

# Package ‘mclogit’

May 8, 2026

**Type** Package

**Title** Multinomial Logit Models for Categorical Responses and Discrete Choices

**Version** 0.9.15

**Date** 2025-12-15

**Maintainer** Martin Elff <martin@elff.eu>

**Description** Provides estimators for multinomial logit models in their conditional logit (for discrete choices) and baseline logit variants (for categorical responses), optionally with overdispersion or random effects. Random effects models are estimated using the PQL technique (based on a Laplace approximation) or the MQL technique (based on a Solomon-Cox approximation). Estimates should be treated with caution if the group sizes are small.

**License** GPL-2

**Depends** stats, Matrix

**Imports** MASS, memisc, methods, nlme

**Suggests** nnet, ucminf, knitr, rmarkdown

**LazyLoad** Yes

**VignetteBuilder** knitr

**URL** <http://melff.github.io/mclogit/>, <https://github.com/melff/mclogit/>

**BugReports** <https://github.com/melff/mclogit/issues>

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**NeedsCompilation** no

**Author** Martin Elff [aut, cre]

**Repository** CRAN

**Date/Publication** 2025-12-16 06:10:30 UTC

## Contents

|                              |           |
|------------------------------|-----------|
| dispersion . . . . .         | 2         |
| electors . . . . .           | 4         |
| getSummary-methods . . . . . | 5         |
| mblogit . . . . .            | 7         |
| mclogit . . . . .            | 10        |
| mclogit.control . . . . .    | 14        |
| mclogit.fit . . . . .        | 16        |
| predict . . . . .            | 17        |
| rebase . . . . .             | 19        |
| simulate.mclogit . . . . .   | 19        |
| Transport . . . . .          | 21        |
| <b>Index</b>                 | <b>22</b> |

---

|            |   |
|------------|---|
| dispersion | <i>Overdispersion in Multinomial Logit Models</i> |
|------------|---|

---

### Description

The function `dispersion()` extracts the dispersion parameter from a multinomial logit model or computes a dispersion parameter estimate based on a given method. This dispersion parameter can be attached to a model using `update()`. It can also be given as an argument to `summary()`.

### Usage

```
dispersion(object, method, ...)
## S3 method for class 'mclogit'
dispersion(object, method=NULL, ...)
```

### Arguments

|                     |   |
|---------------------|---|
| <code>object</code> | an object that inherits class "mclogit". When passed to <code>dispersion()</code> , it should be the result of a call of <code>mclogit()</code> or <code>mblogit()</code> , <i>without</i> random effects.                                    |
| <code>method</code> | a character string, either "Afroz", "Fletcher", "Pearson", or "Deviance", that specifies the estimator of the dispersion; or NULL, in which case the default estimator, "Afroz" is used. The estimators are discussed in Afroz et al. (2019). |
| <code>...</code>    | other arguments, ignored or passed to other methods.  |

### References

Afroz, Farzana, Matt Parry, and David Fletcher. (2020). "Estimating Overdispersion in Sparse Multinomial Data." *Biometrics* 76(3): 834-842. doi:10.1111/biom.13194.

**Examples**

```

library(MASS) # For 'housing' data

# Note that with a factor response and frequency weighted data,
# Overdispersion will be overestimated:
house.mblogit <- mblogit(Sat ~ Infl + Type + Cont,
                        weights = Freq,
                        data = housing)
dispersion(house.mblogit, method = "Afroz")
dispersion(house.mblogit, method = "Deviance")

# In order to be able to estimate overdispersion accurately,
# data like the above (which usually comes from applying
# 'as.data.frame' to a contingency table) the model has to be
# fitted with the optional argument 'aggregate=TRUE' or
# by requesting the dispersion in advance.
house.mblogit.agg <- mblogit(Sat ~ Infl + Type + Cont,
                            weights = Freq,
                            data = housing,
                            aggregate = TRUE)

# Now the estimated dispersion parameter is no longer larger than 20,
# but just bit over 1.0.
dispersion(house.mblogit.agg, method = "Afroz")
dispersion(house.mblogit.agg, method = "Deviance")

# It is possible to obtain the dispersion after estimating the coefficients:
phi.Afroz <- dispersion(house.mblogit.agg, method = "Afroz")
summary(house.mblogit.agg, dispersion = phi.Afroz)

summary(update(house.mblogit.agg, dispersion = "Afroz"))

# If an estimate of the (over-)dispersion is requested, 'aggregate' is set to
# TRUE by default:
house.mblogit.odsp <- mblogit(Sat ~ Infl + Type + Cont,
                             weights = Freq,
                             data = housing,
                             dispersion = "Afroz")

summary(house.mblogit.odsp)
dispersion(house.mblogit.odsp, method = "Deviance")

# Note that aggregation (either implicitly or explicitly required) affects
# the reported deviance in surprising ways:
house.mblogit.o.00 <- mblogit(Sat ~ Infl,
                              weights = Freq,
                              data = housing,
                              dispersion = TRUE)

deviance(house.mblogit.o.00)
dispersion(house.mblogit.o.00)
# The deviance is (almost) zero, because aggregation leads to a two-way
# table and a single-predictor model is already saturated.

# In order to make models comparable, one will need to set the groups:

