## Package ‘ModTools’

**January 24, 2024**

<table>
<thead>
<tr>
<th>Type</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Title</strong></td>
<td>Tools for Building Regression and Classification Models</td>
</tr>
<tr>
<td><strong>Version</strong></td>
<td>0.9.6</td>
</tr>
<tr>
<td><strong>Date</strong></td>
<td>2024-01-25</td>
</tr>
<tr>
<td><strong>Encoding</strong></td>
<td>UTF-8</td>
</tr>
</tbody>
</table>

**Description**

Collection of tools for regression and classification tasks. The package implements a consistent user interface to the most popular regression and classification algorithms, such as random forest, neural networks, C5 trees and support vector machines, and complements it with a handful of auxiliary functions, such as variable importance and a tuning function for the parameters.

**Depends**

DescTools, MASS, nnet, survival, R (>= 3.5.0)

**License**

GPL (>= 2)

**Imports**

e1071, C50, rpart, randomForest, pROC, methods, relaimpo,

rpart.plot, lattice, lmtest, car, robustbase, class,

NeuralNetTools, naivebayes, sandwich, AER

**LazyLoad**

yes

**LazyData**

yes

**NeedsCompilation**

no

**Author**

Andri Signorell [aut, cre], Bernhard Compton [ctb], Marcel Dettling [ctb], Alexandre Hainard [ctb], Max Kuhn [ctb], Frédérique Lisacek [ctb], Michal Majka [ctb], Markus Müller [ctb], Dan Putler [ctb], Jean-Charles Sanchez [ctb], Nataliu Tiberti [ctb], Natacha Turck [ctb], Jarek Tuszynski [ctb], Robin Xavier [ctb], Achim Zeileis [ctb]
Maintainer Andri Signorell <andri@signorell.net>
Repository CRAN
Date/Publication 2024-01-24 00:52:47 UTC

R topics documented:

ModTools-package ......................................................... 2
BestCut ................................................................. 7
bioChemists ............................................................... 9
BreuschPaganTest ......................................................... 9
CoeffDiffCI ............................................................... 11
CP .......................................................................... 12
d.glass .................................................................. 13
d.pima ................................................................. 14
FitMod ................................................................. 15
LeafRates ................................................................. 16
LogitBoost ............................................................... 18
Node ................................................................. 19
Over-/Undersample ....................................................... 20
PlotLift ................................................................. 22
predict.zeroinfl ......................................................... 23
PredictCI ................................................................. 25
RefLevel ................................................................. 26
Response ................................................................. 27
RobSummary .............................................................. 28
ROC ................................................................. 29
Rules ................................................................. 30
SplitTrainTest ........................................................... 31
TModC ................................................................. 32
Tobit ................................................................. 33
Tune ................................................................. 34
VarImp ................................................................. 36
zeroinfl ................................................................. 37
zeroinfl.control ......................................................... 41

Index 43
ModTools-package

Description

There is a rich selection of R packages implementing algorithms for classification and regression tasks out there. The authors legitimately take the liberty to tailor the function interfaces according to their own taste and needs. For us other users, however, this often results in struggling with user interfaces, some of which are rather weird - to put it mildly - and almost always different in terms of arguments and result structures. ModTools pursues the goal of offering uniform handling for the most important regression and classification models in applied data analyses.

The function FitMod() is designed as a simple and consistent interface to these original functions while maintaining the flexibility to pass on all possible arguments. print, plot, summary and predict operations can so be carried out following the same logic. The results will again be reshaped to a reasonable standard.

For all the functions of this package Google styleguides are used as naming rules (in absence of convincing alternatives). The 'BigCamelCase' style has been consequently applied to functions borrowed from contributed R packages as well.

As always: Feedback, feature requests, bugreports and other suggestions are welcome!

Details

The ModTools::FitMod() function comprises interfaces to the following models:

Regression:

- **lm()**
  - Linear model OLS (base)
- **lmrob()**
  - Robust linear model (robustbase)
- **poisson()**
  - GLM model with family poisson (base)
- **negbin()**
  - GLM model with family negative.binomial (MASS)
- **gamma()**
  - GLM model with family gamma (base)
- **tobit()**
  - Tobit model for censored responses (package AER)

Classification:

- **lda()**
  - Linear discriminant analysis (MASS)
- **qda()**
  - Quadratic discriminant analysis (MASS)
- **logit()**
  - Logistic Regression model glm, family binomial(logit)(base)
- **multinom()**
  - Multinomial Regression model (nnet)
- **polr()**
  - Proportional odds model (MASS)
- **rpart()**
  - Regression and classification trees (rpart)
- **nnet()**
  - Neuronal networks (nnet)
- **randomForest()**
  - Random forests (randomForest)
- **C5.0()**
  - C5.0 tree (C50)
- **svm()**
  - Support vector machines (e1071)
- **naive_bayes()**
  - Naive Bayes classifier (naivebayes)
- **LogitBoost()**
  - Logit boost (using decision stumps as weak learners) (ModTools)

Preprocess:

- **SplitTrainTest()**
  - Splits a data frame or index vector into a training and a test sample
ModTools-package

**OverSample()**
Get balanced datasets by sampling with replacement.

**Manipulating rpart objects:**
- **CP()**
  Extract and plot complexity table of an rpart tree.
- **Node()**
  Accessor to the most important properties of a node, being a split or a leaf.
- **Rules()**
  Extract the decision rules from top to the end node of an rpart tree.
- **LeafRates()**
  Returns the misclassification rates in all end nodes.

**Prediction and Validation:**
- **Response()**
  Extract the response variable of any model.
- **predict()**
  Consistent predict for FitMod models.
- **VarImp()**
  Variable importance for most FitMod models.
- **ROC()**
  ROC curves for all dichotomous classification FitMod models.
- **BestCut()**
  Find the optimal cut for a classification based on the ROC curve.
- **PlotLift()**
  Produces a lift chart for a binary classification model.
- **TModC()**
  Aggregated results for multiple FitMod classification models.
- **Tune()**
  Tuning approaches to find optimal parameters for FitMod classification models.
- **RobSummary()**
  Robust summary for GLM models (poisson).

**Tests:**
- **BreuschPaganTest()**
  Breusch-Pagan test against heteroskedasticity.

**Warning**
This package is still under development. You should be aware that everything in the package might be subject to change. Backward compatibility is not yet guaranteed. Functions may be deleted or renamed and new syntax may be inconsistent with earlier versions. By release of version 1.0 the "deprecated-defunct process" will be installed.

**Author(s)**
Andri Signorell
Helsana Versicherungen AG, Health Sciences, Zurich
HWZ University of Applied Sciences in Business Administration Zurich.

Includes R source code and/or documentation previously published by (in alphabetical order):
Bernhard Compton, Marcel Dettling, Max Kuhn, Michal Majka, Dan Putler, Jarek Tuszynski, Robin Xavier, Achim Zeileis

The good things come from all these guys, any problems are likely due to my tweaking. Thank you all!
Maintainer: Andri Signorell <andri@signorell.net>

