Package ‘MuMIn’

August 8, 2014

Type Package
Title Multi-model inference
Version 1.10.5
Date 2014-08-08
Encoding UTF-8
Author Kamil Bartoń
Maintainer Kamil Bartoń <kamil.barton@go2.pl>
Description Model selection and model averaging based on information criteria (AICc and alike).
License GPL-2
Depends R (>= 3.0.0)
Imports stats
Suggests stats4, nlme, mgcv (>= 1.7.5), lme4 (>= 0.999375-16), gamm4, MASS, nnet, survival, geepack
Enhances aod, coxme, glmmML, MCMCglmm, pscl, spdep, unmarked, ordinal, gee, splm, logistf, caper, betareg, aods3, cplm
LazyLoad yes

R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MuMIn-package</td>
<td>2</td>
</tr>
<tr>
<td>AICc</td>
<td>3</td>
</tr>
<tr>
<td>Beetle</td>
<td>5</td>
</tr>
<tr>
<td>Cement</td>
<td>7</td>
</tr>
<tr>
<td>dredge</td>
<td>8</td>
</tr>
<tr>
<td>Formula manipulation</td>
<td>12</td>
</tr>
<tr>
<td>get.models</td>
<td>13</td>
</tr>
<tr>
<td>importance</td>
<td>14</td>
</tr>
<tr>
<td>Information criteria</td>
<td>16</td>
</tr>
<tr>
<td>merge.model.selection</td>
<td>17</td>
</tr>
<tr>
<td>Model utilities</td>
<td>18</td>
</tr>
<tr>
<td>model.avg</td>
<td>19</td>
</tr>
</tbody>
</table>
The package **MuMIn** contains functions to streamline information-theoretic model selection and carry out model averaging based on the information criteria.

The collection of functions includes:

- *dredge* performs automated model selection with subsets of the supplied ‘global’ model, and optional choices of other model properties (such as different link functions). The set of models may be generated either with ‘all possible’ combinations, or tailored according to the conditions specified.
- *pdredge* does the same, but can parallelize model fitting process using a cluster.
- *model.sel* creates a model selection table from hand-picked models.
- *model.avg* calculates model averaged parameters, with standard errors and confidence intervals.
- *AICc* calculates second-order Akaike information criterion.

For a complete list of functions, use `library(help = "MuMIn")`.

By default, **AIC** is used to rank the models and to obtain model selection probabilities, though any other information criteria can be utilised. At least the following ones are currently implemented in **R**: **AIC** and **BIC** in package **stats**, and **QAIC**, **QAICc**, **ICOMP**, **CAICF**, and Mallows’ **Cp** in **MuMIn**.

There is also **DIC** extractor for MCMC models, and **QIC** for GEE.

Most of **R**’s common modelling functions are supported, for a full inventory see list of supported models.

**Author(s)**

Kamil Bartoń

**References**

See Also

AIC, step or stepAIC for stepwise model selection by AIC.

Examples

data(Cement)

options(na.action = "na.fail") # change the default "na.omit" to prevent models
# from being fitted to different datasets in
# case of missing values.

fm1 <- lm(y ~ ., data = Cement)
ms1 <- dredge(fm1)
plot(ms1)

model.avg(ms1, subset = delta < 4)

confset.95p <- get.models(ms1, cumsum(weight) <= .95)
avgmod.95p <- model.avg(confset.95p)
summary(avgmod.95p)

confint(avgmod.95p)

AICc

Second-order Akaike Information Criterion

Description

Calculate second-order Akaike information criterion for one or several fitted model objects (AIC<sub>c</sub>,
AIC for small samples).

Usage

AICc(object, ..., k = 2, REML = NULL)

Arguments

object

... a fitted model object for which there exists a logLik method, or a logLik object.

k

the ‘penalty’ per parameter to be used; the default k = 2 is the classical AIC.

REML

optional logical value, passed to the logLik method indicating whether the re-
stricted log-likelihood or log-likelihood should be used. The default is to use the
method used for model estimation.

Value

If just one object is provided, returns a numeric value with the corresponding AIC<sub>c</sub>; if more than
one object are provided, returns a data.frame with rows corresponding to the objects and columns
representing the number of parameters in the model (df) and AIC<sub>c</sub>.
Note

AICc should be used instead of AIC when the sample size is small in comparison to the number of estimated parameters (Burnham & Anderson 2002 recommend its use when $n/K < 40$).

Author(s)

Kamil Bartoń

References


See Also

Akaike’s An Information Criterion: AIC

Other implementations: AICc in package AICcmodavg, AICc in package bbmle and aicc in package glmulti

Examples

```r
# Model-averaging mixed models
if(require(nlme)) {
  data(Orthodont, package = "nlme")

  # Fit model by REML
  fm2 <- lme(distance ~ Sex*age, data = Orthodont,
             random = ~ 1|Subject / Sex, method = "REML")

  # Model selection: ranking by AICc using ML
  ms2 <- dredge(fm2, trace = TRUE, rank = "AICc", REML = FALSE)

  (attr(ms2, "rank.call"))

  # Get the models (fitted by REML, as in the global model)
  fmList <- get.models(ms2, 1:4)

  # Because the models originate from dredge(..., rank=AICc, REML=FALSE),
  # the default weights in model.avg are ML based:
  summary(model.avg(fmList))

  # same result
  #model.avg(fmList, rank = "AICc", rank.args = list(REML=FALSE))
}
```
Flour beetle mortality data

Description
Mortality of flour beetles (*Tribolium confusum*) due to exposure to gaseous carbon disulfide CS$_2$, from Bliss (1935).

Usage
data(Beetle)

Format
Beetle is a data frame with 5 elements.

- **dose** The dose of CS$_2$ in mg/L.
- **n.tested** Number of beetles tested
- **n.killed** Number of beetles killed
- **Prop** A matrix with two columns named **n.killed** and **n.survived**
- **mortality** Observed mortality rate.

Source

References

Examples

# "Logistic regression example"
# from Burnham & Anderson (2002) chapter 4.11
data(Beetle)

# Fit a global model with all the considered variables
globmod <- glm(Prop ~ dose + I(dose^2) + log(dose) + I(log(dose)^2),
               data = Beetle, family = binomial)

# A logical expression defining the subset of models to use:
# * either log(dose) or dose
# * the quadratic terms can appear only together with linear terms
msubset <- expression(xor(dose,log(dose)) & (dose | !I(dose^2))
                      & (log(dose) | !I(log(dose)^2)))
# Table 4.6
# Use varying argument to fit models with different link functions
# Note the use of alist rather than list in order to keep the
# family objects unevaluated
varying.link <- list(family = alist(
    logit = binomial("logit"),
    probit = binomial("probit"),
    cloglog = binomial("cloglog")
))

(ms12 <- dredge(globmod, subset = msubset, varying = varying.link,
    rank = AIC))

# Table 4.7 "models justifiable a priori"
(ms3 <- subset(ms12, has(dose, !I(dose^2))))
# The same result, but would fit the models again:
# ms3 <- update(ms12, update(globmod, . ~ dose), subset =,
#    fixed = ~dose)

mod3 <- get.models(ms3, 1:3)

# Table 4.8. Predicted mortality probability at dose 40.
# calculate confidence intervals on logit scale
logit.ci <- function(p, se, quantile = 2) {
  C. <- exp(quantile * se / (p * (1 - p)))
  p / (p + (1 - p) * c(C., 1/C.))
}

mavg3 <- model.avg(mod3, revised.var = FALSE)
pred <- sapply(mod3, predict, newdata = list(dose = 4/zero.noslash), se.fit = TRUE,
    type = "response")

# get predictions both from component and averaged models
pred <- lapply(c(component = mod3, list(averaged = mavg3)), predict,
    newdata = list(dose = 40), type = "response", se.fit = TRUE)
# reshape predicted values
pred <- t(sapply(pred, function(x) unlist(x)[1:2]))

# build the table
tab <- cbind(
  c(Weights(ms3), NA),
  pred,
  matrix(logit.ci(pred[,"fit"], pred[,"se.fit"],
    quantile = c(rep(1.96, 3), 2)), ncol = 2)
)

colnames(tab) <- c("Akaike weight", "Predicted(40)", "SE", "Lower CI",
    "Upper CI")
rownames(tab) <- c(as.character(ms3$family), "model averaged")
print(tab, digits = 3, na.print = "")

# Figure 4.3
newdata <- list(dose = seq(min(Beetle$dose), max(Beetle$dose), length.out = 25))
Cement hardening data

Description

Cement hardening data from Woods et al (1939).