```

```

house.mblogit.o.0 <- mblogit(Sat ~ Infl,
                            weights = Freq,
                            data = housing,
                            groups = ~ Infl + Type + Cont,
                            dispersion = TRUE)

deviance(house.mblogit.o.0)
dispersion(house.mblogit.o.0)

anova(house.mblogit.o.0,
      house.mblogit.odsp)

# These complications with the deviances do not arise if no aggregation is
# requested:
house.mblogit.0 <- mblogit(Sat ~ Infl,
                          weights = Freq,
                          data = housing)

anova(house.mblogit.0,
      house.mblogit)

# Using frequencies on the left-hand side is perhaps the safest option:
housing.wide <- memisc::Aggregate(table(Sat) ~ Infl + Type + Cont,
                                data = housing) # Note that 'Aggregate' uses
                                                # variable 'Freq' for weighting.
house.mblogit.wide <- mblogit(cbind(Low,Medium,High) ~ Infl + Type + Cont,
                             data = housing.wide)

summary(house.mblogit.wide)
dispersion(house.mblogit.wide, method = "Afroz")

house.mblogit.wide.0 <- mblogit(cbind(Low,Medium,High) ~ Infl,
                              data = housing.wide)

summary(house.mblogit.wide.0)
dispersion(house.mblogit.wide.0, method="Afroz")

anova(house.mblogit.wide.0,
      house.mblogit.wide)

```

---

electors

*Class, Party Position, and Electoral Choice*

---

### Description

This is an artificial data set on electoral choice as influenced by class and party positions.

### Usage

```
data(electors)
```

**Format**

A data frame containing the following variables:

**class** class position of voters

**party** party that runs for election

**Freq** frequency by which each party list is chosen by members of each class

**time** time variable, runs from zero to one

**econ.left** economic-policy "leftness" of each party

**welfare** emphasis of welfare expansion of each party

**auth** position on authoritarian issues

**Examples**

```
data(electors)

summary(mclogit(
  cbind(Freq, interaction(time, class))~econ.left+welfare+auth,
  data=electors))

summary(mclogit(
  cbind(Freq, interaction(time, class))~econ.left/class+welfare/class+auth/class,
  data=electors))

## Not run: # This takes a bit longer.
summary(mclogit(
  cbind(Freq, interaction(time, class))~econ.left/class+welfare/class+auth/class,
  random=~1|party.time,
  data=within(electors, party.time<-interaction(party, time))))

summary(mclogit(
  cbind(Freq, interaction(time, class))~econ.left/(class*time)+welfare/class+auth/class,
  random=~1|party.time,
  data=within(electors, {
    party.time <-interaction(party, time)
    econ.left.sq <- (econ.left-mean(econ.left))^2
  })))

## End(Not run)
```

---

getSummary-methods      *'getSummary' Methods*

---

**Description**

[getSummary](#) methods for use by [mtable](#)

**Usage**

```
## S3 method for class 'mblogit'
getSummary(obj,
            alpha=.05,
            ...)
## S3 method for class 'mclogit'
getSummary(obj,
            alpha=.05,
            rearrange=NULL,
            ...)
## S3 method for class 'mmblogit'
getSummary(obj,
            alpha=.05,
            ...)
## S3 method for class 'mmclogit'
getSummary(obj,
            alpha=.05,
            rearrange=NULL,
            ...)
```

**Arguments**

|           |   |
|-----------|---|
| obj       | an object returned by <code>mblogit</code> or <code>mclogit</code>  |
| alpha     | level of the confidence intervals; their coverage should be $1-\alpha/2$  |
| rearrange | an optional named list of character vectors. Each element of the list designates a column in the table of estimates, and each element of a character vector refers to a coefficient. Names of list elements become column heads and names of the character vector elements become coefficient labels. |
| ...       | further arguments; ignored.   |

