Package ‘MuMIn’

June 23, 2010

Type Package

Title Multi-model inference

Version 0.12.4

Date 2010-06-21

Encoding UTF-8

Author Kamil Bartoń

Maintainer Kamil Bartoń `<kamil.barton@go2.pl>`

Description Functions for model selection and model averaging based on information criteria (AICc and alikes).

License GPL

Suggests lme4

LazyLoad yes

R topics documented:

MuMIn-package 2
AICc 3
Cement 4
dredge 4
dRedging.functions 6
get.models 7
model.avg 8
par.avg 10
QAIC 11

Index **13**

MuMIn-package *Multi model inference*

Description

The package MuMIn contains functions for 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)
dd
#list of models with delta.aicc < 4
top.models.1 <- get.models(dd, subset = delta < 4)
model.avg(top.models.1) # get averaged coefficients

#or as a 95% confidence set:
top.models.2 <- get.models(dd, cumsum(weight) <= .95)

model.avg(top.models.2) # get averaged coefficients

# Mixed models:

data(Orthodont, package="nlme")

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

```
require(lme4)
fm3 <- lmer(distance ~ age + Sex + (1 | Subject), data = Orthodont, REML=FALSE)
dd3 <- dredge(fm3)

# Get top-most models, but fitted by REML:
(top.models.3 <- get.models(dd3, subset = delta < 4, REML=TRUE))
# use: method = "REML" for older versions of lme4
```

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)
```

Arguments

`object`	a fitted model object
`...`	optionally more fitted model objects
`k`	the “penalty” per parameter to be used; the default `k` = 2 is the classical `AIC`

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), AICc and the `AIC`.

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`

`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` *data dredging*

Description

runs models with all possible combinations of the explanatory variables in the supplied model.

Usage

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

Arguments

`global.model`	a fitted 'global' model object. Currently, it can be a `lm`, `glm`, `gam`, `lme`, `lmer`, `sarlm` or `spautolm`.
`beta`	logical should standardized coefficients be returned rather than normal ones?
`eval`	whether to evaluate and rank the models. If set to FALSE only a list of all possible model formulas is returned
`rank`	custom rank function (information criterion) to use 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`	maximum number of terms to be included in single model, defaults to the number of terms in `global.model`
`...`	optional arguments for the `rank` function

Details

Models are run one by one by calling `update` with modified `formula` argument. While is robust in, it is not very efficient, so may be time (and memory) consuming.

Handling interactions, `dredge` respects marginality constraints, so "all possible combinations" do not include models containing interactions without their respective main effects.

`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.

Function `rank` must be able to accept model as a first argument and must always return a scalar.

Value

`dredge` returns a `data.frame` of class `model.selection` with models' coefficients, k, deviance/RSS, r-squared, AIC, AICc, etc. This depends on a type of model. Models are ordered according to `AICc` (lowest on top), or by `rank` function if specified. It has also a `formulas` attribute - a list containing all the model formulas.

Note

Use the `lmer` from **lme4** library rather than the old `lmer` from **Matrix** package. Complex expressions (like `log(y)`) cannot be used as the response variable, the response must be specified as simple variable name. They may be used on the right side of the formula, though. Make sure there is no a `na.action` set to `na.omit` in `global.model`. This can result with models fitted to different data sets, if there are NA's present. `dredge` cannot handle nested model designs (formulas such as `y ~ a/b`).

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

get.models, model.avg. QAIC has examples of using custom rank function.

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

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

dRedging.functions *Helper functions for package dRedging*

Description

`beta.weights` - computes standardized coefficients (beta weights) for the model; `coeffs` - gets model coefficients; `getAllTerms` - extracts independent variables' names from a fitted model; `tTable` - returns `summary` table of a fitted model with coefficients, standard errors, and p-values.

Usage

```
beta.weights(model)
coeffs(model)
getAllTerms(x, ...)
tTable(model, ...)
```

Arguments

`model`	a fitted model object
`x`	a fitted model object or a `formula`
`...`	other arguments, not used

Author(s)

Kamil Bartoń

See Also

`dredge`

`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"))
```

`model.avg` *Model averaging*

Description

averages models according to an information criterion

Usage

```
model.avg(m1, ..., beta = FALSE, method = c("0", "NA"), rank = NULL, rank.args =

## S3 method for class 'averaging':
coef(object, ...)

## S3 method for class 'averaging':
predict(object, newdata, se.fit = NULL, interval = NULL,
type=NULL, ...)
```

Arguments

`m1`	A fitted model object or a list of such objects.
`beta`	Logical, should standardized coefficients be returned rather than normal ones?
`method`	If set to "0" (default), terms missing in one model are assumed to be 0's, otherwise they are omitted from the weighted average.
`rank`	Custom rank function (information criterion) to use instead of `AICc`, e.g. `QAIC` or `BIC`, may be omitted if `m1` is a list returned by `dredge`. See 'Details'.
`rank.args`	Optional `list` of arguments for the `rank` function.
`alpha`	Significance level for calculatinq confidence intervals.
`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,interval`	Currently not used.
`type`	Ignored. Only predictions on the link scale are allowed. Warning is given if user tries something else here.
`...`	for `model.avg` - more fitted model objects, for `predict` - arguments to be passed to respective `predict` method

Details

`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.
Function `rank` must be able to accept model as a first argument and must always return a scalar. \
Apart from `predict` and `coef`, other default methods, such as `formula` and `residuals` may be used.