Usage

data(Cement)

Format

Cement is a data frame with 5 variables. x1-x4 are four predictor variables expressed as a percentage of weight.

X1 calcium aluminate
X2 tricalcium silicate
X3 tetracalcium alumino ferrite
X4 dicalcium silicate
y calories of heat evolved per gram of cement after 180 days of hardening.

Source


References

dredge

Automated model selection

Description

Generate a set of models with combinations (subsets) of the terms in the global model, with optional rules for model inclusion.

Usage

```r
dredge(global.model, beta = FALSE, evaluate = TRUE, rank = "AICc",
fixed = NULL, m.max = NA, m.min = 0, subset, marg.ex = NULL,
trace = FALSE, varying, extra, ct.args = NULL, ...)
```

## S3 method for class model.selection

```r
print(x, abbrev.names = TRUE, warnings = getOption("warn") != -1L, ...)
```

Arguments

- **global.model**: a fitted ‘global’ model object. See ‘Details’ for a list of supported types.
- **beta**: logical, should standardized coefficients be returned?
- **evaluate**: whether to evaluate and rank the models. If FALSE, a list of model calls is returned.
- **rank**: optional custom rank function (information criterion) to be used instead AICc, e.g. AIC, QAIC or BIC. See ‘Details’.
- **fixed**: optional, either a single sided formula or a character vector giving names of terms to be included in all models. See ‘Subsetting’.
- **m.max, m.min**: optionally, the maximum and minimum number of terms in a single model (excluding the intercept), m.max defaults to the number of terms in global.model. See ‘Subsetting’.
- **subset**: logical expression describing models to keep in the resulting set. See ‘Subsetting’.
- **marg.ex**: a character vector specifying names of variables for which NOT to check for marginality restrictions when generating model formulas. If this argument is set to TRUE, all combinations of terms are used (i.e. no checking). If NA or missing, the exceptions will be found based on the terms of global.model. See ‘Details’.
- **trace**: if TRUE, all calls to the fitting function (i.e. updated global.model calls) are printed before actual fitting takes place.
- **varying**: optionally, a named list describing the additional arguments to vary between the generated models. Item names correspond to the arguments, and each item provides a list of choices (i.e. list(arg1 = list(choice1, choice2, ...), ...)). Complex elements in the choice list (such as family objects) should be either named (uniquely) or quoted (unevaluated, e.g. using alist, see quote), otherwise it may produce rather unpleasant effects. See example in Beetle.
- **extra**: optional additional statistics to include in the result, provided as functions, function names or a list of such (best if named or quoted). Similarly as in rank argument, each function must accept fitted model object as an argument and
return a (value coercible to a) numeric vector. These can be e.g. additional information criterions or goodness-of-fit statistics. The character strings \texttt{"R^2"} and \texttt{"adjR^2"} are treated in a special way, and will add a likelihood-ratio based $R^2$ and modified-$R^2$ respectively to the result (this is more efficient than using \texttt{r.squaredLR} directly).

- \texttt{x} a model.selection object, returned by \texttt{dredge}.
- \texttt{abbrev.names} should printed variable names be abbreviated? (useful with many variables).
- \texttt{warnings} if \texttt{TRUE}, errors and warnings issued during the model fitting are printed below the table (currently, only with \texttt{pdredge}). To permanently remove the warnings, set the object’s attribute \texttt{"warnings"} to \texttt{NULL}.
- \texttt{ct.args} optional list of arguments to be passed to \texttt{coefTable} (e.g. dispersion parameter for \texttt{glm} affecting standard errors used in subsequent model averaging).
- \texttt{...} optional arguments for the \texttt{rank} function. Any can be an expression (of mode \texttt{call}), in which case any \texttt{x} within it will be substituted with a current model.

Details

Models are fitted through repeated evaluation of modified calls to the \texttt{global.model} (in a similar fashion as with \texttt{update}). This approach, while robust in that it can be applied to a variety of different model object types is not very efficient and may be time-intensive.

Note that the number of combinations grows exponentially with number of predictor variables ($2^N$, less when interactions are present, see below).

The fitted model objects are not stored in the result. To get (a subset of) the models, use \texttt{get.models} on the object returned by \texttt{dredge}.

For a list of model types that can be used as a \texttt{global.model} see \texttt{list of supported models}. Modelling functions not storing \texttt{call} in their result should be evaluated \texttt{via} the wrapper function created by \texttt{updateable}.

Information criterion: \texttt{rank} is found by a call to \texttt{match.fun} and may be specified as a function or a symbol or a character string specifying a function to be searched for from the environment of the call to \texttt{dredge}. The function \texttt{rank} must accept model object as its first argument and always return a scalar.

Interactions: By default marginality constraints are respected, so “all possible combinations” do not include models containing interactions without their respective main effects and all lower order terms. This behaviour can be altered by \texttt{marg.ex} argument, which can be used to allow for simple nested designs. For example, with global model of form \texttt{a / (x + z)}, one would use \texttt{marg.ex = "a" and fixed = "a"}. If \texttt{global.model} uses such a formula and \texttt{marg.ex} is missing or \texttt{NA}, it will be adjusted automatically.

Subsetting: There are three ways to constrain the resulting set of models: setting limits to the number of terms in a model with \texttt{m.max} and \texttt{m.min}, binding term(s) to all models with \texttt{fixed}, and more complex rules can be applied using argument \texttt{subset}. To be included in the selection table, the model formulation must satisfy all these conditions.

\texttt{subset} can take either a form of an \texttt{expression} or a \texttt{matrix}. The latter should be a lower triangular matrix with logical values, where columns and rows correspond to \texttt{global.model} terms. Value \texttt{subset["a", "b"] = FALSE} will exclude any model containing both terms \texttt{a} and \texttt{b}.

In the form of expression, the argument \texttt{subset} acts in a similar fashion to that in the function \texttt{subset} for \texttt{data.frames}: model terms can be referred to by name as variables in the expression, with the difference being that they are always logical (i.e. \texttt{TRUE} if a term exists in the model).
There is also .(x) and .(+x) notation that indicate respectively any or all model terms including a variable x. This concerns only interactions containing a particular main effect x (e.g. x:z, v:x:z), and not a variable in a complex expression, such as x in log(x) or I(x^2). This is only useful with marginality exceptions (marg.ex argument).

The expression can contain any of the global.model terms (getAllTerms(global.model) lists them), as well as names of the varying argument items. Names of global.model terms take precedence when identical to names of varying, so to avoid ambiguity varying variables in subset expression should be enclosed in V() (e.g. subset = V(family) == "Gamma" assuming that varying is something like list(family = c(..., "Gamma"))).

If item names in varying are missing, the items themselves are coerced to names. Call and symbol elements are represented as character values (via deparse), and everything except numeric, logical, character and NULL values is replaced by item numbers (e.g. varying = list(family = c(..., Gamma)) should be referred to as subset = V(family) == 2. This can quickly become confusing, therefore it is recommended to use named lists. demo(dredge.varying) provides examples.

The subset expression can also contain variable `*nvar*` (backtick-quoted), equal to number of terms in the model (not the number of estimated parameters).

To make inclusion of a variable conditional on presence of some other variable, the function dc ("dependency chain") can be used in the subset expression. dc takes any number of variables as arguments, and allows a variable to be included only if all preceding variables are also present (e.g. subset = dc(a, b, c) allows for models of form a, a+b and a+b+c but not b, c, b+c or a+c).

subset expression can have a form of an unevaluated call, expression object, or a one sided formula. See 'Examples'.

Compound model terms (such as interactions, ‘as-is’ expressions within I() or smooths in gam) should be treated as non-syntactic names and enclosed in backticks (e.g. subset = `a:b` & `s(x, k = 2)` | `I(log(x))`) see Quotes. Mind the spacing, names must match exactly the term names as given by getAllTerms.

To simply keep certain terms in all models, use of argument fixed is much more efficient. The fixed formula is interpreted in the same manner as model formula and so the terms need not to be quoted.

subset expression syntax summary:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a &amp; b</td>
<td>indicates that model terms a and b must be present (see Logical Operators)</td>
</tr>
<tr>
<td>V(x)</td>
<td>indicates a varying variable x</td>
</tr>
<tr>
<td>.(x)</td>
<td>indicates that at least one term containing variable x must be present</td>
</tr>
<tr>
<td>.(+x)</td>
<td>indicates that all the terms containing variable x must be present</td>
</tr>
<tr>
<td>dc(a, b, c,...)</td>
<td>‘dependency chain’: b is allowed only if a is present, and c only if both a and b are present, etc.</td>
</tr>
<tr>
<td><code>*nvar*</code></td>
<td>number of variables</td>
</tr>
</tbody>
</table>

Missing values: Use of na.action = "na.omit" (R’s default) or "na.exclude" in global.model must be avoided, as it results with sub-models fitted to different data sets, if there are missing values. Error is thrown if it is detected.