**Examples**

```
## Not run:
summary(classd.model <- mclogit(cbind(Freq,choice.set)~
  (econdim1.sq+nonmatdim1.sq+nonmatdim2.sq)+
  (econdim1+nonmatdim1+nonmatdim2)+
  (econdim1+nonmatdim1+nonmatdim2):classd,
  data=mvoteint.classd,random=~1|mvoteint/eb,
  subset=classd!="Farmers"))
myGetSummary.classd <- function(x)getSummary.mclogit(x,rearrange=list(
  "Econ. Left/Right"=c(
    "Squared effect"="econdim1.sq",
    "Linear effect"="econdim1",
    " x Intermediate/Manual worker"="econdim1:classdIntermediate",
    " x Service class/Manual worker"="econdim1:classdService class",
    " x Self-employed/Manual worker"="econdim1:classdSelf-employed"
  ),
  "Lib./Auth."=c(
```

```

    "Squared effect"="nonmatdim1.sq",
    "Linear effect"="nonmatdim1",
    " x Intermediate/Manual worker"="nonmatdim1:classdIntermediate",
    " x Service class/Manual worker"="nonmatdim1:classdService class",
    " x Self-employed/Manual worker"="nonmatdim1:classdSelf-employed"
  ),
  "Mod./Trad."=c(
    "Squared effect"="nonmatdim2.sq",
    "Linear effect"="nonmatdim2",
    " x Intermediate/Manual worker"="nonmatdim2:classdIntermediate",
    " x Service class/Manual worker"="nonmatdim2:classdService class",
    " x Self-employed/Manual worker"="nonmatdim2:classdSelf-employed"
  )
))

library(memisc)
mtable(classd.model,getSummary=myGetSummary.classd)
# Output would look like so:
# =====
#                               Econ. Left/Right   Lib./Auth.   Mod./Trad.
# -----
# Squared effect                 0.030           0.008          -0.129**
#                               (0.081)         (0.041)         (0.047)
# Linear effect                  -0.583***        -0.038          0.137**
#                               (0.063)         (0.041)         (0.045)
# x Intermediate/Manual worker    0.632***        -0.029          -0.015
#                               (0.026)         (0.020)         (0.019)
# x Service class/Manual worker   1.158***         0.084**         0.000
#                               (0.040)         (0.032)         (0.030)
# x Self-employed/Manual worker  1.140***         0.363***        0.112***
#                               (0.035)         (0.027)         (0.026)
# Var(mvoint)                   1.080***
#                               (0.000)
# Var(mvoint x eb)              0.118***
#                               (0.000)
# -----
# Dispersion                     1.561
# Deviance                      15007.0
# N                             173445
# =====

## End(Not run)

```

mblogit

*Baseline-Category Logit Models for Categorical and Multinomial Responses*

## Description

The function `mblogit` fits baseline-category logit models for categorical and multinomial count responses with fixed alternatives.

**Usage**

```
mblogit(
  formula,
  data = parent.frame(),
  random = NULL,
  catCov = c("free", "diagonal", "single"),
  subset,
  weights = NULL,
  offset = NULL,
  na.action = getOption("na.action"),
  model = TRUE,
  x = FALSE,
  y = TRUE,
  contrasts = NULL,
  method = NULL,
  estimator = c("ML", "REML"),
  dispersion = FALSE,
  start = NULL,
  aggregate = FALSE,
  groups = NULL,
  from.table = FALSE,
  control = if (length(random)) mmclogit.control(...) else mclogit.control(...),
  ...
)
```

**Arguments**

|         |   |
|---------|---|
| formula | the model formula. The response must be a factor or a matrix of counts.   |
| data    | an optional data frame, list or environment (or object coercible by <a href="#">as.data.frame</a> to a data frame) containing the variables in the model. If not found in data, the variables are taken from <code>environment(formula)</code> , typically the environment from which <code>glm</code> is called.                                 |
| random  | an optional formula or list of formulas that specify the random-effects structure or <code>NULL</code> .  |
| catCov  | a character string that specifies optional restrictions on the covariances of random effects between the logit equations. "free" means no restrictions, "diagonal" means that random effects pertinent to different categories are uncorrelated, while "single" means that the random effect variances pertinent to all categories are identical. |
| subset  | an optional vector specifying a subset of observations to be used in the fitting process.   |
| weights | an optional vector of weights to be used in the fitting process. Should be <code>NULL</code> or a numeric vector.   |
| offset  | an optional model offset. If not <code>NULL</code> , must be a matrix if as many columns as the response has categories or one less.  |

|            |  |
|------------|--|
| na.action  | a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of <code>options</code> , and is <code>na.fail</code> if that is unset. The 'factory-fresh' default is <code>na.omit</code> . Another possible value is NULL, no action. Value <code>na.exclude</code> can be useful.   |
| model      | a logical value indicating whether <i>model frame</i> should be included as a component of the returned value.   |
| x, y       | logical values indicating whether the response vector and model matrix used in the fitting process should be returned as components of the returned value.   |
| contrasts  | an optional list. See the <code>contrasts.arg</code> of <code>model.matrix.default</code> .  |
| method     | NULL or a character string, either "PQL" or "MQL", specifies the type of the quasiliikelihood approximation to be used if a random-effects model is to be estimated.   |
| estimator  | a character string; either "ML" or "REML", specifies which estimator is to be used/approximated.   |
| dispersion | a logical value or a character string; whether and how a dispersion parameter should be estimated. For details see <a href="#">dispersion</a> .  |
| start      | an optional matrix of starting values (with as many rows as logit equations). If the model has random effects, the matrix should have a "VarCov" attribute with starting values for the random effects (co-)variances. If the random effects model is estimated with the "PQL" method, the starting values matrix should also have a "random.effects" attribute, which should have the same structure as the "random.effects" component of an object returned by <code>mblogit()</code> .  |
| aggregate  | a logical value; whether to aggregate responses by covariate classes and groups before estimating the model if the response variable is a factor.<br><br>This will not affect the estimates, but the dispersion and the residual degrees of freedom. If <code>aggregate=TRUE</code> , the dispersion will be relative to a saturated model; it will be much smaller than with <code>aggregate=FALSE</code> . In particular, with only a single covariate and no grouping, the deviance will be close to zero. If <code>dispersion</code> is not <code>FALSE</code> , then the default value of <code>aggregate</code> will be <code>TRUE</code> . For details see <a href="#">dispersion</a> .<br><br>This argument has consequences only if the response in <code>formula</code> is a factor. |
| groups     | an optional formula that specifies groups of observations relevant for the estimation of overdispersion. For details see <a href="#">dispersion</a> .  |
| from.table | a logical value; should be <code>FALSE</code> . This argument only exists for the sake of compatibility and will be removed in the next release.   |
| control    | a list of parameters for the fitting process. See <a href="#">mclgfit.control</a>  |
| ...        | arguments to be passed to <code>mclgfit.control</code> or <code>mmclgfit.control</code>  |

