

# Package ‘MuMIn’

November 3, 2011

**Type** Package

**Title** Multi-model inference

**Version** 2.0.0

**Date** 2011-11-03

**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 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

**BuildVignettes** true

## R topics documented:

MuMIn-package	2
AICc	3
Beetle	4
Cement	6
dredge	7
gamm-wrapper	11
get.models	12
importance	13
Information criteria	14
Miscellaneous	15
model.avg	16
model.sel	18
MuMIn-devel	20

par.avg . . . . .	20
predict.averaging . . . . .	22
QAIC . . . . .	24
r.squaredLR . . . . .	25
subset.model.selection . . . . .	27
Weights . . . . .	28
<b>Index</b>	<b>29</b>

---

MuMIn-package	<i>Multi-model inference</i>
---------------	------------------------------

---

**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 with subsets of the supplied ‘global’ model, and optional choices of other model properties (such as different link functions). Model set may be generated either with ‘all possible’ combinations, or tailored according to the conditions specified
- [model.sel](#) creates a model selection table from hand-picked models
- [model.avg](#) calculates model averaged parameters, with standard errors and confidence intervals
- [AICc](#) calculates second-order Akaike information criterion.

Model selection can be done according to any information criterion, such as [AIC](#), [AICc](#), [BIC](#), [QAIC](#), [ICOMP](#) or [Mallows’ Cp](#).

**Author(s)**

Kamil Barton

**References**

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

**See Also**

[AIC](#), [step](#) or [stepAIC](#) for stepwise model selection by AIC.

## Examples

```
data(Cement)

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

ms1 <- dredge(fm1)
confset.d4 <- get.models(ms1, subset = delta < 4)
model.avg(confset.d4)

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

---

AICc	<i>Second-order Akaike Information Criterion</i>
------	--

---

## 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

<code>object</code>	a fitted model object for which there exists a <code>logLik</code> method, or a <code>logLik</code> 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 <a href="#">AIC</a>
<code>REML</code>	optional logical value, passed to the <code>logLik</code> 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 sample size is small in comparison to the number of estimated parameters (Burnham & Anderson 2002 recommend its use when  $n/K < 40$ ).

## Author(s)

Kamil Bartoń

## References

- Burnham, K. P. and Anderson, D. R (2002) *Model selection and multimodel inference: a practical information-theoretic approach*. 2nd ed. New York, Springer-Verlag.
- 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
ms2 <- dredge(fm2, trace = TRUE, rank = "AICc", REML = FALSE)

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

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

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

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

---

Beetle

*Flour beetle mortality data*

---

## Description

Mortality of flour beetles (*Tribolium confusum*) due to exposure to gaseous carbon disulfide CS<sub>2</sub>, from Bliss (1935)

## Usage

