Package ‘MuMIn’

October 16, 2011

Type Package

Title Multi-model inference

Version 1.5.0

Date 2011-10-16

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 (>= 2.12.0)

Imports stats

Suggests stats4, nlme, mgcv (>= 1.7.5), lme4 (>= 0.999375-
16), gamm4,MASS, nnet, spdep, survival, unmarked, glmmML

LazyLoad yes

R topics documented:

MuMIn-package *Multi-model inference*

Description

The package MuMIn contains functions to streamline model selection and perform model averaging based on information criteria (AIC, AICc and alike).

Details

User level functions include:

- dredge performs automated model selection based on subsets of the supplied 'global' model, and optional list of choices for other model characteristics (e.g. different link functions). Model set may be generated either with "all possible" combinations, or tailored according to complex inclusion rules
- model.sel creates a model selection table from handpicked models
- model.avg calculates model averaged parameters
- AICc calculates second-order Akaike information criterion for one or several fitted model objects.

Model selection can be done according to any information criterion, including AIC, AICc, BIC, QAIC, ICOMP or Mallows' Cp.

Author(s)

Kamil Bartoń

References

Burnham, K. P. and Anderson, D. R (2002) *Model selection and multimodel inference: a practical information-theoretic approach*. 2nd ed.

See Also

AIC, step

Examples

```
fm1 <- lm(Fertility ~ . , data = swiss)

dd <- dredge(fm1)
top.models.1 <- get.models(dd, subset = delta < 4)
model.avg(top.models.1) # get averaged coefficients

top.models.2 <- get.models(dd, cumsum(weight) <= .95)
model.avg(top.models.2)

# Mixed models:
# modified example(lme)
data(Orthodont, package="nlme")
require(nlme)
```

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

`AICc` *Second-order Akaike Information Criterion*

Description

Calculates second-order Akaike information criterion for one or several fitted model objects (AIC_c, 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
`...`	optionally more fitted model objects
`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 restricted 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_c; 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_c

Note

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

Author(s)

Kamil Bartoń

References

Burnham, K. P. and Anderson, D. R (2002) *Model selection and multimodel inference: a practical information-theoretic approach*. 2nd ed.

Hurvich, C. M. and Tsai, C.-L. (1989) Regression and time series model selection in small samples, *Biometrika* 76: 297–307.

See Also

Akaike's An Information Criterion: `AIC`

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

Examples

```
#Model-averaging mixed models

library(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
dd <- dredge(fm2, trace=TRUE, rank="AICc", REML=FALSE)

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

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

# Because the models originate from 'dredge(..., rank=AICc, REML=FALSE)',
# the default weights in 'model.avg' are ML based:
model.avg(gm, method = "NA")

# same result
#model.avg(gm, method = "NA", rank="AICc", rank.args = list(REML=FALSE))
# REML based weights
model.avg(gm, method = "NA", rank="AICc", rank.args = list(REML=TRUE))
```

`Beetle` *Flour beetle mortality data*

Description

Mortality of flour beetles (*Tribolium confusu*) due to exposure to gaseous carbon disulfide CS_2, from Bliss (1935)

Usage

```
data(Beetle)
```

Format

`Beetle` is a data frame with 4 variables.

dose The dose of CS_2 in mg/L

n.tested Number of beetles tested

n.killed Number of beetles killed

mortality Observed mortality rate

Source

Bliss C. I. (1935) The calculation of the dosage-mortality curve. *Annals of Applied Biology*, 22: 134–167

References

Burnham, K. P. and Anderson, D. R. (2002) *Model selection and multimodel inference: a practical information-theoretic approach*. 2nd ed.