It is a common mistake to give na.action as an argument in the call to dredge (typically resulting in an error from the rank function to which the argument is passed through ...), while the correct way is either to pass na.action in the call to the global model or to set it as a global option.

Methods: There are subset and plot methods, the latter produces a graphical representation of model weights and variable relative importance. Coefficients can be extracted with coef or coefTable.
**dredge**

**Value**

dredge returns an object of class `model.selection`, being a `data.frame` with models’ coefficients (or presence/NA for factors), `df` - number of parameters, log-likelihood, the information criterion value, delta-IC and *Akaike weight*. Models are ordered by the value of the information criterion specified by `rank` (lowest on top).

The attribute "calls" is a list containing the model calls used (arranged in the same order as the models). A model call can be retrieved with `getCall(*, i)` where `i` is a vector of model index or name (if given as character string).

Other attributes: "global" - the `global.model` object, "rank" - the rank function used, "call" - the matched call, and "warnings" - list of errors and warnings given by the modelling function during the fitting, with model number appended to each.

**Note**

Users should keep in mind the hazards that a “thoughtless approach” of evaluating all possible models poses. Although this procedure is in certain cases useful and justified, it may result in selecting a spurious “best” model, due to the model selection bias.

*“Let the computer find out” is a poor strategy and usually reflects the fact that the researcher did not bother to think clearly about the problem of interest and its scientific setting* (Burnham and Anderson, 2002).

**Author(s)**

Kamil Bartoń

**See Also**

- `pdredge` is a parallelized version of this function (uses a cluster).
- `get.models`, `model.avg`, `model.sel` for manual model selection tables.

Possible alternatives: `glmulti` in package `glmulti` and `bestglm`(`bestglm`). `regsubsets` in package `leaps` also performs all-subsets regression.

*Lasso* variable selection provided by various packages, e.g. `glmnet`, `lars` or `glmmLasso`.

**Examples**

```r
# Example from Burnham and Anderson (2002), page 100:
data(Cement)
options(na.action = "na.fail")  # prevent fitting models to different datasets

fm1 <- lm(y ~ ., data = Cement)
dd <- dredge(fm1)
subset(dd, delta < 4)

# Visualize the model selection table:
if(require(graphics))
  plot(dd)

# Model average models with delta AICc < 4
model.avg(dd, subset = delta < 4)

# or as a 95% confidence set:
```
model.avg(dd, subset = cumsum(weight) <= .95) # get averaged coefficients

#Best model
summary(get.models(dd, 1)[[1]])

## Not run:
# Examples of using subset:
# keep only models containing X3
dredge(fm1, subset = ~ X3) # subset as a formula
dredge(fm1, subset = expression(X3)) # subset as expression object
# the same, but more effective:
dredge(fm1, fixed = "X3")
# exclude models containing both X1 and X2 at the same time
dredge(fm1, subset = !(X1 && X2))
# Fit only models containing either X3 or X4 (but not both);
# include X3 only if X2 is present, and X2 only if X1 is present.
dredge(fm1, subset = dc(X1, X2, X3) && xor(X3, X4))
# the same as above, but without using "dc"
dredge(fm1, subset = (X1 | !X2) && (X2 | !X3) && xor(X3, X4))

# Include only models with up to 2 terms (and intercept)
dredge(fm1, m.max = 2)

## End(Not run)

# Add R^2 and F-statistics, use the extra argument
dredge(fm1, m.max = 1, extra = c("R^2", F = function(x)
  summary(x)$fstatistic[[1]])

# with summary statistics:
dredge(fm1, m.max = 1, extra = list(
  "R^2", "x" = function(x) {
    s <- summary(x)
    c(Rsq = s$r.squared, adjRsq = s$adj.r.squared,
      F = s$fstatistic[[1]])
  })

# Add other information criterions (but rank with AICc):
dredge(fm1, m.max = 1, extra = alist(AIC, BIC, ICOMP, Cp))

Formula manipulation  Manipulate model formulas

Description
simplify.formula rewrites a formula using shorthand notation. Currently only the factor crossing operator * is applied, so that expanded expression such as a+b+a:b becomes a*b. expand.formula does the opposite, additionally expanding other expressions, i.e. all nesting (/), grouping and ^.
get.models

Usage

simplify.formula(x)
expand.formula(x)

Arguments

x  a formula or an object from which it can be extracted (such as a fitted model object).

Author(s)

Kamil Bartoń

See Also

formula
delete.response, drop.terms, and reformulate

Examples

simplify.formula(y ~ a + b + a:b + (c + b)^2)
simplify.formula(y ~ a + b + a:b + 0)
expand.formula(~ a * b)

get.models  Evaluate models from selection table

Description

Generate a list of fitted model objects from a model.selection table, optionally using parallel computation in a cluster.

Usage

get.models(object, subset, cluster = NA, ...)

Arguments

object  object returned by dredge.
subset  subset of models, an expression evaluated within the model selection table (see 'Details').
cluster  optionally, a cluster object. If it is a valid cluster, models are evaluated using parallel computation.
...  additional arguments to update the models. For example, in lme one may want to use method = "REML" while using "ML" for model selection.
importance

Details
The argument subset must be explicitly provided. This is to assure that a potentially long list of models is not fitted unintentionally. To evaluate all models, set subset to NA or TRUE.

If subset is a character vector, it is interpreted as names of rows to be selected.

Value
list of fitted model objects.

Note
pget.models is still available, but is deprecated.

Author(s)
Kamil Bartoń

See Also
dredge and pdredge, model.avg
makeCluster in packages parallel and snow

Examples
# Mixed models:

fm2 <- lme(distance ~ age + Sex, data = Orthodont,
random = ~ 1 | Subject, method = "ML")
ms2 <- dredge(fm2)

# Get top-most models, but fitted by REML:
(confset.d4 <- get.models(ms2, subset = delta < 4, method = "REML"))

## Not run:
# Get the top model:
get.models(ms2, subset = 1)[[1]]

## End(Not run)

<table>
<thead>
<tr>
<th>importance</th>
<th>Relative variable importance</th>
</tr>
</thead>
</table>

Description
Sum of ‘Akaike weights’ over all models including the explanatory variable.

Usage
importance(x)
importance

Arguments

x Either a list of fitted model objects, or a "model.selection" or "averaging" object.

Value

a numeric vector of relative importance values, named as the predictor variables.

Author(s)

Kamil Bartoń

See Also

Weights
dredge, model.avg, mod.sel

Examples

# Generate some models
data(Cement)
fm1 <- lm(y ~ ., data = Cement)
ms1 <- dredge(fm1)

# Importance can be calculated/extracted from various objects:
importance(ms1)
## Not run:
importance(subset(mod.sel(ms1), delta <= 4))
importance(model.avg(ms1, subset = delta <= 4))
importance(subset(ms1, delta <= 4))
importance(get.models(ms1, delta <= 4))
## End(Not run)

# Re-evaluate the importances according to BIC
# note that re-ranking involves fitting the models again

# nobs is not used here for backwards compatibility
lognobs <- log(length(resid(fm1)))
importance(subset(mod.sel(ms1, rank = AIC, rank.args = list(k = lognobs)),
    cumsum(weight) <= .95))

# This gives a different result than previous command, because subset is
# applied to the original selection table that is ranked with AICc
importance(model.avg(ms1, rank = AIC, rank.args = list(k = lognobs),
        subset = cumsum(weight) <= .95))
Information criteria

Various information criteria

Description

Calculate Mallows’ Cp and Bozdogan’s ICOMP and CAICF information criteria.

Extract or calculate Deviance Information Criterion from MCMCglmm and mer object.

Usage

Cp(object, ..., dispersion = NULL)
ICOMP(object, ..., REML = NULL)
CAICF(object, ..., REML = NULL)
DIC(object, ...)

Arguments

object a fitted model object (in case of ICOMP and CAICF, logLik and vcov methods must exist for the object). For DIC, an object of class MCMCglmm or mer.

... optionally more fitted model objects.

dispersion the dispersion parameter. If NULL, it is inferred from object.

REML optional logical value, passed to the logLik method indicating whether the restricted log-likelihood or log-likelihood should be used. The default is to use the method used for model estimation.