## Details

The function `mblogit` internally rearranges the data into a 'long' format and uses `mclgfit.fit` to compute estimates. Nevertheless, the 'user data' are unaffected.

## Value

mblogit returns an object of class "mblogit", which has almost the same structure as an object of class "glm". The difference are the components coefficients, residuals, fitted.values, linear.predictors, and y, which are matrices with number of columns equal to the number of response categories minus one.

## References

Agresti, Alan. 2002. *Categorical Data Analysis*. 2nd ed, Hoboken, NJ: Wiley. doi:10.1002/0471249688

Breslow, N.E. and D.G. Clayton. 1993. "Approximate Inference in Generalized Linear Mixed Models". *Journal of the American Statistical Association* 88 (421): 9-25. doi:10.1080/01621459.1993.10594284

## See Also

The function `multinom` in package `nnet` also fits multinomial baseline-category logit models, but has a slightly less convenient output and does not support overdispersion or random effects. However, it provides some other options. Baseline-category logit models are also supported by the package `VGAM`, as well as some reduced-rank and (semi-parametric) additive generalisations. The package `mnlogit` estimates logit models in a way optimized for large numbers of alternatives.

## Examples

```
library(MASS) # For 'housing' data
library(nnet)
library(memisc)

(house.mult<- multinom(Sat ~ Infl + Type + Cont, weights = Freq,
                      data = housing))

(house.mblogit <- mblogit(Sat ~ Infl + Type + Cont, weights = Freq,
                        data = housing))

summary(house.mult)

summary(house.mblogit)

mtable(house.mblogit)
```

## Description

mclgit fits conditional logit models and mixed conditional logit models to count data and individual choice data, where the choice set may vary across choice occasions.

Conditional logit models without random effects are fitted by Fisher-scoring/IWLS. Models with random effects (mixed conditional logit models) are estimated via maximum likelihood with a simple Laplace approximation (aka PQL).

## Usage

```
mclgit(formula, data=parent.frame(), random=NULL,
       subset, weights = NULL, offset=NULL, na.action = getOption("na.action"),
       model = TRUE, x = FALSE, y = TRUE, contrasts=NULL,
       method = NULL, estimator=c("ML","REML"),
       dispersion = FALSE,
       start=NULL,
       groups = NULL,
       control=if(length(random))
                mmclogit.control(...)
                else mclgit.control(...), ...)
```

```
## S3 method for class 'mclgit'
update(object, formula., dispersion, ...)
```

```
## S3 method for class 'mclgit'
summary(object, dispersion = NULL, correlation = FALSE,
        symbolic.cor = FALSE, ...)
```

## Arguments

|         |   |
|---------|---|
| formula | <p>a model formula: a symbolic description of the model to be fitted. The left-hand side should result in a two-column matrix. The first column contains the choice counts or choice indicators (alternative is chosen=1, is not chosen=0). The second column contains unique numbers for each choice set.</p> <p>The left-hand side can either take the form <code>cbind(choice, set)</code> or (from version 0.9.1) <code>choice set</code></p> <p>If individual-level data is used, choice sets correspond to individuals, if aggregated data with choice counts are used, choice sets usually correspond to covariate classes.</p> <p>The right-hand of the formula contains choice predictors. It should be noted that constants are deleted from the formula as are predictors that do not vary within choice sets.</p> |
| data    | <p>an optional data frame, list or environment (or object coercible by <code>as.data.frame</code> to a data frame) containing the variables in the model. If not found in data, the variables are taken from <code>environment(formula)</code>, typically the environment from which <code>glm</code> is called.</p>  |
| random  | <p>an optional formula or list of formulas that specify the random-effects structure or <code>NULL</code>.</p>  |

