Package 'ToTweedieOrNot'

December 1, 2014

Type Package

Title Code for the paper Generalised linear models for aggregate claims; to Tweedie or not?

Version 1.0

Date 2014-11-27

Author Oscar Alberto Quijano Xacur

Maintainer Oscar Alberto Quijano Xacur <oscar.quijano@use.startmail.com>

Description Main functions used for the simulations and graphs of the paper Generalised linear models for aggregate claims; to Tweedie or not?

License GPL (>=3)

R topics documented:

`ToTweedieOrNot-package`

Code for the Article "Generalised linear models for aggregate claims; to Tweedie or not?"

Description

This package contains the main functions used for the simulations and graphs of the article "Generalised linear models for aggregate claims; to Tweedie or not?"

Details

Package:	ToTweedieOrNot
Type:	Package
Version:	1.0
Date:	2014-11-27
License:	GPL(>=3)

Author(s)

Oscar Alberto Quijano Xacur.

Maintainer: Oscar Alberto Quijano Xacur <oscar.quijano@use.startmail.com>.

References

Quijano Xacur, Oscar A. and Garrido, José. (2014) *Generalised linear models for aggregate claims; to Tweedie or not?*

Jørgensen, Bent.(1987) *The Theory of Dispersion Models* Chapman and Hall.

`generate.aggregated.loss.data` *Simulate aggregated loss data.*

Description

This function returns simulated data for aggregate loss given GLM parameters for frequency and severity. It assumes the same covariates for these two.

Usage

```
generate.aggregated.loss.data(classes.vector,
                              beta.vector.freq,
                              beta.vector.sev,
                              phi.freq,
                              phi.sev,
                              inverse.link.function.freq,
                              inverse.link.function.sev,
                              observations.per.class.freq,
                              names,
                              sim.function.freq,
                              sim.function.sev,
                              response.variable.name = "Response",
                              weights.function.freq)
```

Arguments

`classes.vector`	Vector of integers. Each integer represents the number of classes for each variable.
`beta.vector.freq`	Coefficients vector for the frequency model.
`beta.vector.sev`	Coefficients vector for the severity model.
`phi.freq`	Dispersion parameter for the frequency model.
`phi.sev`	Dispersion parameter for the severity model.
`inverse.link.function.freq`	Inverse of the link function for the frequency model.
`inverse.link.function.sev`	Inverse of the link function for the severity model.
`observations.per.class.freq`	Number of observations for each class for the frequency simulation.
`names`	Vector of strings. The strings correspond to the names of the variables.
`sim.function.freq`	Function that receives the mean-parameter and dispersion parameter and gives back a simulated value according to the desired random variable for the frequency simulation.
`sim.function.sev`	Function that receives the mean-parameter and dispersion parameter and gives back a simulated value according to the desired random variable for the severity simulation.
`response.variable.name`	Name for the response variable of the aggregated data.
`weights.function.freq`	Function that receives an integer and gives back a vector of weights. This weights are for the frequency model.

Details

This function generates aggregated data assuming independence between frequency and severity.

The coefficient vectors, observations per class, names and classes.vector should be as for the function generate.simulated.glm.data separately for frequency and severity.

There is no need for a function that gives weights for the severity. These weights are the number of claims given by the frequency simulation.

Value

This function returns a data frame whose entries are the simulated responses, and the covariates. The name of the simulated response in the list is the value given to the variable response.variable.name. This list also has the following entries

`Claims`	Number of claims for each class.
`Exposure`	Total weight for each class.

Note

See notes for the function generate.simulated.glm.data

Author(s)

Oscar Alberto Quijano Xacur <oscar.quijano@use.startmail.com>.