```
data(Beetle)
```

**Format**

Beetle is a data frame with 5 elements.

**dose** The dose of CS<sub>2</sub> in mg/L

**n.tested** Number of beetles tested

**n.killed** Number of beetles killed

**Prop** A matrix with two columns named **n.killed** and **n.survived**

**mortality** Observed mortality rate

**Source**

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. New York, Springer-Verlag.

**Examples**

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

library(MuMIn)

data(Beetle)

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

# A logical expression defining the subset of models to use:
# * either log(dose) or dose
# * the quadratic terms can appear only together with linear terms
msubset <- expression(xor(dose, 'log(dose)') & (dose | !'I(dose^2)')
  & ('log(dose)' | !'I(log(dose)^2)'))

# Table 4.6

# Use 'varying' argument to fit models with different link functions
# Note the use of 'alist' rather than 'list' in order to keep the 'family'
# objects unevaluated
varying.link <- list(family = alist(
  logit = binomial("logit"),
  probit = binomial("probit"),
  cloglog = binomial("cloglog")
))

dredge(globmod, subset = msubset, varying = varying.link, rank=AIC)
```

```

# Table 4.7 "models justifiable a priori"
(ms3 <- dredge(update(globmod, . ~ dose), fixed = ~dose, rank = AIC,
  varying = varying.link))

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

# Table 4.8. Predicted mortality probability at dose 40.

# a helper functions 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"], Weights(ms3),
  revised.var = FALSE)[1:2])
tab <- cbind(
  c(Weights(ms3), 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(ms3$family), "model averaged")

print(tab, digits = 3, na.print = "")

# Figure 4.3
newdata <- list(dose = seq(min(Beetle$dose), max(Beetle$dose), length.out = 25))
matplot(newdata$dose, sapply(mod3, predict, newdata, type="response"),
  type = "l", xlab = quote(list("Dose of" ~ CS[2],(mg/L))),
  ylab = "Mortality", col = 2:4, lty = 3, lwd = 1
)

# add model-averaged prediction with CI, using the same method as above
pred <- lapply(mod3, predict, newdata, type = "response", se.fit = TRUE)
pred.y <- sapply(pred, "[", "fit")
pred.se <- sapply(pred, "[", "se.fit")
avpred <- sapply(1:25, function(i) par.avg(pred.y[i, ], pred.se[i, ],
  weight = Weights(ms3), revised.var = FALSE)[1:2])
avci <- matrix(logit.ci(avpred[1, ], avpred[2, ], quantile = 2), ncol = 2)
matplot(newdata$dose, cbind(avpred[1, ], avci), type = "l", add = TRUE, lwd = 1,
  lty = c(1, 2, 2), col = 1)
legend("topleft", NULL, c(as.character(ms3$family), expression('averaged' %+-% CI)),
  lty = c(3, 3, 3, 1), col = c(2:4, 1))

```

**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. New York, Springer-Verlag.

---

dredge

*Automated model selection*


---

**Description**

Generate a set of 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, extra, ...)
```

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

## Arguments

<code>global.model</code>	a fitted 'global' model object. See 'Details' for a list of supported types.
<code>beta</code>	logical, should standardized coefficients be returned?
<code>evaluate</code>	whether to evaluate and rank the models. If FALSE, a list of model calls is returned.
<code>rank</code>	optional custom rank function (information criterion) to be used instead AICc, e.g. QAIC or BIC. 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, m.min</code>	optionally the maximum and minimum number of terms in a single model (excluding the intercept), <code>m.max</code> defaults to the number of terms in <code>global.model</code> .
<code>subset</code>	logical expression describing models to keep in the resulting set. See 'Details'.
<code>marg.ex</code>	a character vector specifying names of variables for which NOT to check for marginality restrictions when generating model formulas. If this argument is set to TRUE, all combinations of terms are used (i.e. no checking).
<code>trace</code>	if TRUE, all calls to the fitting function (i.e. updated <code>global.model</code> calls) are printed before actual fitting takes place.
<code>varying</code>	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 <a href="#">alist</a> , see <a href="#">quote</a> ), otherwise it may produce rather unpleasant effects. See example in <a href="#">Beetle</a> .
<code>extra</code>	optional additional statistics to include in the result, provided as functions, function names or a list of such (best if named or quoted). Similarly as in <code>rank</code> argument, each function must accept fitted object <code>model</code> as an argument and return (object coercible to) a numeric vector. These can be e.g. additional information criteria or goodness-of-fit statistics. The character string " $R^2$ " is treated in a special way, and will add a likelihood-ratio based $R^2$ to the resulting table (this is more efficient than using <code>r.squaredLR</code> directly).
<code>x</code>	a <code>model.selection</code> object, returned by <code>dredge</code> .
<code>abbrev.names</code>	should variable names be abbreviated when printing? (useful with many variables).
<code>...</code>	optional arguments for the rank function. Any can be an expression (of model call), in which case any <code>x</code> within it will be substituted with a current model.

## Details

`dredge` currently is known to work with `lm`, `glm`, `rlm`, `polr`, `multinom`, `gam`, `gls`, `lme`, `lmer`, `coxph`, `survreg`, `glmmML`, `sarlm`, `spautolm`, `gamm` and `gamm4` (the last two should be fitted *via* the wrapper `MuMIn::gamm`), and several classes of models from package **unmarked**.

Models are run one by one by repeated evaluation of modified calls to `global.model` (in a similar fashion to that of `update`). This method, while robust in that it can be applied to a variety of different model objects is not very efficient, and may be 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](#) on the object returned by `dredge`.



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**).

The argument `subset` acts in a similar fashion to that in the function `subset` for `data.frames`: the model terms can be referred to by name as variables in the expression, with the difference that they are always logical (i.e. `TRUE` if a term exists in the model). The expression 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. See ‘Examples’. Compound model terms (such as ‘as-is’ expressions within `I()` or the 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.

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  $\geq 0.13.17$  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 presence/NA for factors), `df` - number of parameters, log-likelihood, the information criterion value, delta-IC and *Akaike weight*. 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).

XXX "global", "global.call", "rank", "call"

## 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 Barton

## See Also

[get.models](#), [model.avg](#).

There are [subset](#) and [plot](#) methods.

Possible alternatives: [glmulti](#) in package **glmulti** and [bestglm](#) (**bestglm**), or [aictab](#) (**AICcmodavg**) and [Ictab](#) (**bbmle**) for "hand-picked" model selection tables.

[regsubsets](#) in package **leaps** also performs all-subsets regression.

## Examples