Details

Mallows’ Cp statistic is the residual deviance plus twice the estimate of $\sigma^2$ times the residual degrees of freedom. It is closely related to AIC (and a multiple of it if the dispersion is known).

ICOMP (I for informational and COMP for complexity) penalizes the covariance complexity of the model, rather than the number of parameters directly.

CAICF (C is for ‘consistent’ and F denotes the use of the Fisher information matrix) includes with penalty the natural logarithm of the determinant of the estimated Fisher information matrix.

Value

If just one object is provided, the functions return a numeric value with the corresponding IC; otherwise a data frame with rows corresponding to the objects is returned.

References

merge.model.selection

Combine model selection tables

Description

Combine two or more model selection tables.

Usage

## S3 method for class model.selection
merge(x, y, suffixes = c(".x", ".y"), ...)

Arguments

x, y model.selection objects to be combined.
suffixes a character vector with two elements that are appended respectively to row names of the combined tables
... ignored

Value

A model.selection object containing the selected models (rows).

Note

both delta IC values and IC-weights are recalculated in the resulting tables.

Unlike the merge method for data.frame, this method simply appends second table to the first.

Author(s)

Kamil Bartoń

See Also

dredge, model.sel, merge, rbind.
### Examples

```r
## Not run:
require(mgcv)
data(Beetle)

ms1 <- dredge(glm(Prop ~ dose + I(dose^2) + log(dose) + I(log(dose)^2),
                   data = Beetle, family = binomial, na.action = na.fail))

fm2 <- gam(Prop ~ s(dose, k = 3), data = Beetle, family = binomial)

merge(ms1, model.sel(fm2))

## End(Not run)
```

---

## Model utilities

### Model utility functions

#### Description

These functions extract or calculate various values from provided fitted model objects(s). They are mainly meant for internal use, but may be also useful for end-users.

- `beta.weights` computes standardized coefficients (beta weights) for a model;
- `coeffs` extracts model coefficients;
- `getAllTerms` extracts independent variable names from a model object;
- `coefTable` extracts a table of coefficients, standard errors and associated degrees of freedom when possible;
- `model.names` generates shorthand (alpha)numeric names for one or several fitted models;

#### Usage

```r
beta.weights(model)

coeffs(model)

ggetAllTerms(x, ...)
## S3 method for class terms
ggetAllTerms(x, offset = TRUE, intercept = FALSE, ...)

ccoefTable(model, ...)
## S3 method for class lme
ccoefTable(model, adjustSigma, ...)
## S3 method for class gee
ccoefTable(model, ..., type = c("naive", "robust"))

model.names(object, ..., labels = NULL, use.letters = FALSE)
```
Arguments

model a fitted model object.
object a fitted model object or a list of such objects.
x a fitted model object or a formula.
offset should ‘offset’ terms be included?
intercept should terms names include the intercept?
labels optionally, a character vector with names of all the terms, e.g. from a global model. model.names enumerates the model terms in order of their appearance in the list and in the models. So, changing the order of the models would lead to different names. The argument ‘labels’ can be used to prevent this happening.
... for model.names, more fitted model objects. For coefTable arguments that are passed to appropriate vcov or summary method (e.g. dispersion parameter for glm may be used here). In other functions often not used.
use.letters logical, whether letters should be used instead of numeric codes.
type for GEE models, the type of covariance estimator to calculate returned standard errors on. Either "naive" or "robust" ('sandwich').
adjustSigma See summary.lme.

Details

The functions coeffs, getAllTerms and coefTable provide interface between the model object and model.avg (and dredge). Custom methods can be written to provide support for additional classes of models.

Note

coeffs’s value is in most cases identical to that returned by coef, the only difference being it returns fixed effects’ coefficients for mixed models, and the value is always a named numeric vector.

Use of tTable is deprecated in favour of coefTable.

Author(s)

Kamil Bartoń
Usage

model.avg(object, ..., revised.var = TRUE)

## Default S3 method:
model.avg(object, ..., beta = FALSE, rank = NULL, rank.args = NULL, 
   revised.var = TRUE, dispersion = NULL, ct.args = NULL)

## S3 method for class model.selection
model.avg(object, subset, fit = FALSE, ..., revised.var = TRUE)

Arguments

object  a fitted model object or a list of such objects, or a model.selection object. See ‘Details’.

... for default method, more fitted model objects. Otherwise, arguments that are passed to the default method.

beta   logical, should standardized coefficients be returned?

rank optionally, a custom rank function (information criterion) to use instead of AICc, e.g. BIC or QAIC, may be omitted if object is a model list returned by get.models or a model.selection object. See ‘Details’.

rank.args optional list of arguments for the rank function. If one is an expression, an x within it is substituted with a current model.

revised.var logical, indicating whether to use revised formula for standard errors. See par.avg.

dispersion the dispersion parameter for the family used. See summary.glm. This is used currently only with glm, is silently ignored otherwise.

ct.args optional list of arguments to be passed to coefTable (besides dispersion).

subset see subset method for model.selection object.

fit if TRUE, the component models are fitted using get.models. See ‘Details’.

Details

model.avg may be used either with a list of models, or directly with a model.selection object (e.g. returned by dredge). In the latter case, the models from the model selection table are not evaluated unless the argument fit is set to TRUE or some additional arguments are present (such as rank or dispersion). This results in much faster calculation, but has certain drawbacks, because the fitted component model objects are not stored, and some methods (e.g. predict, fitted, model.matrix or vcov) would not be available with the returned object. Otherwise, get.models is called prior to averaging, and ... are passed to it.

For a list of model types that are accepted see list of supported models.

rank is found by a call to match.fun and typically is specified as a function or a symbol (e.g. a backquoted name) or a character string specifying a function to be searched for from the environment of the call to lapply. rank must be a function able to accept model as a first argument and must always return a scalar.

Several standard methods for fitted model objects exist for class averaging, including summary, predict, coef, confint, formula, and vcov.
The model.avg, vcov, confint and coefTable accept argument full that if set to TRUE, the full model-averaged coefficients are returned, rather than subset-averaged ones (when full = FALSE, being the default).

logLik returns a list of logLik objects for the component models.

**Value**

An object of class averaging is a list with components:

- **summary**: a data.frame with log-likelihood, IC, Delta(IC) and Akaike weights for the component models.
- **coef.shrinkage**: a vector of full model-averaged coefficients, see 'Note'.
- **coefArray**: an array of component models’ coefficients, their standard errors, and degrees of freedom.
- **term.codes**: names of the terms with numerical codes used in the summary.
- **avg.model**: the model averaged parameters. A data.frame containing averaged coefficients, unconditional standard error, adjusted SE (if df's are available) and z-values (coefficient and SE) and significance (assuming a normal error distribution).
- **importance**: relative importance of the predictor variables (including interactions), calculated as a sum of the Akaike weights over all of the models in which the parameter of interest appears.
- **term.names**: character vector giving names of all terms in the model.
- **x, formula**: the model matrix and formula corresponding to the one that would be used in a single model. formula contains only the averaged coefficients.
- **call**: the matched call.

In addition, the object has following attributes:

- **modelList**: a list of component model objects.
- **beta**: logical, were standardized coefficients used?
- **revised.var**: if TRUE, the standard errors were calculated with the revised formula (See par.avg).

**Note**

The ‘subset’ (or ‘conditional’) average only averages over the models where the parameter appears. An alternative, the ‘full’ average assumes that a variable is included in every model, but in some models the corresponding coefficient (and its respective variance) is set to zero. Unlike the ‘subset average’, it does not have a tendency of biasing the value away from zero. The ‘full’ average is a type of shrinkage estimator and for variables with a weak relationship to the response they are smaller than ‘subset’ estimators.

Averaging models with different contrasts for the same factor would yield nonsense results, currently no checking for contrast consistency is done.

From version 1.0.1, print method provides only a concise output (similarly as for lm). To print a full summary of the results use summary function. Confidence intervals can be obtained with confint.

**Author(s)**

Kamil Bartoń
References


See Also

See par.avg for more details of model averaged parameter calculation.
dredge, get.models
AICc has examples of averaging models fitted by REML.
modavg in package AICcmodavg, and coef.glmulti in package glmulti also perform model averaging.

