Package ‘mclogit’

October 27, 2022

Type Package
Title Multinomial Logit Models, with or without Random Effects or Overdispersion
Version 0.9.6
Date 2022-10-27
Author Martin Elff
Maintainer Martin Elff <mclogit@elff.eu>
Description Provides estimators for multinomial logit models in their conditional logit and baseline logit variants, with or without random effects, with or without overdispersion. 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 memisc, methods
Suggests MASS, nnet
Enhances emmeans
LazyLoad Yes
BugReports https://github.com/melff/mclogit/issues
RoxygenNote 7.1.2
NeedsCompilation no
Repository CRAN
Date/Publication 2022-10-27 10:02:37 UTC
**R topics documented:**

- dispersion .................................................. 2
- electors ....................................................... 3
- getSummary-methods ....................................... 4
- mblogit ....................................................... 6
- mclogit ....................................................... 9
- mclogit.control ............................................. 13
- mclogit.fit .................................................. 14
- predict ....................................................... 15
- simulate.mclogit .......................................... 16
- Transport .................................................... 18

**Index**

<table>
<thead>
<tr>
<th>topic</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dispersion</td>
<td>19</td>
</tr>
</tbody>
</table>

---

**dispersion**  

*Overdispersion in Multinomial Logit Models*

**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 given as an argument to `summary()`.

**Usage**

```r
dispersion(object, method, ...)
```

## S3 method for class 'mclogit'

dispersion(object, method=NULL, ...)```

**Arguments**

- **object**  
an object that inherits class "mclogit". When passed to `dispersion()`, it should be the result of a call of `mclogit()` of `mblogit()`, *without* random effects.

- **method**  
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).

- **...**  
other arguments, ignored or passed to other methods.

**References**

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")
summary(house.mblogit)

phi.Afroz <- dispersion(house.mblogit, method = "Afroz")
summary(house.mblogit, dispersion = phi.Afroz)
summary(update(house.mblogit, dispersion = "Afroz"))

# 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 'from.table=TRUE':
house.mblogit.corrected <- mblogit(Sat ~ Infl + Type + Cont, weights = Freq,
data = housing, from.table = TRUE,
dispersion = "Afroz")

# Now the estimated dispersion parameter is no longer larger than 20,
# but just bit over 1.0.
summary(house.mblogit.corrected)

data(electors)

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
getSummary-methods

**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,
    econ.left.sq <- (econ.left-mean(econ.left))^2
  )))

## End(Not run)

---

**getSummary-methods**  ‘getSummary’ Methods

**Description**

*getSummary* methods for use by *mtable*

**Usage**

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

## 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 mblogit or mclogit
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",
...
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(mvoteint)    | 1.080***  |          |          |
# |                  | (0.000)   |          |          |
# | Var(mvoteint 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,
na.action =getOption("na.action"),
model = TRUE,
x = FALSE,
y = TRUE,
contrasts = NULL,
method = NULL,
estimator = c("ML", "REML"),
dispersion = FALSE,
start = NULL,
from.table = FALSE,
groups = NULL,
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 `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `glm` is called.
- **random**: an optional formula or list of formulas that specify the random-effects structure or NULL.
- **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 NULL or a numeric vector.
- **na.action**: a function which indicates what should happen when the data contain NAs. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The ‘factory-fresh’ default is `na.omit`. Another possible value is NULL, no action. Value `na.exclude` can be useful.
- **model**: a logical value indicating whether model frame 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 contrasts.arg of `model.matrix.default`.
- **method**: `NULL` or a character string, either "PQL" or "MQL", specifies the type of the quasilikelihood 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 `dispersion`.

