping x-labels)
remove_outliers
do not show observations above this number in the plot. This might be helpful for outliers.
...
other plotting parameters to affect the plot

Value

a ggplot object

Author(s)

Martin Haringa

Examples

## Not run:
library(ggplot2)
library(dplyr)
fit_gam(MTPL, nclaims = nclaims, x = age_policyholder, exposure = exposure) %>%
  autoplot(., show_observations = TRUE)

## End(Not run)

 autoplot.riskfactor  Automatically create a ggplot for objects obtained from rating_factors()

Description

Takes an object produced by univariate(), and plots the available input.

Usage

## S3 method for class 'riskfactor'
autoplot(
  object,
  risk_factors = NULL,
  ncol = 1,
  labels = TRUE,
  dec.mark = ",",
  ylab = "rate",
  color_bg = "#E7B800",
  linetype = FALSE,
  ...
)
Arguments

- **object**: riskfactor object produced by `rating_factors()`
- **risk_factors**: character vector to define which factors are included. Defaults to all risk factors.
- **ncol**: number of columns in output (default is 1)
- **labels**: show labels with the exposure (default is TRUE)
- **dec.mark**: control the format of the decimal point, as well as the mark between intervals before the decimal point, choose either "." (default) or ",."
- **ylab**: modify label for the y-axis
- **color_bg**: change the color of the histogram ("#E7B800" is default)
- **linetype**: use different linetypes (default is FALSE)
- **...**: other plotting parameters to affect the plot

Value

A ggplot2 object

Examples

```r
library(dplyr)
df <- MTPL2 %>%
  mutate_at(vars(area), as.factor) %>%
  mutate_at(vars(area), ~biggest_reference(., exposure))

mod1 <- glm(nclaims ~ area + premium, offset = log(exposure), family = poisson(), data = df)
mod2 <- glm(nclaims ~ area, offset = log(exposure), family = poisson(), data = df)
x <- rating_factors(mod1, mod2, model_data = df, exposure = exposure)
autoplot(x)
```

Description

Automatically create a ggplot for objects obtained from `univariate()`.

Usage

```r
## S3 method for class 'univariate'
autoplot(
  object,
  show_plots = 1:9,
  ncol = 1,
  background = TRUE,
)```
autoplot.univariate

```r
labels = TRUE,
sort = FALSE,
sort_manual = NULL,
dec.mark = ",",
color = "dodgerblue",
color_bg = "#E7B800",
label_width = 10,
coord_flip = FALSE,
...
)
```

**Arguments**

- **object**: univariate object produced by `univariate()`
- **show_plots**: numeric vector of plots to be shown (default is c(1,2,3,4,5,6,7,8,9)), there are nine available plots:
  - 1. frequency (i.e. number of claims / exposure)
  - 2. average severity (i.e. severity / number of claims)
  - 3. risk premium (i.e. severity / exposure)
  - 4. loss ratio (i.e. severity / premium)
  - 5. average premium (i.e. premium / exposure)
  - 6. exposure
  - 7. severity
  - 8. nclaims
  - 9. premium
- **ncol**: number of columns in output (default is 1)
- **background**: show exposure as a background histogram (default is TRUE)
- **labels**: show labels with the exposure (default is TRUE)
- **sort**: sort (or order) risk factor into descending order by exposure (default is FALSE)
- **sort_manual**: sort (or order) risk factor into own ordering; should be a character vector (default is NULL)
- **dec.mark**: control the format of the decimal point, as well as the mark between intervals before the decimal point, choose either "," (default) or "."
- **color**: change the color of the points and line ("dodgerblue" is default)
- **color_bg**: change the color of the histogram ("#E7B800" is default)
- **label_width**: width of labels on the x-axis (10 is default)
- **coord_flip**: flip cartesian coordinates so that horizontal becomes vertical, and vertical, horizontal (default is FALSE)
- **...**: other plotting parameters to affect the plot

**Value**

- a ggplot2 object
Examples

```r
library(ggplot2)
x <- univariate(MTPL2, x = area, severity = amount, nclaims = nclaims, exposure = exposure)
autoplot(x)
autoplot(x, show_plots = c(6,1), background = FALSE, sort = TRUE)