Examples

# Example from Burnham and Anderson (2002), page 100:
data(Cement)
fm1 <- lm(y ~ ., data = Cement)
(ms1 <- dredge(fm1))
#models with delta.aicc < 4
summary(model.avg(ms1, subset = delta < 4))
#or as a 95% confidence set:
avgmod.95p <- model.avg(ms1, cumsum(weight) <= .95)
confint(avgmod.95p)

## Not run:
# The same result, but re-fitting the models via get.models
confset.95p <- get.models(ms1, cumsum(weight) <= .95)
model.avg(confset.95p)
# Force re-fitting the component models
model.avg(ms1, cumsum(weight) <= .95, fit = TRUE)
# Models are also fitted if additional arguments are given
model.avg(ms1, cumsum(weight) <= .95, rank = "AIC")

## End(Not run)

## Not run:
# using BIC (Schwarzs Bayesian criterion) to rank the models
BIC <- function(x) AIC(x, k = log(length(residuals(x))))
model.avg(confset.95p, rank = BIC)
# the same result, using AIC directly, with argument k
# x in a quoted rank argument is substituted with a model object
# (in this case it does not make much sense as the number of observations is
# common to all models)
model.avg(confset.95p, rank = AIC, rank.args = alist(k = log(length(residuals(x)))))

## End(Not run)
model.sel

model selection table

Description

Build a model selection table.

Usage

model.sel(object, ...)

## S3 method for class model.selection
model.sel(object, rank = NULL, rank.args = NULL, ..., beta = FALSE, extra)

## Default S3 method:
model.sel(object, ..., rank = NULL, rank.args = NULL, beta = FALSE, extra)

Arguments

object  
A fitted model object, a list of such objects, or a "model.selection" object.

...  
More fitted model objects.

rank  
Optional, custom rank function (information criterion) to use instead of AICc, e.g. QAIC or BIC, may be omitted if object is a model list returned by get.models.

rank.args  
Optional list of arguments for the rank function. If one is an expression, an x within it is substituted with a current model.

beta  
logical, should standardized coefficients be returned?

extra  
optional additional statistics to include in the result, provided as functions, function names or a list of such (best if named or quoted). See dredge for details.

Value

An object of class "model.selection" with columns containing useful information about each model: the coefficients, df, log-likelihood, the value of the information criterion used, Delta(IC) and ‘Akaike weight’. If any arguments differ between the modelling function calls, the result will include additional columns showing them (except for formulas and some other arguments).

Author(s)

Kamil Bartoń

See Also

dredge, AICc, list of supported models.

Possible alternatives: ICtab (in package bbmle), or aictab (AICmodavg).
Examples

```r
data(Cement)
Cement$X1 <- cut(Cement$X1, 3)
Cement$X2 <- cut(Cement$X2, 2)

fm1 <- glm(formula = y ~ X1 + X2 * X3, data = Cement)
fm2 <- update(fm1, . ~ . - X1 - X2)
fm3 <- update(fm1, . ~ . - X2 - X3)

## ranked with AICc by default
(msAICc <- model.sel(fm1, fm2, fm3))

## ranked with BIC
model.sel(fm1, fm2, fm3, rank = AIC, rank.args = alist(k = log(nobs(x))))
# or
# model.sel(msAICc, rank = AIC, rank.args = alist(k = log(nobs(x))))
# or
# update(msAICc, rank = AIC, rank.args = alist(k = log(nobs(x))))
```

MuMIn-models

List of supported models

Description

List of model classes accepted by `model.avg`, `model.sel`, and `dredge`.

Details

Fitted model objects that can be used with model selection and model averaging functions include those returned by:

- `lm`, `glm` (package `stats`);
- `rlm`, `glm.nb` and `polr` (package `MASS`);
- `multinom` (package `nnet`);
- `lme`, `glm` (package `nlme`);
- `lmer`, `glmer` (package `lme4`);
- `cplm` (package `cplm`);
- `gam`, `gamm` (package `mgcv`);
- `gamm4` (package `gamm4`);
- `glmmML` (package `glmmML`);
- `glmmadmb` (package `glmmADMB` from R-Forge);
- `asreml` (package `asreml`, non-free commercial package; REML comparisons only);
- `hurdle`, `zeroinfl` (package `pscl`);
- `negbin`, `betabin` (package `glimML`);
- `aodml`, `aodql` (package `aods3`);
par.avg

Description

Average a single model coefficient based on provided weights. It is mostly intended for internal use.

Usage

par.avg(x, se, weight, df = NULL, level = 1 - alpha, alpha = 0.05, revised.var = TRUE, adjusted = TRUE)
Arguments

- \(x\) vector of parameters.
- \(se\) vector of standard errors.
- \(weight\) vector of weights.
- \(df\) (optional) vector of degrees of freedom.
- \(alpha, level\) significance level for calculating confidence intervals.
- \(revised.var\) logical, should the revised formula for standard errors be used? See ‘Details’.
- \(adjusted\) logical, should the inflated standard errors be calculated? See ‘Details’.

Details

Unconditional standard errors are square root of the variance estimator, calculated either according to the original equation in Burnham and Anderson (2002, equation 4.7), or a newer, revised formula from Burnham and Anderson (2004, equation 4) (if \(revised.var = \text{TRUE}\), this is the default). If \(adjusted = \text{TRUE}\) (the default) and degrees of freedom are given, the adjusted standard error estimator and confidence intervals with improved coverage are returned (see Burnham and Anderson 2002, section 4.3.3).

Value

\(par.avg\) returns a vector with named elements:

- \(\text{Coefficient}\) model coefficients,
- \(\text{SE}\) unconditional standard error,
- \(\text{Adjusted SE}\) adjusted standard error,
- \(\text{Lower CI, Upper CI}\) unconditional confidence intervals.

Author(s)

Kamil Bartoń

References


See Also

- \(model.avg\) for model averaging.
Automated model selection using parallel computation

Description
Parallelized version of dredge.

Usage
pdredge(global.model, cluster = NA, beta = FALSE, evaluate = TRUE, rank = "AICc",
fixed = NULL, m.max = NA, m.min = 0, subset, marg.ex = NULL, trace = FALSE,
varying, extra, ct.args = NULL, check = FALSE, ...)

Arguments
- global.model, beta, evaluate, rank
  see dredge.
- fixed, m.max, m.min, subset, marg.ex, varying, extra, ct.args, ...
  see dredge.
- trace
displays the generated calls, but may not work as expected since the models are
evaluated in batches rather than one by one.
- cluster
either a valid cluster object, or NA for a single threaded execution.
- check
either integer or logical value controlling how much checking for existence and
correctness of dependencies is done on the cluster nodes. See ‘Details’.

Details
All the dependencies for fitting the global.model, including the data and any objects the modelling
function will use must be exported into the cluster worker nodes (e.g. via clusterExport). The re-
quired packages must be also loaded thereinto (e.g. via clusterEvalQ(..., library(package)),
before the cluster is used by pdredge.

If check is TRUE or positive, pdredge tries to check whether all the variables and functions used in
the call to global.model are present in the cluster nodes’ .GlobalEnv before proceeding further.
This causes false errors if some arguments of the model call (other than subset) would be evaluated
in data environment. In that case using check = FALSE (the default) is desirable.

If check is TRUE or greater than one, pdredge will compare the global.model updated at the cluster
nodes with the one given as argument.

Value
See dredge.

Author(s)
Kamil Bartoń

See Also
makeCluster and other cluster related functions in packages parallel or snow.
Examples

# One of these packages is required:
## Not run: require(parallel) || require(snow)

# From example(Beetle)
data(Beetle)

Beetle100 <- Beetle[sample(nrow(Beetle), 100, replace = TRUE),]

fm1 <- glm(Prop ~ dose + I(dose^2) + log(dose) + I(log(dose)^2),
            data = Beetle100, family = binomial)

msubset <- expression(xor(dose, log(dose)) & (dose | !I(dose^2))
                        & (log(dose) | !I(log(dose)^2))
)

varying.link <- list(family = alist(logit = binomial("logit"),
                                probit = binomial("probit"),
                                cloglog = binomial("cloglog" ))

# Set up the cluster
clusterType <- if(length(find.package("snow", quiet = TRUE))) "SOCK" else "PSOCK"
clust <- try(makeCluster(getOption("cl.cores", 2), type = clusterType))
clusterExport(clust, "Beetle100")

# noticeable gain only when data has about 3000 rows (Windows 2-core machine)
print(system.time(dredge(fm1, subset = msubset, varying = varying.link)))
print(system.time(pdredge(fm1, cluster = FALSE, subset = msubset,
                          varying = varying.link)))
print(system.time(pdd <- pdredge(fm1, cluster = clust, subset = msubset,
                                 varying = varying.link)))
print(pdd)

