

# Package ‘MuMIn’

January 23, 2012

**Type** Package

**Title** Multi-model inference

**Version** 1.7.0

**Date** 2012-01-22

**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, coxme

**LazyLoad** yes

**BuildVignettes** true

## R topics documented:

MuMIn-package . . . . .	2
AICc . . . . .	3
Beetle . . . . .	4
Cement . . . . .	7
dredge . . . . .	7
gamm-wrapper . . . . .	11
get.models . . . . .	12
importance . . . . .	13
Information criteria . . . . .	14
Model utilities . . . . .	15
model.avg . . . . .	16
model.sel . . . . .	20
par.avg . . . . .	21

pdredge . . . . .	22
predict.averaging . . . . .	24
QAIC . . . . .	27
r.squaredLR . . . . .	28
subset.model.selection . . . . .	29
Weights . . . . .	31
<b>Index</b>	<b>32</b>

---

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

---

**Description**

The package **MuMIn** contains functions to streamline information-theoretic model selection and carry out model averaging based on the information criteria.

**Details**

The collection of functions includes:

- [dredge](#) performs automated model selection with subsets of the supplied ‘global’ model, and optional choices of other model properties (such as different link functions). The set of models may be generated either with ‘all possible’ combinations, or tailored according to the conditions specified.
- [pdredge](#) does the same, but can parallelize model fitting process using a cluster.
- [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.

For a complete list of functions, use `library(help = "MuMIn")`.

By default,  $AIC_c$  is used to rank the models and to obtain model selection probabilities, though any other information criteria can be utilised. At least the following ones are currently implemented in R: [AIC](#) and [BIC](#) in package **stats**, and [QAIC](#), [QAICc](#), [ICOMP](#), [CAICF](#), and [Mallows’ Cp](#) in **MuMIn**.

Most of R’s common modelling functions are supported, for a full list refer to the help pages for `dredge` and `model.avg`.

**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**

[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

*Second-order Akaike Information Criterion*

## Description

Calculate 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 AIC.
<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:
summary(model.avg(fmList))

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

---

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

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")
))

(msl2 <- dredge(globmod, subset = msubset, varying = varying.link,
  rank = AIC))

# Table 4.7 "models justifiable a priori"
```

```

(ms3 <- subset(ms12, has(dose, !`I(dose^2)`)))
# The same result, but would fit the models again:
# ms3 <- update(ms12, update(globmod, . ~ dose), subset =,
#   fixed = ~dose)

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

# Table 4.8. Predicted mortality probability at dose 40.

# 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))

```

---

Cement

*Cement hardening data*


---

**Description**

Cement hardening data from Woods et al (1939).

**Usage**

```
data(Cement)
```

**Format**

Cement is a data frame with 5 variables. x1-x4 are four predictor variables expressed as a percentage of weight.

**X1** calcium aluminate

**X2** tricalcium silicate

**X3** tetracalcium alumino ferrite

**X4** dicalcium silicate

**y** calories of heat evolved per gram of cement after 180 days of hardening.

**Source**

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

**References**

Burnham, K. P. and Anderson, D. R (2002) *Model selection and multimodel inference: a practical information-theoretic approach*. 2nd ed. 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, warnings = getOption("warn") != -1L,
      ...)
```

## 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 <code>FALSE</code> , a list of model calls 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, 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 <code>TRUE</code> , all combinations of terms are used (i.e. no checking).
<code>trace</code>	if <code>TRUE</code> , 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 <code>family</code> objects) should be either named (uniquely) or quoted (unevaluated, e.g. using <code>alist</code> , see <code>quote</code> ), otherwise it may produce rather unpleasant effects. See example in <code>Beetle</code> .
<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 (an object coercible to) a numeric vector. These can be e.g. additional information criteria or goodness-of-fit statistics. The character strings <code>"R^2"</code> and <code>"adjR^2"</code> are treated in a special way, and will add a likelihood-ratio based $R^2$ and modified- $R^2$ respectively to the result (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>warnings</code>	if <code>TRUE</code> , errors and warnings issued during the model fitting are printed below the table (currently, only with <code>pdredge</code> ). To permanently remove the warnings, set the object’s attribute <code>"warnings"</code> to <code>NULL</code> .
<code>...</code>	optional arguments for the <code>rank</code> function. Any can be an expression (of mode <code>call</code> ), in which case any <code>x</code> within it will be substituted with a current model.

