Package 'mlmodels'

May 8, 2026

Title Maximum Likelihood Models and Tools for Estimation, Prediction, and Testing

Version 0.1.2

Description Provides a collection of maximum likelihood estimators with a consistent S3 interface. Supported models include Gaussian (linear and log-normal), logit, probit, Poisson, negative binomial (NB1 and NB2), gamma, and beta regression. A distinctive feature is flexible modeling of the scale parameter (variance, dispersion, precision, or shape) alongside the location/mean parameters. The package offers unified predict() methods, multiple variance-covariance estimators (observed information, outer product of gradients, robust/Huber-White, cluster-robust, bootstrap, jackknife), and a full suite of hypothesis tests (Wald, likelihood ratio, information matrix, Vuong, overdispersion, and goodness-of-fit). It is fully compatible with 'marginaleffects' for post-estimation analysis. Methods implemented include Cameron and Trivedi (1990) <doi:10.1016/0304-4076(90)90014-K>, for Poisson overdispersion testing, Manjon and Martinez (2014) <doi:10.1177/1536867X1401400406>, for goodness-of-fit testing of count data models, Vuong (1989) <doi:10.2307/1912557>, for non-nested likelihood ratio testing, and White (1982) <doi:10.2307/1912526>, for information matrix tests.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Imports cli, hardhat, insight, marginaleffects, MASS, matrixcalc, maxLik, rlang, tibble

Suggests boot, dplyr, e1071, ggplot2, knitr, patchwork, pkgdown, rmarkdown, testthat (>= 3.0.0), wooldridge

VignetteBuilder knitr

Config/testthat/edition 3

Config/pkgdown/logo man/figures/logo.png

URL https://alfisankipan.github.io/mlmodels/

BugReports https://github.com/alfisankipan/mlmodels/issues

Depends R (>= 4.1.0)

NeedsCompilation no

Author Alfonso Sanchez-Penalver [aut, cre] (ORCID:
<https://orcid.org/0000-0001-8491-4632>)

Maintainer Alfonso Sanchez-Penalver <oneiros_spain@yahoo.com>

Repository CRAN

Date/Publication 2026-05-08 15:40:02 UTC

Contents

`AIC.mlmodel` *Extract AIC from mlmodel objects*

Description

Extract AIC from mlmodel objects

Usage

```
## S3 method for class 'mlmodel'
AIC(object, ..., k = 2)

## S3 method for class 'summary.mlmodel'
AIC(object, ..., k = 2)
```

Arguments

`object`	An object of class `"mlmodel"` or `"summary.mlmodel"`.
`...`	Further arguments passed to methods.
`k`	Numeric. The penalty per parameter. Default is `k = 2` (standard AIC). See `stats::AIC()` for details.

Details

For `mlmodel` objects, AIC is computed as `-2 * logLik(object) + k * npar`.

For `summary.mlmodel` objects, the pre-computed AIC (with `k = 2`) is returned; the `k` argument is accepted for compatibility but ignored.

Value

A numeric value with the AIC.

`BIC.mlmodel` *Extract BIC from mlmodel objects*

Description

Extract BIC from mlmodel objects

Usage

```
## S3 method for class 'mlmodel'
BIC(object, ...)

## S3 method for class 'summary.mlmodel'
BIC(object, ...)
```

Arguments

`object`	An object of class `"mlmodel"` or `"summary.mlmodel"`.
`...`	Further arguments passed to methods.

Details

BIC is computed as `-2 * logLik(object) + log(nobs) * npar`.

Value

A numeric value with the BIC.

`coef.mlmodel` *Extract Model Coefficients*

Description

Extract Model Coefficients

Usage

```
## S3 method for class 'mlmodel'
coef(object, ...)
```

Arguments

`object`	An `mlmodel` object.
`...`	Currently not used.

Value

A named numeric vector of estimated coefficients.

`confint.mlmodel` *Confidence Intervals for mlmodel Coefficients*

Description

Confidence Intervals for mlmodel Coefficients

Usage

```
## S3 method for class 'mlmodel'
confint(
  object,
  parm,
  level = 0.95,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)
```

Arguments

`object`	An `mlmodel` object.
`parm`	A specification of which parameters are to be given confidence intervals (names or numeric indices). If missing, all parameters are used.
`level`	The confidence level required. Default is 0.95.
`vcov`	Optional user-supplied variance-covariance matrix.
`vcov.type`	Type of variance-covariance matrix to use. See vcov.mlmodel.
`cl_var`	Clustering variable (name as string or vector).
`repetitions`	Number of bootstrap replications when `vcov.type = "boot"`.
`seed`	Random seed for bootstrap/jackknife.
`progress`	Show progress bar? Default `FALSE`.
`...`	Further arguments passed to methods.