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 `mblogit()`.

from.table  a logical value; do the data represent a contingency table, e.g. were created by applying `as.data.frame()` a the result of `table()` or `xtabs()`. This relevant only if the response is a factor. This argument should be set to `TRUE` if the data do come from a contingency table. Correctly setting `from.table=TRUE` in this case, will lead to efficiency gains in computing, but more importantly overdispersion will correctly be computed if present.

groups  an optional formula that specifies groups of observations relevant for the specification of overdispersed response counts.

control  a list of parameters for the fitting process. See `mclogit.control`

...  arguments to be passed to `mclogit.control` or `mmclogit.control`

Details

The function `mblogit` internally rearranges the data into a 'long' format and uses `mclogit.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


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.
mclogit

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)

---

mclogit

Conditional Logit Models and Mixed Conditional Logit Models

Description

mclogit 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

mclogit(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, control=if(length(random))

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

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formula</td>
<td>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. The left-hand side can either take the form <code>cbind(choice, set)</code> or (from version 0.9.1) `choice</td>
</tr>
<tr>
<td>data</td>
<td>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.</td>
</tr>
<tr>
<td>random</td>
<td>an optional formula or list of formulas that specify the random-effects structure or NULL.</td>
</tr>
<tr>
<td>weights</td>
<td>an optional vector of weights to be used in the fitting process. Should be NULL or a numeric vector.</td>
</tr>
<tr>
<td>offset</td>
<td>an optional model offset. Currently only supported for models without random effects.</td>
</tr>
<tr>
<td>subset</td>
<td>an optional vector specifying a subset of observations to be used in the fitting process.</td>
</tr>
<tr>
<td>na.action</td>
<td>a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The 'factory-fresh' default is na.omit. Another possible value is NULL, no action. Value na.exclude can be useful.</td>
</tr>
<tr>
<td>start</td>
<td>an optional numerical vector of starting values for the conditional logit parameters. If the model has random effects, the vector should have a &quot;VarCov&quot; attribute with starting values for the random effects (co-)variances. If the random effects model is estimated with the &quot;PQL&quot; method, the starting values matrix should also have a &quot;random.effects&quot; attribute, which should have the same structure as the &quot;random.effects&quot; component of an object returned by <code>mblogit()</code>.</td>
</tr>
<tr>
<td>model</td>
<td>a logical value indicating whether model frame should be included as a component of the returned value.</td>
</tr>
<tr>
<td>x, y</td>
<td>logical values indicating whether the response vector and model matrix used in the fitting process should be returned as components of the returned value.</td>
</tr>
<tr>
<td>contrasts</td>
<td>an optional list. See the contrasts.arg of <code>model.matrix.default</code>.</td>
</tr>
<tr>
<td>method</td>
<td>NULL or a character string, either &quot;PQL&quot; or &quot;MQL&quot;, specifies the type of the quasilinear approximation to be used if a random-effects model is to be estimated.</td>
</tr>
</tbody>
</table>
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 dispersion.

control a list of parameters for the fitting process. See mclogit.control

... arguments to be passed to mclogit.control or mmclogit.control

object an object that inherits class "mclogit". When passed to dispersion(), it should be the result of a call of mclogit() of mblogit(), without random effects.

formula. a changes to the model formula, see update.default and update.formula.

correlation logical; see summary.lm.

symbolic.cor logical; see summary.lm.

Value

mclogit returns an object of class "mclogit", 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 mclogit.

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 mclogit will complain about such model a misspecification explicitly.

References


See Also

Conditional logit models are also supported by \texttt{gmlnl}, \texttt{mlogit}, and \texttt{survival}. \texttt{survival} supports conditional logit models for binary panel data and case-control studies. \texttt{mlogit} and \texttt{gmlnl} 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 (indipendence from irrelevant alterantives) 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

```r
data(Transport)
summary(mclogit(
  cbind(resp, suburb) ~ distance + cost,
  data = Transport
))
# New syntactic sugar:
summary(mclogit(
  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(mclogit(
  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(mclogit(
  Freq | time.class ~ econ.left/class + welfare/class + auth/class,
  random = list(~ 1 | party, ~ econ.left + 0 | time),
  data = electors))

## End(Not run)
```
mclogit.control

Control Parameters for the Fitting Process

Description

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

Usage

mclogit.control(epsilon = 1e-08,
    maxit = 25, trace=TRUE)
mclogit.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)

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?