Value

`model.avg` returns a list with elements:

`summary`	Model table with deviance, AICc, Delta and weight.
`coefficients`	the model coefficients
`variance`	variance of coefficients
`avg.model`	averaged model summary (`data.frame` with columns: coef - averaged coefficients, var - unconditional variance estimator, ase - adjusted standard error estimator, lci, uci - unconditional confidence intervals)
`relative.importance`	relative variable importances
`variable.codes`	Variable names with numerical codes used in the summary
`relative.importance`	Relative importance of variables
`weights`	
`beta`	(logical) were standardized coefficients used?
`model`	the model matrix, analogical to one that would be used in a single model.
`residuals`	the residuals (response minus fitted values).

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

`dredge`, `get.models`. `QAIC` has examples of using custom rank function.

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

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

```
## 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)


# Predicted values

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, 25)

# Predictions from each of the models in a set:
pred <- sapply(top.models, predict, newdata=newdata)
# Add predictions from the averaged model:
pred <- cbind(pred, averaged=predict(model.avg(top.models), newdata))

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

legend("topleft", legend=c(lapply(top.models, formula), "Averaged model"),
   col=seq(ncol(pred)), lty=1:5, lwd=c(rep(1,ncol(pred)-1), 2))
```

`par.avg` *Parameter averaging*

Description

averages single parameter based on given weights

Usage

```
par.avg (x, se, npar, weight, alpha = 0.05)
```

Arguments

`x`	vector of parameters
`se`	vector of standard errors
`npar`	vector giving numbers of estimated parameters
`weight`	vector of weights
`alpha`	significance level for calculatinq confidence intervals

Value

par.avg returns a vector with named elements:

`Coefficient`	model coefficients
`Variance`	unconditional variance of coefficients
`Unconditional SE`	
`Lower CI, Upper CI`	relative variable importances

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

`model.avg` for averaging models.

`QAIC` *Quasi AIC*

Description

Calculates "quasi AIC" for one or several fitted model objects. This function is provided just as an example of custom rank function for use with `model.avg` and `dredge`

Usage

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

Arguments

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

Details

`rank` is specified as a function or a symbol (e.g. a backquoted name) or a character string specifying a function.

Function `rank` must be able to accept model as a first argument and must always return a scalar.

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.

Author(s)

Kamil Bartoń

Examples

```
budworm <- data.frame(ldose = rep(0:5, 2), numdead = c(1, 4, 9, 13, 18, 20, 0, 2, 6, 10,
budworm$SF <- cbind(budworm$numdead, 20 - budworm$numdead)
budworm.qlg <- glm(SF ~ sex*ldose, family = quasibinomial, data = budworm)

dd1 <- dredge(budworm.qlg, rank = "QAIC", chat = summary(budworm.qlg)$dispersion)
gm1 <- get.models(dd1, 1:4)

model.avg(gm1)

model.avg(gm1[[1]], gm1[[2]], rank = "QAIC", rank.args = list(chat = 1))
```

Index

∗Topic **datasets**
    Cement, 3
∗Topic **misc**
    dRedging.functions, 6
∗Topic **models**
    AICc, 2
    dredge, 4
    get.models, 6
    model.avg, 7
    MuMIn-package, 1
    par.avg, 9
    QAIC, 10
∗Topic **package**
    MuMIn-package, 1

AIC, *2*, *3*
AICc, 2, *5*

beta.weights
        (*dRedging.functions*), 6

Cement, 3
coef.averaging (*model.avg*), 7
coeffs (*dRedging.functions*), 6

data.frame, *5*, *8*
dredge, 4, *6*, *7*, *9*, *10*
dRedging.functions, 6

formula, *6*

get.models, *5*, 6, *9*
getAllTerms (*dRedging.functions*),
        6

list, *7*

model.avg, *5*, *7*, 7, *10*
MuMIn (*MuMIn-package*), 1
MuMIn-package, 1

par.avg, 9
predict.averaging (*model.avg*), 7
print.averaging (*model.avg*), 7
print.model.selection (*dredge*), 4

QAIC, *5*, *9*, 10

step, *2*
summary, *6*

tTable (*dRedging.functions*), 6

update, *4*