Examples

```r
r.swiss <- FitMod(Fertility ~ ., swiss, fitfn="lm")
r.swiss
# PlotTA(r.swiss)
# PlotQQNorm(r.swiss)

## Count models

data(housing, package="MASS")

# poisson count
r.pois <- FitMod(Freq ~ Infl*Type*Cont + Sat, family=poisson, data=housing, fitfn="poisson")

# negative binomial count
r.nb <- FitMod(Freq ~ Infl*Type*Cont + Sat, data=housing, fitfn="negbin")
summary(r.nb)

r.log <- FitMod(log(Freq) ~ Infl*Type*Cont + Sat, data=housing, fitfn="lm")
summary(r.log)

r.ols <- FitMod(Freq ~ Infl*Type*Cont + Sat, data=housing, fitfn="lm")
summary(r.ols)

r.gam <- FitMod(Freq ~ Infl*Type*Cont + Sat, data=housing, fitfn="gamma")
summary(r.gam)

r.gami <- FitMod(Freq ~ Infl*Type*Cont + Sat, data=housing, fitfn="gamma", link="identity")
summary(r.gami)

old <- options(digits=3)
TMod(r.pois, r.nb, r.log, r.ols, r.gam, r.gami)
options(old)

## Ordered Regression

r.polr <- FitMod(Sat ~ Infl + Type + Cont, data=housing, fitfn="polr", weights = Freq)

# multinomial Regression
# r.mult <- FitMod(factor(Sat, ordered=FALSE) ~ Infl + Type + Cont, data=housing,
#                  weights = housing$Freq, fitfn="multinom")

# Regression tree
r.rp <- FitMod(factor(Sat, ordered=FALSE) ~ Infl + Type + Cont, data=housing,
               weights = housing$Freq, fitfn="rpart")

# compare predictions
```
d.p <- expand.grid(Infl=levels(housing$Infl), Type=levels(housing$Type), Cont=levels(housing$Cont))
d.p$polr <- predict(r.polr, newdata=d.p)
# ??
# d.p$ols <- factor(round(predict(r.ols, newdata=d.p)^2), labels=levels(housing$Sat))
# d.p$mult <- predict(r.mult, newdata=d.p)
d.p$rp <- predict(r.rp, newdata=d.p, type="class")
d.p

# Classification with 2 classes ***************

r.pima <- FitMod(diabetes ~ ., d.pima, fitfn="logit")
r.pima
Conf(r.pima)
plot(ROC(r.pima))
OddsRatio(r.pima)

# rpart tree
rp.pima <- FitMod(diabetes ~ ., d.pima, fitfn="rpart")
rp.pima
Conf(rp.pima)
lines(ROC(rp.pima), col=hblue)
# to be improved
plot(rp.pima, col=SetAlpha(c("blue","red"), 0.4), cex=0.7)

# Random Forest
rf.pima <- FitMod(diabetes ~ ., d.pima, method="class", fitfn="randomForest")
rf.pima
Conf(rf.pima)
lines(ROC(rf.pima), col=hred)

# more models to compare

r.glm <- FitMod(mdiab, data=d.pim$train, fitfn="logit")
r.rp <- FitMod(mdiab, data=d.pim$train, fitfn="rpart")
r.rf <- FitMod(mdiab, data=d.pim$train, fitfn="randomForest")
r.svm <- FitMod(mdiab, data=d.pim$train, fitfn="svm")
r.c5 <- FitMod(mdiab, data=d.pim$train, fitfn="C5.0")
r.nn <- FitMod(mdiab, data=d.pim$train, fitfn="nnet")
r.nb <- FitMod(mdiab, data=d.pim$train, fitfn="naive_bayes")
r.lda <- FitMod(mdiab, data=d.pim$train, fitfn="lda")
r.qda <- FitMod(mdiab, data=d.pim$train, fitfn="qda")
r.lb <- FitMod(mdiab, data=d.pim$train, fitfn="lb")
BestCut

mods <- list(glm=r glm, rp=r rp, rf=r rf, svm=r svm, c5=r c5
         , nn=r nn, nb=r nb, lda=r lda, qda=r qda, lb=r lb)

# insight in the Regression tree
plot(r rp, box.palette = as.list(Pal("Helsana", alpha = 0.5)))

# Insample accuracy ...
TModC(mods, ord="auc")
# ... is substantially different from the out-of-bag:
TModC(mods, newdata=d.pim$test, reference=d.pim$test$diabetes, ord="bs")
# C5 and SVM turn out to be show-offs! They overfit quite ordinary
# whereas randomforest and logit keep their promises. ...
sapply(mods, function(z) VarImp(z))

# Multinomial classification problem with n classes **************
d.gl <- SplitTrainTest(d.glass, p = 0.2)
glass <- formula(Type ~ RI + Na + Mg + Al + Si + K + Ca + Ba + Fe)

# *** raises an unclear error in CRAN-Debian tests *** ??
# r.mult <- FitMod(glass, data=d.gl$train, maxit=600, fitfn="multinom")
r rp <- FitMod(glass, data=d.gl$train, fitfn="rpart")
r.rf <- FitMod(glass, data=d.gl$train, fitfn="randomForest")
r.svm <- FitMod(glass, data=d.gl$train, fitfn="svm")
r.c5 <- FitMod(glass, data=d.gl$train, fitfn="C5.0")
r.nn <- FitMod(glass, data=d.gl$train, fitfn="nnet")
r.nbay <- FitMod(glass, data=d.gl$train, fitfn="naive_bayes")
r.lda <- FitMod(glass, data=d.gl$train, fitfn="lda")
# r.qda <- FitMod(glass, data=d.glass, fitfn="qda")
r.lb <- FitMod(glass, data=d.gl$train, fitfn="lb")

mods <- list(rp=r rp, rf=r rf, svm=r svm, c5=r c5,
             nn=r nn, nbay=r nbay, lda=r lda, lb=r lb)

# confusion matrix and other quality measures can be calculated with Conf()
Conf(r rf)

# we only extract the general accuracy
sapply(lapply(mods, function(z) Conf(z)), "[[", "acc")

# let's compare r.mult with a model without RI as predictor
# Conf(r.mult)
# Conf(update(r.mult, . ~ - RI))

BestCut

Best Cutpoint for a ROC Curve

Description

Returns the best cutpoint for a given classification model.
Usage

BestCut(x, method = c("youden", "closest.topleft"))

Arguments

x a roc object from the roc function
method one of "youden" or "closest.topleft", controls how the optimal threshold is determined. See details.

Details

The method argument controls how the optimal threshold is determined.

'youden' Youden’s J statistic (Youden, 1950) is employed. The optimal cut-off is the threshold that maximizes the distance to the identity (diagonal) line. Can be shortened to “y”.

The optimality criterion is:

\[
\max(sensitivities + specificities)
\]

'closest.topleft' The optimal threshold is the point closest to the top-left part of the plot with perfect sensitivity or specificity. Can be shortened to “c” or “t”.

The optimality criterion is:

\[
\min((1 - sensitivities)^2 + (1 - specificities)^2)
\]

Value

the threshold value

Author(s)

Robin Xavier <pROC-cran@xavier.robin.name>, Andri Signorell <andri@signorell.net> (interface)

References


See Also

ROC

Examples

```r
glm <- FitMod(diabetes ~ ., data = d.pima, fitfn="logit")

ROC(glm)
BestCut(ROC(glm))
```
bioChemists

| bioChemists | article production by graduate students in biochemistry Ph.D. programs |

**Description**

A sample of 915 biochemistry graduate students.

**Usage**

```r
data(bioChemists)
```

**Format**

- `art` count of articles produced during last 3 years of Ph.D.
- `fem` factor indicating gender of student, with levels Men and Women
- `mar` factor indicating marital status of student, with levels Single and Married
- `kid5` number of children aged 5 or younger
- `phd` prestige of Ph.D. department
- `ment` count of articles produced by Ph.D. mentor during last 3 years

**References**


---

**BreuschPaganTest**

**Breusch-Pagan Test**

**Description**

Performs the Breusch-Pagan test against heteroskedasticity.

**Usage**

```r
BreuschPaganTest(formula, varformula = NULL, studentize = TRUE, data = list())
```
**Arguments**

- `formula`: a symbolic description for the model to be tested (or a fitted "lm" object).
- `varformula`: a formula describing only the potential explanatory variables for the variance (no dependent variable needed). By default the same explanatory variables are taken as in the main regression model.
- `studentize`: logical. If set to `TRUE` Koenker's studentized version of the test statistic will be used.
- `data`: an optional data frame containing the variables in the model. By default the variables are taken from the environment which `BreuschPaganTest` is called from.

**Details**

The Breusch-Pagan test fits a linear regression model to the residuals of a linear regression model (by default the same explanatory variables are taken as in the main regression model) and rejects if too much of the variance is explained by the additional explanatory variables.