Examples

```
# The "Logistic regression example"
# from Burnham & Anderson (2002) chapter 4.11

data(Beetle)

# Fit a global model with all the considered variables
globmod <- glm(cbind(n.killed, n.tested- n.killed)~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")
))

(dd12 <- dredge(globmod, subset = msubset, varying = varying.link, rank=AIC))

# Table 4.7 "models justifiable a priori"
(dd3 <- dredge(update(globmod, . ~ dose), m.min = 1, rank=AIC,
varying = varying.link))

mod3 <- get.models(dd3, 1:3)

# Table 4.8. Predicted mortality probability at dose 40.

# helper function to 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.))
}

pred <- sapply(mod3, predict, newdata=list(dose=40), se.fit=TRUE, type="response")
pred <- apply(pred, 1, unlist)[,1:2] # simplify

# build the table
tab <- rbind(pred, par.avg(pred[,"fit"], pred[,"se.fit"], dd3$weight,
revised.var = FALSE)[1:2])
tab <- cbind(
c(dd3$weight, NA),
tab,
matrix(logit.ci(tab[,"fit"], tab[,"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(dd3$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))
matplot(newdata$dose, sapply(mod3, predict, newdata, type="response"),
type="l", xlab=quote(list("Dose of"~ CS[2],(mg/L))),
ylab="Mortality"
)
```

`Cement` *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

Woods H., Steinour H.H., Starke H.R. (1932) Effect of composition of Portland cement on heat evolved during hardening. *Industrial & Engineering Chemistry* 24, 1207-1214

References

Burnham, K. P. and Anderson, D. R (2002) *Model selection and multimodel inference: a practical information-theoretic approach*. 2nd ed.

`dredge` *Evaluate "all possible" models*

Description

Automatically generate models with combinations of the terms in the global model, with optional rules for inclusion.

Usage

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

## S3 method for class 'model.selection'
print(x, abbrev.names = TRUE, ...)
```

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. `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.
`m.max, m.min`	optional, maximum and minimum number of terms in a single model (excluding the intercept), `m.max` defaults to the number of terms in `global.model`.
`subset`	logical expression to put constraints for the set of models. Can contain any of the `global.model` terms (use `getAllTerms(global.model)` to list them). It can have a form of an unevaluated `call`, `expression` object, or a one sided `formula`. Compound model terms (such as 'as-is' expressions within `I` or smooths in `gam`) should be treated as non-syntactic names and enclosed in back-ticks (see Quotes). Mind the spacing, names must match exactly the term names in model's formula. To simply keep certain variables in all models, use of `fixed` is preferred.

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 model formulas are used (i.e. no checking). See 'Details'.

trace — if TRUE, all calls to the fitting function (i.e. updated global.model calls) are printed.

varying — optionally, a named list describing the additional arguments to vary between the generated models. Names are the names of the arguments, and each item provides a list of choices. Complex items 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.

x — a model.selection object, returned by dredge.

abbrev.names — should variable names be abbreviated when printing? (useful with many variables).

... — optional arguments for the rank function. Any can be an expression (of mode call), in which case any x within it will be substituted with a current model.

Details

dregde currently is known to work with lm, glm, rlm, polr, multinom, gam, gls, lme, lmer, coxph, glmmML, sarlm, spautolm, gamm and gamm4 (the last two should be fitted *via* the wrapper MuMIn::gamm).

Models are run one by one by repeated evaluation of the call to global.model with modified formula argument (or fixed in lme). This method, while robust in that it can be applied to a variety of different models is not very efficient and may be considerably time-intensive.

Note that the number of combinations grows exponentially with number of predictor variables (2^N). Because there is potentially a large number of models to evaluate, to avoid memory overflow the fitted model objects are not stored. To get (a subset of) the models, use get.models with the object returned by dredge as an argument.

Handling interactions, dredge respects marginality constraints, so "all possible combinations" do not include models containing interactions without their respective main effects. This behaviour can be altered by marg.ex argument. It can be used to allow for simple nested designs. For example, with global model of form a / (x + z), use marg.ex = "a" and fixed = "a".

rank is found by a call to match.fun and may be 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 dredge.

Function rank must be able to accept model as a first argument and must always return a scalar. Typical choice for rank would be "AIC", "QAIC" or "BIC" (**stats** or **nlme**).

Use of na.action = na.omit (R's default) in global.model should be avoided, as it results with sub-models fitted to different data sets, if there are missing values. In versions >= 0.13.17 a warning is given in such a case.

Value

dredge returns an object of class model.selection, being a data.frame with models' coefficients (or TRUE/FALSE for factors), k, deviance/RSS, R-squared, AIC, AICc, delta and weight. This depends on a type of model. Models are ordered according to the used information criterion (lowest on top), specified by rank.

The attribute "calls" is a list containing the model calls used (arranged in the same order as the models).

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 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

get.models, model.avg.

There is also subset.model.selection method.

Consider the alternatives: glmulti in package **glmulti** and bestglm (**bestglm**), or aictab (**AICcmodavg**) and ICtab (**bbmle**) for "hand-picked" model selection tables.

Examples