MTPL2a <- MTPL2
MTPL2a$jaar <- sample(2015:2019, nrow(MTPL2a), replace = TRUE)
x1 <- univariate(MTPL2a, x = area, severity = amount, nclaims = nclaims,
exposure = exposure, by = jaar)
autoplot(x1, show_plots = 1:2)
```

biggest_reference

Set reference group to the group with largest exposure

Description

This function specifies the first level of a factor to the level with the largest exposure. Levels of factors are sorted using an alphabetic ordering. If the factor is used in a regression context, then the first level will be the reference. For insurance applications it is common to specify the reference level to the level with the largest exposure.

Usage

`biggest_reference(x, weight)`

Arguments

- `x`: an unordered factor
- `weight`: a vector containing weights (e.g. exposure). Should be numeric.

Value

A factor of the same length as `x`

Author(s)

Martin Haringa

References

### Examples

```r
## Not run:
library(dplyr)
 df <- chickwts %>%
  mutate(across(where(is.character), as.factor)) %>%
  mutate(across(where(is.factor), ~biggest_reference(., weight)))

## End(Not run)
```

---

**bootstrap_rmse**

*Bootstrap RMSE*

#### Description

Generate \( n \) bootstrap replicates to compute \( n \) root mean squared errors.

#### Usage

```r
bootstrap_rmse(
  model,
  data,
  n = 50,
  frac = 1,
  show_progress = TRUE,
  rmse_model = NULL
)
```

#### Arguments

- `model`: a model object
- `data`: data used to fit model object
- `n`: number of bootstrap replicates (defaults to 50)
- `frac`: fraction used in training set if cross-validation is applied (defaults to 1)
- `show_progress`: show progress bar (defaults to TRUE)
- `rmse_model`: numeric RMSE to show as vertical dashed line in autoplot() (defaults to NULL)

#### Details

To test the predictive ability of the fitted model it might be helpful to determine the variation in the computed RMSE. The variation is calculated by computing the root mean squared errors from \( n \) generated bootstrap replicates. More precisely, for each iteration a sample with replacement is taken from the data set and the model is refitted using this sample. Then, the root mean squared error is calculated.
Check overdispersion of Poisson GLM

Description

Check Poisson GLM for overdispersion.

Usage

check_overdispersion(object)

Arguments

object fitted model of class glm and family Poisson
Details

A dispersion ratio larger than one indicates overdispersion, this occurs when the observed variance is higher than the variance of the theoretical model. If the dispersion ratio is close to one, a Poisson model fits well to the data. A p-value < .05 indicates overdispersion. Overdispersion > 2 probably means there is a larger problem with the data: check (again) for outliers, obvious lack of fit. Adopted from performance::check_overdispersion().

Value

A list with dispersion ratio, chi-squared statistic, and p-value.

Author(s)

Martin Haringa

References

• Bolker B et al. (2017): GLMM FAQ.

Examples

```r
x <- glm(nclaims ~ area, offset = log(exposure), family = poisson(), data = MTPL2)
check_overdispersion(x)
```

---

`check_residuals`  
Check model residuals

Description

Detect overall deviations from the expected distribution.

Usage

```r
check_residuals(object, n_simulations = 30)
```

Arguments

- `object`: a model object
- `n_simulations`: number of simulations (defaults to 30)
Details

Misspecifications in GLMs cannot reliably be diagnosed with standard residual plots, and GLMs are thus often not as thoroughly checked as LMs. One reason why GLMs residuals are harder to interpret is that the expected distribution of the data changes with the fitted values. As a result, standard residual plots, when interpreted in the same way as for linear models, seem to show all kind of problems, such as non-normality, heteroscedasticity, even if the model is correctly specified. `check_residuals()` aims at solving these problems by creating readily interpretable residuals for GLMs that are standardized to values between 0 and 1, and that can be interpreted as intuitively as residuals for the linear model. This is achieved by a simulation-based approach, similar to the Bayesian p-value or the parametric bootstrap, that transforms the residuals to a standardized scale. This explanation is adopted from `simulateResiduals`.

Value

Invisibly returns the p-value of the test statistics. A p-value < 0.05 indicates a significant deviation from expected distribution.

Author(s)

Martin Haringa

References


Gelman, A. & Hill, J. Data analysis using regression and multilevel/hierarchical models Cambridge University Press, 2006


Examples

```r
## Not run:
m1 <- glm(nclaims ~ area, offset = log(exposure), family = poisson(), data = MTPL2)
check_residuals(m1, n_simulations = 50) %>% autoplot()
## End(Not run)
```
Description

Constructs insurance tariff classes to fitgam objects produced by fit_gam. The goal is to bin the continuous risk factors such that categorical risk factors result which capture the effect of the covariate on the response in an accurate way, while being easy to use in a generalized linear model (GLM).

Usage