## Not run:
# Time consuming example with unmarked model, based on example(pcount).
# Having enough patience you can run this with demo(pdredge.pcount).
library(unmarked)
data(mallard)
mallardUMF <- unmarkedFramePCount(mallard.y, siteCovs = mallard.site,
                                   obsCovs = mallard.obs)
(ufm.mallard <- pcount(~ ivel + date + I(date^2) ~ length + elev + forest,
                        mallardUMF, K = 30))
clusterEvalQ(clust, library(unmarked))
clusterExport(clust, "mallardUMF")

# stats4 is needed for AIC to work with unmarkedFit objects but is not
# loaded automatically with unmarked.
require(stats4)
invisible(clusterCall(clust, "library", "stats4", character.only = TRUE))

#system.time(print(pdd1 <- pdredge(ufm.mallard, 
# subset = p(date) | !p(I(date^2)), rank = AIC)))

system.time(print(pdd2 <- pdredge(ufm.mallard, clust,
predict.averaging

subset = p(date) | !p(I(date^2)), rank = AIC, extra = "adjR^2"))

# best models and null model
subset(pdd2, delta < 2 | df == min(df))

# Compare with the model selection table from unmarked
# the statistics should be identical:
models <- get.models(pdd2, delta < 2 | df == min(df), cluster = clust)

modSel(fitList(fits = structure(models, names = model.names(models,
labes = getAllTerms(ufm.mallard))), nullmod = "(Null)")

## End(Not run)
stopCluster(clust)

predict.averaging Predict method for the averaged model

Description
Model-averaged predictions with optional standard errors.

Usage
## S3 method for class averaging
predict(object, newdata = NULL, se.fit = FALSE,
interval = NULL, type = NA, backtransform = FALSE, full = TRUE, ...)

Arguments
object An object returned by model.avg.
newdata An optional data frame in which to look for variables with which to predict. If omitted, the fitted values are used.
se.fit logical, indicates if standard errors should be returned. This has any effect only if the predict methods for each of the component models support it.
interval Currently not used.
type The type of predictions to return (see documentation for predict appropriate for the class of used component models). If omitted, the default type is used. See 'Details'.
backtransform If TRUE, the averaged predictions are back-transformed from link scale to response scale. This makes sense provided that all component models use the same family, and the prediction from each of the component models is calculated on the link scale (as specified by type. For glm, use type = "link"). See 'Details'.
full If TRUE, the full model averaged coefficients are used (only if se.fit = FALSE and the component objects are a result of lm).
... Arguments to be passed to respective predict method (e.g. level for lme model).
Predicting is possible only with averaging objects with "modelList" attribute, i.e. those created via model.avg from a model list, or from model.selection object with argument fit = TRUE (note this will recreate the model objects which may be time consuming, see model.avg).

If all the component models are ordinary linear models, the prediction can be made either with the full averaged coefficients (the argument full = TRUE this is the default) or subset-averaged coefficients. Otherwise the prediction is obtained by calling predict on each component model and weighted averaging the results, which corresponds to the assumption that all predictors are present in all models, but those not estimated are equal zero. See 'Note' in model.avg. Predictions from component models with standard errors are passed to par.avg and averaged in the same way as the coefficients.

Predictions on the response scale from generalized models can be calculated by averaging predictions of each model on the link scale, followed by inverse transformation (this is achieved with type = "link" and backtransform = TRUE). This is only possible if all component models use the same family and link function. Alternatively, predictions from each model on response scale may be averaged (with type = "response" and backtransform = FALSE). Note that this leads to results differing from those calculated with the former method. See also predict.glm.

Value

If se.fit = FALSE, a vector of predictions, otherwise a list with components: fit containing the predictions, and se.fit with the estimated standard errors.

Note

This method relies on availability of the predict methods for the component model classes (except when all component models are of class lm).

The package MuMIn includes predict methods for lme, gls and lmer (lme4), all of which can calculate standard errors of the predictions (with se.fit = TRUE). The former two enhance the original predict methods from package nlme, and with se.fit = FALSE they return identical result. MuMIn’s versions are always used in averaged model predictions (so it is possible to predict with standard errors), but from within global environment they will be found only if MuMIn is before nlme on the search list (or directly extracted from namespace as MuMIn:::predict.lme).

predict method for mer models currently can only calculate values on the outermost level (equivalent to level = 0 in predict.lme).

Author(s)

Kamil Bartoń

See Also

model.avg and par.avg for details of model-averaged parameter calculation.

Examples

if(require(graphics)) {

  # Example from Burnham and Anderson (2002), page 100:
  data(Cement)
**QAIC**

**Quasi AIC or AICc**

**Description**

Calculate a modification of Akaike’s Information Criterion for overdispersed count data (or its version corrected for small sample, “quasi-AICc”), for one or several fitted model objects.
QAIC

Usage

QAIC(object, ..., chat, k = 2)
QAICc(object, ..., chat, k = 2)

Arguments

object: a fitted model object.
...
optionally, more fitted model objects.
chat: \( \hat{c} \), the variance inflation factor.
k: the ‘penalty’ per parameter.

Value

If only one object is provided, returns a numeric value with the corresponding QAIC or QAICc; otherwise returns a data.frame with rows corresponding to the objects.

Note

\( \hat{c} \) is the dispersion parameter estimated from the global model, and can be calculated by dividing model’s deviance by the number of residual degrees of freedom.

In calculation of QAIC, the number of model parameters is increased by 1 to account for estimating the overdispersion parameter. Without overdispersion, \( \hat{c} = 1 \) and QAIC is equal to AIC.

Note that glm does not compute maximum-likelihood estimates in models within the quasi-family. In case it is justified, and with a proper caution, a workaround could be used by ‘borrowing’ the aic element from the corresponding ‘non-quasi’ family (see ‘Example’).

Author(s)

Kamil Bartoń

See Also

AICc, quasi family used for models with over-dispersion

Examples

# Based on "example(predict.glm)", with one number changed to create # overdispersion
budworm <- data.frame(
  ldose = rep(0:5, 2), sex = factor(rep(c("M", "F"), c(6, 6))),
  numdead = c(10, 4, 9, 12, 18, 20, 0, 2, 6, 10, 12, 16))
budworm$SF = cbind(numdead = budworm$numdead,
  numalive = 20 - budworm$numdead)
budworm.lg <- glm(SF ~ sex*ldose, data = budworm, family = binomial)
(chat <- deviance(budworm.lg) / df.residual(budworm.lg))
dredge(budworm.lg, rank = "QAIC", chat = chat)
dredge(budworm.lg, rank = "AIC")

# Not run:
# A hacked constructor for quasibinomial family object, that allows for # ML estimation
QIC

x.quasibinomial <- function(...) {
  res <- quasibinomial(...)
  res$aic <- binomial(...)$aic
  res
}
QAIC(update(budworm.lg, family = x.quasibinomial), chat = chat)

## End(Not run)

QIC

QIC and quasi-Likelihood for GEE

Description

Calculate quasi-likelihood under the independence model criterion (QIC) for Generalized Estimating Equations.

Usage

QIC(object, ..., typeR = FALSE)
QICu(object, ..., typeR = FALSE)
quasiLik(object, ...)

Arguments

object a fitted model object of class gee, geepack or yags.
... for QIC and QICu, optionally more fitted model objects.
typeR logical, whether to calculate QIC(R), QIC(R) is based on quasi-likelihood of a working correlation $R$ model. Defaults to FALSE, and QIC(I) based on independence model is returned.

Value

If just one object is provided, returns a numeric value with the corresponding QIC; if more than one object are provided, returns a data.frame with rows corresponding to the objects and one column representing QIC or QICu.

Note

This implementation is based partly on (revised) code from packages yags (R-Forge) and ape. The functions are still in experimental stage and should be used with caution.

Author(s)

Kamil Bartoń

References


See Also

Methods exist for gee (package gee), geeglm (geepack), and yags (yags on R-Forge). yags and compar.gee from package ape both provide QIC values.

Examples

```r
if(require(geepack)) {
  data(ohio)

  fm1 <- geeglm(resp ~ age * smoke, id = id, data = ohio,
                 family = binomial, corstr = "exchangeable", scale.fix = TRUE)
  fm2 <- update(fm1, corstr = "ar1")
  fm3 <- update(fm1, corstr = "unstructured")

  model.sel(fm1, fm2, fm3, rank = "QIC")

  ## Not run:
  # same result:
  # same result:
  dredge(fm1, m.min = 3, rank = "QIC", varying = list(
    corstr = list("exchangeable", "unstructured", "ar1"))
  )

  ## End(Not run)
}
```

---

### r.squaredGLMM

Pseudo-R-squared for Generalized Mixed-Effect models

**Description**

Calculate a conditional and marginal coefficient of determination for Generalized mixed-effect models ($R^2_{GLMM}$).