```
# Example from Burnham and Anderson (2002), page 100:
data(Cement)
lm1 <- lm(y ~ ., data = Cement)
dd <- dredge(lm1)
subset(dd, delta < 4)

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


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

#or as a 95% confidence set:
top.models <- get.models(dd, cumsum(weight) <= .95)

model.avg(top.models) # get averaged coefficients

#topmost model:
top.models[[1]]

## Not run:
# Examples of using 'subset':
# exclude models containing both X1 and X2
dredge(lm1, subset = !(X1 & X2))
# keep only models containing X3
dredge(lm1, subset = X3)
# the same, but more effective:
dredge(lm1, fixed = "X3")

#Reduce the number of generated models, by including only those with
# up to 2 terms (and intercept)
```

```
dredge(lm1, m.max = 2)

## End(Not run)
```

`gamm-wrapper` *Updatable* `gamm`

Description

Enables updating of the model objects fitted by `gamm` and `gamm4` from packages **mgcv** and **gamm4**.

Usage

```
gamm(formula, random = NULL, ..., lme4 = inherits(random, "formula"))
```

Arguments

`formula, random, ...`
: arguments passed to `gamm` or `gamm4`

`lme4`
: logical, if `TRUE` `gamm4` is used rather than `gamm`. If `TRUE`, the `random` argument must be provided as a `formula`.

Details

This function is just a wrapper for `gamm` and `gamm4`. It allows `update` on the returned object, so also makes possible using it in model selection with `dredge`.

This is only a temporary workaround and it is likely be removed soon.

Value

Depending on the value of the 'lme4' switch, either a `gamm` or `gamm4` fitted model object. The only difference from the original object is an addition of the `call` component.

Note

To assure `gamm` is called *via* this wrapper (in case it is masked by the original `gamm` from **mgcv**), use `MuMIn::gamm`.

Author(s)

Kamil Bartoń

See Also

`gamm` and `gamm4`

`get.models` *Get models*

Description

Gets list of models from a `model.selection` object

Usage

```
get.models(dd, subset = delta <= 4, ...)
```

Arguments

`dd`	object returned by `dredge`
`subset`	subset of models
`...`	additional parameters passed to `update`, for example, in `lme`/`lmer` one may want to use `method = "REML"` while using "ML" for model selection

Value

`list` of models.

Author(s)

Kamil Bartoń

See Also

`dredge`, `model.avg`

Examples

```
# Mixed models:

require(nlme)
fm2 <- lme(distance ~ age + Sex, data = Orthodont,
    random = ~ 1 | Subject, method="ML")
dd2 <- dredge(fm2)

# Get top-most models, but fitted by REML:
(top.models.2 <- get.models(dd2, subset = delta < 4, method = "REML"))
```

importance *Relative variable importance*

Description

Sum of 'Akaike weights' over all models that include the explanatory variable.

Usage

```
importance(x)
```

Arguments

`x`	Either a model list or a "`model.selection`" or "`averaging`" object

Value

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

Author(s)

Kamil Bartoń

See Also

`dredge`, `model.avg`, `mod.sel`

Examples