Under $H_0$ the test statistic of the Breusch-Pagan test follows a chi-squared distribution with parameter (the number of regressors without the constant in the model) degrees of freedom.

Examples can not only be found on this page, but also on the help pages of the data sets `bondyield`, `currencysubstitution`, `growthofmoney`, `moneydemand`, `unemployment`, `wages`.

**Value**

A list with class "htest" containing the following components:

- `statistic`: the value of the test statistic.
- `p.value`: the p-value of the test.
- `parameter`: degrees of freedom.
- `method`: a character string indicating what type of test was performed.
- `data.name`: a character string giving the name(s) of the data.

**Author(s)**

Achim Zeileis <Achim.Zeileis@R-project.org>

**References**


**See Also**

`lm`, `ncvTest`
Examples

```r
## generate a regressor
x <- rep(c(-1,1), 50)

## generate heteroskedastic and homoskedastic disturbances
err1 <- rnorm(100, sd = rep(c(1,2), 50))
err2 <- rnorm(100)

## generate a linear relationship
y1 <- 1 + x + err1
y2 <- 1 + x + err2

## perform Breusch-Pagan test
BreuschPaganTest(y1 ~ x)
BreuschPaganTest(y2 ~ x)
```

**CoeffDiffCI**

Confidence Interval for the Difference of Two Coefficients in a Linear Model

Description

Calculate the confidence interval for the difference of two coefficients in a linear model.

Usage

`CoeffDiffCI(x, coeff, conf.level = 0.95, sides = c("two.sided", "left", "right"))`

Arguments

- `x`: the linear model object
- `coeff`: a vector of length two, containing either the names or the index of the two coefficients whose difference should be used
- `conf.level`: confidence level of the interval.
- `sides`: a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". You can specify just the initial letter. "left" would be analogue to a hypothesis of "greater" in a t.test.

Details

This is quite useful in the course of the modelling process.

Value

A numeric vector with 3 elements:

- `mean`
- `lwr.ci`: lower bound of the confidence interval
- `upr.ci`: upper bound of the confidence interval
Author(s)
Andri Signorell <andri@signorell.net>

See Also
linearHypothesis()

Examples
# get some model first...
 r.lm <- FitMod(Fertility ~ ., data=swiss, fitfn="lm")

# calculate the confidence interval for the difference of the
# coefficients Examination and Education
 CoeffDiffCI(r.lm, c("Examination", "Education"))

# the test could be calculated as
car::linearHypothesis(r.lm, "Education = Examination")

CP

Complexity Parameter of an rpart Model

Description
Extracts, prints and plots the complexity table of an rpart model.

Usage
CP(x, ...)

## S3 method for class 'CP'
print(x, digits = getOption("digits") - 2L, ...)

## S3 method for class 'CP'
plot(x, minline = TRUE, lty = 3, col = 1,
    upper = c("size", "splits", "none"), ...)

Arguments

x     fitted model object of class "rpart". This is assumed to be the result of some
       function that produces an object with the same named components as that re-
       turned by the rpart function.
digits the number of digits of numbers to print.
minline whether a horizontal line is drawn 1SE above the minimum of the curve.
lty     line type for this line
col     colour for this line
upper what is plotted on the top axis: the size of the tree (the number of leaves)
       ("size"), the number of splits ("splits") or nothing ("none").
...     further arguments passed to print and plot
Details

The complexity parameter table is hidden deep in the entrails of the \texttt{rpart} result object, it is convenient to have a function to extract it.

Value

A list containing the following components:

\begin{itemize}
  \item \texttt{cp} : the complexity table
  \item \texttt{x} : the \texttt{rpart} object
\end{itemize}

Author(s)

Andri Signorell <andri@signorell.net>

See Also

\texttt{printcp, plotcp}

Examples

\begin{verbatim}
r.rp <- FitMod(diabetes ~ ., d.pima, fitfn="rpart")
CP(r.rp)
plot(CP(r.rp))
\end{verbatim}

---

d.glass  Measurements of Forensic Glass Fragments

Description

The \texttt{d.glass} data frame has 214 rows and 10 columns. It was collected by B. German on fragments of glass collected in forensic work.

Usage

d.glass

Format

This data frame contains the following columns:

- \texttt{RI} : refractive index; more precisely the refractive index is 1.518xxxx.
  The next 8 measurements are percentages by weight of oxides.
- \texttt{Na} : sodium.
- \texttt{Mg} : manganese.
- \texttt{Al} : aluminium.
Si silicon.
K potassium.
Ca calcium.
Ba barium.
Fe iron.

Type  The fragments were originally classed into seven types, one of which was absent in this dataset. The categories which occur are window float glass (WinF: 70), window non-float glass (WinNF: 76), vehicle window glass (Veh: 17), containers (Con: 13), tableware (Tabl: 9) and vehicle headlamps (Head: 29).

References

---

**d.pima**

*Diabetes survey on Pima Indians*

**Description**

The National Institute of Diabetes and Digestive and Kidney Diseases conducted a study on 768 adult female Pima Indians living near Phoenix.

**Usage**

data(d.pima)
data(d.pima2)

**Format**

The dataset contains the following variables

- **pregnant** Number of times pregnant
- **glucose** Plasma glucose concentration at 2 hours in an oral glucose tolerance test
- **diastolic** Diastolic blood pressure (mm Hg)
- **triceps** Triceps skin fold thickness (mm)
- **insulin** 2-Hour serum insulin (mu U/ml)
- **bmi** Body mass index (weight in kg/(height in metres squared))
- **diabetes** Diabetes pedigree function
- **age** Age (years)
- **test** test whether the patient shows signs of diabetes (coded 0 if negative, 1 if positive)

**Details**

d.pima2 is the same dataset as d.pima with the only change, that invalid 0-values are replaced by NAs.
Note

This dataset has been borrowed from Julian Faraway’s package:

faraway: Functions and datasets for books by Julian Faraway, 2015

Source

The data may be obtained from the package MASS.

---

**FitMod**

*Wrapper for Several Model Functions*

---

**Description**

Popular implementations of algorithms are characterized by partly unconventional implementations of the operating standards in R. For example, the function `e1071::SVM()` returns the predicted values as attributes!

FitMod() is designed as a wrapping function to offer a consistent interface for a selection of most often used classification and regression models.

**Usage**

FitMod(formula, data, ..., subset, na.action = na.pass, fitfn = NULL)

## S3 method for class 'Var'

FitMod/Var

## S3 method for class 'Var'

predict(object, ...)

## S3 method for class 'Var'

plot(x, ...)

## S3 method for class 'Var'

summary(object, ...)

## S3 method for class 'Var'

drop1(object, ...)

**Arguments**

**x**

a fitted object of class "FitMod".

**formula**

a formula expression as for classification and regression models, of the form response ~ predictors. The response should be a factor or a matrix with K columns, which will be interpreted as counts for each of K classes. See the documentation of `formula()` for other details.

**data**

an optional data frame in which to interpret the variables occurring in formula.

**subset**

expression saying which subset of the rows of the data should be used in the fit. All observations are included by default.

**na.action**

a function to filter missing data.
fitfn  code for the fitting function to be used for regression or classifying. So far implemented are: lm, lmrob, poisson, quasipoisson, gamma, negbin, poisson, polr, tobit, zeroinfl, multinom, poisson, rpart, randomForest, logit, nnet, C5.0, lda, qda, svm, naive_bayes, lb.

object the model object.

... further arguments passed to the underlying functions.

Details

The function will in general return the original object, extended by a further class FitMod, which allows to capture the output and plot routines.

The classifying algorithms will at the minimum offer the predicting options type = c("class", "prob") additionally to those implemented by the underlying function.