Details

Confidence intervals are constructed as $\hat{\beta} \pm z_{1-\alpha/2} \times SE(\hat{\beta})$, where the standard errors come from the requested variance-covariance matrix.

The function supports all variance types available in vcov.mlmodel, including robust, clustered, bootstrap, and jackknife estimators.

Value

Matrix with the confidence intervals for the requested parameters.

Examples

```
data(mroz)
mroz$incthou <- mroz$faminc / 1000

fit <- ml_lm(incthou ~ age + I(age^2) + huswage + educ + unem,
             data = mroz)

# Default 95% confidence intervals (using OIM)
confint(fit)

# 90% confidence intervals
confint(fit, level = 0.90)

# Confidence intervals for specific parameters
confint(fit, parm = c("value::educ", "value::huswage"))
confint(fit, parm = 4:5)                    # by position

# Using different variance types
confint(fit, vcov.type = "robust")

# Clustered confidence intervals
confint(fit, vcov.type = "robust", cl_var = "age")

# Using a pre-computed bootstrap variance matrix
v_boot <- vcov(fit, type = "boot", repetitions = 100, seed = 123)
confint(fit, vcov = v_boot)
```

---

docvis *U.S. Medical Expenditure Panel Survey*

Description

Cross sectional sample for 2003.

Usage

```
docvis
```

Format

A data frame with 3677 rows and 22 variables.

offer Binary: employer offers insurance

ssiratio Ratio of SSI income to total income

age Age in years

educyr Years of education

physician Number of visits to doctor

nonphysician Number of visits to health professional (not doctor)

medicaid Binary: has medicaid public insurance

private Binary: has private supplementary insurance

female Binary: female

phylim Binary: physical limitation

actlim Binary: activity limitation

income Income (in thousands)

totchr Number of chronic conditions

insured Same as `private`

age2 `age` squared

linc `log(income)`

bh Binary: black or hispanic

docvis Number of doctor visits

ldocvis `log(docvis)` if `docvis > 0`

ldocvisa `log(docvis + .01)`

one Constant term (1)

docbin Binary: `docvis > 0`

Source

https://www.stata-press.com/data/mus2.html `mus220mepsdocvis.dta`

References

Cameron, A. C., and P. K. Trivedi. 2022. *Microeconometrics Using Stata, Second Edition*. Volumes I and II. College Station, TX: Stata Press.

`find_predictors.mlmodel` *Extract the predictors used in the model (for insight/marginaleffects compatibility)*

Description

Extract the predictors used in the model (for insight/marginaleffects compatibility)

Usage

```
## S3 method for class 'mlmodel'
find_predictors(x, ...)
```

Arguments

x	An object of class "mlmodel"
...	Further arguments passed to methods

Value

A character vector with the names of the predictor variables.

---

find_variables.mlmodel *Extract the variables used in the model (for insight/marginaleffects compatibility)*

Description

Extract the variables used in the model (for insight/marginaleffects compatibility)

Usage

```
find_variables.mlmodel(x, ...)
```

Arguments

x	An object of class "mlmodel"
...	Further arguments passed to methods

Value

A list with components response (character) and conditional (character vector).

---

fitted.mlmodel *Extract Fitted Values from mlmodel*

Description

Extract Fitted Values from mlmodel

Usage

```
## S3 method for class 'mlmodel'
fitted(object, ...)

## S3 method for class 'values.mlmodel'
fitted(object, ...)
```

Arguments

object	An mlmodel object.
...	Further arguments passed to methods (currently ignored).

Value

A numeric vector of fitted values, aligned to the original data. Dropped observations (due to NAs or subset) return NA.

---

formula.mlmodel *Extract value formula from mlmodel objects*

Description

Extract value formula from mlmodel objects

Usage

```
## S3 method for class 'mlmodel'
formula(x, ...)
```

Arguments

x	An mlmodel object
...	Currently not implemented

Value

The formula of the value equation (object of class formula).