```
# Generate some models
data(Cement)
lm1 <- lm(y ~ ., data = Cement)
dd <- dredge(lm1)

# Importance can be calculated/extracted from various objects:
importance(dd)
## Not run:
importance(subset(mod.sel(dd), delta <= 4))
importance(model.avg(dd, subset = delta <= 4))
importance(subset(dd, delta <= 4))
importance(get.models(dd, 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(lm1)))

importance(subset(mod.sel(dd, 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(dd, rank=AIC, rank.args=list(k = lognobs),
subset=cumsum(weight) <= .95))
```

Information criteria *Various information criteria*

Description

Calculate Mallows' *Cp* and Bozdogan's ICOMP information criterion

Usage

```
Cp(object, dispersion = NULL)
ICOMP(object, ..., REML = NULL)
```

Arguments

`object`	a fitted model object (in case of ICOMP, a `logLik` method must exist for the object)
`...`	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 σ^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 free parameters directly.

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

Mallows, C. L. (1973) Some Comments on CP. *Technometrics* 15: 661–675

Bozdogan, H. and Haughton, D.M.A. (1998) Information complexity criteria for regression models. *Comp. Stat. & Data Analysis* 28: 51-76

This implementation of ICOMP is a generalized version of `ICOMP.lm` in package **icomp** by J. Ferguson.

See Also

`AIC`, `AICc` and `BIC`

`Miscellaneous` *Helper functions*

Description

`beta.weights` - computes standardized coefficients (beta weights) for a model;

`coeffs` - extracts model coefficients;

`getAllTerms` - extracts independent variable names from a model object;

`tTable` - extracts a table of coefficients, standard errors, and p-values from a model object;

`Weights` - calculates Akaike weights (normalized models likelihoods)

Usage

```
beta.weights(model)
coeffs(model)
getAllTerms(x, ...)
## S3 method for class 'terms'
getAllTerms(x, offset = TRUE, intercept = FALSE, ...)
tTable(model, ...)
Weights(aic, ...)

cbindDataFrameList(x)
rbindDataFrameList(x)
```

Arguments

`model`	a fitted model object
`x`	a fitted model object or a `formula`. for `*bindDataFrameList`, a list of `data.frames`
`offset`	should 'offset' terms be included?
`intercept`	should terms names include the intercept?
`...`	other arguments, often not used
`aic`	a vector of AIC (or other information criterion) values

Details

The functions `coeffs`, `getAllTerms` and `tTable` provide an interface between the model and `model.avg` (as well as `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 is that it returns fixed effects' coefficients for mixed models.

Whimsically, the functions `*bindDataFrameList` are not exported from the name space, use `MuMIn:::cbindDataFrameList` to access them.

Author(s)

Kamil Bartoń

See Also

Vignette 'Extending **MuMIn**'s functionality' has information on how to use model selection functions with other model types.

`model.avg` *Model averaging*

Description

Model averaging based on an information criterion.

Usage

