Package ‘MuMIn’

September 12, 2011

Type Package

Title Multi-model inference

Version 1.3.6

Date 2011-09-12

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 methods, R (>= 2.13.0)

Suggests stats4, lme4, MASS, mgcv, gamm4, nlme (>= 3.1.99), nnet, spdep, glmmML, unmarked

Enhances stats, survival

LazyLoad yes

BuildVignettes false

Repository R-Forge

Repository/R-Forge/Project mumin

Repository/R-Forge/Revision 92

Date/Publication 2011-09-12 18:21:07

R topics documented:

`MuMIn-package` *Multi-model inference*

Description

The package `MuMIn` contains functions for (automated) model selection and model averaging based on information criteria (AIC alike).

Details

User level functions include:

`model.avg` does model averaging.

`get.models` evaluates models from the table returned by dredge.

`dredge` runs models with combinations of terms of the supplied 'global.model'.

`AICc` calculates second-order Akaike information criterion for one or several fitted model objects.

Author(s)

Kamil Bartoń `<kamil.barton@go2.pl>`

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 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 AICc; 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 AICc.

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

Akaike's An Information Criterion: `AIC`

`AICc` in package **AICcmodavg**, `aicc` in package **glmulti**

Examples

```
#Model-averaging mixed models

library(nlme)
data(Orthodont, package = "nlme")

# Fit model by REML
fm2 <- lme(distance ~ Sex*age + age*Sex, 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))
```

`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

Author(s)

Kamil Bartoń

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 restrictions.

Usage

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

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

Arguments

- `global.model` — a fitted 'global' model object. Currently, it can be a `lm`, `glm`, `rlm`, `polr`, `multinom`, `gam`, `gls`, `lme`, `lmer`, `coxph`, `glmmML`, `sarlm` or `spautolm`, but also other types are likely to work (yet this is untested).
- `beta` — logical, should standardized coefficients be returned?
- `eval` — whether to evaluate and rank the models. If `FALSE`, a list of all possible model formulas 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` — optional, maximum number of terms to be included in single model, 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). Complex expressions (e.g smooth functions in `gam` models) 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.
- `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

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 back-quoted 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 such 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`. `QAIC` has examples of using custom rank function.

There is also `subset.model.selection` method.

Consider the alternatives: `glmulti` in package **glmulti** and `bestglm` (**bestglm**), or `aictab` (**AICcmodavg**) and `ICtab` (**bbmle**) for a "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)

#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

#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)
```

`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"))
```

---

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, ...)
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?

`...`	other arguments, 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.

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 using with other model types

---

`mod.sel` *model selection table*

Description

Builds a model selection table

Usage

```
mod.sel(object, ..., rank = AICc, rank.args = NULL)
```

Arguments

`object`	A fitted model object or a list of such objects.
`...`	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 default AICc
mod.sel(fm1, fm2, fm3)

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

---

model.avg *Model averaging*

Description

Model averaging based on an information criterion.

Usage

```
model.avg(object, ..., beta = FALSE, method = c("0", "NA"),
    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	The method of averaging parameter estimators 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. QAIC or BIC, 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 (**mgcv**)
- lme, gls (**nlme**)
- lmer (**lme4**)
- rlm, glm.nb polr (**MASS**)
- multinom (**nnet**)
- sarlm, spautolm (**spdep**)
- glmmML (**glmmML**)
- coxph (**survival**)

model.avg may be used with a list of models, or an 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.

Other model types are also likely to be supported, in particular those inheriting from one of the above classes. See 'Details' section of the 'Miscellaneous' page to see how to provide support for other types of models.

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.

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` 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 the predictor variables: calculated as a sum of the Akaike weights over all of the models in which the parameter of interest appears.
- `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

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 averaged model calculation.

`dredge`, `get.models`. `QAIC` has examples of using custom rank function and prediction with confidence intervals.
`AICc` has examples of averaging models fitted by REML.

`modavg` in package **AICcmodavg**, and `coef.glmulti` in package **glmulti** also perform model averaging.

Examples

```
require(graphics)

# 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)
```

---

par.avg *Parameter averaging*

Description

Averages single model coefficient based on provided weights

Usage

```
par.avg(x, se, weight, df = NULL, 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`	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.