```
# Example from Burnham and Anderson (2002), page 100:
data(Cement)
fm1 <- lm(y ~ ., data = Cement)
dd <- dredge(fm1)
subset(dd, delta < 4)

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

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

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

model.avg(confset.95p) # get averaged coefficients

#'Best' model
summary(confset.95p[[1]])

## Not run:
# Examples of using 'subset':
# exclude models containing both X1 and X2
dredge(fm1, subset = !(X1 & X2))
# keep only models containing X3
dredge(fm1, subset = ~ X3) # subset as a formula
dredge(fm1, subset = expression(X3)) # subset as expression object
# the same, but more effective:
dredge(fm1, fixed = "X3")

#Reduce the number of generated models, by including only those with
# up to 2 terms (and intercept)
dredge(fm1, m.max = 2)

## End(Not run)

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

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

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

---

gamm-wrapper	<i>Updateable</i> 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

<code>formula</code> , <code>random</code> , ...	arguments passed to <code>gamm</code> or <code>gamm4</code>
<code>lme4</code>	logical, if TRUE <code>gamm4</code> is used rather than <code>gamm</code> . If TRUE, the <code>random</code> argument must be provided as a formula.

## Details

This function is just a wrapper for `gamm` and `gamm4`. The only purpose of it is to add a `call` component, that is not provided by `gamm*` as such. 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** (when **MuMIn** was loaded after **mgcv**), use `MuMIn::gamm`.

## Author(s)

Kamil Bartoń

## See Also

[gamm](#) and [gamm4](#)

get.models

*Get models***Description**

Generates a list of fitted model objects from a `model.selection` table

**Usage**

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

**Arguments**

<code>object</code>	object returned by <a href="#">dredge</a>
<code>subset</code>	subset of models, an expression evaluated within the model selection table, see <a href="#">subset method</a>
<code>...</code>	additional parameters passed to update. 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 fitted model objects

**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")
ms2 <- dredge(fm2)

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

---

importance	<i>Relative variable importance</i>
------------	-------------------------------------

---

**Description**

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

**Usage**

```
importance(x)
```

**Arguments**

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

**Value**

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

**Author(s)**

Kamil Bartoń

**See Also**

[dredge](#), [model.avg](#), [mod.sel](#)

**Examples**

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

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

## End(Not run)

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

# 'nobs' is not used here for backwards compatibility
lognobs <- log(length(resid(fm1)))

importance(subset(mod.sel(ms1, rank = AIC, rank.args = list(k = lognobs)),
cumsum(weight) <= .95))
```

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

---

## Information criteria    *Various information criteria*

---

### Description

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

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

### 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

### 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.

**Usage**