Value

A list.
mclogit.fit

Internal functions used for model fit.

Description

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

Usage

mclogit.fit(y, s, w, X,
       dispersion=FALSE,
       start = NULL, offset = NULL,
       control = mclogit.control())

mmclogit.fitPQLMQL(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 = mmclogit.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.
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 quasilikelihood 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 mclogit.control
predict

Value

A list with components describing the fitted model.

predict

Predicting responses or linear parts of the baseline-category and conditional logit models

Description

The predict() methods allow to obtain within-sample and out-of-sample predictions from models fitted with mclogit() 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

```r
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))
```

---

**simulate.mclogit**  
*Simulating responses from baseline-category and conditional logit models*

Description

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

Usage

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

**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 `link[stats]{simulate}`
- ... other arguments, ignored.

**Value**

The result of the `simulate` method for objects created by `mclogit` 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**

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

housing.long <- housing[rep(seq.int(nrow(housing)),housing$Freq),]
(housel.mblogit <- mblogit(Sat ~ Infl + Type + Cont,
                          data=housing.long))
sml <- simulate(housel.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]])

<table>
<thead>
<tr>
<th>Transport</th>
<th>Choice of Means of Transport</th>
</tr>
</thead>
</table>

**Description**

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

**Usage**

```r
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, 3
  - Transport, 18
* models
  - mclogit, 9
* regression
  - mclogit, 9
  - AIC.mclogit (mclogit), 9
  - anova.mclogit (mclogit), 9
  - as.data.frame, 7, 10, 17
  - BIC.mclogit (mclogit), 9
  - deviance.mclogit (mclogit), 9
  - dispersion, 2, 8, 11, 14
  - electors, 3
  - fitted.mblogit (mblogit), 6
  - fitted.mclogit (mclogit), 9
  - getSummary, 4
  - getSummary-methods, 4
  - getSummary.mblogit (getSummary-methods), 4
  - getSummary.mclogit (getSummary-methods), 4
  - getSummary.mmblogit (getSummary-methods), 4
  - getSummary.mmclogit (getSummary-methods), 4
  - glm, 8, 11
  - logLik.mclogit (mclogit), 9
  - mblogit, 5, 6, 17
  - mclogit, 5, 9, 17
  - mclogit.control, 8, 11, 13, 14
  - mclogit.fit, 8, 14
  - mmclogit.control (mclogit.control), 13
  - mmclogit.fitPQLMQL (mclogit.fit), 14
  - mtable, 4
  - multinom, 8
  - na.exclude, 7, 10
  - na.fail, 7, 10
  - na.omit, 7, 10
  - options, 7, 10
  - predict, 15
  - print.mblogit (mblogit), 6
  - print.mclogit (mclogit), 9
  - print.mmblogit (mmblogit), 6
  - print.summary.mblogit (mblogit), 6
  - print.summary.mclogit (mclogit), 9
  - print.summary.mmblogit (mmblogit), 6
  - print.summary.mmclogit (mclogit), 9
  - residuals.mclogit (mclogit), 9
  - simulate, 17
  - simulate.mblogit (simulate.mclogit), 16
  - simulate.mclogit, 16
  - simulate.mmblogit (simulate.mclogit), 16
  - simulate.mmclogit (simulate.mclogit), 16
  - summary.lm, 11
  - summary.mblogit (mblogit), 6
  - summary.mclogit (mclogit), 9
  - summary.mmblogit (mmblogit), 6
  - summary.mmclogit (mclogit), 9
  - Transport, 18
  - update.default, 11
  - update.formula, 11
  - update.mclogit (mclogit), 9
  - vcov.mclogit (mclogit), 9
  - weights.mblogit (mblogit), 6
  - weights.mclogit (mclogit), 9