```
model.avg(object, ..., beta = FALSE, method = c("0", "NA"),
method.var = c("NA", "0"), rank = NULL, rank.args = NULL,
revised.var = TRUE)
```

Arguments

- `object` — A fitted model object or a list of such objects. Alternatively an object of class `model.selection`. See 'Details'.
- `...` — more fitted model objects
- `beta` — Logical, should standardized coefficients be returned?
- `method, method.var` — The method of averaging estimators and their variance for parameters that are not common for all the models. Either `"0"` (default) or `"NA"`. See 'Details'.
- `rank` — Optional, 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`.

Details

`model.avg` has been tested to work with the following model classes: `lm`, `glm`; `gam`, `gamm` (**mgcv**); `gamm4` (**gamm4**); `lme`, `gls` (**nlme**); `lmer` (**lme4**); `rlm`, `glm.nb` `polr` (**MASS**); `multinom` (**nnet**); `sarlm`, `spautolm` (**spdep**); `glmmML` (**glmmML**); `coxph` (**survival**); and `unmarkedFit` (**unmarked**).

Other model types are also likely to be supported, in particular those inheriting from one of the above classes. See package vignette 'Extending **MuMIn**'s functionality' for a demonstration on how to provide support for other types of models.

`model.avg` may be used with a list of models, but also directly a `model.selection` object returned by `dredge`. In the latter case, the models from the model selection table are evaluated (with a call to `get.models`) prior to averaging. A warning is given if the `subset` argument is not provided, and the default `delta <= 4` will be used.

With `method = "0"` (default) all predictors are averaged as if they were present in all models in the set, and the value of parameter estimate is taken to be 0 if it is not present in a particular model. If `method = "NA"`, the predictors are averaged only over the models in which they appear, which biases them away from zero. Analogically, the argument `method.var` defines the method of calculating unconditional variance estimators (in this case the default is `"NA"`). Example in `link{par.avg}` shows comparison of the two methods for both the coefficients and variance.

`rank` is found by a call to `match.fun` and typically is specified as a function or a symbol (e.g. a back-quoted 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.

Some generic methods such as `predict.averaging`, `coef`, `formula`, `residuals` and `vcov` are supported.
`logLik` method returns a list of `logLik` objects for the component models.

Value

An object of class `averaging` is a list with components:

List of 13

`summary` a `data.frame` with deviance, AICc, Delta and weights for the component models

`coefficients, se, dfs`
matrices of component models' coefficients, their standard errors, and degrees of freedom

`variable.codes`
names of the variables with numerical codes used in `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 / SE) and significance (assuming a normal error distribution)

`importance` relative importance of the predictor variables, 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

`residuals` model averaged residuals (response minus fitted values)

`call` the matched call.

In addition, the object has following attributes:

`mList` a list of component model objects

`method` a character vector of length 2 (for coefficient and variance), describing how the missing parameters were handled ("NA" or "0").

`beta` logical, were standardized coefficients used?

`revised.var` if TRUE, the standard errors were calculated with the revised formula (See `par.avg`)

Note

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

Burnham, K. P. and Anderson, D. R (2002) *Model selection and multimodel inference: a practical information-theoretic approach*. 2nd ed.

See Also

See `par.avg` for 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)
lm1 <- lm(y ~ ., data = Cement)
dd <- dredge(lm1)
dd

#models with delta.aicc < 4
model.avg(get.models(dd, subset = delta < 4)) # get averaged coefficients

#or as a 95% confidence set:
top.models <- get.models(dd, cumsum(weight) <= .95)

model.avg(top.models) # get averaged coefficients
## Not run:
# The same result
model.avg(dd, cumsum(weight) <= .95)

## End(Not run)

## Not run:
# using BIC (Schwarz's Bayesian criterion) to rank the models
BIC <- function(x) AIC(x, k=log(length(residuals(x))))
mav <- model.avg(top.models, rank=BIC)

## End(Not run)
```

`model.sel` *model selection table*

Description

Builds a model selection table

Usage

```
mod.sel(object, ...)
model.sel(object, ...)

## S3 method for class 'model.selection'
mod.sel(object, rank = NULL, rank.args = NULL, ...)
## Default S3 method:
mod.sel(object, ..., rank = NULL, rank.args = NULL)
```

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.

Value

An object of class `"model.selection"` with columns containing useful information about each model: the coefficients, value of the information criterion used, Delta(IC) and weight.

Author(s)

Kamil Bartoń

See Also

`dredge`

Examples

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

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

```
fm3 <- update(fm1, . ~ . - X2 - X3)

# ranked with AICc by default
mod.sel(fm1, fm2, fm3)


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

`par.avg` *Parameter averaging*

Description

Averages single model coefficient based on provided weights

Usage

```
par.avg(x, se, weight, df = NULL, level = 1 - alpha, alpha = 0.05,
    revised.var = 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'.