## Details

Fitted model objects that can be used as a `global.model` include ones returned by `lm`, `glm` (package **stats**); `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 `coxme` (**coxme**); `rq` (**quantreg**); and all models from package **unmarked**. `gamm` and `gamm4` should be evaluated *via* the wrapper `MuMin`: `gamm`.



Models are fitted one by one through repeated evaluation of modified calls to the `global.model` (in a similar fashion as with `update`). This method, while robust in that it can be applied to a variety of different model object types 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 can be potentially a large number of models to evaluate, to avoid memory overflow the fitted model objects are not stored in the result. 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 by the value of the information criterion specified by `rank` (lowest on top).

The attribute `"calls"` is a list containing the model calls used (arranged in the same order as the models). Other attributes: `"global"` - the `global.model` object, `"rank"` - the rank function used, `"call"` - the matched call, and `"warnings"` - list of errors and warnings given by the modelling function during the fitting, with model number appended to each. The associated model call can be found with `attr(*, "calls")[[i]]`, where *i* is the model number.

## 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 the 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**

[pdredge](#) is a parallelized version of this function, which may be faster if execution of [dredge](#) takes very long.

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

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

Possible alternatives: [glmulti](#) in package **glmulti** and [bestglm](#) (**bestglm**), or [aictab](#) (**AICc-modavg**) and [lctab](#) (**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)
fml <- lm(y ~ ., data = Cement)
dd <- dredge(fml)
subset(dd, delta < 4)

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

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

#or as a 95% confidence set:
model.avg(dd, subset = cumsum(weight) <= .95) # get averaged coefficients

#'Best' model
summary(get.models(dd, 1))[[1]]

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

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

## End(Not run)

# Add R^2 and F-statistics, use the 'extra' argument
dredge(fml, m.max = 1, extra = c("R^2", F = function(x)
```

```
summary(x)$fstatistic[[1]])

# with summary statistics:
dredge(fml, m.max = 1, extra = list(
  "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:

# there is no BIC in R < 2.13.0, so need to add it:
if(!exists("BIC", mode = "function"))
  BIC <- function(object, ...)
    AIC(object, k = log(length(resid(object))))

dredge(fml, m.max = 1, extra = alist(AIC, BIC, ICOMP, Cp))
```

---

gamm-wrapper

*Updateable* 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**

Generate a list of fitted model objects from a `model.selection` table. `pget.models` can use parallel computation in a cluster to do that.

**Usage**

```
get.models(object, subset, ...)
pget.models(object, cluster = NA, subset, ...)
```

**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 the <a href="#">subset method</a> . If it is a character vector, it is interpreted as names of rows to be selected. By default, <b>all</b> model objects are fitted and returned.
<code>...</code>	additional arguments to update the models. For example, in <code>lme</code> one may want to use <code>method = "REML"</code> while using "ML" for model selection.
<code>cluster</code>	a cluster object. See <a href="#">pdredge</a> for details.

**Value**

[list](#) of fitted model objects.

**Note**

As of version 1.6.3, the default behaviour (if `subset` argument is missing) is to return all the models, rather than a ‘confidence set’ with `delta <= 4`.

**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**

[Weights](#)  
[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 and CAICF information criteria.

## Usage

```

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

```

## Arguments

<code>object</code>	a fitted model object (in case of ICOMP and CAICF, <code>logLik</code> and <code>vcov</code> methods must exist for the object).
<code>...</code>	optionally more fitted model objects.
<code>dispersion</code>	the dispersion parameter. If <code>NULL</code> , it is inferred from object.
<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.

## 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.

CAICF (C is for 'consistent' and F denotes the use of the Fisher information matrix) includes with penalty the natural logarithm of the determinant of the estimated Fisher information matrix.

**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](#) and [BIC](#) in **stats**, [AICc](#)

---

Model utilities	<i>Model utility functions</i>
-----------------	--------------------------------

---

**Description**

These functions extract or calculate various values from provided fitted model objects(s). They are mainly meant for internal use, but may be also useful for end-users.

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

`coeffs` extracts model coefficients;

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

`coefTable` extracts a table of coefficients, standard errors and, when possible, associated p-values from a model object;

`model.names` generates shorthand numeric names for one or several fitted models.

**Usage**