**Usage**

```r
r.squaredGLMM(x)
```

**Arguments**

- `x` a fitted linear model object.

**Details**

For mixed-effects models, $R^2$ can be categorized into two types. **Marginal** $R^2_{GLMM}$ represents the variance explained by fixed factors, and is defined as:

$$R^2_{GLMM}^{(m)} = \frac{\hat{\sigma}_j^2}{\hat{\sigma}_j^2 + \sum_{l=1}^{L} \hat{\sigma}_l^2 + \sigma_d^2}$$

**Conditional** $R^2_{GLMM}$ is interpreted as variance explained by both fixed and random factors (i.e. the entire model), and is calculated according to the equation:
$$R^2_{\text{GLMM}(c)} = \frac{\sigma_f^2 + \sum_{l=1}^{u} \sigma_l^2}{\sigma_f^2 + \sum_{l=1}^{u} \sigma_l^2 + \sigma_c^2 + \sigma_d^2}$$

where $\sigma_f^2$ is the variance of the fixed effect components, and $\sum \sigma_l^2$ is the sum of all $u$ variance components (group, individual, etc.), $\sigma_c^2$ is the variance due to additive dispersion and $\sigma_d^2$ is the distribution-specific variance.

Value

r.squaredGLMM returns a numeric vector with two values for marginal and conditional $R^2_{\text{GLMM}}$.

Note

$R^2_{\text{GLMM}}$ can be calculated also for fixed-effect models. In the simplest case of OLS it reduces to $\text{var(fitted) / (var(fitted) + deviance / 2)}$. Unlike likelihood-ratio based $R^2$ for OLS, value of this statistic differs from that of the classical $R^2$.

Currently methods exist for classes: mer(Mod), lme, glmmlr() and (g)lm.

See note in r.squaredLR help page for comment on using $R^2$ in model selection.

This function is still in experimental stage and should be used with caution.

Author(s)

This implementation is based on R code from ‘Supporting Information’ for Nakagawa & Schielzeth (2012), and its extension by Paul Johnson.

References


Johnson, P. (in press). Generalization of Nakagawa & Schielzeth’s $R^2_{\text{GLMM}}$ to all random effects specifications. Methods in Ecology and Evolution

See Also

summary.lm, r.squaredLR

Examples

data(Orthodont, package = "nlme")
fm1 <- lme(distance ~ Sex * age, ~ 1 | Subject, data = Orthodont)
r.squaredGLMM(fm1)
r.squaredLR(fm1)
r.squaredLR(fm1, null.RE = TRUE)
r.squaredLR

Likelihood-ratio based pseudo-R-squared

Description

Calculate a coefficient of determination based on the likelihood-ratio test ($R^2_{LR}$).

Usage

```r
r.squaredLR(x, null = NULL, null.RE = FALSE)
null.fit(x, evaluate = FALSE, RE.keep = FALSE, envir = NULL)
```

Arguments

- `x`: a fitted model object.
- `null`: a fitted null model. If not provided, `null.fit` will be used to construct it. `null.fit`'s capabilities are limited to only a few model classes, for others the `null` model has to be specified manually.
- `null.RE`: logical, should the null model contain random factors? Only used if no `null` model is given, otherwise omitted, with a warning.
- `evaluate`: if `TRUE` evaluate the fitted model object else return the call.
- `RE.keep`: if `TRUE`, the random effects of the original model are included.
- `envir`: the environment in which the `null` model is to be evaluated, defaults to the environment of the original model’s formula.

Details

This statistic is is one of the several proposed pseudo-$R^2$’s for nonlinear regression models. It is based on an improvement from `null` (intercept only) model to the fitted model, and calculated as

$$R^2_{LR} = 1 - \exp\left(-\frac{2}{n}(\log \text{Lik}(x) - \log \text{Lik}(0))\right)$$

where logLik(x) and logLik(0) are the log-likelihoods of the fitted and the null model respectively. ML estimates are used for this purpose in when models have been fitted by REstricted ML (by calling `logLik` with argument REML = FALSE). Note that the `null` model can include the random factors of the original model, in which case the statistic represents the ‘variance explained’ by fixed effects.

For OLS models the value is consistent with classical $R^2$. In some cases (e.g. in logistic regression), the maximum $R^2_{LR}$ is less than one. The modification proposed by Nagelkerke (1991) adjusts the $R^2_{LR}$ to achieve 1 at its maximum: $\bar{R}^2 = R^2_{LR} / \max (R^2_{LR})$ where $\max (R^2_{LR}) = 1 - \exp\left(\frac{2}{n} \log \text{Lik}(0)\right)$.

`null.fit` tries to guess the `null` model call, given the provided fitted model object. This would be usually a glm. The function will give an error for an unrecognized class.

Value

`r.squaredLR` returns a value of $R^2_{LR}$, and the attribute "adj.r.squared" gives the Nagelkerke’s modified statistic. Note that this is not the same as nor equivalent to the classical ‘adjusted R squared’.

`null.fit` returns the fitted `null` model object (if `evaluate = TRUE`) or an unevaluated call to fit a `null` model.
Note

$R^2$ is a useful goodness-of-fit measure as it has the interpretation of the proportion of the variance ‘explained’, but it performs poorly in model selection, and is not suitable for use in the same way as the information criterions.

References


See Also

`summary.lm`, `r.squaredGLMM`

subset.model.selection

*Subsetting model selection table*

Description

Return subsets of a model selection table returned by `dredge` or `model.sel`.

Usage

```r
## S3 method for class model.selection
subset(x, subset, select, recalc.weights = TRUE,
  recalc.delta = FALSE, ...)
## S3 method for class model.selection
x[i, j, recalc.weights = TRUE, recalc.delta = FALSE, ...]
```

Arguments

- `x` a `model.selection` object to be subsetted.
- `subset`, `select` logical expressions indicating columns and rows to keep. See `subset`.
- `i, j` indices specifying elements to extract.
- `recalc.weights` logical value specifying whether Akaike weights should be normalized across the new set of models to sum to one.
- `recalc.delta` logical value specifying whether $\Delta IC$ should be calculated for the new set of models (not done by default).
- `...` further arguments passed to `[.data.frame`.

Value

A `model.selection` object containing only the selected models (rows). When columns are selected (arguments `select` or `j` are provided), a plain `data.frame` is returned.
Note

Unlike the method for `data.frame`, extracting with only one index (i.e. `x[i]`) will select rows rather than columns.

To select rows according to presence or absence of the variables (rather than their value), a pseudo-function `has` may be used, e.g. `subset(x, has(a, !b))` will select rows with `a` and without `b` (this is equivalent to `!is.na(a) & is.na(b)`). `has` can take any number of arguments. Importantly, the `has()` notation cannot be used in the `subset` argument for `dredge`, where the variable names should be given directly, with the same effect.

To select rows where one variable can be present conditional on the presence of other variable(s), the function `dc` (dependency chain) can be used. `dc` takes any number of variables as arguments, and allows a variable to be included only if all the preceding arguments are also included (e.g. `subset = dc(a, b, c)` allows for models of form `a`, `a+b` and `a+b+c` but not `b, c, b+c` or `a+c`).

Author(s)

Kamil Bartoń

See Also

`dredge`, `subset` and `.[data.frame]` for subsetting and extracting from `data.frame`.

Examples

data(Cement)
fm1 <- lm(formula = y ~ X1 + X2 + X3 + X4, data = Cement)

# generate models where each variable is included only if the previous # are included too, e.g. X2 only if X1 is there, and X3 only if X2 and X1 dredge(fm1, subset = dc(X1, X2, X3, X4))
# which is equivalent to
# dredge(fm1, subset = (!X2 | X1) & (!X3 | X2) & (!X4 | X3))

# alternatively, generate "all possible" combinations
ms0 <- dredge(fm1)
# ...and afterwards select the subset of models
subset(ms0, dc(X1, X2, X3, X4))
# which is equivalent to
# subset(ms0, (has(!X2) | has(X1)) & (has(!X3) | has(X2)) & (has(!X4) | has(X3)))

# Different ways of finding a confidence set of models:
# delta(AIC) cutoff
subset(ms0, delta <= 4, recalc.weights = FALSE)
# cumulative sum of Akaike weights
subset(ms0, cumsum(weight) <= .95, recalc.weights = FALSE)
# relative likelihood
subset(ms0, (weight / weight[1]) > (1/8), recalc.weights = FALSE)
**updateable**

Make a function return updateable result

**Description**

Creates a function wrapper that stores a call in the values returned by its argument FUN.