Value

model object as returned by the calculating function extended with the FitMod class.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

lm, rpart

Examples

r.lm <- FitMod(Fertility ~ ., data=swiss, fitfn="lm")
r.logit <- FitMod(diabetes ~ glucose + pressure + mass + age, data=d.pima, fitfn="logit")
r.svm <- FitMod(diabetes ~ glucose + pressure + mass + age, data=d.pima, fitfn="svm")

LeafRates

Leafrates for the Nodes of an 'rpart' Tree

Description

Return the frequencies of correct and wrong classifications in given node(s) in tabular form. The 'purity', denoting the relative frequency of correctly classified elements, is a useful information for the interpretation of regression and classification trees and a measure for its quality.
LeafRates

Usage

LeafRates(x)

## S3 method for class 'LeafRates'
plot(x, col = NULL, which = c("rel", "abs"),
     layout = NULL, ylim = NULL, ...)

Arguments

x        fitted model object of class rpart.
col      color for the bars in the plot
which     one out of "rel" or "abs", denoting whether relative or absolute frequencies should be used for the plot.
layout    vector defining the layout
ylim      the y limits of the plot.
...       further arguments (not used).

Details

The result comprises absolute and relative frequencies per leaf.

Value

A list with 5 elements consisting of:

node      the node id (of the leaf)
freq      the absolute frequency of correct and wrong classifications
p.row     the relative frequency of correct and wrong classifications
mfreq     the total number of cases
mperc     the percentage of the sample in the leaf

Author(s)

Andri Signorell <andri@signorell.net>

See Also

Node, Rules

Examples

```r
r,rp <- FitMod(Species ~ ., data=iris, fitfn="rpart")
LeafRates(r,rp)
plot(LeafRates(r,rp))
```
Description

Train logitboost classification algorithm using decision stumps (one node decision trees) as weak learners.

Usage

LogitBoost(x, ...)

## S3 method for class 'formula'
LogitBoost(formula, data, ..., subset, na.action)

## Default S3 method:
LogitBoost(x, y, nIter=ncol(x), ...)

Arguments

- `formula`: a formula expression as for regression models, of the form `response ~ predictors`. The response should be a factor or a matrix with K columns, which will be interpreted as counts for each of K classes. See the documentation of `formula()` for other details.
- `data`: an optional data frame in which to interpret the variables occurring in formula.
- `...`: additional arguments for nnet
- `subset`: expression saying which subset of the rows of the data should be used in the fit. All observations are included by default.
- `na.action`: a function to filter missing data.
- `x`: A matrix or data frame with training data. Rows contain samples and columns contain features.
- `y`: Class labels for the training data samples. A response vector with one label for each row/component of xlearn. Can be either a factor, string or a numeric vector.
- `nIter`: An integer, describing the number of iterations for which boosting should be run, or number of decision stumps that will be used.

Details

The function was adapted from logitboost.R function written by Marcel Dettling. See references and "See Also" section. The code was modified in order to make it much faster for very large data sets. The speed-up was achieved by implementing a internal version of decision stump classifier instead of using calls to rpart. That way, some of the most time consuming operations were precomputed once, instead of performing them at each iteration. Another difference is that training and testing phases of the classification process were split into separate functions.
Value

An object of class "LogitBoost" including components:

Stump          List of decision stumps (one node decision trees) used:

  • column 1: feature numbers or each stump, or which column each stump
    operates on
  • column 2: threshold to be used for that column
  • column 3: bigger/smaller info: 1 means that if values in the column are
    above threshold than corresponding samples will be labeled as lablist[1].
    Value "-1" means the opposite.

If there are more than two classes, than several "Stumps" will be cbind’ed

lablist         names of each class

Author(s)

Jarek Tuszynski (SAIC) <jaroslaw.w.tuszynski@saic.com>

References

Dettling and Buhlmann (2002), *Boosting for Tumor Classification of Gene Expression Data.*

Examples

```r
# basic interface
r.lb <- LogitBoost(Species ~ ., data=iris, nIter=20)
pred <- predict(r.lb)
prob <- predict(r.lb, type="prob")
d.res <- data.frame(pred, prob)
d.res[1:10, ]

# accuracy increases with nIter (at least for train set)
table(predict(r.lb, iris, type="class", nIter= 2), iris$Species)
table(predict(r.lb, iris, type="class", nIter=10), iris$Species)
table(predict(r.lb, iris, type="class"),         iris$Species)

# example of splitting the data into train and test set
d.set <- SplitTrainTest(iris)
r.lb <- LogitBoost(Species ~ ., data=d.set$train, nIter=10)
table(predict(r.lb, d.set$test, type="class", nIter=2), d.set$test$Species)
table(predict(r.lb, d.set$test, type="class"),           d.set$test$Species)
```
Nodes and Splits in an rpart Tree

Description
The rpart result object has a complex and compact design. This can make practical use tedious for occasional users as it is difficult to figure out how to access some specific information. The function Node() is designed as accessor to the most important properties of a node, being a ‘split’ or a ‘leaf’ (aka. ‘endnode’). It also serves as base for further convenience functions as e.g. LeafRates().

Usage
Node(x, node = NULL, type = c("all", "split", "leaf"), digits = 3)

Arguments
- x: fitted model object of class rpart.
- node: integer vector, defining the nodes whose details are required.
- type: one out of "all" (default), "split", "leaf", where the latter two restrict the result set to splits or end nodes only. Can be abbreviated.
- digits: the number of digits for numeric values

Details
Node() returns detailed information for a single node in the tree. It reports all the data in the summary of a node, but with the option to provide a nodelist. The structure of the result is organised as a list.

Value
A list containing:
- id: int, id of the node
- vname: character, one out of 'leaf' or 'split'
- isleaf: logical, TRUE for leaves FALSE else
- nobs: integer, number of observation in the node
- group: character, the predicted class for the node
- ycount: numeric, the number of observation per class in the node
- yprob: numeric, the relative frequencies for the each class
- nodeprob: the global probability for an observation to fall in the node
- complexity: numeric, the complexity parameter for the node
- tprint: character, the text to be printed
Over-/Undersample

Author(s)
Andri Signorell <andri@signorell.net>

See Also
LeafRates, Rules

Examples

```r
r.rpart <- FitMod(Species ~ ., data=iris, fitfn="rpart")
# return Node nr. 3
Node(r.rpart, node=3)

r.rp <- FitMod(Type ~ ., data = d.glass, fitfn="rpart")
# return all the splits
Node(r.rpart, type="split")
```

Description
For classification purposes we might want to have balanced datasets. If the response variable has not a prevalence of 50%, we can sample records for getting as much response A cases as response B. This is called oversample. Undersample means to sample the (lower) number of cases A from the records of case B.

Usage

```r
OverSample(x, vname)
UnderSample(x, vname)
```

Arguments

- `x` a data frame containing predictors and response
- `vname` the name of the response variable to be used to over/undersample

Value

a data frame with balanced response variable

Author(s)
Andri Signorell <andri@signorell.net>

See Also
BestCut
Examples

OverSample(d.pima2, "diabetes")
UnderSample(d.pima2, "diabetes")

Description

Provides either a total cumulative response or incremental response rate lift chart for the purposes of comparing the predictive capability of different binary predictive models.

Usage

PlotLift(modelList, data, targLevel, trueResp, type = "cumulative", sub = "")

Arguments

- **modelList**: A character vector containing the names of the different models to be compared. The selected models must have the same y variable that must be a binary factor, and have been estimated using the same data set.
- **data**: The dataframe that constitutes the comparison sample. If this dataframe is not the same as the dataframe used to estimated models, the dataframe must contain all the variables used in the models to be compared.
- **targLevel**: The label for the level of the binary factor of interest. For example, in a database marketing application, this level could be "Yes" for a variable that takes on the values "Yes" and "No" to indicate if a customer responded favorably to a promotion offer.
- **trueResp**: The true rate of the target level for the master database the estimation and comparison dataframes were originally drawn from.
- **type**: A character string that must either have the value of "cumulative" (to produce a total cumulative response chart) or "incremental" (to produce an incremental response rate chart).
- **sub**: A sub-title for the plot, typically to identify the sample used.