```
beta.weights(model)
```

```
coeffs(model)
```

```
getAllTerms(x, ...)
## S3 method for class 'terms'
getAllTerms(x, offset = TRUE, intercept = FALSE, ...)
```

```
coefTable(model, ...)
## S3 method for class 'lme'
coefTable(model, adjustSigma, ...)
```

```
model.names(object, ..., labels = NULL)
```

**Arguments**

<code>model</code>	a fitted model object.
<code>object</code>	a fitted model object or a list of such objects.
<code>x</code>	a fitted model object or a formula.
<code>offset</code>	should 'offset' terms be included?
<code>intercept</code>	should terms names include the intercept?
<code>labels</code>	optionally, a character vector with names of all the terms, e.g. from a global model. <code>model.names</code> enumerates the model terms in order of their appearance in the list and in the models. So, changing the order of the models would lead to different names. The argument 'labels' can be used to prevent this happening.
<code>...</code>	For <code>model.names</code> , more fitted model objects. For <code>coefTable</code> arguments that are passed to appropriate <code>summary</code> method (e.g. <code>dispersion</code> parameter for <code>glm</code> may be used here). In other functions often not used.
<code>adjustSigma</code>	See <a href="#">summary.lme</a> .

**Details**

The functions `coeffs`, `getAllTerms` and `coefTable` provide interface between the 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.

Use of `tTable` is deprecated in favour of `coefTable`.

**Author(s)**

Kamil Barton

---

`model.avg`

*Model averaging*

---

**Description**

Model averaging based on an information criterion.

**Usage**

```
model.avg(object, ..., revised.var = TRUE)

## Default S3 method:
model.avg(object, ..., beta = FALSE, rank = NULL, rank.args = NULL,
          revised.var = TRUE, dispersion = NULL)
```