|              |   |
|--------------|---|
| subset       | an optional vector specifying a subset of observations to be used in the fitting process.   |
| weights      | an optional vector of weights to be used in the fitting process. Should be NULL or a numeric vector.  |
| offset       | an optional model offset.   |
| na.action    | a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of <code>options</code> , and is <code>na.fail</code> if that is unset. The ‘factory-fresh’ default is <code>na.omit</code> . Another possible value is NULL, no action. Value <code>na.exclude</code> can be useful.  |
| start        | an optional numerical vector of starting values for the conditional logit parameters. If the model has random effects, the vector should have a "VarCov" attribute with starting values for the random effects (co-)variances. If the random effects model is estimated with the "PQL" method, the starting values matrix should also have a "random.effects" attribute, which should have the same structure as the "random.effects" component of an object returned by <code>mblogit()</code> . |
| model        | a logical value indicating whether <i>model frame</i> should be included as a component of the returned value.  |
| x, y         | logical values indicating whether the response vector and model matrix used in the fitting process should be returned as components of the returned value.  |
| contrasts    | an optional list. See the <code>contrasts.arg</code> of <code>model.matrix.default</code> .   |
| method       | NULL or a character string, either "PQL" or "MQL", specifies the type of the quasiliikelihood approximation to be used if a random-effects model is to be estimated.  |
| estimator    | a character string; either "ML" or "REML", specifies which estimator is to be used/approximated.  |
| dispersion   | a real number used as dispersion parameter; a character vector that specifies the method to compute the dispersion; a logical value – if TRUE the default method ("Afroz") is used, if FALSE, the dispersion parameter is set to 1, that is, no dispersion. For details see <a href="#">dispersion</a> .  |
| groups       | an optional formula that specifies groups of observations relevant for the estimation of overdispersion. Covariates should be constant within groups, otherwise a warning is generated since the overdispersion estimate may be imprecise.  |
| control      | a list of parameters for the fitting process. See <a href="#">mclgfit.control</a>   |
| ...          | arguments to be passed to <code>mclgfit.control</code> or <code>mmclgfit.control</code>   |
| object       | an object that inherits class "mclgfit". When passed to <code>dispersion()</code> , it should be the result of a call of <code>mclgfit()</code> or <code>mblogit()</code> , <i>without</i> random effects.  |
| formula.     | a changes to the model formula, see <a href="#">update.default</a> and <a href="#">update.formula</a> .   |
| correlation  | logical; see <a href="#">summary.lm</a> .   |
| symbolic.cor | logical; see <a href="#">summary.lm</a> .   |

### Value

`mclgfit` returns an object of class "mclgfit", which has almost the same structure as an object of class "[glm](#)".

**Note**

Covariates that are constant within choice sets are automatically dropped from the model formula specified by the `formula` argument of `mclgit`.

If the model contains random effects, these should

- either vary within choice sets (e.g. the levels of a factor that defines the choice sets should not be nested within the levels of factor)
- or be random coefficients of covariates that vary within choice sets.

In earlier versions of the package (prior to 0.6) it will lead to a failure of the model fitting algorithm if these conditions are not satisfied. Since version 0.6 of the package, the function `mclgit` will complain about such model a misspecification explicitly.

From version 0.9.7 it is possible to choose the optimization technique used for the inner iterations of the PQL/MQL: either `nlminb` (the default), `nlm`, or any of the algorithms (other than "Brent" supported by `optim`). To choose the optimizer, use the appropriate argument for `mmclgit.control`.

**References**

- Agresti, Alan (2002). *Categorical Data Analysis*. 2nd ed, Hoboken, NJ: Wiley. [doi:10.1002/0471249688](https://doi.org/10.1002/0471249688)
- Breslow, N.E. and D.G. Clayton (1993). "Approximate Inference in Generalized Linear Mixed Models". *Journal of the American Statistical Association* 88 (421): 9-25. [doi:10.1080/01621459.1993.10594284](https://doi.org/10.1080/01621459.1993.10594284)
- Elff, Martin (2009). "Social Divisions, Party Positions, and Electoral Behaviour". *Electoral Studies* 28(2): 297-308. [doi:10.1016/j.electstud.2009.02.002](https://doi.org/10.1016/j.electstud.2009.02.002)
- McFadden, D. (1973). "Conditional Logit Analysis of Qualitative Choice Behavior". Pp. 105-135 in P. Zarembka (ed.). *Frontiers in Econometrics*. New York: Wiley. <https://eml.berkeley.edu/reprints/mcfadden/zarembka.pdf>

**See Also**

Conditional logit models are also supported by **gmn**, **mlogit**, and **survival**. **survival** supports conditional logit models for binary panel data and case-control studies. **mlogit** and **gmn** treat conditional logit models from an econometric perspective. Unlike the present package, they focus on the random utility interpretation of discrete choice models and support generalisations of conditional logit models, such as nested logit models, that are intended to overcome the IIA (independence from irrelevant alternatives) assumption. Mixed multinomial models are also supported and estimated using simulation-based techniques. Unlike the present package, mixed or random-effects extensions are mainly intended to fit repeated choices of the same individuals and not aggregated choices of many individuals facing identical alternatives.

**Examples**

```
data(Transport)

summary(mclgit(
  cbind(resp,suburb)~distance+cost,
  data=Transport
```

```

))
# New syntactic sugar:
summary(mclgit(
  resp|suburb~distance+cost,
  data=Transport
))

## Not run: # This takes a bit longer.
data(electors)

electors <- within(electors,{
  party.time <-interaction(party,time)
  time.class <- interaction(time,class)
})

# Time points nested within parties
summary(mclgit(
  Freq|time.class~econ.left/class+welfare/class+auth/class,
  random=~1|party/time,
  data=electors))

# Party-level random intercepts and random slopes varying over time points
summary(mclgit(
  Freq|time.class~econ.left/class+welfare/class+auth/class,
  random=list(~1|party,~econ.left+0|time),
  data=electors))

## End(Not run)

```

---

mclgit.control

*Control Parameters for the Fitting Process*


---

## Description

mclgit.control returns a list of default parameters that control the fitting process of mclgit.