```r
construct_tariff_classes(
  object,
  alpha = 0,
  niterations = 10000,
  ntrees = 200,
  seed = 1
)
```

Arguments

- `object`: fitgam object produced by `fit_gam`
- `alpha`: complexity parameter. The complexity parameter (alpha) is used to control the number of tariff classes. Higher values for alpha render less tariff classes. (alpha = 0 is default).
- `niterations`: in case the run does not converge, it terminates after a specified number of iterations defined by `niterations`.
- `ntrees`: the number of trees in the population.
- `seed`: an numeric seed to initialize the random number generator (for reproducibility).

Details

Evolutionary trees are used as a technique to bin the fitgam object produced by fit_gam into risk homogeneous categories. This method is based on the work by Henckaerts et al. (2018). See Grubinger et al. (2014) for more details on the various parameters that control aspects of the evtree fit.

Value

A list of class `constructTariffClasses` with components

- `prediction`: data frame with predicted values
- `x`: name of continuous risk factor for which tariff classes are constructed
- `model`: either 'frequency', 'severity' or 'burning'
- `data`: data frame with predicted values and observed values
- `x_obs`: observations for continuous risk factor
- `splits`: vector with boundaries of the constructed tariff classes
- `tariff_classes`: values in vector `x` coded according to which constructed tariff class they fall
ficher

Fisher’s natural breaks classification

Description
The function provides an interface to finding class intervals for continuous numerical variables, for example for choosing colours for plotting maps.

Usage
ficher(vec, n = 7, diglab = 2)

Arguments
vec a continuous numerical variable
n number of classes required (n = 7 is default)
diglab number of digits (n = 2 is default)
Details

The "fisher" style uses the algorithm proposed by W. D. Fisher (1958) and discussed by Slocum et al. (2005) as the Fisher-Jenks algorithm. This function is adopted from the classInt package.

Value

Vector with clustering

Author(s)

Martin Haringa

References


Description

Fits a generalized additive model (GAM) to continuous risk factors in one of the following three types of models: the number of reported claims (claim frequency), the severity of reported claims (claim severity) or the burning cost (i.e. risk premium or pure premium).

Usage

```r
fit_gam(
  data, nclaims, x, exposure, amount = NULL, pure_premium = NULL, model = "frequency", round_x = NULL
)
```

**Arguments**

- **data**: data.frame of an insurance portfolio
- **nclaims**: column in data with number of claims
- **x**: column in data with continuous risk factor
- **exposure**: column in data with exposure
- **amount**: column in data with claim amount
- **pure_premium**: column in data with pure premium
- **model**: choose either 'frequency', 'severity' or 'burning' (model = 'frequency' is default). See details section.
- **round_x**: round elements in column x to multiple of round_x. This gives a speed enhancement for data containing many levels for x.

**Details**

The 'frequency' specification uses a Poisson GAM for fitting the number of claims. The logarithm of the exposure is included as an offset, such that the expected number of claims is proportional to the exposure.

The 'severity' specification uses a lognormal GAM for fitting the average cost of a claim. The average cost of a claim is defined as the ratio of the claim amount and the number of claims. The number of claims is included as a weight.

The 'burning' specification uses a lognormal GAM for fitting the pure premium of a claim. The pure premium is obtained by multiplying the estimated frequency and the estimated severity of claims. The word burning cost is used here as equivalent of risk premium and pure premium. Note that the functionality for fitting a GAM for pure premium is still experimental (in the early stages of development).

**Value**

A list with components

- **prediction**: data frame with predicted values
- **x**: name of continuous risk factor
- **model**: either 'frequency', 'severity' or 'burning'
- **data**: data frame with predicted values and observed values
- **x_obs**: observations for continuous risk factor

**Author(s)**

Martin Haringa
References


Examples

```r
fit_gam(MTPL, nclaims = nclaims, x = age_policyholder, exposure = exposure)
```

---

**histbin**

Create a histogram with outlier bins

**Description**

Visualize the distribution of a single continuous variable by dividing the x axis into bins and counting the number of observations in each bin. Data points that are considered outliers can be binned together. This might be helpful to display numerical data over a very wide range of values in a compact way.

**Usage**