```

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

cbindDataFrameList(x)
rbindDataFrameList(x)
  
```

**Arguments**

<code>model</code>	a fitted model object
<code>x</code>	a fitted model object or a <a href="#">formula</a> . for <code>*bindDataFrameList</code> , a list of data.frames
<code>offset</code>	should 'offset' terms be included?
<code>intercept</code>	should terms names include the intercept?
<code>...</code>	other arguments, often not used

**Details**

The functions `coeffs`, `getAllTerms` and `tTable` provide interface between the fitted model object and `model.avg` (and `dredge`). Custom methods can be written to provide support for additional classes of models. The vignette 'Extending **MuMIn**'s functionality' describes it in more detail.

**Note**

`coeffs`'s value is in most cases identical to that returned by `coef`, the only difference being it returns fixed effects' coefficients for mixed models.

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

**Author(s)**

Kamil Bartoń

---

model.avg	<i>Model averaging</i>
-----------	------------------------

---

## Description

Model averaging based on an information criterion.

## Usage

```
model.avg(object, ..., beta = FALSE, 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 <code>model.selection</code> . See ‘Details’.
...	more fitted model objects
beta	Logical, should standardized coefficients be returned?
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 <code>get.models</code> or a <code>model.selection</code> 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 <a href="#">par.avg</a> .

## Details

`model.avg` has been tested to work with the fitted objects from following modelling functions: `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`, `survreg` (**survival**); and several models within `unmarkedFit` class (**unmarked**). Other classes 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.

`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.

Several standard methods for fitted model objects exist for class averaging, including `summary`, `predict`, `coef`, `confint`, `formula`, `residuals`, `vcov`. The `coef` method accepts argument `full`, if set to `TRUE` the full model-averaged coefficients are returned, rather than subset-averaged ones. `logLik` 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
coef.shrinkage	a vector of full model-averaged coefficients, see 'Note'
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 <i>dfs</i> are available) and z-values (coefficient and 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. formula contains only the averaged coefficients
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
beta	logical, were standardized coefficients used?
revised.var	if TRUE, the standard errors were calculated with the revised formula (See <a href="#">par.avg</a> )

**Note**

The 'subset' (or 'conditional') average only averages over the models where the parameter appears. An alternative, the 'full' average assumes that a variable is included in every model, but in some models the corresponding coefficient is set to zero. Unlike the 'subset average', it does not have a tendency of biasing the value away from zero. It is, however, an unresolved issue how the variance of this estimate should be calculated, therefore the standard errors and confidence interval are returned only for the subset-averaged coefficients (as from version  $\geq 1.5.0$  argument `method` is no longer accepted, use of it throws an error).

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. New York, Springer-Verlag.

**See Also**

See [par.avg](#) for more details of model averaged parameter calculation.

[dredge](#), [get.models](#)

[AICc](#) has examples of averaging models fitted by REML.

[modavg](#) in package **AICcmodavg**, and [coef.glmulti](#) in package **glmulti** also perform model averaging.

**Examples**

```
# Example from Burnham and Anderson (2002), page 100:
data(Cement)
fm1 <- lm(y ~ ., data = Cement)
(ms1 <- dredge(fm1))

#models with delta.aicc < 4
summary(model.avg(get.models(ms1, subset = delta < 4))) # get averaged coefficients

#or as a 95% confidence set:
confset.95p <- get.models(ms1, cumsum(weight) <= .95)

avgmod.95p <- model.avg(confset.95p) # get averaged coefficients
confint(avgmod.95p)

## Not run:
# The same result
model.avg(ms1, 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))))
model.avg(confset.95p, rank=BIC)
# the same result, using AIC directly, with argument k
# 'x' in a quoted 'rank' argument is substituted with a model object
# (in this case it does not make much sense as the number of observations is
# common to all models)
model.avg(confset.95p, rank=AIC, rank.args=alist(k=log(length(residuals(x)))))

## End(Not run)
```

---

model.sel

*model selection table*


---

**Description**

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**

<code>object</code>	A fitted model object, a list of such objects, or a "model.selection" object.
<code>...</code>	More fitted model objects
<code>rank</code>	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 <code>get.models</code> .
<code>rank.args</code>	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))))

```

MuMin-devel

*Under development***Description**

This is a dummy manual for exported functions which are currently under development, just to satisfy R check.

**Usage**