**Usage**

```r
updateable(FUN)
updateable2(FUN, Class)
```

# updateable wrapper for mgcv::gamm and gamm4::gamm4
tupGamm(formula, random = NULL, ..., lme4 = inherits(random, "formula"))

**Arguments**

- **FUN**
  function to be modified, found via `match.fun`.
- **Class**
  optional character vector naming class(es) to be set onto the result of `FUN` (not possible with formal S4 objects).
- **formula, random, ...**
  arguments to be passed to `gamm` or `gamm4`
- **lme4**
  if TRUE, `gamm4` is called, `gamm` otherwise.

**Details**

Most model fitting functions in R return an object that can be updated or re-fitted via `update`. This is thanks to the call stored in the object, which can be used (possibly modified) later on. It is also utilised by dredge to generate sub-models.

Some functions (such as `gamm` or `MCMCglmm`) do not provide their result with the call element. In this case `updateable` can be used on that function to add it. The resulting wrapper should be used in exactly the same way as the original function.

**Value**

A function with the same arguments as `FUN`, wrapping a call to `FUN` and adding an element named `call` to its result if possible, or an attribute "call" (if the returned value is atomic or a formal S4 object).

**Note**

`uGamm` sets also an appropriate class onto the result ("gamm4" and/or "gamm"), which is needed for some generics defined in MuMIn to work (note that unlike the functions created by `updateable` it has no formal arguments of the original function). As of version 1.9.2, MuMIn::gamm is no longer available.

**MuMIn** replaces the default method for `getCall` (defined originally in package stats), with a function that can extract the call also when it is an attribute (rather than an element of the object).

**Author(s)**

Kamil Bartoń
See Also

update, getCall, getElement, attributes
gamm, gamm4

Examples

# Simple example with cor.test:
# From example(cor.test)
x <- c(44.4, 45.9, 41.9, 53.3, 44.7, 44.1, 5/zero.noslash.7, 45.2, 6/zero.noslash.1)
y <- c( 2.6,  3.1,  2.5,  5.0,  3.6,  4.0,  5.2,  2.8,  3.8)

ct1 <- cor.test(x, y, method = "kendall", alternative = "greater")
uCor.test <- updateable(cor.test)
ct2 <- uCor.test(x, y, method = "kendall", alternative = "greater")

getCall(ct1) # --> NULL
getCall(ct2)

#update(ct1, method = "pearson") --> Error
update(ct2, method = "pearson")
update(ct2, alternative = "two.sided")

## predefined wrapper for gamm:
if(require(mgcv)) {
  set.seed(0)
dat <- gamSim(6, n = 100, scale = 5, dist = "normal")

fmm1 <- uGamm(y ~s(x/zero.noslash)+ s(x3) + s(x2), family = gaussian, data = dat,
               random = list(fac = ~1))

getCall(fmm1)
class(fmm1)
}

## Not run:
library(caper)
data(shorebird)
shorebird <- comparative.data(shorebird.tree, shorebird.data, Species)

fm1 <- crunch(Egg.Mass ~ F.Mass * M.Mass, data = shorebird)
uCrunch <- updateable(crunch)
fm2 <- uCrunch(Egg.Mass ~ F.Mass * M.Mass, data = shorebird)

getCall(fm1)
getCall(fm2)
update(fm2) # Error with fm1
Weights

dredge(fm2)

## End(Not run)

<table>
<thead>
<tr>
<th>Weights</th>
<th>Akaike weights</th>
</tr>
</thead>
</table>

**Description**

Calculate or extract normalized model likelihoods (‘Akaike weights’).

**Usage**

`Weights(x)`

**Arguments**

- `x` a numeric vector of information criterion values such as AIC, or objects returned by functions like `AIC`. There are also methods for extracting Akaike weights from a `model.selection` or `averaging` objects.

**Value**

a numeric vector of normalized likelihoods.

**Author(s)**

Kamil Bartoń

**See Also**

- `importance`
- `weights`, which extracts fitting weights from model objects.

**Examples**

data(Beetle)

```r
fml <- glm(Prop ~ dose, data=Beetle, family=binomial)
fml2 <- update(fml, . ~ . + I(dose^2))
fml3 <- update(fml, . ~ log(dose))
fml4 <- update(fml3, . ~ . + I(log(dose)^2))

round(Weights(AICc(fml, fml2, fml3, fml4)), 3)
```
Index

*Topic **datasets**
  Beetle, 5
  Cement, 7

*Topic **manip**
  Formula manipulation, 12
  model.sel, 16
  merge.model.selection, 17
  model.utilties, 18
  subset.model.selection, 37

*Topic **models**
  AICc, 3
  dredge, 8
  get.models, 13
  importance, 14
  Information criteria, 16
  Model utilities, 18
  model.avg, 19
  model.sel, 23
  MuMIn-package, 2
  par.avg, 25
  pdredge, 27
  predict.averaging, 29
  QAIC, 31
  QIC, 33
  r.squaredGLMM, 34
  r.squaredLR, 36
  Weights, 41

*Topic **package**
  MuMIn-models, 24
  MuMIn-package, 2

*Topic **utils**
  updateable, 39
  [.data.frame, 37, 38
  [.model.selection
    (subset.model.selection), 37
  AIC, 2–4, 17
  AICc, 2, 3, 4, 17, 22, 23, 32
  aicc, 4
  aictab, 23
  alixt, 8
  append.model.selection
    (merge.model.selection), 17
  attribute, 39
  attributes, 40

  Beetle, 5, 8
  betas, 11
  beta.weights (Model utilities), 18
  BIC, 2, 17

  CAICF, 2
  CAICF (Information criteria), 16
  Cement, 7
  coef, 19
  coef.glmulti, 22
  coeff (Model utilities), 18
  coefTable, 9, 10, 20
  coefTable (Model utilities), 18
  compar.gee, 34
  confint, 21
  Cp (Information criteria), 16

  dc (dredge), 8
  delete.response, 13
  DIC, 2, 25
  DIC (Information criteria), 16
  dredge, 2, 8, 13–15, 17, 22, 23, 25, 27, 38
  drop.terms, 13

  expand.formula (Formula manipulation), 12
  extractDIC, 17

  formula, 13, 20
  Formula manipulation, 12

  gamm, 40
  gamm-wrapper (updateable), 39
  gamm4, 40
  gee, 25, 34
  geeglm, 25, 34
  get.models, 9, 11, 13, 22
  getallTerms (Model utilities), 18
  getCall, 40
  getElement, 40
  glmulti, 11
  global option, 10

  has (subset.model.selection), 37
INDEX

IC (Information criteria), 16
ICOMP, 2
ICOMP (Information criteria), 16
ICtab, 23
importance, 14, 41
Information criteria, 16

list, 14
list of supported models, 2, 9, 20, 23
lme, 29
Logical Operators, 10
logLik, 21
makeCluster, 14
Mallows’ Cp, 2
Mallows’ Cp (Information criteria), 16
match.fun, 20, 39
merge, 17
merge.model.selection, 17
mod.sel, 15
mod.sel (model.sel), 23
modavg, 22
Model utilities, 18
model.avg, 2, 11, 14, 15, 19, 25, 26, 30
model.names (Model utilities), 18
model.sel, 2, 11, 17, 23, 25
MuMIn (MuMIn-package), 2
MuMIn-gamm (updateable), 39
MuMIn-model-utils (Model utilities), 18
MuMIn-models, 24
MuMIn-package, 2

null.fit (r.squaredLR), 36

par.avg, 20–22, 25, 30
pdredge, 2, 11, 14, 27
pget.models (get.models), 13
predict, 20
predict.averaging, 29
predict.glm, 30
predict.gls, 30
predict.lme, 30
print.averaging (model.avg), 19
print.model.selection (dredge), 8

QAIC, 2, 31
QAICc, 2
QAICc (QAIC), 31
QIC, 2, 17, 25, 33
QICu (QIC), 33
quasi, 32
quasilik (QIC), 33
quote, 8

Quotes, 10
r.squaredGLMM, 34, 37
r.squaredLR, 9, 35, 36
rbind, 17
rbind.model.selection
   (merge.model.selection), 17
reformulate, 13
regsubsets, 11

search list, 30
simplify.formula (Formula manipulation), 12
step, 3
stepAIC, 3
subset, 10, 20, 37, 38
subset.model.selection, 37
summary.glm, 20
summary.lm, 35, 37
summary.lme, 19
tTable (Model utilities), 18
uGamm (updateable), 39
update, 39, 40
updateable, 9, 25, 39
updateable2 (updateable), 39
V (dredge), 8
vcov, 19, 20

Weights, 15, 41
weights, 41