`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"`).

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 “quasi AIC” (or “quasi AICc”) for one or several fitted model objects. This function is provided mainly as an example of custom rank function for use with `model.avg` and `dredge`

Usage

```
QAIC(object, ..., chat)
QAICc(object, ..., chat)
```

Arguments

`object`	a fitted model object.
`...`	optionally more fitted model objects.
`chat`	c - hat

Value

If just one object is provided, returns a numeric value with the corresponding QAIC; if more than one object are provided, returns a data.frame with rows corresponding to the objects.

Note

This implementation of QAIC uses model `deviance` rather than a likelihood itself. While this allows calculation also with models fitted using quasi-likelihood (where `logLik = NA`), the absolute values returned may differ from those obtained with the use of plain log-likelihood, since deviance is sometimes adjusted by a constant, so that the saturated model has deviance zero (see `glm`).

`dredge` will use `QAICc` instead of default `AICc` with `glm` with quasi* family.

Author(s)

Kamil Bartoń

See Also

`AICc`
`quasi` family used for models with over-dispersion.
`AIC` and `BIC` may also be used as a custom rank function in `dredge` and `model.avg`.

'Dealing with quasi- models in R', a vignette in the **bbmle** package.

Examples

```
# Based on "example(predict.glm)"
require(graphics)

budworm <- data.frame(
    ldose = rep(0:5, 2),
    numdead = c(1, 4, 9, 13, 18, 20, 0, 2, 6, 10, 12, 16),
    sex = factor(rep(c("M", "F"), c(6, 6))))
budworm$SF = cbind(
    numdead = budworm$numdead,
    numalive = 20 - budworm$numdead)

budworm.lg <- glm(SF ~ sex*ldose, data = budworm, family = quasibinomial)

dd <- dredge(budworm.lg, rank = "QAIC",
    chat = summary(budworm.lg)$dispersion)
```

```
# Average all models
budworm.avg <- model.avg(get.models(dd, seq(nrow(dd))), method="NA")
#model.avg(mod[[1]], mod[[2]], rank = "QAIC", rank.args = list(chat = 1))

plot(c(1,32), c(0,1), type = "n", xlab = "dose",
     ylab = "prob", log = "x")
text(2^budworm$ldose, budworm$numdead/20, as.character(budworm$sex))
ld <- seq(0, 5, 0.1)

newdata <- data.frame(ldose=ld, sex=factor(rep("M", length(ld)),
    levels=levels(budworm$sex)))

# Predictions from global model / Males
pred.lg <- predict(budworm.lg, newdata, se.fit=TRUE, type="response")
matplot(2^ld, cbind(pred.lg$fit, pred.lg$fit - (2 * pred.lg$se.fit),
    pred.lg$fit + (2 * pred.lg$se.fit)), add=TRUE, type="l", col=1)

# Predictions from averaged model / Males
pred.avg <- predict(budworm.avg, newdata, se.fit=TRUE, type="response")
matplot(2^ld, cbind(pred.avg$fit, pred.avg$fit - (2 * pred.avg$se.fit),
    pred.avg$fit + (2 * pred.avg$se.fit)), add=TRUE, type="l", col=2)

newdata$sex[] <- "F"

# Predictions from global model / Females
pred.lg <- predict(budworm.lg, newdata, se.fit=TRUE, type="response")
matplot(2^ld, cbind(pred.lg$fit, pred.lg$fit - (2 * pred.lg$se.fit),
    pred.lg$fit + (2 * pred.lg$se.fit)), add=TRUE, type="l", col=1)

# Predictions from averaged model / Females
pred.avg <- predict(budworm.avg, newdata, se.fit=TRUE, type="response")
matplot(2^ld, cbind(pred.avg$fit, pred.avg$fit - (2 * pred.avg$se.fit),
    pred.avg$fit + (2 * pred.avg$se.fit)), add=TRUE, type="l", col=2)

legend("bottomright", legend=c("full", "averaged"), title="Model",
       col=1:2, lty=1)
```

`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