Details

Lift charts are a commonly used tool in business data mining applications. They are used to assess how well a model is able to predict a desirable (from an organization's point-of-view) response on the part of a customer compared to alternative estimated models and a benchmark model of approaching customers randomly. The total cumulative response chart shows the percentage of the total response the organization would receive from only contacting a given percentage (grouped by deciles) of its entire customer base. This chart is best for selecting between alternative models, and in predicting the revenues the organization will receive by contacting a given percentage of their customers that the model predicts are most likely to favorably respond. The incremental response rate chart provides the response rate among each of ten decile groups of the organization's customers, with the decile groups ordered by their estimated likelihood of a favorable response.
Value

The function returns the sample response invisibly.

Author(s)

original Dan Putler, tweaks Andri Signorell <andri@signorell.net>

Examples

d.pim <- SplitTrainTest(d.pima, p = 0.2)

r.rp <- FitMod(diabetes ~ pregnant + glucose + pressure + triceps
+ insulin + mass + pedigree + age
, data=d.pim$train, fitfn="rpart")

r(glm <- FitMod(diabetes ~ pregnant + glucose + pressure + triceps
+ insulin + mass + pedigree + age
, data=d.pim$train, fitfn="logit")

r.nn <- FitMod(diabetes ~ pregnant + glucose + pressure + triceps
+ insulin + mass + pedigree + age
, data=d.pim$train, fitfn="nnet")

oldpar <- par(mfrow=c(1,2))
PlotLift(c("r.rp", "r.glm", "r.nn"), data = d.pim$train,
targLevel = "pos", trueResp =0.34, type = "cumulative")
PlotLift(c("r.rp", "r.glm", "r.nn"), data = d.pim$train,
targLevel = "pos", trueResp =0.34, type = "incremental")
par(oldpar)

predict.zeroinfl  Methods for zeroinfl Objects

Description

Methods for extracting information from fitted zero-inflated regression model objects of class "zeroinfl".

Usage

## S3 method for class 'zeroinfl'
predict(object, newdata,
  type = c("response", "prob", "count", "zero"), na.action = na.pass,
  at = NULL, ...)

## S3 method for class 'zeroinfl'
residuals(object, type = c("pearson", "response"), ...)

## S3 method for class 'zeroinfl'
coef(object, model = c("full", "count", "zero"), ...)
A set of standard extractor functions for fitted model objects is available for objects of class "zeroinfl", including methods to the generic functions print and summary which print the estimated coefficients along with some further information. The summary in particular supplies partial Wald tests based on the coefficients and the covariance matrix (estimated from the Hessian in the numerical optimization of the log-likelihood). As usual, the summary method returns an object of class "summary.zeroinfl" containing the relevant summary statistics which can subsequently be printed using the associated print method.

The methods for coef and vcov by default return a single vector of coefficients and their associated covariance matrix, respectively, i.e., all coefficients are concatenated. By setting the model argument, the estimates for the corresponding model components can be extracted.

Both the fitted and predict methods can compute fitted responses. The latter additionally provides the predicted density (i.e., probabilities for the observed counts), the predicted mean from the count component (without zero inflation) and the predicted probability for the zero component. The residuals method can compute raw residuals (observed - fitted) and Pearson residuals (raw residuals scaled by square root of variance function).

The terms and model.matrix extractors can be used to extract the relevant information for either component of the model.

A logLik method is provided, hence AIC can be called to compute information criteria.
Provided as confidence intervals for predictions of a GLM.

**Usage**

```
PredictCI(mod, newdata, conf.level = 0.95)
```

**Arguments**

- `mod`: the binomial model
- `newdata`: the data to be predicted
- `conf.level`: confidence level of the interval. Default is 0.95.

**Details**

The confidence intervals for predictions are calculated with the se of the model and the normal quantile.

**Value**

A matrix with 3 columns for the fit, the lower confidence interval and the upper confidence interval.

**Author(s)**

Andri Signorell <andri@signorell.net>
References

https://stackoverflow.com/questions/14423325/confidence-intervals-for-predictions-from-logistic-regression

See Also

FitMod

Examples

r.logit <- FitMod(diabetes ~ age, d.pima, fitfn = "logit")
head(PredictCI(r.logit, newdata=d.pima))

RefLevel

Description

Returns all the reference levels in the factors used in a linear model. It is customer friendly to report also the reference level in lm summaries, which normally are suppressed.

Usage

RefLevel(x)

Arguments

x

lm object, linear model with factors as predictors.

Details

For reporting tables of linear models we might want to include an information about the used reference levels, which remain uncommented in the default lm result output. RefLevel() allows to add a footnote or integrate the reference levels in the coefficient table.

Value

a named vector containing the reference levels of all factors

Note

It’s not clear how general the used algorithm is for more exotic models. dummy.coef could in such cases be an alternative.

Author(s)

Andri Signorell <andri@signorell.net>
See Also
dummy.coef, `Response`, `relevel`, `lm`

Examples

```r
RefLevel(lm(breaks ~ wool + tension, data = warpbreaks))
```

Description

Time after time, in the course of our daily work, we experience that the response variable is hidden very deeply in the object. This again leads to superfluous consultation of the documentation. `Response()` relieves us of this work.

Usage

```r
Response(x, ...)
```

Arguments

- `x` : the model to use
- `...` : more arguments

Details

The function implements the extraction of the response variables for all the models listed in the package’s help text.

Value

- the response of model `x`

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`model.frame`, `model.response`, `RefLevel`

Examples

```r
r.rpart <- FitMod(diabetes ~ ., d.pima, fitfn="rpart")
Response(r.rpart)
```

```r
# up to the attribute "response" this is the same
identical(StripAttr(Response(r.rpart), "response"),
          model.response(model.frame(r.rpart)))
```
RobSummary

Robust Summary for Linear Models

Description

For poisson models with mild violation of the distribution assumption that the variance equals the mean, Cameron and Trivedi (2009) recommended using robust standard errors for the parameter estimates. The function uses the function `vcovHC` from the package `sandwich` to obtain the robust standard errors and calculate the p-values accordingly. It returns a matrix containing the usual results in the model summary, comprising the parameter estimates, their robust standard errors, p-values, extended with the 95% confidence interval.

Usage

```
RobSummary(mod, conf.level = 0.95, type = "HC0")
```

Arguments

- `mod`: the model for which robust standard errors should be calculated
- `conf.level`: the confidence level, default is 95%.
- `type`: a character string specifying the estimation type. Details in `vcovHC()`.

Details

Further details in https://stats.oarc.ucla.edu/r/dae/poisson-regression/

Value

A $p \times 6$ matrix with columns for the estimated coefficient, its standard error, t- or z-statistic, the corresponding (two-sided) p-value, the lower and upper confidence interval.

Author(s)

Andri Signorell <andri@signorell.net>

References

Cameron, A. C. and Trivedi, P. K. (2009) Microeconometrics Using Stata. College Station, TX: Stata Press.

See Also

- `summary.lm`, `summary.glm`

Examples

```r
r.lm <- lm(Fertility ~ ., swiss)
RobSummary(r.lm)
```
Description

This is a wrapper to the main function `pROC` of the `pROC` package (by Xavier Robin et al.). It builds a ROC curve and returns a "roc" object, a list of class "roc".

Usage

```r
ROC(x, resp = NULL, ...) 
```

Arguments

- `x` a model object, or the predicted probabilities, when `resp` is not `NULL`.
- `resp` the response
- `...` all arguments are passed to `roc()`.

Details

Partial ROC is calculated following Peterson et al. (2008; doi:10.1016/j.ecolmodel.2007.11.008). This function is a modification of the PartialROC funcion, available at https://github.com/narayanibarve/ENMGadgets.

Value

A data.frame containing the AUC values and AUC ratios calculated for each iteration.