```
## S3 method for class 'model.selection'
model.avg(object, subset, fit = FALSE, ..., revised.var = TRUE)
```

## Arguments

<code>object</code>	a fitted model object or a list of such objects, or a <code>model.selection</code> object. See ‘Details’.
<code>...</code>	more fitted model objects.
<code>beta</code>	logical, should standardized coefficients be returned?
<code>rank</code>	optionally, a custom rank function (information criterion) to use instead of AICc, e.g. BIC or QAIC, may be omitted if <code>object</code> is a model list returned by <code>get.models</code> or a <code>model.selection</code> object. See ‘Details’.
<code>rank.args</code>	optional list of arguments for the <code>rank</code> function. If one is an expression, an <code>x</code> within it is substituted with a current model.
<code>revised.var</code>	logical, indicating whether to use revised formula for standard errors. See <a href="#">par.avg</a> .
<code>dispersion</code>	the dispersion parameter for the family used. See <a href="#">summary.glm</a> . This value is used only with the model types which accept it (currently only <code>glm</code> ), and will be silently ignored otherwise.
<code>subset</code>	see <a href="#">subset</a> method for <code>model.selection</code> object.
<code>fit</code>	if TRUE, the component models are fitted using <code>get.models</code> . See ‘Details’.

## Details

`model.avg` has been tested to work with the fitted objects from the 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**); `coxme` (**coxme**); and models within the class `unmarkedFit` (**unmarked**). Other classes are also likely to be supported, in particular those inheriting from one of the above classes.

`model.avg` may be used either with a list of models, or directly with a `model.selection` object (e.g. returned by `dredge`). In the latter case, the models from the model selection table are not evaluated unless the argument `fit` is set to TRUE or some additional arguments are present (such as `rank` or `dispersion`). This results in much faster calculation, but has certain drawbacks, because the fitted component model objects are not stored and some methods (e.g. `predict`, `fitted`, `model.matrix` or `vcov`) would not be available with the returned object. Otherwise, `get.models` is called prior to averaging, and `...` are passed to it.

`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:

<code>summary</code>	a <code>data.frame</code> with log-likelihood, IC, Delta(IC) and Akaike weights for the component models.
<code>coef.shrinkage</code>	a vector of full model-averaged coefficients, see ‘Note’.
<code>coefArray</code>	an array of component models’ coefficients, their standard errors, and degrees of freedom.
<code>term.codes</code>	names of the terms with numerical codes used in the <code>summary</code> .
<code>avg.model</code>	the model averaged parameters. A <code>data.frame</code> 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).
<code>importance</code>	relative importance of the predictor variables (including interactions), calculated as a sum of the <i>Akaike weights</i> over all of the models in which the parameter of interest appears.
<code>term.names</code>	character vector giving names of all terms in the model.
<code>x, formula</code>	the model matrix and formula corresponding to the one that would be used in a single model. <code>formula</code> contains only the averaged coefficients.
<code>residuals</code>	model averaged residuals (response minus fitted values).
<code>call</code>	the matched call.

In addition, the object has following attributes:

<code>modelList</code>	a list of component model objects.
<code>beta</code>	logical, were standardized coefficients used?
<code>revised.var</code>	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).

Averaging models with different contrasts for the same factor would yield nonsense results.

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.

Vignette ‘Extending **MuMIn**’s functionality’ demonstrates how to provide support for other types of models.

[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:
library(MuMIn)
data(Cement)
fml <- lm(y ~ ., data = Cement)
(msl <- dredge(fml))

#models with delta.aicc < 4
summary(model.avg(msl, subset = delta < 4))

#or as a 95% confidence set:
avgmod.95p <- model.avg(msl, cumsum(weight) <= .95)
confint(avgmod.95p)

## Not run:
# The same result, but re-fitting the models via 'get.models'
confset.95p <- get.models(msl, cumsum(weight) <= .95)
model.avg(confset.95p)

# Force re-fitting the component models
model.avg(msl, cumsum(weight) <= .95, fit = TRUE)
# Models are also fitted if additional arguments are given
model.avg(msl, cumsum(weight) <= .95, rank = "AIC")

## 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	<i>model selection table</i>
-----------	------------------------------

---

## Description

Build a model selection table.

## Usage

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

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

## Arguments

object	A fitted model object, a list of such objects, or a "model.selection" object.
...	More fitted model objects.
rank	Optional, custom rank function (information criterion) to use instead of AICc, e.g. QAIC or BIC, may be omitted if object is a model list returned by get.models.
rank.args	Optional list of arguments for the rank function. If one is an expression, an x within it is substituted with a current model.

## Value

An object of class "model.selection" with columns containing useful information about each model: the coefficients, df, log-likelihood, the value of the information criterion used, Delta(IC) and 'Akaike weight'. If any arguments differ between the modelling function calls, the result will include additional columns showing them (except for formulas and some other arguments).

## Author(s)

Kamil Barton

## See Also

[dredge](#)

## Examples

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

```

fm1 <- glm(formula = y ~ X1 + X2 * X3, data = Cement)
fm2 <- update(fm1, . ~ . - X1 - X2)
fm3 <- update(fm1, . ~ . - X2 - X3)

