

Package ‘MuMIn’

July 8, 2010

Type Package

Title Multi-model inference

Version 0.12.12

Date 2010-07-08

Encoding UTF-8

Author Kamil Barton

Maintainer Kamil Barton <kamil.barton@go2.pl>

Description 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
get.models	6
miscellaneous	7
model.avg	8
par.avg	11
QAIC	12
Index	14

Description

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

Details

User level functions include:

1. `model.avg` - does model averaging.
2. `get.models` - evaluates models from the table returned by `dredge`.
3. `dredge` - runs models with combinations of terms of the supplied 'global' model.
4. `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

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

dd <- dredge(fml)
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

<code>object</code>	a fitted model object
<code>...</code>	optionally more fitted model objects
<code>k</code>	the “penalty” per parameter to be used; the default <code>k = 2</code> 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

Evaluate "all possible" models

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, subset, ...)
```

Arguments

<code>global.model</code>	a fitted ‘global’ model object. Currently, it can be a <code>lm</code> , <code>glm</code> , <code>gam</code> , <code>lme</code> , <code>lmer</code> , <code>sarlm</code> or <code>spautolm</code> , but also other types are likely to work (untested).
<code>beta</code>	logical should standardized coefficients be returned rather than normal ones?
<code>eval</code>	whether to evaluate and rank the models. If <code>FALSE</code> , a list of all possible model formulas is returned
<code>rank</code>	optional custom rank function (information criterion) to be used instead <code>AICc</code> , e.g. <code>QAIC</code> or <code>BIC</code> , See ‘Details’
<code>fixed</code>	optional, either a single sided formula or a character vector giving names of terms to be included in all models
<code>m.max</code>	optional, maximum number of terms to be included in single model, defaults to the number of terms in <code>global.model</code>
<code>subset</code>	logical expression to put additional constraints for the set of models. Can contain any of the <code>global.model</code> terms. Run <code>getAllTerms(global.model)</code> to list all the terms. Complex expressions (e.g smooth functions in <code>gam</code> models) should be treated as non-syntactic names and enclosed in backticks (see Quotes). Mind the spacing, names must match exactly the term names in model’s formula.
<code>...</code>	optional arguments for the <code>rank</code> function

Details

Models are run one by one by calling `update` with modified `formula` argument. This method, while robust in that it can be applied to a variety of different models, 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 an object of class `model.selection`, being a `data.frame` with models’ coefficients, `k`, deviance/RSS, R-squared, `AIC`, `AICc`, `delta` and `weight`. This depends on a type of model. Models are ordered according to `AICc` (lowest on top), or by `rank` function if specified. The attribute “formulas” is a list containing model formulas.

Note

Make sure there is no `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`) properly.

`subset` and `[` methods return a plain `data.frame` when columns are selected.

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)
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:
# Example of using 'subset':
# for some reason, exclude models with both X1 and X2
dredge(lm1, subset = !X1 | !X2)

# keep only models with X3
dredge(lm1, subset = X3)
# the same, but slightly more effective:
dredge(lm1, fixed = ~ X3)

## 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

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

Value

`list` of models.

Author(s)

Kamil Barton

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 relative likelihoods)

Usage

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

```
cbindDataFrameList(x)
rbindDataFrameList(x)
```

Arguments

<code>model</code>	a fitted model object
<code>x</code>	a fitted model object or a formula . for <code>*bindDataFrameList</code> , a list of <code>data.frames</code>
<code>...</code>	other arguments, not used
<code>aic</code>	a vector of AIC (or other information criterion) values

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 e.g. `MuMIn:::cbindDataFrameList` to access them.

Author(s)

Kamil Barton

See Also

[dredge](#)

model.avg

Model averaging

Description

Model averaging based on an information criterion.

Usage

```
model.avg(m1, ..., beta = FALSE, method = c("0", "NA"), rank = NULL,
rank.args = NULL, alpha = 0.05)

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

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


Arguments

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

`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"`).

Apart from `predict` and `coef`, other default methods, such as `formula` and `residuals` may be used.

Value

Ann object of class `averaging`, which is a list with elements:

<code>summary</code>	Model table with deviance, AICc, Delta and weight.
<code>coefficients</code>	the model coefficients
<code>variance</code>	variance of coefficients
<code>avg.model</code>	averaged model summary (<code>data.frame</code> with columns: <code>coef</code> - averaged coefficients, <code>var</code> - unconditional variance estimator, <code>ase</code> - adjusted standard error estimator, <code>lci</code> , <code>uci</code> - unconditional confidence intervals)
<code>relative.importance</code>	relative variable importances
<code>variable.codes</code>	Variable names with numerical codes used in the summary
<code>relative.importance</code>	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, 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(model.avg(top.models, method="0"), newdata),
averaged.NA=predict(model.avg(top.models, 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 smooth OR linear term (but not both) for each variable:
dd <- dredge(gam1, subset=(!`s(x1)` | !x1) & (!`s(x2)` | !x2) & (!`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", lwd=c(rep(1,ncol(pred)-2), 2, 2),
xlab="x1", ylab="y")

## End(Not run)

```

par.avg

Parameter averaging

Description

Averages single parameter based on provided 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 calculating 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

<code>object</code>	a fitted model object.
<code>...</code>	optionally more fitted model objects.
<code>chat</code>	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 Barton

Examples

```
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$SF <- cbind(budworm$numdead, 20 - budworm$numdead)

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

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

model.avg(gml)

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

Index

*Topic **datasets**

Cement, [3](#)

*Topic **misc**

miscellaneous, [7](#)

*Topic **models**

AICc, [2](#)

dredge, [4](#)

get.models, [6](#)

model.avg, [8](#)

MuMin-package, [1](#)

par.avg, [11](#)

QAIC, [12](#)

*Topic **package**

MuMin-package, [1](#)

*Topic **utilities**

miscellaneous, [7](#)

AIC, [2](#), [3](#)

AICc, [2](#), [5](#)

beta.weights (*miscellaneous*), [7](#)

cbindDataFrameList
(*miscellaneous*), [7](#)

Cement, [3](#)

coef, [7](#)

coef.averaging (*model.avg*), [8](#)

coeffs (*miscellaneous*), [7](#)

data.frame, [5](#), [9](#)

dredge, [4](#), [6](#), [8](#), [9](#), [12](#)

formula, [7](#)

gam, [4](#)

get.models, [5](#), [6](#), [9](#)

getAllTerms (*miscellaneous*), [7](#)

list, [6](#)

miscellaneous, [7](#)

model.avg, [5](#), [6](#), [8](#), [12](#)

MuMin (*MuMin-package*), [1](#)

MuMin-package, [1](#)

par.avg, [11](#)

predict.averaging (*model.avg*), [8](#)

print.averaging (*model.avg*), [8](#)

print.model.selection (*dredge*), [4](#)

QAIC, [5](#), [9](#), [12](#)

Quotes, [4](#)

rbindDataFrameList
(*miscellaneous*), [7](#)

step, [2](#)

tTable (*miscellaneous*), [7](#)

update, [5](#)

Weights (*miscellaneous*), [7](#)