Examples

```
sim.poisson.function <- function(mean=lambda,phi=1){
    rpois(1,lambda=mean/phi)
}

sim.gamma.function <- function(mean,phi){
  rgamma(1,shape=mean,rate=phi);
}

generate.weights <- function(n){
  # Function that generayes n weights.
  p <- 0.7 #probability of weight being 1
    bern <- rbinom(n,size=1,prob=p);
    bern+(1-bern)*runif(n)
 }

gamma.beta.vector <- c(5,-0.1,-0.2,-0.3,-0.4,-0.5,-0.6,-0.7,-0.8,-0.9,-0.10,-0.11,-0.12);
poisson.beta.vector <- -2*gamma.beta.vector;
poisson.beta.vector[1] <- 3;
observations.for.poisson.classes <- sample(200:300,60,replace=TRUE);

# This simulates aggregated loss data assuming a Poisson distribution
# for the frequency and a gamma distribution for the severity.
aggregated.data <- generate.aggregated.loss.data(
                          classes.vector=c(2,3,10),
                          beta.vector.freq=poisson.beta.vector,
                          beta.vector.sev=gamma.beta.vector,
                          phi.freq=1,
                          phi.sev=100,
                          inverse.link.function.freq=exp,
                          inverse.link.function.sev=exp,
                          observations.per.class.freq=observations.for.poisson.classes,
                          names=c("Gender","Driver","Make"),
                          sim.function.freq=sim.poisson.function,
                          sim.function.sev=sim.gamma.function,
                          response.variable.name="Loss",
                          weights.function.freq=generate.weights
                          )
```

`generate.simulated.glm.data` *Simulate GLM data.*

Description

Simulates GLM data given the response, link function, weights and response distribution. It assumes that all the covariates are categorical.

Usage

```
generate.simulated.glm.data(classes.vector,
                            beta,
                            phi,
                            inverse.link.function,
                            observations.per.class,
                            names,
                            sim.function,
                            response.variable.name = "Response",
                            weights.function,
                            is.tweedie = FALSE,
                            ...)
```

Arguments

`classes.vector`	Vector of integers. Each integer represents the number of classes for each variable.
`beta`	Coefficients vector.
`phi`	Dispersion parameter.
`inverse.link.function`	Inverse of the link function.
`observations.per.class`	Number of observations for each class.
`names`	Vector of strings. The strings correspond to the names of the variables.
`sim.function`	Function that receives the mean-parameter and dispersion parameter and gives back a simulated value according to the desired random variable.
`response.variable.name`	String that has the name of the response variable.
`weights.function`	Function that receives an integer and gives back a vector of weights.
`is.tweedie`	Should be TRUE if the response distribution is tweedie with variance function power between 1 and 2.
`...`	Used in case it is needed to pass extra parameters to the simulation function.

Details

beta should have 1+sum(classes.vector[i]-1) elements. observations.per.class should have prod(classes.vector) elements. names and classes.vector should have the same length.

Value

This function returns a data frame whose entries are the simulated responses, and the covariates. The name of the simulated response in the list is the value given to the variable response.variable.name. There is also an entry called Exposure which has the total weight for each class.

Note

Notice that for a discrete random variable the mean of the distribution is given by mean-parameter/disp while for a continuous random variable the mean is equal to the mean parameter. This is important to take into consideration when writing the simulation functions to pass as a parameter.

For a distribution that does not have a dispersion parameter, like the Poisson, disp should be set to 1.

Author(s)

Oscar Alberto Quijano Xacur <oscar.quijano@use.startmail.com>.

Examples

```
#This function simulates one gamma random variable given the mean and
#dispersion parameter.
sim.gamma.function <- function(mean,phi){
  rgamma(1,shape=mean,rate=phi);
}

generate.weights <- function(n){
  rep(1,n);
 }


beta.vector <- c(3,-1.2,0.8,-1.35,1.4,-0.55,0.68,-0.77,1.46,.38,-1.11,1.13,1.14);
observations <- sample(100:200,60,replace=TRUE);

simulated.gamma.glm <- generate.simulated.glm.data(classes.vector=c(2,3,10),
                                                   beta=beta.vector,
                                                   phi=1000,
                                                   inverse.link.function=exp,
                                                   observations.per.class=observations,
                                                   names=c("Gender","Driver","Make"),
                                                   sim.function=sim.gamma.function,
                                                   response.variable.name="Payments",
                                                   weights.function=generate.weights,
                                                   is.tweedie=FALSE);

sim.poisson.function <- function(mean=lambda,phi=1){
```

```
    rpois(1,lambda=mean/phi) # The mean in this case is the
                             # mean-parameter divided by phi.
}

poisson.beta.vector <- c(5,1.2,0.8,1.35,1.4,0.55,0.68,0.77,1.46,.38,1.11,1.13,1.14)
observations.for.poisson.classes <- sample(50:100,60,replace=TRUE);

fake.poisson.data <- generate.simulated.glm.data(classes.vector=c(2,3,10),
                                                 beta=poisson.beta.vector,
                                                 phi=1,
                                                 inverse.link.function=exp,
                                                 observations.per.class=observations.for.poisson.classes,
                                                 names=c("Gender","Driver","Make"),
                                                 sim.function=sim.poisson.function,
                                                 response.variable.name="Claims",
                                                 weights.function=generate.weights,
                                                 is.tweedie=FALSE);
```

`lift.curve` *The Lift Chart*

Description

Plots a lift chart given the predictions and the observed values or a glm object.