```r
histbin(
  data,  # data.frame
  x,  # variable name in data.frame 'data' that should be mapped
  left = NULL,
  right = NULL,
  line = FALSE,
  bins = 30,
  fill = "#E7B800",
  fill_outliers = "#a7d1a7"
)
```

**Arguments**

- `data` : data.frame
- `x` : variable name in data.frame 'data' that should be mapped
model_performance

left numeric indicating the floor of the range
right numeric indicating the ceiling of the range
line show density line (default is FALSE)
bins numeric to indicate number of bins
fill color used to fill bars
fill_outliers color used to fill outlier bars

Details
Wrapper function around ‘ggplot2::geom_histogram()’. The method is based on suggestions from https://edwinth.github.io/blog/outlier-bin/.

Value
a ggplot2 object

Examples
histbin(MTPL2, premium)
histbin(MTPL2, premium, left = 30, right = 120, line = TRUE, bins = 30)

model_performance Performance of fitted GLMs

Description
Compute indices of model performance for (one or more) GLMs.

Usage
model_performance(...)

Arguments
... One or more objects of class glm.

Details
The following indices are computed:

- **AIC** Akaike’s Information Criterion, see AIC
- **BIC** Bayesian Information Criterion, see BIC
- **RMSE** Root mean squared error, rmse

Adopted from performance::model_performance().
MTPL

Value
   data frame

Author(s)
   Martin Haringa

Examples
   m1 <- glm(nclaims ~ area, offset = log(exposure), family = poisson(), data = MTPL2)
   m2 <- glm(nclaims ~ area, offset = log(exposure), family = poisson(), data = MTPL2)
   model_performance(m1, m2)

Description
   A dataset containing the age, number of claims, and exposure of almost 33,000 policyholders

Usage
   MTPL

Format
   A data frame with 32,731 rows and 4 variables:

   age_policyholder  age of policyholder, in years.
   nclaims          number of claims.
   exposure          exposure, for example, if a vehicle is insured as of July 1 for a certain year, then during that year, this would represent an exposure of 0.5 to the insurance company.
   amount            claim amount in Euros.

Author(s)
   Martin Haringa

Source
   The data is derived from the portfolio of a large Dutch motor insurance company.
MTPL2

Characteristics of 3,000 policyholders in a Motor Third Party Liability (MTPL) portfolio.

Description
A dataset containing the area, number of claims, exposure, claim amount, exposure, and premium of 3,000 policyholders

Usage
MTPL2

Format
A data frame with 3,000 rows and 6 variables:

- customer_id: customer id
- area: region where customer lives
- nclaims: number of claims
- amount: claim amount (severity)
- exposure: exposure
- premium: earned premium

Author(s)
Martin Haringa

Source
The data is derived from the portfolio of a large Dutch motor insurance company.

period_to_months

Split period to months

Description
The function splits rows with a time period longer than one month to multiple rows with a time period of exactly one month each. Values in numeric columns (e.g. exposure or premium) are divided over the months proportionately.

Usage
period_to_months(df, begin, end, ...)

Arguments

- df: data.frame
- begin: column in df with begin dates
- end: column in df with end dates
- ...: numeric columns in df to split

Details

In insurance portfolios it is common that rows relate to periods longer than one month. This is for example problematic in case exposures per month are desired.

Since insurance premiums are constant over the months, and do not depend on the number of days per month, the function assumes that each month has the same number of days (i.e. 30).

Value

data.frame with same columns as in df, and one extra column called id

Author(s)

Martin Haringa

Examples

```r
library(lubridate)
portfolio <- data.frame(
    begin1 = ymd(c("2014-01-01", "2014-01-01")),
    end = ymd(c("2014-03-14", "2014-05-10")),
    termination = ymd(c("2014-03-14", "2014-05-10")),
    exposure = c(0.2025, 0.3583),
    premium = c(125, 150))
period_to_months(portfolio, begin1, end, premium, exposure)
```

rating_factors

Include reference group in regression output

Description

Extract coefficients in terms of the original levels of the coefficients rather than the coded variables.

Usage