Details

Unconditional standard errors are square root of the variance estimator, calculated either according to the original formula in Burnham and Anderson (2002, p. 160, equation 4.7), or a newer, revised formula from Burnham and Anderson (2004, equation 4) (if `revised.var = TRUE`, this is the default). If degrees of freedom are given, the confidence intervals are based on adjusted standard error estimator (Burnham and Anderson 2002, page 164).

Value

`par.avg` returns a vector with named elements:

`Coefficient`	model coefficients
`SE`	unconditional standard error
`Adjusted SE`	adjusted standard error
`Lower CI, Upper CI`	unconditional confidence intervals

Author(s)

Kamil Bartoń

References

Burnham, K. P. and Anderson, D. R (2002) *Model selection and multimodel inference: a practical information-theoretic approach*. 2nd ed.

Burnham, K. P. and Anderson, D. R. (2004). *Multimodel inference - understanding AIC and BIC in model selection*. Sociological Methods & Research 33(2): 261-304.

See Also

`model.avg` for model averaging.

Examples

```
# Compare estimates and variance, model averaged with different methods
# ("NA", "0", see ?model.avg)

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

avgmod <- model.avg(dredge(fm1), cumsum(weight) <= .95, method="NA")
weight <- avgmod$summary$Weight

na2z <- function(x) ifelse(is.na(x), 0, x) #helper function (changes NA to 0)

ret <- sapply(colnames(avgmod$coefficients), function(i) {
x <- avgmod$coefficients[, i]
se <- avgmod$se[, i]
df <- avgmod$dfs[, i]
rbind(
'coef(x)se(x)' = par.avg(x, se, weight, df),
'coef(0)se(x)' = par.avg(na2z(x), se, weight, df),
'coef(0)se(0)' = par.avg(na2z(x), na2z(se), weight, df)
)
}, simplify=FALSE)


if(require(graphics)) {

# Plot the comparison: coefficients +- confidence intervals
y <- 1:3
cap <- .1
op <- par(mfrow=c(2,3), mar=c(3,5,1,3), oma=c(3,0,3,0))
lapply(names(ret), function(i) {
x <- ret[[i]]
plot(y, x[,1], xlim=c(.5,3.5), ylim=range(x[,c(1,4,5)]), pch=20, cex=2,
ylab=i, xlab=NA, axes=FALSE, col=c(1,2,2))
segments(y, x[,4], y, x[,5], lty=2, col=c(1,1,2))
segments(y - cap, x[,4], y + cap, x[,4], col=c(1,1,2))
segments(y - cap, x[,5], y + cap, x[,5], col=c(1,1,2))
box(); axis(2); axis(1, at=y, labels=rownames(x))
abline(h=0, lty=3)
```

```
})
title("Comparison of model averaging methods",
sub="Cement hardening example", outer=TRUE, line=1)
par(op)

}
```

`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, se.fit = NULL, interval = NULL,
    type=c("link", "response"), ...)
```

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`	Predictions on response scale are only possible if all component models use the same `family`. See `predict.glm`
`...`	Arguments to be passed to respective `predict` method (e.g. `level` for `lme` model).

Details

`predict.averaging` supports `method = "NA"` only for linear, fixed effect models. In other cases (e.g. nonlinear or mixed models), prediction is obtained using “brute force”, i.e. by calling `predict` on each component model and weighted averaging the results, which is equivalent to assuming that missing coefficients equal zero (`method = "0"`).

Predictions from generalized models are calculated by averaging predictions of each model on the link scale, followed by inverse transformation.

Besides `predict` and `coef`, other generic methods such as `formula`, `residuals` and `vcov` are supported.
`logLik` method returns a list of `logLik` objects for the component models.

Value

An object of class `averaging` with following elements:

- `summary` — a `data.frame` with deviance, AICc, Delta and weights for the component models.
- `coefficients, variance` — matrices of component models' coefficients and their variances
- `variable.codes` — names of the variables with numerical codes used in the summary
- `avg.model` — the averaged model summary, (`data.frame` containing averaged coefficients, unconditional standard error, adjusted SE, and confidence intervals)
- `importance` — the relative importance of variables
- `beta` — (logical) were standardized coefficients used?
- `term.names` — character vector giving names of all terms in the model
- `residuals` — the residuals (response minus fitted values).
- `x, formula` — the model matrix and formula analogical to those that would be used in a single model.
- `method` — how the missing terms were handled ("NA" or "0").
- `call` — the matched call.

Note

`predict.averaging` relies on availability of the `predict` methods for the component model classes (except for `lm/glm`).

Author(s)

Kamil Bartoń

See Also

`model.avg` See `par.avg` for details of model-averaged parameter calculation.

Examples