---

get_data.mlmodel *Extract data used to fit the model (for insight/marginaleffects compatibility)*

Description

Extract data used to fit the model (for insight/marginaleffects compatibility)

Usage

```
## S3 method for class 'mlmodel'
get_data(x, ...)

get_modeldata.mlmodel(x, ...)
```

Arguments

x	An object of class "mlmodel"
...	Further arguments (currently ignored)

Value

The original data frame used when fitting the model

GOFtest *Goodness-of-Fit Test for Count Models*

Description

Performs the Manjon and Martinez (2014) chi-squared goodness-of-fit test for count data models.

Usage

```
GOFtest(object, bins = 0:5)

## S3 method for class 'mlmodel'
GOFtest(object, bins = 0:5)
```

Arguments

object	An object of class "mlmodel.count" (typically from ml_poisson() or ml_negbin()).
bins	Integer vector. Defines the boundaries of the bins used to group counts. Default is 0:5.

Details

The test compares the observed frequencies with the expected frequencies predicted by the model across different count bins. It produces both a binned comparison table and an overall regression-based chi-squared test statistic.

A low p-value indicates that the model's predicted probabilities do not adequately match the observed count distribution (model misspecification).

Value

An object of class "GOFtest.mlmodel" with components:

model Description of the fitted model.

matrix A table with observed and predicted frequencies, proportions, absolute differences, and Pearson contributions per bin.

test A list containing teststat, df, and pval for the overall goodness-of-fit test.

Author(s)

Alfonso Sanchez-Penalver

References

Manjon, M., & Martinez, O. (2014). 'The chi-squared goodness-of-fit test for count-data models.' *The Stata Journal*, 14(4), 798-816. doi:10.1177/1536867X1401400406

Examples

```
# Poisson model
fit_pois <- ml_poisson(docvis ~ private + medicaid + age + I(age^2) +
                       educyr + actlim + totchr, data = docvis)

GOFtest(fit_pois, bins = 0:5)

# Negative binomial model
fit_nb2 <- ml_negbin(docvis ~ private + medicaid + age + I(age^2) +
                     educyr + actlim + totchr, data = docvis)

GOFtest(fit_nb2)
```

---

gradientObs *Gradient (Score) by Observation*

Description

Extract the per-observation gradients (scores) evaluated at the estimated parameters from an `mlmodel` object.

Usage

```
gradientObs(object)

## S3 method for class 'mlmodel'
gradientObs(object)
```

Arguments

`object`	An `mlmodel` object.

Details

These are the individual contributions to the score vector. They are mainly useful for advanced users who want to implement custom tests or diagnostics.

Value

A numeric matrix with one row per observation and one column per parameter. Each row contains the gradient of the log-likelihood for that observation.

---

hessianObs *Hessian by Observation*

Description

Extract the per-observation Hessian matrices evaluated at the estimated parameters from an `mlmodel` object.

Usage

```
hessianObs(object)

## S3 method for class 'mlmodel'
hessianObs(object)
```

Arguments

`object`	An `mlmodel` object.

Details

This is mainly intended for advanced use (e.g., custom diagnostics or information matrix tests). For most users, the functions `IMtest` or `vcov` are more convenient.

Value

A numeric matrix of dimension `(N*K) x K`, where `N` is the number of observations and `K` is the number of parameters. The Hessian for each observation is stacked vertically.

IMtest *Information Matrix Test for Model Misspecification*

Description

Performs the Information Matrix (IM) test for misspecification on models fitted with the `mlmodels` package.

Usage

```
IMtest(object, ...)

## S3 method for class 'mlmodel'
IMtest(object, method = "quad", repetitions = 999, seed = 1234L, ...)
```

Arguments

`object`	A fitted model object inheriting from `"mlmodel"`.
`...`	Further arguments passed to methods (currently not used).
`method`	Character string. Specifies the version of the test: • `"quad"` (default): Quadratic form of the Information Matrix test (most common). • `"opg"`: Outer Product of Gradients version (Chesher-Lancaster). • `"boot_quad"`: Analytical chi-square and p-value, plus bootstrap p-value for the quadratic form. • `"boot_opg"`: Analytical chi-square and p-value, plus bootstrap p-value for the OPG version.
`repetitions`	Integer. Number of bootstrap replications when using a bootstrap method. Default is 999.
`seed`	Integer. Random seed for reproducibility in bootstrap methods. If `NULL`, a random seed is generated.

Details

The Information Matrix test checks whether the model is correctly specified by testing the equality between the Hessian and the outer product of the gradient (information matrix equality). Rejection of the null hypothesis indicates model misspecification (e.g., incorrect functional form, heteroskedasticity not properly modeled, omitted variables, etc.).