Author(s)

Andri Signorell <andri@signorell.net>

References


See Also

`pROC`
Examples

```r
r.glm <- FitMod(diabetes ~ ., data = d.pima, fitfn="logit")
ROC(r.glm)

# plot ROC curves for a list of models
r.rp <- FitMod(diabetes ~ ., data = d.pima, fitfn="rpart")

# combine models to a list
mlst <- list(r.glm, r.rp)

# do the plot
for(i in seq_along(mlst))
  if(i==1){
    plot(ROC(mlst[[i]], grid=TRUE, col=c(hred, hblue)[i])
  } else {
    lines(ROC(mlst[[i]], col=c(hred, hblue)[i])
  }
```

Rules

Extract Rules from 'rpart' Object

Description

Extract rules from an rpart object. This can be useful, if the rules must be implemented in another system. The rules contain all the criteria for the binary splits of an rpart tree from the root node down to the specified leaf.

Usage

```r
Rules(x, node = NULL, leafonly = FALSE)
```

Arguments

- `x`: the rpart object to extract the rules from
- `node`: integer vector, defining the nodes whose details are required.
- `leafonly`: boolean, defining if only the rules leading to end nodes ("leafs") should be returned.

Details

The function builds upon the original function `path.rpart`, which is bulky in some situations.

Value

- a list with the rules
  - `frame`: the frame of the rpart
  - `ylevels`: the y values of the node
  - `ds.size`: the size of the dataset
  - `path`: a list of character vectors containing the rules
**SplitTrainTest**

**Author(s)**

Andri Signorell <andri@signorell.net>

**See Also**

rpart, path.rpart

**Examples**

```r
r.rp <- FitMod(diabetes ~ ., data=d.pima, fitfn="rpart")
Rules(r.rp)
```

---

**SplitTrainTest**  
*Split DataFrame in Train an Test Sample*

**Description**

For modeling we usually split our data frame in a train sample, where we train our model on, and a test sample, where we test, how good it works. This function splits a given data frame in two parts, one being the training sample and the other the test sample in form of a list with two elements.

**Usage**

```r
SplitTrainTest(x, p = 0.1, seed = NULL, logical = FALSE)
```

**Arguments**

- `x`: data.frame
- `p`: proportion for test sample. Default is 10%.
- `seed`: initialization for random number generator.
- `logical`: logical, defining if a logical vector should be returned or the list with train and test data. Default is FALSE.

**Details**

In order to obtain reasonable models, we should ensure two points. The dataset must be large enough to yield statistically meaningful results and it should be representative of the data set as a whole. Assuming that our test set meets the preceding two conditions, our goal is to create a model that generalizes well to new data. We are aiming for a model that equally well predicts training and test data. We should never train on test data. If we are seeing surprisingly good results on the evaluation metrics, it might be a sign that we're accidentally training on the test set.

**Value**

- If `logical` is FALSE a list with two data frames, `train` and `test`, of the same structure as the given data in `x`.
- If `logical` is TRUE a logical vector containing `nrow` elements of TRUE and FALSE.
Author(s)
Andri Signorell <andri@signorell.net>

Examples
SplitTrainTest(d.pima)

TModC  

Compare Classification Models

Description
For the comparison of several classification models, the AUC values and BrierScore values of the models are determined and tabulated. Both the absolute values and the relative values are reported, each related to the model with the highest corresponding value.

Usage
TModC(..., newdata = NULL, reference = NULL, ord = NULL)

## S3 method for class 'TModC'
plot(x, col = NULL, args.legend = NULL,...)

Arguments

... the models to be compared

x TModC object to plot

newdata the data to use for predicting. If not provided, the model.frame will be used.

reference the reference values

ord character defining the order of the results table, can be any of "auc", "bs", "auc_p", "bs_p", "bs_rnk", "auc_rnk", "ensemble" (using the mean of "auc_p" and "bs_p" for the ranking).

col the color for the lines in the ROC plot

args.legend the legend to be placed in the ROC plot

Value

a matrix with the columns

| auc  | absolute value of area under the ROC curve (AUC) |
| auc_p| percentage of the auc based on the best observed AUC |
| auc_rnk | the rank of the auc |
| bs | absolute value of the Brier score |
| bs_p | percentage of the Brier score based on the best observed BS |
| bs_rnk | the rank of the BS |
| auc_grnk | character representation of the AUC rank |
| bs_grnk | character representation of the BS rank |
Tobit

Author(s)
Andri Signorell <andri@signorell.net>

See Also
TMod, BrierScore, AUC, ROC

Examples

d.pim <- SplitTrainTest(d.pima, p = 0.2)
mdiab <- formula(diabetes ~ pregnant + glucose + pressure + triceps +
                  insulin + mass + pedigree + age)

r(glm) <- FitMod(mdiab, data=d.pim$train, fitfn="logit")
r.rp <- FitMod(mdiab, data=d.pim$train, fitfn="rpart")
mods <- list(glm=r(glm), rp=r.rp)

# the table with the measures
(tm <- TModC(mods, ord="auc"))

# plotting the ROC curves
plot(tm, col=c("darkmagenta", "dodgerblue"))

---

Tobit

Tobit Regression

Description
Fitting and testing Tobit regression models for censored data.

Usage
Tobit(formula, left = 0, right = Inf, dist = "gaussian",
      subset = NULL, data = list(), ...)

Arguments

formula a symbolic description of a regression model of type y ~ x1 + x2 + ....
left left limit for the censored dependent variable y. If set to -Inf, y is assumed not
to be left-censored.
right right limit for the censored dependent variable y. If set to Inf, the default, y is
assumed not to be right-censored.
dist assumed distribution for the dependent variable y. This is passed to survreg,
see the respective man page for more details.
subset a specification of the rows to be used.
data a data frame containing the variables in the model.
... further arguments passed to survreg.
Details

The function Tobit is an alias for the AER function `tobit` (Achim Zeileis <Achim.Zeileis@R-project.org>). All details can be found there.

Value

An object of class "Tobit" inheriting from class "survreg".

Author(s)

Andri Signorell

Examples

```r
# still to do
```

Description

Some classifiers benefit more from adjusted parameters to a particular dataset than others. However, it is often not clear from the beginning how the parameters have to be determined. What often only remains is a grid search when several parameters have to be found in combination. The present function uses a grid search approach for the decisive arguments (typically for a neural network, a random forest or a classification tree). However it’s not restricted to these models, any model fulfilling weak interface standards could be provided.

Usage

```r
Tune(x, ..., testset = NULL, keepmod = TRUE)
```

Arguments

- `x` the model to be tuned, best (but not necessarily) trained with `FitMod`.
- `...` a list of parameters, containing the values to be used for a grid search.
- `testset` a testset containing all variables required in the model to be used for calculating independently the accuracy (normally a subset of the original dataset).
- `keepmod` logical, defining if all fitted models should be returned in the result set. Default is TRUE. (Keep an eye on your RAM!)
Details

The function creates a n-dimensional grid according to the given parameters and calculates the model with the combinations of all the parameters. The accuracy for the models are calculated insample and on a test set, if one has been provided.

It makes sense to avoid overfitting to provide a test set to also be evaluated. A matrix with all combination of the values for the given parameters, sorted by accuracy, either by the accuracy achieved in the test set or the insample accuracy is returned.