```
makeArgs(obj, termNames, comb, opt, ...)
fixCoefNames(x)
```

**Arguments**

obj, termNames, comb, opt	More undescribed arguments.
x	Even more undescribed arguments.
...	optional arguments

**Details**

None.

**Value**

Various.

**Author(s)**

Kamil Bartoń

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, adjusted = TRUE)
```

**Arguments**

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

**Details**

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

**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	<i>Predict Method for the Averaged Model</i>
-------------------	--

---

## Description

Model-averaged predictions with optional standard errors.

## Usage

```
## S3 method for class 'averaging'
predict(object, newdata = NULL, se.fit = FALSE,
        interval = NULL, type = c("link", "response"), full = TRUE, ...)
```

## Arguments

object	An object returned by <code>model.avg</code>
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 <a href="#">family</a> . See <a href="#">predict.glm</a> .
full	If TRUE, the full model averaged coefficients are used (only if <code>se.fit = FALSE</code> and the component objects are a result of <code>lm</code> ).
...	Arguments to be passed to respective predict method (e.g. <code>level</code> for <a href="#">lme</a> model).

## Details

For an ordinary linear model, the predicted values can be averaged either using the full averaged coefficients (if argument `full = TRUE`, the default), or subset-averaged coefficients. In other cases the prediction is obtained by calling `predict` on each component model and weighted averaging the results, which corresponds to the assumption that all predictors are present in all models, but those not estimated are equal zero. See ‘Note’ in [model.avg](#). Predictions from component models with standard errors are passed to `par.avg` and averaged in the same way as the coefficients.

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

## Value

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

## Note

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

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

ms1 <- dredge(fm1)
confset.95p <- get.models(ms1, subset=cumsum(weight) <= .95)
avgm <- model.avg(confset.95p)

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))

n <- length(confset.95p)

# Predictions from each of the models in a set, and with averaged coefficients
pred <- data.frame(
  model = sapply(confset.95p, predict, newdata = newdata),
  averaged.subset = predict(avgm, newdata),
  averaged.full = predict(avgm, newdata, full = TRUE)
)

opal <- palette(c(topo.colors(n), "black", "red", "orange"))
matplot(newdata$X1, pred, type = "l",
  lwd = c(rep(2,n),3,3), lty = 1,
  xlab = "X1", ylab = "y", col=1:7)

# For comparison, prediction obtained by averaging predictions of the component
# models
pred.se <- predict(avgm, newdata, se.fit = TRUE)
y <- pred.se$fit
ci <- pred.se$se.fit * 2
matplot(newdata$X1, cbind(y, y - ci, y + ci), add = TRUE, type="l",
  lty = 2, col = n + 3, lwd = 3)

legend("topleft",
  legend=c(lapply(confset.95p, formula),
    paste(c("subset", "full"), "averaged"), "averaged predictions CI"),
  lty = 1, lwd = c(rep(2,n),3,3,3), cex = .75, col=1:8)

palette(opal)

```

---

QAIC	<i>Quasi AIC or AIC<sub>c</sub></i>
------	-------------------------------------

---

### Description

Calculates a modification of Akaike's Information Criterion for overdispersed count data (or its version corrected for small sample, "quasi AIC<sub>c</sub>"), for one or several fitted model objects.

### Usage

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

### Arguments

<code>object</code>	a fitted model object.
<code>...</code>	optionally, more fitted model objects.
<code>chat</code>	$\hat{c}$ , the variance inflation factor
<code>k</code>	the 'penalty' per parameter

### Value

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

### Note

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

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

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

### Author(s)

Kamil Bartoń

### See Also

[AICc](#), [quasi](#) family used for models with over-dispersion



## Examples