```r
rating_factors(
    ...,
    model_data = NULL,
    exposure = NULL,
    exponentiate = TRUE,
    signif_stars = TRUE
)
```
Arguments

... glm object(s) produced by glm()
model_data data.frame used to create glm object(s), this should only be specified in case the exposure is desired in the output, default value is NULL
exposure column in model_data with exposure, default value is NULL
exponentiate logical indicating whether or not to exponentiate the the coefficient estimates. Defaults to TRUE.
signif_stars show significance stars for p-values (defaults to TRUE)

Details

A fitted linear model has coefficients for the contrasts of the factor terms, usually one less in number than the number of levels. This function re-expresses the coefficients in the original coding. This function is adopted from dummy.coef(). Our adoption prints a data.frame as output.

Value
data.frame

Author(s)

Martin Haringa

Examples

library(dplyr)
df <- MTPL2 %>%
  mutate_at(vars(area), as.factor) %>%
  mutate_at(vars(area), ~biggest_reference(., exposure))

mod1 <- glm(nclaims ~ area + premium, offset = log(exposure), family = poisson(), data = df)
mod2 <- glm(nclaims ~ area, offset = log(exposure), family = poisson(), data = df)

rating_factors(mod1, mod2, model_data = df, exposure = exposure)

Description

Extract coefficients in terms of the original levels of the coefficients rather than the coded variables. Use rating_factors() to compare the output obtained from two or more glm objects.
Usage

rating_factors1(
  model,
  model_data = NULL,
  exposure = NULL,
  colname = "estimate",
  exponentiate = TRUE
)

Arguments

model a single glm object produced by glm()
model_data data.frame used to create glm object, this should only be specified in case the exposure is desired in the output, default value is NULL
exposure the name of the exposure column in model_data, default value is NULL
colname the name of the output column, default value is "estimate"
exponentiate logical indicating whether or not to exponentiate the the coefficient estimates. Defaults to TRUE.

Examples

MTPL2a <- MTPL2
MTPL2a$area <- as.factor(MTPL2a$area)
x <- glm(nclaims ~ area, offset = log(exposure), family = poisson(), data = MTPL2a)


reduce
Reduce portfolio by merging redundant date ranges

Description

Transform all the date ranges together as a set to produce a new set of date ranges. Ranges separated by a gap of at least min.gapwidth days are not merged.

Usage

reduce(df, begin, end, ..., agg_cols = NULL, agg = "sum", min.gapwidth = 5)

Arguments

df data.frame
begin name of column df with begin dates
end name of column in df with end dates
... names of columns in df used to group date ranges by
**reduce**

**agg_cols**  
list with columns in df to aggregate by (defaults to NULL)

**agg**  
aggregation type (defaults to "sum")

**min.gapwidth**  
ranges separated by a gap of at least min.gapwidth days are not merged. Defaults to 5.

**Details**

This function is adopted from IRanges::reduce().

**Value**

An object of class "reduce". The function summary is used to obtain and print a summary of the results. An object of class "reduce" is a list usually containing at least the following elements:

- **df**  
data frame with reduced time periods
- **begin**  
name of column in df with begin dates
- **end**  
name of column in df with end dates
- **cols**  
names of columns in df used to group date ranges by

**Author(s)**

Martin Haringa

**Examples**

```r
portfolio <- structure(list(policy_nr = c("12345", "12345", "12345", "12345",  
"12345", "12345", "12345", "12345", "12345", "12345",  
productgroup = c("fire", "fire", "fire", "fire", "fire", "fire",  
"fire", "fire", "fire", "fire"), product = c("contents",  
"contents", "contents", "contents", "contents", "contents",  
"contents", "contents", "contents", "contents"), begin_dat = structure(c(16709,  
16740, 17410, 17440, 17805, 17897, 17956, 17987, 18017,  
18262), class = "Date"), end_dat = structure(c(16739, 16800,  
16831, 17439, 17531, 17896, 17955, 17986, 18016, 18261, 18292),  
class = "Date"), premium = c(89L, 58L, 83L, 73L, 69L, 94L,  
91L, 97L, 57L, 65L, 55L)), row.names = c(NA, -11L), class = "data.frame")

# Merge periods
reduce(portfolio, begin = begin_dat, end = end_dat, policy_nr,  
productgroup, product, min.gapwidth = 5)

# Merge periods and sum premium per period
reduce(portfolio, begin = begin_dat, end = end_dat, policy_nr,  
productgroup, product, agg_cols = list(premium), min.gapwidth = 5)
```
Description

Compute root mean squared error.

Usage

\texttt{rmse(object, data)}

Arguments

- \texttt{object}: fitted model
- \texttt{data}: data.frame (defaults to NULL)

Details

The RMSE is the square root of the average of squared differences between prediction and actual observation and indicates the absolute fit of the model to the data. It can be interpreted as the standard deviation of the unexplained variance, and is in the same units as the response variable. Lower values indicate better model fit.

Value

numeric value

Author(s)

Martin Haringa

Examples

\begin{verbatim}
x <- glm(nclaims - area, offset = log(exposure), family = poisson(), data = MTPL2)
rmse(x, MTPL2)
\end{verbatim}
**Summary.reduce**

Automatically create a summary for objects obtained from reduce()

### Description

Takes an object produced by reduce(), and counts new and lost customers.

### Usage