Value

a matrix with all supplied parameters and a column "acc" and "test_acc" (if a test set has been provided)

Author(s)

Andri Signorell <andri@signorell.net>

Examples

d.pim <- SplitTrainTest(d.pima, p = 0.2)
diabetes ~ pregnant + glucose + pressure + triceps + insulin + mass + pedigree + age

# tune a neural network for size and decay
r.nn <- FitMod(mdiab, data=d.pim$train, fitfn="nnet")
(tu <- Tune(r.nn, size=12:17, decay = 10^(-4:-1), testset=d.pim$test))

# tune a random forest
r.rf <- FitMod(mdiab, data=d.pim$train, fitfn="randomForest")
(tu <- Tune(r.rf, mtry=seq(2, 20, 2), testset=d.pim$test))

# tune a SVM model
r.svm <- FitMod(mdiab, data=d.pim$train, fitfn="svm")


tu <- Tune(r.svm,
kernel=c("radial", "sigmoid"),
cost=c(0.1,1,10,100,1000),
gamma=c(0.5,1,2,3,4), testset=d.pim$test)

# let's get some more quality measures

tu$modpar$Sens <- sapply(tu$mods, Sens) # Sensitivity
tu$modpar$Spec <- sapply(tu$mods, Spec) # Specificity

Sort(tu$modpar, ord="test_acc", decreasing=TRUE)
Variable importance is an expression of the desire to know how important a variable is within a group of predictors for a particular model. But in general it is not a well defined concept, say there is no theoretically defined variable importance metric. Nevertheless, there are some approaches that have been established in practice for some regression and classification algorithms. The present function provides an interface for calculating variable importance for some of the models produced by fitMod, comprising linear models, classification trees, random forests, C5 trees and neural networks. The intention here is to provide reasonably homogeneous output and plot routines.

Usage

VarImp(x, scale = FALSE, sort = TRUE, ...)  

## S3 method for class 'VarImp'  
VarImp(x, scale = FALSE, sort = TRUE, type=NULL, ...)  

## Default S3 method:  
VarImp(x, scale = FALSE, sort = TRUE, ...)  

## S3 method for class 'VarImp'  
plot(x, sort = TRUE, maxrows = NULL,  
main = "Variable importance", ...)  

## S3 method for class 'VarImp'  
print(x, digits = 3, ...)  

Arguments

x the fitted model  
scale logical, should the importance values be scaled to 0 and 100?  
... parameters to pass to the specific VarImp methods  
sort the name of the column, the importance table should be ordered after  
maxrows the maximum number of rows to be reported  
main the main title for the plot  
type some models have more than one type available to produce a variable importance. Linear models accept one of "lmg","pmvd","first","last","betasq","pratt".  
digits the number of digits for printing the "VarImp" table
Details

**Linear Models:** For linear models there’s a fine package `relaimpo` available on CRAN containing several interesting approaches for quantifying the variable importance. See the original documentation.

**rpart, Random Forest:** `VarImp.rpart` and `VarImp.randomForest` are wrappers around the importance functions from the `rpart` or `randomForest` packages, respectively.

**C5.0:** C5.0 measures predictor importance by determining the percentage of training set samples that fall into all the terminal nodes after the split. For example, the predictor in the first split automatically has an importance measurement of 100 percent since all samples are affected by this split. Other predictors may be used frequently in splits, but if the terminal nodes cover only a handful of training set samples, the importance scores may be close to zero. The same strategy is applied to rule-based models and boosted versions of the model. The underlying function can also return the number of times each predictor was involved in a split by using the option `metric="usage"`.

**Neural Networks:** The method used here is "Garson weights".

**SVM, GLM, Multinom:** There are no implementations for these models so far.

Value

A data frame with class `c("VarImp.train", "data.frame")` for `VarImp.train` or a matrix for other models.

Author(s)

Andri Signorell <andri@signorell.net>

References


---

**zeroinfl**

Zero-inflated Count Data Regression

Description

Fit zero-inflated regression models for count data via maximum likelihood.