Two main versions are implemented:

- **Quadratic form** (`"quad"`): Generally preferred for its better finite-sample properties.
- **OPG version** (`"opg"`): Chesher and Lancaster (1983) version.

Bootstrap versions (`"boot_quad"` and `"boot_opg"`) provide p-values based on the empirical distribution of the test statistic and are useful when asymptotic approximations may be unreliable.

Value

An object of class `"IMtest.mlmodel"` containing the analytical test statistic, degrees of freedom and p-value, plus the bootstrapped p-value (if a bootstrap method was selected).

Author(s)

Alfonso Sanchez-Penalver

References

Chesher, A. (1983). The information matrix test: Simplified calculation via a score test interpretation. Economics Letters, 13(1), 45-48.

Lancaster, T. (1984). The covariance matrix of the information matrix test. Econometrica, 52(4), 1051-1053.

White, H. (1982). Maximum likelihood estimation of misspecified models. Econometrica, 50(1), 1-25.

See Also

waldtest(), lrtest(), vuongtest()

Examples

```
# Linear model example
data(mroz)
mroz$incthou <- mroz$faminc / 1000

fit <- ml_lm(incthou ~ age + I(age^2) + huswage + educ + unem,
             data = mroz)

# Default quadratic form test
IMtest(fit)

# OPG version
IMtest(fit, method = "opg")

# Bootstrap p-value (quadratic form)
IMtest(fit, method = "boot_quad", repetitions = 50, seed = 123)

# Heteroskedastic model
fit_het <- ml_lm(incthou ~ age + I(age^2) + huswage + educ + unem,
                 scale = ~ educ, data = mroz)
IMtest(fit_het)
```

---

logLik.mlmodel *Extract Log-Likelihood from mlmodel objects*

Description

Extract Log-Likelihood from mlmodel objects

Usage

```
## S3 method for class 'mlmodel'
logLik(object, ...)

## S3 method for class 'summary.mlmodel'
logLik(object, ...)
```

Arguments

object	An object of class mlmodel or summary.mlmodel.
...	Additional arguments passed to methods.

Details

The returned object is of class `"logLik"` and has two important attributes:

- `nobs`: number of observations used in estimation.
- `df`: number of estimated parameters (usually called K), computed as `length(coef(object))`. This includes coefficients from both the location (mean/value) and scale equations when present.

Value

An object of class `"logLik"` with the log-likelihood value and the attributes `nobs` and `df`.

`loglikeObs` *Log-Likelihood by Observation*

Description

Extract the per-observation log-likelihood contributions from an `mlmodel` object.

Usage

```
loglikeObs(object)

## S3 method for class 'mlmodel'
loglikeObs(object)
```

Arguments

`object`	An `mlmodel` object.

Details

These individual contributions are useful for Vuong tests, robust variance estimation, or custom model diagnostics.

Value

A numeric vector of length `nobs(object)` containing the log-likelihood contribution of each observation.

`lrtest` *Likelihood Ratio Test for Nested mlmodel Objects*

Description

Performs a likelihood ratio test comparing two nested models fitted with the same estimator (e.g. `ml_lm`, `ml_logit`, `ml_negbin`, etc.).

Usage

```
lrtest(object_1, object_2, ...)

## S3 method for class 'mlmodel'
lrtest(object_1, object_2, ...)
```

Arguments

`object_1`	A fitted model object inheriting from `"mlmodel"`. Typically the restricted (smaller) model.
`object_2`	A fitted model object inheriting from `"mlmodel"`. Typically the unrestricted (larger) model. The order of `object_1` and `object_2` does not matter — the function automatically determines which is the restricted model.
`...`	Further arguments passed to methods (currently not used).

Details

The likelihood ratio test statistic is calculated as:

$$LR = 2 \times (\log L_{\text{unrestricted}} - \log L_{\text{restricted}})$$

Under the null hypothesis that the restricted model is correct, `LR` follows a χ^2 distribution with degrees of freedom equal to the difference in the number of parameters between the two models.

Important: The two models must be nested (the restricted model must be a special case of the unrestricted one) and fitted on exactly the same sample. The restricted model must have a lower (or equal) log-likelihood.

Value

An object of class `"lrtest.mlmodel"` with the test statistic, degrees of freedom, and p-value.