Usage

```
lift.curve(prediction,
           observed,
           nsep = 10,
           weights = NULL,
           title = "Lift curve",
           xtitle = "",
           ytitle = "Total claim size")

lift.curve.glm(modelo.glm,
               nsep = 10,
               title = "Lift Curve",
               xtitle = "",
               ytitle = "")
```

Arguments

`prediction`	Vector that contains the predicted values.
`observed`	Vector that contains the observed values.
`modelo.glm`	glm object.
`nsep`	Number of groups to use for the chart.

weights	NULL or a vector with the weights for each class.
title	Main title for the chart.
xtitle	Title for the x-axis.
ytitle	Title for the y-axis.

Details

The following steps are used in order to create a lift chart:

1) Using the model generate predictions for the observations. 2) Order the observations increasingly with respect to the predictions. 3) Divide the ordered data in groups with equal number of observations. 4) Plot the mean observation and mean prediction for each group. 5) If you include weights, add bars representing the weights for each class.

If weights is not NULL, there will be bars in the chart to represent the weight of each class.

Author(s)

Oscar Alberto Quijano Xacur <oscar.quijano@use.startmail.com>.

References

Article To Tweedie or Not.

Examples

```
observed <- 1:20+rnorm(20)
predicted<-1:20
lift.curve(predicted,observed)

library(datasets)

glm.object <- glm(weight~group,data=PlantGrowth,family=gaussian)
lift.curve.glm(glm.object)
```

---

resids.plot *Residual plot for GLMs.*

Description

Gives residual plots for GLMs. It gives three reference lines, one for x=0 and two others that correspond the observed mean plus and minus a factor of the observed standard deviance of the residuals.

Usage

```
resids.plot(glm.object,
               sd.factor = 1.5,
               rtype = "deviance",
               ptype = "link",
               xlab = "Predicted value (linear predictor)",
               ylab = "Deviance residuals")
```

Arguments

glm.object	glm object to plot the residuals.
sd.factor	Number of standard deviances away for the mean to use for the reference lines.
rtype	String indicating the type of the residuals. It can be any type supported by the residuals function in the stats package.
ptype	String indicating the type of prediction. This is used for the x-axis of the plot. It can be any type supported by the predict function in the predict function in the stats package.
xlab	Label for the x-axis.
ylab	Label for the y-axis.

Details

This function also puts the class number under those residuals that are outside of the region enclosed by the two reference lines.

Author(s)

Oscar Alberto Quijano Xacur <oscar.quijano@use.startmail.com>.

Examples

```
library(datasets)
glm.object <- glm(weight~group,data=PlantGrowth,family=gaussian)
resids.plot(glm.object)
```

---

tweedie.fs *Frequency and Severity means from Tweedie.*

Description

Returns frequency and severity prediction from a Tweedie GLM.

Usage

```
tweedie.fs(mt, p, disp)
```

Arguments

mt	glm object with a tweedie response.
p	variance function power of the tweedie.
disp	dispersion parameter.

Details

When a Tweedie GLM is used to model the aggregate loss of a portfolio it is possible to also obtain estimates for the mean frequency and severity out of it.

p should be between 1 and 2.

Value

This function returns a list with the following elements:

mean.poisson	Vector that contains the predicted frequency mean for each class.
mean.gamma	Vector that contains the predicted severity mean for each class.
mean.tweedie	Vector that contains the predicted aggregate loss mean

Author(s)

Oscar Alberto Quijano Xacur. <oscar.quijano@use.startmail.com>

Examples

```
library(statmod)# Needed for using the tweedie family with glm
library(tweedie)# Needed in order to simulate the tweedie distribution.

simulated.response <- rtweedie(20,xi=1.5,10,3)
tweedie.glm <- glm(simulated.response~1,family=tweedie(var.power=1.5))
tweedie.fs(tweedie.glm,1.5,3)
```

Index