## Usage

```

mclgit.control(epsilon = 1e-08,
               maxit = 25, trace=TRUE)
mmclgit.control(epsilon = 1e-08,
               maxit = 25, trace=TRUE,
               trace.inner=FALSE,
               avoid.increase = FALSE,
               break.on.increase = FALSE,
               break.on.infinite = FALSE,
               break.on.negative = FALSE,
               inner.optimizer = "nlminb",

```

```

maxit.inner = switch(inner.optimizer,
                     SANN           = 10000,
                     `Nelder-Mead` = 500,
                     100),
CG.type = 1,
NM.alpha = 1,
NM.beta = 0.5,
NM.gamma = 2.0,
SANN.temp = 10,
SANN.tmax = 10,
grtol = 1e-6,
xtol = 1e-8,
maxeval = 100,
gradstep = c(1e-6, 1e-8),
use.gradient = c("analytic", "numeric"))

```

### Arguments

|                   |   |
|-------------------|---|
| epsilon           | positive convergence tolerance $\epsilon$ ; the iterations converge when $ dev - dev_{old}  / ( dev  + 0.1) < \epsilon$ .   |
| maxit             | integer giving the maximal number of IWLS or PQL iterations.  |
| trace             | logical indicating if output should be produced for each iteration.   |
| trace.inner       | logical; indicating if output should be produced for each inner iteration of the PQL method.  |
| avoid.increase    | logical; should an increase of the deviance be avoided by step truncation?  |
| break.on.increase | logical; should an increase of the deviance be avoided by stopping the algorithm?   |
| break.on.infinite | logical; should an infinite deviance stop the algorithm instead of leading to step truncation?  |
| break.on.negative | logical; should a negative deviance stop the algorithm?   |
| inner.optimizer   | a character string, one of "nlminb", "nlm", "ucminf", "Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN". See <a href="#">nlminb</a> , <a href="#">nlm</a> , <a href="#">ucminf</a> , or <a href="#">optim</a> . |
| maxit.inner       | integer; the maximum number of inner iterations   |
| CG.type           | integer; the type argument passed to <a href="#">optim</a> if "CG" is selected as inner optimizer.  |
| NM.alpha          | integer; the alpha argument passed to <a href="#">optim</a> if "Nelder-Mead" is selected as inner optimizer.  |
| NM.beta           | integer; the beta argument passed to <a href="#">optim</a> if "Nelder-Mead" is selected as inner optimizer.   |
| NM.gamma          | integer; the gamma argument passed to <a href="#">optim</a> if "Nelder-Mead" is selected as inner optimizer.  |

|              |   |
|--------------|---|
| SANN.temp    | integer; the temp argument passed to <code>optim</code> if "SANN" is selected as inner optimizer.                                     |
| SANN.tmax    | integer; the tmax argument passed to <code>optim</code> if "SANN" is selected as inner optimizer.                                     |
| grtol        | numeric; the grtol control parameter for <code>ucminf</code> if "ucminf" is selected as inner optimizer.                              |
| xtol         | numeric; the xtol control parameter for <code>ucminf</code> if "ucminf" is selected as inner optimizer.                               |
| maxeval      | integer; the maxeval control parameter for <code>ucminf</code> if "ucminf" is selected as inner optimizer.                            |
| gradstep     | a numeric vector of length; the gradstep control parameter for <code>ucminf</code> if "ucminf" is selected as inner optimizer.        |
| use.gradient | a character string; whether the gradient should be computed analytically or whether a finite-difference approximation should be used. |

**Value**

A list.

---

|         |   |
|---------|---|
| mclgfit | <i>Internal functions used for model fit.</i> |
|---------|---|

---

**Description**

These functions are exported and documented for use by other packages. They are not intended for end users.

**Usage**

```
mclgfit(y, s, w, X,
        dispersion=FALSE,
        start = NULL, offset = NULL,
        control = mclgfit.control())

mmclgfitPQLMQL(y, s, w, X, Z, d,
               start = NULL,
               start.Phi = NULL,
               start.b = NULL,
               offset = NULL, method=c("PQL", "MQL"),
               estimator = c("ML", "REML"),
               control = mmclgfit.control())
```

**Arguments**

|            |   |
|------------|---|
| y          | a response vector. Should be binary.  |
| s          | a vector identifying individuals or covariate strata  |
| w          | a vector with observation weights.  |
| X          | a model matrix; required.   |
| dispersion | a logical value or a character string; whether and how a dispersion parameter should be estimated. For details see <a href="#">dispersion</a> . |
| Z          | the random effects design matrix.   |
| d          | dimension of random effects. Typically $d=1$ for random intercepts only, $d>1$ for models with random intercepts.                               |
| start      | an optional numerical vector of starting values for the coefficients.   |
| offset     | an optional model offset. Currently only supported for models without random effects.   |
| start.Phi  | an optional matrix of starting values for the (co-)variance parameters.   |
| start.b    | an optional list of vectors with starting values for the random effects.  |
| method     | a character string, either "PQL" or "MQL", specifies the type of the quasiliikelihood approximation.  |
| estimator  | a character string; either "ML" or "REML", specifies which estimator is to be used/approximated.  |
| control    | a list of parameters for the fitting process. See <a href="#">mclgfit.control</a>   |

**Value**

A list with components describing the fitted model.