```r
## S3 method for class 'reduce'
summary(object, period = "days", ...)
```

### Arguments

- `object`: reduce object produced by reduce()
- `period`: a character string indicating the period to aggregate on. Four options are available: "quarters", "months", "weeks", and "days" (the default option)
- `...`: names of columns to aggregate counts by

### Value

data.frame

### Examples

```r
portfolio <- structure(list(policy_nr = c("12345", "12345", "12345", "12345", 
"12345", "12345", "12345", "12345", "12345", "12345"),
productgroup = c("fire", "fire", "fire", "fire", "fire", "fire", 
"fire", "fire", "fire", "fire"), product = c("contents", 
"contents", "contents", "contents", "contents", "contents", 
"contents", "contents", "contents", "contents"), begin_dat = structure(c(16709, 
16740, 16801, 17410, 17440, 17805, 17897, 17956, 17987, 18017, 
18262), class = "Date"), end_dat = structure(c(16739, 16800, 
16831, 17439, 17531, 17896, 17955, 17986, 18016, 18261, 18292), 
class = "Date"), premium = c(89L, 58L, 83L, 73L, 69L, 94L, 
91L, 9L, 5L, 65L, 55L)), row.names = c(NA, -11L), class = "data.frame")
pt1 <- reduce(portfolio, begin = begin_dat, end = end_dat, policy_nr, 
productgroup, product, min.gapwidth = 5)
summary(pt1, period = "days", policy_nr, productgroup, product)
pt2 <- reduce(portfolio, begin = begin_dat, end = end_dat, policy_nr, 
productgroup, product, agg.cols = list(premium), min.gapwidth = 5)
summary(pt2, period = "weeks", policy_nr, productgroup, product)
```
Univariate analysis for discrete risk factors

Description

Univariate analysis for discrete risk factors in an insurance portfolio. The following summary statistics are calculated:

- frequency (i.e. number of claims / exposure)
- average severity (i.e. severity / number of claims)
- risk premium (i.e. severity / exposure)
- loss ratio (i.e. severity / premium)
- average premium (i.e. premium / exposure)

If input arguments are not specified, the summary statistics related to these arguments are ignored.

Usage

```r
univariate(
  df,
  x,
  severity = NULL,
  nclaims = NULL,
  exposure = NULL,
  premium = NULL,
  by = NULL
)
```

Arguments

df             data.frame with insurance portfolio
x              column in df with risk factor
severity       column in df with severity (default is NULL)
nclaims        column in df with number of claims (default is NULL)
exposure       column in df with exposure (default is NULL)
premium        column in df with premium (default is NULL)
by             column(s) in df to group by

Value

A list of class univ_all with components

df             data frame
xvar           name of column in df with risk factor
severity name of column in df with severity
nclaims name of column in df with number of claims
exposure name of column in df with exposure
premium name of column in df with premium

Examples

univariate(MTPL2, x = area, severity = amount, nclaims = nclaims,
               exposure = exposure, premium = premium)

# The summary statistics related to premium are not calculated
univariate(MTPL2, x = area, severity = amount, nclaims = nclaims, exposure = exposure)
Index

* datasets
  MTPL, 21
  MTPL2, 22

  add_prediction, 2
  AIC, 20
  autoplot.bootstrap_rmse, 3
  autoplot.check_residuals, 4
  autoplot.construct_tariff_classes, 4
  autoplot.fitgam, 6
  autoplot.riskfactor, 7
  autoplot.univariate, 8

  BIC, 20
  biggest_reference, 10
  bootstrap_rmse, 11

  check_overdispersion, 12
  check_residuals, 13
  construct_tariff_classes, 14

  fisher, 16
  fit_gam, 17

  histbin, 19

  model_performance, 20
  MTPL, 21
  MTPL2, 22

  period_to_months, 22

  rating_factors, 23
  rating_factors1, 24
  reduce, 25
  rmse, 20, 27

  simulateResiduals, 14
  summary.reduce, 28

  univariate, 29