```
# Based on "example(predict.glm)", with one number changed to create
# overdispersion
budworm <- data.frame(
  ldose = rep(0:5, 2), sex = factor(rep(c("M", "F"), c(6, 6))),
  numdead = c(10, 4, 9, 12, 18, 20, 0, 2, 6, 10, 12, 16))
budworm$SF = cbind(numdead = budworm$numdead,
  numalive = 20 - budworm$numdead)

budworm.lg <- glm(SF ~ sex*ldose, data = budworm, family = binomial)
(chat <- deviance(budworm.lg) / df.residual(budworm.lg))

dredge(budworm.lg, rank = "QAIC", chat = chat)
dredge(budworm.lg, rank = "AIC")

## Not run:
# 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)
```

---

r.squaredLR

*likelihood-ratio based pseudo R-squared*


---

## Description

Calculate likelihood-ratio based coefficient of determination,  $R_{LR}^2$

## Usage

```
r.squaredLR(x, null = null.fit(x, TRUE))

null.fit(x, evaluate = FALSE,
  envir = environment(as.formula(formula(x))))
```

## Arguments

x	a fitted model object
null	a fitted <i>null</i> model, if not provided, a glm with only intercept and appropriate family will be used
evaluate	If TRUE returns the fitted model object else return the call
envir	the environment in which the <i>null</i> model is to be evaluated, by default an environment of the original model's formula.

## Details

This is one of the several pseudo R-squared goodness-of-fit statistics that have been proposed, mainly with application in logistic regression models. It is based on an improvement from *null* (intercept only) model to the fitted model, and calculated as

$$R_{LR}^2 = 1 - \exp\left(-\frac{2}{n}(\logLik(x) - \logLik(0))\right)$$

where  $\logLik(x)$  and  $\logLik(0)$  are the log likelihoods of the fitted and the *null* model respectively.

For ordinary linear models  $R_{LR}^2$  is consistent with classical  $R^2$ . For discrete models, which likelihood is a product of probabilities (such as logistic regression), the maximum  $R_{LR}^2$  is less than one, and equals

$$\max(R_{LR}^2) = 1 - \exp\left(\frac{2}{n}\logLik(0)\right)$$

The modification proposed by Nagelkerke (1991) adjusts the  $R_{LR}^2$  to achieve 1 at its maximum:  $\bar{R}^2 = R_{LR}^2 / \max(R_{LR}^2)$

`null.fit` tries to guess an intercept only (*null*) model (currently only as a `glm`), given the provided fitted model object. It is primarily intended for internal use.

## Value

`r.squaredLR` returns a value of  $R_{LR}^2$ , with additional attribute "adj.r.squared" gives the modification proposed by Nagelkerke (1991). Note that this is not the same as the classical 'adjusted R squared'.

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

## Note

Although this statistic is not related to information theory, it is provided here because it may be often useful to compare it with the value of information criterions.

## References

- Cox, D. R. and Snell, E. J. (1989) *The analysis of binary data*, 2nd ed. London, Chapman and Hall.
- Nagelkerke, N. J. D. (1991) A note on a general definition of the coefficient of determination. *Biometrika* 78: 691-692 (and references for the original formulation therein).

## See Also

[summary.lm](#)

---

`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

<code>x</code>	a <code>model.selection</code> object to be subsetting.
<code>subset, select</code>	logical expressions indicating columns and rows to keep. See <a href="#">subset</a> .
<code>i, j</code>	indices specifying elements to extract.
<code>recalc.weights</code>	logical value specifying whether Akaike weights should be normalized across the new set of models to sum to one.
<code>...</code>	further arguments passed to <a href="#">[.data.frame]</a> .

## 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`.

---

**Weights***Akaike weights*

---

**Description**

Calculate or extract Akaike weights (normalized model likelihoods)

**Usage**

```
Weights(x)
```

**Arguments**

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

**Value**

a numeric vector of normalized likelihoods

**Author(s)**

Kamil Bartoń

**See Also**

[importance](#)

[weights](#), which extracts fitting weights from model objects

**Examples**