---

|         |   |
|---------|---|
| predict | <i>Predicting responses or linear parts of the baseline-category and conditional logit models</i> |
|---------|---|

---

**Description**

The `predict()` methods allow to obtain within-sample and out-of-sample predictions from models fitted with `mclgfit()` and `mblogit()`.

For models with random effects fitted using the PQL-method, it is possible to obtain responses that are conditional on the reconstructed random effects.

**Usage**

```
## S3 method for class 'mblogit'
predict(object, newdata=NULL, type=c("link", "response"), se.fit=FALSE, ...)
## S3 method for class 'mclogit'
predict(object, newdata=NULL, type=c("link", "response"), se.fit=FALSE, ...)
## S3 method for class 'mmblogit'
predict(object, newdata=NULL, type=c("link", "response"), se.fit=FALSE,
        conditional=TRUE, ...)
## S3 method for class 'mmclogit'
predict(object, newdata=NULL, type=c("link", "response"), se.fit=FALSE,
        conditional=TRUE, ...)
```

**Arguments**

|             |  |
|-------------|--|
| object      | an object in class "mblogit", "mmblogit", "mclogit", or "mmclogit"   |
| newdata     | an optional data frame with new data   |
| type        | a character string specifying the kind of prediction   |
| se.fit      | a logical value; whether predictions should be accompanied with standard errors  |
| conditional | a logical value; whether predictions should be made conditional on the random effects (or whether they are set to zero, i.e. their expectation). This argument is consequential only if the "mmblogit" or "mmclogit" object was created with method="PQL". |
| ...         | other arguments, ignored.  |

**Value**

The predict methods return either a matrix (unless called with `se.fit=TRUE`) or a list with two matrix-valued elements "fit" and "se.fit".

**Examples**

```
library(MASS)
(house.mblogit <- mblogit(Sat ~ Infl + Type + Cont,
                        data = housing,
                        weights=Freq))

head(pred.house.mblogit <- predict(house.mblogit))
str(pred.house.mblogit <- predict(house.mblogit, se=TRUE))

head(pred.house.mblogit <- predict(house.mblogit,
                                  type="response"))
str(pred.house.mblogit <- predict(house.mblogit, se=TRUE,
                                  type="response"))

# This takes a bit longer.
data(electors)
(mcre <- mclogit(
  cbind(Freq, interaction(time, class))~econ.left/class+welfare/class+auth/class,
  random=~1|party.time,
  data=within(electors, party.time<-interaction(party, time))))
```

```
str(predict(mcre))
str(predict(mcre, type="response"))

str(predict(mcre, se.fit=TRUE))
str(predict(mcre, type="response", se.fit=TRUE))
```

---

|        |   |
|--------|---|
| rebase | <i>Change baseline category of multinomial logit or similar model</i> |
|--------|---|

---

### Description

'rebase' returns an model object that is equivalent to the one given as argument but differs in parameterization

### Usage

```
rebase(object, to, ...)

## S3 method for class 'mblogit'
rebase(object, to, ...)
```

### Arguments

|        |  |
|--------|--|
| object | a statistical model object               |
| to     | usually, a string; the baseline category |
| ...    | other arguments, currently ignored       |

---

|                 |   |
|-----------------|---|
| simulate.mclgit | <i>Simulating responses from baseline-category and conditional logit models</i> |
|-----------------|---|

---

### Description

The simulate() methods allow to simulate responses from models fitted with mclgit() and mblogit(). Currently only models *without* random effects are supported for this.

**Usage**

```
## S3 method for class 'mblogit'
simulate(object, nsim = 1, seed = NULL, ...)
## S3 method for class 'mclgit'
simulate(object, nsim = 1, seed = NULL, ...)

# These methods are currently just 'stubs', causing an error
# message stating that simulation from models with random
# effects are not supported yet
## S3 method for class 'mmblogit'
simulate(object, nsim = 1, seed = NULL, ...)
## S3 method for class 'mmclogit'
simulate(object, nsim = 1, seed = NULL, ...)
```

**Arguments**

|        |  |
|--------|--|
| object | an object from the relevant class  |
| nsim   | a number, specifying the number of simulated responses for each observation.   |
| seed   | an object specifying if and how the random number generator should be initialized ('seeded'). The interpretation of this argument follows the default method, see <code>link[stats]{simulate}</code> |
| ...    | other arguments, ignored.  |

**Value**

The result of the `simulate` method for objects created by `mclgit` is a data frame with one variable for each requested simulation run (their number is given by the `nsim=` argument). The contents of the columns are counts (or zero-one values), with group-wise multinomial distribution (within choice sets) just like it is assumed for the original response.