## ranked with AICc by default
(msAICc <- model.sel(fm1, fm2, fm3))

## ranked with BIC
model.sel(fm1, fm2, fm3, rank = AIC, rank.args = alist(k = log(nobs(x))))
# or
model.sel(msAICc, rank = AIC, rank.args = alist(k = log(nobs(x))))
# or
# update(msAICc, rank = AIC, rank.args = alist(k = log(nobs(x))))

```

---

par.avg

---

*Parameter averaging*


---

## Description

Average a 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 adjusted standard error estimator and confidence intervals with improved coverage are returned (see 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.

---

pdredge

*Automated model selection using parallel computation*

---

**Description**

Parallelized version of `dredge`.

**Usage**

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

**Arguments**

<code>global.model</code> , <code>beta</code> , <code>evaluate</code> , <code>rank</code>	see <a href="#">dredge</a> .
<code>fixed</code> , <code>m.max</code> , <code>m.min</code> , <code>subset</code> , <code>marg.ex</code> , <code>varying</code> , <code>extra</code> , ...	see <a href="#">dredge</a> .
<code>trace</code>	displays the generated calls, but may not work as expected since the models are evaluated in batches rather than one by one.
<code>cluster</code>	either a valid <code>cluster</code> object, or <code>NA</code> for a single threaded execution.
<code>check</code>	logical, whether to evaluate the <code>global.model</code> in the cluster and compare with the original one using <code>all.equal</code> .

## Details

All the dependencies for fitting the `global.model`, including the data, and any objects the modelling function will use, must be exported (*via* e.g. `clusterExport`) into the cluster worker nodes, as well as the required packages must be loaded thereinto (e.g. *via* `clusterEvalQ(..., library(package))`), before the cluster is used by `pdredge`.

`pdredge` tries to check whether all the variables and functions used in the call to `global.model` are present in the cluster nodes' `.GlobalEnv` before proceeding further, and also, if `check` is `TRUE`, it will compare the `global.model` updated at the cluster nodes with the one given as argument. This additional test will, however, slow down the execution.

This function is still largely experimental. Use of it should be considered mainly with large datasets and complex models, for which the standard version takes a long time to complete. Otherwise there may be no perceptible improvement or even the parallel version may perform worse than a single-threaded one.

## Value

See [dredge](#).

## Author(s)

Kamil Bartoń

## See Also

`makeCluster` and other cluster related functions in packages **parallel** or **snow**.

## Examples

```
# One of these packages is required:
## Not run: require(parallel) || require(snow)

# From example(Beetle)
data(Beetle)

Beetle100 <- Beetle[sample(nrow(Beetle), 100, replace = TRUE),]

fml <- glm(Prop ~ dose + I(dose^2) + log(dose) + I(log(dose)^2),
  data = Beetle100, family = binomial)

msubset <- expression(xor(dose, `log(dose)` ) & (dose | !`I(dose^2)` )
  & (`log(dose)` | !`I(log(dose)^2)` ))
varying.link <- list(family = alist(logit = binomial("logit"),
  probit = binomial("probit"), cloglog = binomial("cloglog") ))

# Set up the cluster
clusterType <- if(length(.find.package("snow", quiet = TRUE))) "SOCK" else "PSOCK"
clust <- try(makeCluster(getOption("cl.cores", 2), type = clusterType))

if(inherits(clust, "cluster")) {

  clusterExport(clust, "Beetle100")
```

```

# noticeable gain only when data has about 3000 rows (Windows 2-core machine)
print(system.time(dredge(fml, subset = msubset, varying = varying.link)))
print(system.time(pdredge(fml, cluster = FALSE, subset = msubset,
  varying = varying.link)))
print(system.time(pdredge(fml, cluster = clust, subset = msubset,
  varying = varying.link)))

## Not run:
# A time consuming example with 'unmarked' model, based on example(pcount).
# Having enough patience, you can run this with 'demo(pdredge.pcount)'.
library(unmarked)
data(mallard)
mallardUMF <- unmarkedFramePCount(mallard.y, siteCovs = mallard.site,
  obsCovs = mallard.obs)
(ufm.mallard <- pcount(~ ivel + date + I(date^2) ~ length + elev + forest,
  mallardUMF, K = 30))
clusterEvalQ(clust, library(unmarked))
clusterExport(clust, "mallardUMF")

# 'stats4' is needed for AIC to work with unmarkedFit objects but is not
# loaded automatically with 'unmarked'.
require(stats4)
invisible(clusterCall(clust, "library", "stats4", character.only = TRUE))

#system.time(print(pdd1 <- pdredge(ufm.mallard,
#  subset = `p(date)` | !`p(I(date^2))`, rank = AIC)))

system.time(print(pdd2 <- pdredge(ufm.mallard, clust,
  subset = `p(date)` | !`p(I(date^2))`, rank = AIC, extra = "adjR^2")))

# best models and null model
subset(pdd2, delta < 2 | df == min(df))

# Compare with the model selection table from unmarked
# the statistics should be identical:
models <- pget.models(pdd2, clust, delta < 2 | df == min(df))

modSel(fitList(fits = structure(models, names = model.names(models),
  labels = getAllTerms(ufm.mallard))), nullmod = "(Null)")

## End(Not run)

stopCluster(clust)