```
require(graphics)

# Example from Burnham and Anderson (2002), page 100:
data(Cement)
lm1 <- lm(y ~ ., data = Cement)
dd <- dredge(lm1)
top.models <- get.models(dd, subset=cumsum(weight) <= .95)
avgm <- model.avg(top.models)

# helper function
nseq <- function(x, len=length(x)) seq(min(x, na.rm=TRUE),
    max(x, na.rm=TRUE), length=len)

# New predictors: X1 along the range of original data, other
# variables held constant at their means
newdata <- as.data.frame(lapply(lapply(Cement[1:5], mean), rep, 25))
```

```
newdata$X1 <- nseq(Cement$X1, nrow(newdata))

# Predictions from each of the models in a set:
pred <- sapply(top.models, predict, newdata=newdata)
# Add predictions from the models averaged using two methods:
pred <- cbind(pred,
    averaged.0=predict(avgm, newdata),
    averaged.NA=predict(update(avgm, method="NA"), newdata))

matplot(x=newdata$X1, y=pred, type="l", lwd=c(rep(1,ncol(pred)-2), 2, 2),
    xlab="X1", ylab="y")

legend("topleft",
    legend=c(lapply(top.models, formula),
        paste("Averaged model (method=", c("0", "NA"), ")", sep="")),
    col=1:6, lty=1:5, lwd=c(rep(1,ncol(pred)-2), 2, 2), cex = .75)


## Not run:
# Example with gam models (based on "example(gam)")
require(mgcv)
dat <- gamSim(1, n = 500, dist="poisson", scale=0.1)

gam1 <- gam(y ~ s(x0) + s(x1) + s(x2) +  s(x3) + (x1 + x2 + x3)^2,
    family = poisson, data = dat, method = "REML")

cat(dQuote(getAllTerms(gam1)), "\n")

# include only models with either smooth OR linear term (but not both)
# for each predictor variable:
dd <- dredge(gam1, subset=xor(`s(x1)`, x1) & xor(`s(x2)`, x2) &
    xor(`s(x3)`, x3))
# ...this may take a while.

subset(dd, cumsum(weight) < .95)

top.models <- get.models(dd, cumsum(weight) <= .95)

newdata <- as.data.frame(lapply(lapply(dat, mean), rep, 50))
newdata$x1 <- nseq(dat$x1, nrow(newdata))
pred <- cbind(
    sapply(top.models, predict, newdata=newdata),
    averaged=predict(model.avg(top.models), newdata))

matplot(x=newdata$x1, y=pred, type="l", xlab="x1", ylab="y"
    lwd=c(rep(1, ncol(pred) - 2), 2, 2))

## End(Not run)
```

`QAIC` *Quasi AIC(c))*

Description

Calculates 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.

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 analogous '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:
# Ugly hacked constructor for quasibinomial family object, that allows for
# ML estimation
x.quasibinomial <- function(...) {
    res <- quasibinomial(...)
    res$aic <- binomial(...)$aic
    res
}
QAIC(update(budworm.lg, family = x.quasibinomial), chat=chat)

## End(Not run)
```

---

subset.model.selection

Subsetting model selection table

Description

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

Usage

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

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 specyfying whether Akaike weights should be normalized across the new set of models to sum to one.

`...` 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.

Author(s)

Kamil Bartoń

See Also

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

Index