The shape of the result of the `simulate` method for objects created by `mblogit` is also a data frame. The variables within the data frame have a mode or shape that corresponds to the response to which the model was fitted. If the response is a matrix of counts, then the variables in the data frame are also matrices of counts. If the response is a factor and `mblogit` was called with an argument `from.table=FALSE`, the variables in the data frame are factors with the same factor levels as the response to which the model was fitted. If instead the function was called with `from.table=TRUE`, the variables in the data frame are counts, which represent frequency weights that would result from applying `as.data.frame` to a contingency table of simulated frequency counts.

**Examples**

```
library(MASS)
(house.mblogit <- mblogit(Sat ~ Infl + Type + Cont,
  data = housing,
  weights=Freq,
  aggregate=TRUE))
sm <- simulate(house.mblogit,nsim=7)

housing.long <- housing[rep(seq.int(nrow(housing)),housing$Freq),]
```

```

(house1.mblogit <- mblogit(Sat ~ Infl + Type + Cont,
                          data=housing.long))
sml <- simulate(house1.mblogit,nsim=7)

housing.table <- xtabs(Freq~,data=housing)
housing.mat <- memisc::to.data.frame(housing.table)
head(housing.mat)

(housem.mblogit <- mblogit(cbind(Low,Medium,High) ~
                           Infl + Type + Cont,
                           data=housing.mat))
smm <- simulate(housem.mblogit,nsim=7)

str(sm)
str(sml)
str(smm)

head(smm[[1]])

```

---

Transport

*Choice of Means of Transport*


---

### Description

This is an artificial data set on choice of means of transport based on cost and walking distance.

### Usage

```
data(Transport)
```

### Format

A data frame containing the following variables:

**transport** means of transportation that can be chosen.

**suburb** identifying number for each suburb

**distance** walking distance to bus or train station

**cost** cost of each means of transportation

**working** size of working population of each suburb

**prop.true** true choice probabilities

**resp** choice frequencies of means of transportation

# Index

- \* **datasets**
  - electors, [4](#)
  - Transport, [21](#)
- \* **models**
  - mclogit, [10](#)
- \* **regression**
  - mclogit, [10](#)
- AIC.mclogit (mclogit), [10](#)
- anova.mclogit (mclogit), [10](#)
- as.data.frame, [8](#), [11](#), [20](#)
- BIC.mclogit (mclogit), [10](#)
- deviance.mclogit (mclogit), [10](#)
- dispersion, [2](#), [9](#), [12](#), [17](#)
- electors, [4](#)
- fitted.mblogit (mblogit), [7](#)
- fitted.mclogit (mclogit), [10](#)
- getSummary, [5](#)
- getSummary-methods, [5](#)
- getSummary.mblogit
  - (getSummary-methods), [5](#)
- getSummary.mclogit
  - (getSummary-methods), [5](#)
- getSummary.mmblogit
  - (getSummary-methods), [5](#)
- getSummary.mmclogit
  - (getSummary-methods), [5](#)
- glm, [10](#), [12](#)
- logLik.mclogit (mclogit), [10](#)
- mblogit, [6](#), [7](#), [20](#)
- mclogit, [6](#), [10](#), [20](#)
- mclogit.control, [9](#), [12](#), [14](#), [17](#)
- mclogit.fit, [9](#), [16](#)
- mmclogit.control, [13](#)
- mmclogit.control (mclogit.control), [14](#)
- mmclogit.fitPQLMQL (mclogit.fit), [16](#)
- mtable, [5](#)
- multinom, [10](#)
- na.exclude, [9](#), [12](#)
- na.fail, [9](#), [12](#)
- na.omit, [9](#), [12](#)
- nlm, [13](#), [15](#)
- nlminb, [13](#), [15](#)
- optim, [13](#), [15](#), [16](#)
- options, [9](#), [12](#)
- predict, [17](#)
- print.mblogit (mblogit), [7](#)
- print.mclogit (mclogit), [10](#)
- print.mmblogit (mblogit), [7](#)
- print.summary.mblogit (mblogit), [7](#)
- print.summary.mclogit (mclogit), [10](#)
- print.summary.mmblogit (mblogit), [7](#)
- print.summary.mmclogit (mclogit), [10](#)
- ranef.mmclogit (mclogit), [10](#)
- rebase, [19](#)
- residuals.mclogit (mclogit), [10](#)
- simulate, [20](#)
- simulate.mblogit (simulate.mclogit), [19](#)
- simulate.mclogit, [19](#)
- simulate.mmblogit (simulate.mclogit), [19](#)
- simulate.mmclogit (simulate.mclogit), [19](#)
- summary.lm, [12](#)
- summary.mblogit (mblogit), [7](#)
- summary.mclogit (mclogit), [10](#)
- summary.mmblogit (mblogit), [7](#)
- summary.mmclogit (mclogit), [10](#)
- Transport, [21](#)
- ucminf, [15](#)

update.default, [12](#)  
update.formula, [12](#)  
update.mclgit (mclgit), [10](#)  
  
vcov.mclgit (mclgit), [10](#)  
  
weights.mblogit (mblogit), [7](#)  
weights.mclgit (mclgit), [10](#)