} else # if(! inherits(clust, "cluster"))
message("Could not set up the cluster")

```



## 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

<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</code>	logical, indicates if standard errors should be returned. This has any effect only if the <code>predict</code> methods for each of the component models support it.
<code>interval</code>	Currently not used.
<code>type</code>	Predictions on response scale are only possible if all component models use the same <a href="#">family</a> . See <a href="#">predict.glm</a> .
<code>full</code>	If <code>TRUE</code> , 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> ).
<code>...</code>	Arguments to be passed to respective <code>predict</code> method (e.g. <code>level</code> for <a href="#">lme</a> model).

## Details

If all the component models are ordinary linear models, the prediction can be made either with the full averaged coefficients (the argument `full = TRUE` this is the default) or subset-averaged coefficients. Otherwise 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, otherwise 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)
fml <- lm(y ~ X1 + X2 + X3 + X4, data = Cement)

ms1 <- dredge(fml)
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[1:4], 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, full = FALSE),
  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

*Quasi AIC or AICc***Description**

Calculate 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 QAIC<sub>c</sub>; 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 a coefficient of determination based on the likelihood-ratio test ( $R_{LR}^2$ ).

## 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 <code>glm</code> with only intercept and appropriate family will be used.
evaluate	If TRUE evaluate the fitted model object else return the call.
envir	the environment in which the <i>null</i> model is to be evaluated, defaults to the environment of the original model's formula.

## Details

This statistic is one of the several proposed pseudo-R-squared's for nonlinear regression models. It is based on an improvement from *null* (intercept only) model to the fitted model, and calculated as

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

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

For OLS models the value is consistent with classical  $R^2$ . In some cases (e.g. in logistic regression), the maximum  $R_{LR}^2$  is less than one. The modification proposed by Nagelkerke (1991) adjusts the  $R_{LR}^2$  to achieve 1 at its maximum:  $\bar{R}^2 = R_{LR}^2 / \max(R_{LR}^2)$  where  $\max(R_{LR}^2) = 1 - \exp(\frac{2}{n} \log Lik(0))$ .

`null.fit` tries to guess the *null* model call (as a `glm`), given the provided fitted model object.

### Value

`r.squaredLR` returns a value of  $R_{LR}^2$ , and the attribute `"adj.r.squared"` gives the Nagelkerke's modified statistic. 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

$R^2$  is a useful goodness-of-fit measure as it has the interpretation of the proportion of the variance 'explained', but it performs poorly in model selection, and is not suitable for use in the same way as the information criterions.

### References

- Cox, D. R. and Snell, E. J. (1989) *The analysis of binary data*, 2nd ed. London, Chapman and Hall
- Magee, L. (1990)  $R^2$  measures based on Wald and likelihood ratio joint significance tests. *Amer. Stat.* 44: 250-253
- Nagelkerke, N. J. D. (1991) A note on a general definition of the coefficient of determination. *Biometrika* 78: 691-692

### 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**

`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 specifying 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.

To select rows according to presence or absence of the variables (rather than their value), a pseudo-function `has` may be used, e.g. `subset(x, has(a, !b))` will select rows with *a* **and** without *b* (this is equivalent to `!is.na(a) & is.na(b)`). `has` can take any number of arguments. Importantly, the `has()` notation cannot be used in the `subset` argument for `dredge`, where the variable names should be given directly, with the same effect.

**Author(s)**

Kamil Bartoń

**See Also**

[dredge](#), [subset](#) and [\[.data.frame\]](#) for subsetting and extracting from `data.frames`.

**Examples**

```
data(Cement)
fml <- lm(formula = y ~ X1 + X2 + X3 + X4, data = Cement)

# generate models where each variable is included only if the previous
# are included too, e.g. X2 only if X1 is there, and X3 only if X2 and X1
dredge(fml, subset = (!X2 | X1) & (!X3 | X2) & (!X4 | X3))

# alternatively, generate "all possible" combinations
ms0 <- dredge(fml)
# ...and afterwards select the subset of models
subset(ms0, (has(!X2) | has(X1)) & (has(!X3) | has(X2)) & (has(!X4) | has(X3)))
## Not run:
# this way the expression may be more clear
subset(ms0, has(X1, X2, X3, X4) | has(X1, X2, X3) | has(X1, X2) | has(X1)
      | (df == 2))