```
data(Beetle)

fm1 <- glm(Prop ~ dose, data=Beetle, family=binomial)
fm2 <- update(fm1, . ~ . + I(dose^2))
fm3 <- update(fm1, . ~ log(dose))
fm4 <- update(fm3, . ~ . + I(log(dose)^2))

round(Weights(AICc(fm1, fm2, fm3, fm4)), 3)
```

# Index

- \*Topic **datasets**
  - Beetle, [4](#)
  - Cement, [6](#)
- \*Topic **devel**
  - MuMIn-devel, [20](#)
- \*Topic **manip**
  - Miscellaneous, [15](#)
  - subset.model.selection, [27](#)
- \*Topic **models**
  - AICc, [3](#)
  - dredge, [7](#)
  - gamm-wrapper, [11](#)
  - get.models, [12](#)
  - importance, [13](#)
  - Information criteria, [14](#)
  - Miscellaneous, [15](#)
  - model.avg, [16](#)
  - model.sel, [18](#)
  - MuMIn-package, [2](#)
  - par.avg, [20](#)
  - predict.averaging, [22](#)
  - QAIC, [24](#)
  - r.squaredLR, [25](#)
  - Weights, [28](#)
- \*Topic **package**
  - MuMIn-package, [2](#)
- [.data.frame, [27](#)
- [.model.selection
  - (subset.model.selection), [27](#)
- AIC, [2–4](#), [14](#)
- AICc, [2](#), [3](#), [4](#), [14](#), [18](#), [24](#)
- aicc, [4](#)
- aictab, [9](#)
- alist, [8](#)
- Beetle, [4](#), [8](#)
- bestglm, [9](#)
- beta.weights (Miscellaneous), [15](#)
- BIC, [2](#), [14](#)
- cbindDataFrameList (Miscellaneous), [15](#)
- Cement, [6](#)
- coef, [15](#)
- coef.glmulti, [18](#)
- coeffs (Miscellaneous), [15](#)
- confint, [17](#)
- Cp (Information criteria), [14](#)
- dredge, [2](#), [7](#), [12](#), [13](#), [18](#), [19](#), [27](#)
- family, [22](#)
- fixCoefNames (MuMIn-devel), [20](#)
- formula, [15](#), [16](#)
- gamm, [8](#), [11](#)
- gamm (gamm-wrapper), [11](#)
- gamm-wrapper, [11](#)
- gamm4, [11](#)
- get.models, [8](#), [9](#), [12](#), [18](#)
- getAllTerms (Miscellaneous), [15](#)
- glmulti, [9](#)
- IC (Information criteria), [14](#)
- ICOMP, [2](#)
- ICOMP (Information criteria), [14](#)
- ICtab, [9](#)
- importance, [13](#), [28](#)
- Information criteria, [14](#)
- list, [12](#)
- lme, [22](#)
- logLik, [16](#)
- makeArgs (MuMIn-devel), [20](#)
- Mallows' Cp, [2](#)
- Mallows' Cp (Information criteria), [14](#)
- match.fun, [16](#)
- Miscellaneous, [15](#)
- miscellaneous (Miscellaneous), [15](#)
- mod.sel, [13](#)
- mod.sel (model.sel), [18](#)
- modavg, [18](#)
- model.avg, [2](#), [9](#), [12](#), [13](#), [16](#), [21–23](#)
- model.sel, [2](#), [18](#)
- MuMIn (MuMIn-package), [2](#)
- MuMIn-devel, [20](#)
- MuMIn-package, [2](#)

`null.fit(r.squaredLR)`, 25

`par.avg`, 16–18, 20, 23

`predict`, 16

`predict.averaging`, 22

`predict.glm`, 22

`print.averaging(model.avg)`, 16

`print.model.selection(dredge)`, 7

QAIC, 2, 24

QAICc (QAIC), 24

quasi, 24

quote, 8

Quotes, 9

`r.squaredLR`, 8, 25

`rbindDataFrameList` (Miscellaneous), 15

`regsubsets`, 9

`residuals`, 16

`step`, 2

`stepAIC`, 2

`subset`, 9, 27

`subset method`, 12

`subset.model.selection`, 27

`summary.lm`, 26

`tTable` (Miscellaneous), 15

`vcov`, 16

Weights, 28

`weights`, 28