Usage

```r
ger(infl(formula, data, subset, na.action, weights, offset, 
    dist = c("poisson", "negbin", "geometric"), 
    link = c("logit", "probit", "cloglog", "cauchit", "log"), 
    control = zeroinfl.control(...), 
    model = TRUE, y = TRUE, x = FALSE, ...)```

Arguments

- **formula**: symbolic description of the model, see details.
- **data, subset, na.action**: arguments controlling formula processing via `model.frame`.
- **weights**: optional numeric vector of weights.
- **offset**: optional numeric vector with an a priori known component to be included in the linear predictor of the count model. See below for more information on offsets.
- **dist**: character specification of count model family (a log link is always used).
- **link**: character specification of link function in the binary zero-inflation model (a binomial family is always used).
- **control**: a list of control arguments specified via `zeroinfl.control`.
- **model, y, x**: logicals. If TRUE the corresponding components of the fit (model frame, response, model matrix) are returned.
- **...**: arguments passed to `zeroinfl.control` in the default setup.

Details

Zero-inflated count models are two-component mixture models combining a point mass at zero with a proper count distribution. Thus, there are two sources of zeros: zeros may come from both the point mass and from the count component. Usually the count model is a Poisson or negative binomial regression (with log link). The geometric distribution is a special case of the negative binomial with size parameter equal to 1. For modeling the unobserved state (zero vs. count), a binary model is used that captures the probability of zero inflation. in the simplest case only with an intercept but potentially containing regressors. For this zero-inflation model, a binomial model with different links can be used, typically logit or probit.

The formula can be used to specify both components of the model: If a formula of type \( y \sim x_1 + x_2 \) is supplied, then the same regressors are employed in both components. This is equivalent to \( y \sim x_1 + x_2 \mid x_1 + x_2 \). Of course, a different set of regressors could be specified for the count and zero-inflation component, e.g., \( y \sim x_1 + x_2 \mid z_1 + z_2 + z_3 \) giving the count data model \( y \sim x_1 + x_2 \) conditional on \( () \) the zero-inflation model \( y \sim z_1 + z_2 + z_3 \). A simple inflation model where all zero counts have the same probability of belonging to the zero component can be specified by the formula \( y \sim x_1 + x_2 \mid 1 \).

Offsets can be specified in both components of the model pertaining to count and zero-inflation model: \( y \sim x_1 + \text{offset}(x_2) \mid z_1 + z_2 + \text{offset}(z_3) \), where \( x_2 \) is used as an offset (i.e., with coefficient fixed to 1) in the count component and \( z_3 \) analogously in the zero-inflation component. By the rule stated above \( y \sim x_1 + \text{offset}(x_2) \) is expanded to \( y \sim x_1 + \text{offset}(x_2) \mid x_1 + \text{offset}(x_2) \). Instead of using the offset() wrapper within the formula, the offset argument can also be employed which sets an offset only for the count model. Thus, formula = \( y \sim x_1 \) and offset = \( x_2 \) is equivalent to formula = \( y \sim x_1 + \text{offset}(x_2) \mid x_1 \).

All parameters are estimated by maximum likelihood using `optim`, with control options set in `zeroinfl.control`. Starting values can be supplied, estimated by the EM (expectation maximization) algorithm, or by `glm.fit` (the default). Standard errors are derived numerically using the Hessian matrix returned by `optim`. See `zeroinfl.control` for details.
The returned fitted model object is of class "zeroinfl" and is similar to fitted "glm" objects. For elements such as "coefficients" or "terms" a list is returned with elements for the zero and count component, respectively. For details see below.

A set of standard extractor functions for fitted model objects is available for objects of class "zeroinfl", including methods to the generic functions `print`, `summary`, `coef`, `vcov`, `logLik`, `residuals`, `predict`, `fitted`, `terms`, `model.matrix`. See `predict.zeroinfl` for more details on all methods.

Value

An object of class "zeroinfl", i.e., a list with components including:

- **coefficients**: a list with elements "count" and "zero" containing the coefficients from the respective models,
- **residuals**: a vector of raw residuals (observed - fitted),
- **fitted.values**: a vector of fitted means,
- **optim**: a list with the output from the `optim` call for minimizing the negative log-likelihood,
- **control**: the control arguments passed to the `optim` call,
- **start**: the starting values for the parameters passed to the `optim` call,
- **weights**: the case weights used,
- **offset**: a list with elements "count" and "zero" containing the offset vectors (if any) from the respective models,
- **n**: number of observations (with weights > 0),
- **df.null**: residual degrees of freedom for the null model (= n - 2),
- **df.residual**: residual degrees of freedom for fitted model,
- **terms**: a list with elements "count", "zero" and "full" containing the terms objects for the respective models,
- **theta**: estimate of the additional \( \theta \) parameter of the negative binomial model (if a negative binomial regression is used),
- **SE.logtheta**: standard error for \( \log(\theta) \),
- **loglik**: log-likelihood of the fitted model,
- **vcov**: covariance matrix of all coefficients in the model (derived from the Hessian of the `optim` output),
- **dist**: character string describing the count distribution used,
- **link**: character string describing the link of the zero-inflation model,
- **linkinv**: the inverse link function corresponding to `link`,
- **converged**: logical indicating successful convergence of `optim`,
- **call**: the original function call,
- **formula**: the original formula,
- **levels**: levels of the categorical regressors,
contrasts  a list with elements "count" and "zero" containing the contrasts corresponding to levels from the respective models,
model     the full model frame (if model = TRUE),
y        the response count vector (if y = TRUE),
x        a list with elements "count" and "zero" containing the model matrices from the respective models (if x = TRUE),

Author(s)
Achim Zeileis <Achim.Zeileis@R-project.org>

References

See Also
zeroinfl.control, glm, glm.fit, glm.nb, hurdle

Examples
## data
data("bioChemists", package = "ModTools")

## without inflation
## ("art ~ ." is "art ~ fem + mar + kid5 + phd + ment")
fm_pois <- glm(art ~ ., data = bioChemists, family = poisson)
mq_pois <- glm(art ~ ., data = bioChemists, family = quasipoisson)
m_nb <- MASS::glm.nb(art ~ ., data = bioChemists)

## with simple inflation (no regressors for zero component)
fm_zip <- zeroinfl(art ~ 1, data = bioChemists)
m_zip <- zeroinfl(art ~ 1, data = bioChemists, dist = "negbin")

## inflation with regressors
## ("art ~ . | ." is "art ~ fem + mar + kid5 + phd + ment | fem + mar + kid5 + phd + ment")
fm_zip2 <- zeroinfl(art ~ . | ., data = bioChemists)
m_zip2 <- zeroinfl(art ~ . | ., data = bioChemists, dist = "negbin")
zeroinfl.control

Control Parameters for Zero-inflated Count Data Regression

Description

Various parameters that control fitting of zero-inflated regression models using zeroinfl.

Usage

zeroinfl.control(method = "BFGS", maxit = 10000, trace = FALSE, 
EM = FALSE, start = NULL, ...)

Arguments

- **method**: characters string specifying the method argument passed to optim.
- **maxit**: integer specifying the maxit argument (maximal number of iterations) passed to optim.
- **trace**: logical or integer controlling whether tracing information on the progress of the optimization should be produced (passed to optim).
- **EM**: logical. Should starting values be estimated by the EM (expectation maximization) algorithm? See details.
- **start**: an optional list with elements "count" and "zero" (and potentially "theta") containing the coefficients for the corresponding component.
- **...**: arguments passed to optim.

Details

All parameters in zeroinfl are estimated by maximum likelihood using optim with control options set in zeroinfl.control. Most arguments are passed on directly to optim, only trace is also used within zeroinfl and EM/start control the choice of starting values for calling optim.

Starting values can be supplied, estimated by the EM (expectation maximization) algorithm, or by glm.fit (the default). Standard errors are derived numerically using the Hessian matrix returned by optim. To supply starting values, start should be a list with elements "count" and "zero" and potentially "theta" (for negative binomial components only) containing the starting values for the coefficients of the corresponding component of the model.

Value

A list with the arguments specified.

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

See Also

zeroinfl
Examples

## Not run:
data("bioChemists", package = "pscl")

## default start values
fm1 <- zeroinfl(art ~ ., data = bioChemists)

## use EM algorithm for start values
fm2 <- zeroinfl(art ~ ., data = bioChemists, EM = TRUE)

## user-supplied start values
fm3 <- zeroinfl(art ~ ., data = bioChemists,
               start = list(count = c(0.7, -0.2, 0.1, -0.2, 0, 0), zero = -1.7))

## End(Not run)
Index

* **classif**
  * LogitBoost, 18

* **datasets**
  * bioChemists, 9
  * d.glass, 13
  * d.pima, 14

* **htest**
  * BreuschPaganTest, 9

* **misc**
  * PlotLift, 22

* **models**
  * CP, 12
  * FitMod, 15
  * Response, 27
  * Rules, 30
  * TModC, 32
  * VarImp, 36

* **model**
  * ROC, 29
  * SplitTrainTest, 31

* **package**
  * ModTools-package, 2

* **regression**
  * coef, 24, 39
  * coef.zeroinfl (predict.zeroinfl), 23
  * CoeffDiffCI, 11
  * CP, 4, 12
  * currency substitution, 10
  * d.glass, 13
  * d.pima, 14
  * d.pima2 (d.pima), 14
  * drop1.FitMod (FitMod), 15
  * dummy.coef, 26
  * extractAIC.zeroinfl (predict.zeroinfl), 23
  * FitMod, 3, 15, 26, 34
  * fitted, 24, 39
  * fitted.zeroinfl (predict.zeroinfl), 23
  * formula, 15, 18
  * GarsonWeights (VarImp), 36
  * glm, 3, 40
  * glm.fit, 38, 40, 41
  * glm.nb, 40
  * growthofmoney, 10
  * hurdle, 40
  * lda, 3
  * LeafRates, 4, 16, 20, 21
  * linearHypothesis, 12
  * lm, 3, 10, 16, 27
  * lmrob, 3
  * LogitBoost, 3, 18
  * logLik, 24, 39
  * logLik.zeroinfl (predict.zeroinfl), 23
  * model.frame, 27, 38
  * model.matrix, 24, 39

AIC, 24
AUC, 33

BestCut, 4, 7, 21
bioChemists, 9
bondyield, 10
BreuschPaganTest, 4, 9
BrierScore, 33
model.matrix.zeroinfl
(predict.zeroinfl), 23
model.response, 27
ModTools (ModTools-package), 2
ModTools-package, 2
moneydemand, 10
multinom, 3

naive_bayes, 3
ncvTest, 10
nnet, 3
Node, 4, 17, 20

optim, 38, 41
Over-/Undersample, 21
OverSample, 4
OverSample (Over-/Undersample), 21

path.rpart, 30, 31
plot.CP (CP), 12
plot.FitMod (FitMod), 15
plot.LeafRates (LeafRates), 16
plot.TModC (TModC), 32
plot.VarImp (VarImp), 36
plotcp, 13
PlotLift, 4, 22
polr, 3
predict, 4, 24, 39
predict.FitMod (FitMod), 15
PredictCI, 25
predprob.zeroinfl (predict.zeroinfl), 23
print, 24, 39
print.CP (CP), 12
print.FitMod (FitMod), 15
print.summary.zeroinfl
(predict.zeroinfl), 23
print.VarImp (VarImp), 36
print.zeroinfl (zeroinfl), 37
printcp, 13
pROC, 29
Purity (LeafRates), 16

qda, 3

randomForest, 3
RefLevel, 26, 27
relevel, 27
residuals, 24, 39
residuals.zeroinfl (predict.zeroinfl), 23
Response, 4, 27, 27
RobSummary, 4, 28
ROC, 4, 8, 29, 33
rpart, 3, 16, 18, 31
Rules, 4, 17, 21, 30
Splits (Node), 20
SplitTrainTest, 3, 31
summary, 24, 39
summary.FitMod (FitMod), 15
summary.glm, 28
summary.lm, 28
summary.zeroinfl (predict.zeroinfl), 23
survreg, 33
svm, 3
terms, 24, 39
terms.zeroinfl (predict.zeroinfl), 23
TMod, 33
TModC, 4, 32
Tobit, 33
tobit, 34
Tune, 4, 34
UnderSample (Over-/Undersample), 21
unemployment, 10

VarImp, 4, 36
vcov, 24, 39
vcov.zeroinfl (predict.zeroinfl), 23
vcovHC, 28

wages, 10

zeroinfl, 24, 25, 37, 41
zeroinfl.control, 38, 40, 41, 41