## End(Not run)
```

```
# Different ways of finding a confidence set of models:
# delta(AIC) cutoff
subset(ms0, delta <= 4, recalc.weights = FALSE)
# cumulative sum of Akaike weights
subset(ms0, cumsum(weight) <= .95, recalc.weights = FALSE)
# relative likelihood
subset(ms0, (weight / weight[1]) > (1/8), recalc.weights = FALSE)
```

---

Weights

Akaike weights

---

## Description

Calculate or extract normalized model likelihoods ('Akaike weights').

## 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, [7](#)

## \*Topic **manip**

Model utilities, [15](#)  
subset.model.selection, [29](#)

## \*Topic **models**

AICc, [3](#)  
dredge, [7](#)  
gamm-wrapper, [11](#)  
get.models, [12](#)  
importance, [13](#)  
Information criteria, [14](#)  
Model utilities, [15](#)  
model.avg, [16](#)  
model.sel, [20](#)  
MuMin-package, [2](#)  
par.avg, [21](#)  
pdredge, [22](#)  
predict.averaging, [24](#)  
QAIC, [27](#)  
r.squaredLR, [28](#)  
Weights, [31](#)

## \*Topic **package**

MuMin-package, [2](#)  
[.data.frame, [30](#)  
[.model.selection  
(subset.model.selection),  
[29](#)

AIC, [2](#), [4](#), [15](#)

AICc, [2](#), [3](#), [4](#), [15](#), [19](#), [27](#)

aicc, [4](#)

aictab, [10](#)

alist, [8](#)

Beetle, [4](#), [8](#)

bestglm, [10](#)

beta.weights (Model utilities), [15](#)

BIC, [2](#), [15](#)

CAICF, [2](#)

CAICF (Information criteria), [14](#)

Cement, [7](#)

coef, [16](#)

coef.glmulti, [19](#)

coeffs (Model utilities), [15](#)

coefTable (Model utilities), [15](#)

confint, [18](#)

Cp (Information criteria), [14](#)

dredge, [2](#), [7](#), [10](#), [12](#), [13](#), [19](#), [20](#), [22](#), [23](#), [30](#)

family, [25](#)

formula, [17](#)

gamm, [8](#), [12](#)

gamm (gamm-wrapper), [11](#)

gamm-wrapper, [11](#)

gamm4, [12](#)

get.models, [9](#), [10](#), [12](#), [19](#)

getAllTerms (Model utilities), [15](#)

glmulti, [10](#)

has (subset.model.selection), [29](#)

IC (Information criteria), [14](#)

ICOMP, [2](#)

ICOMP (Information criteria), [14](#)

ICtab, [10](#)

importance, [13](#), [31](#)

Information criteria, [14](#)

list, [12](#)

lme, [25](#)

logLik, [17](#)

Mallows' Cp, [2](#)

Mallows' Cp (Information  
criteria), [14](#)

match.fun, [17](#)

mod.sel, [13](#)

mod.sel (model.sel), [20](#)

modavg, [19](#)

Model utilities, [15](#)

model.avg, [2](#), [10](#), [12](#), [13](#), [16](#), [22](#), [25](#), [26](#)

model.names (Model utilities), [15](#)

model.sel, [2](#), [20](#)

MuMin (MuMin-package), [2](#)



MuMIn-model-utils (*Model utilities*), 15  
MuMIn-package, 2

`null.fit(r.squaredLR)`, 28

`par.avg`, 17–19, 21, 26  
`pdredge`, 2, 10, 12, 22  
`pget.models(get.models)`, 12  
`predict`, 17  
`predict.averaging`, 24  
`predict.glm`, 25  
`print.averaging(model.avg)`, 16  
`print.model.selection(dredge)`, 7

QAIC, 2, 27  
QAICc, 2  
QAICc (*QAIC*), 27  
quasi, 27  
quote, 8  
Quotes, 9

`r.squaredLR`, 8, 28  
`regsubsets`, 10  
residuals, 17

`step`, 2  
`stepAIC`, 2  
`subset`, 10, 17, 30  
subset method, 12  
`subset.model.selection`, 29  
`summary.glm`, 17  
`summary.lm`, 29  
`summary.lme`, 16

`tTable(Model utilities)`, 15

`vcov`, 17

Weights, 13, 31  
weights, 31