Author(s)

Alfonso Sanchez-Penalver

See Also

`waldtest()`, `IMtest()`, `vuongtest()`

Examples

```
# Linear model example
data(mroz)
mroz$incthou <- mroz$faminc / 1000

fit_small <- ml_lm(incthou ~ age + huswage, data = mroz)
fit_large <- ml_lm(incthou ~ age + I(age^2) + huswage + educ + unem,
                   data = mroz)

lrtest(fit_small, fit_large)

# You can also reverse the order — the function detects the restricted model
lrtest(fit_large, fit_small)
```

`ml_beta` *Fit Beta Model by Maximum Likelihood*

Description

Fit Beta Model by Maximum Likelihood

Usage

```
ml_beta(
  value,
  scale = NULL,
  weights = NULL,
  data,
  subset = NULL,
  noint_value = FALSE,
  noint_scale = FALSE,
  constraints = NULL,
  start = NULL,
  method = "NR",
  control = NULL,
  ...
)
```

Arguments

`value`	Formula for the conditional log(mean) equation.
`scale`	Formula for log(phi) equation (precision parameter - optional). If `NULL`, a homoskedastic (constant precision) model is fitted.
`weights`	Optional weights variable. It can be either the name of the variable in `data`, or a vector with the weights.
`data`	Data frame.

subset	Optional subset expression. Only observations for which this expression evaluates to TRUE are used in the estimation. This can be a logical vector or an expression (e.g. subset = age > 30).
noint_value	Logical. Should the value equation omit the intercept? Default is FALSE.
noint_scale	Logical. Should the scale equation omit the intercept? Default is FALSE.
constraints	Optional constraints on the parameters. Can be a character vector of string constraints, a named list of string constraints, or a raw maxLik constraints list. See **Details**.
start	Numeric vector of starting values for the coefficients. Required if constraints are being supplied. If supplied without constraints they will be ignored. See **Details**.
method	A string with the method used for optimization. See maxLik for options, and see **Details**.
control	A list of control parameters passed to maxLik. If NULL (default), a sensible set of options is chosen automatically depending on whether constraints are used. See maxControl.
...	Additional arguments passed to maxLik.

Details

Important: Do not use the usual R syntax to remove the intercept in the formula (- 1 or + 0) for the value or scale equations. Use the dedicated arguments noint_value and noint_scale instead.

Coefficient names in the fitted object use the prefixes value:: and scale:: to clearly identify to which equation each coefficient belongs to, and to avoid confusion when the same variable(s) appear(s) in both the value and scale equations.

Either inequality or equality linear constraints are accepted, but not both. A constraint cannot have a linear combination of more than two coefficients.

Important: When constraints are supplied, start cannot be NULL. You **must** provide initial values that yield a feasible log-likelihood. If no constraints are used, any supplied start is ignored.

When constraints are used, ml_lm automatically chooses the optimizer:

- Equality constraints => Nelder-Mead ("NM")
- Inequality constraints => BFGS ("BFGS")

In these cases your supplied method argument (if any) is ignored.

The Beta model is only defined for a strictly fractional response variable in the open interval (0, 1). Observations where the response is y <= 0 or y >= 1 are automatically dropped with a warning. If your data contains boundary values (0 or 1), consider using ml_logit() instead.

Value

An object of class ml_beta that extends mlmodel.

Author(s)

Alfonso Sanchez-Penalver

Examples

```
# Homoskedastic beta regression (fractional response)
data(pw401k)

# Beta regression requires 0 < y < 1.
# Observations at the boundaries (0 or 1) are automatically dropped.
fit_beta <- ml_beta(prate ~ mrate + I(mrate^2) + log(totemp) +
                    I(log(totemp)^2) + age + I(age^2) + sole,
                    data = pw401k,
                    subset = prate < 1)   # drop y = 1

summary(fit_beta, vcov.type = "robust")

# Heteroskedastic beta regression
fit_beta_het <- ml_beta(prate ~ mrate + I(mrate^2) + log(totemp) +
                        I(log(totemp)^2) + age + I(age^2) + sole,
                        scale = ~ totemp + sole,
                        data = pw401k,
                        subset = prate < 1)

summary(fit_beta_het, vcov.type = "robust")


# Note: All predictions (including those from predict(), fitted(), and
# residuals()) return values aligned to the original data, with NA
# for observations dropped due to subset or boundary values.

# Different predict types
head(predict(fit_beta, type = "response")$fit)   # Expected value E[y]
head(predict(fit_beta, type = "variance")$fit)   # Variance of y
head(predict(fit_beta, type = "phi")$fit)        # Precision parameter

# Fitted values and residuals
head(fitted(fit_beta))
head(residuals(fit_beta))
head(residuals(fit_beta, type = "pearson"))
```

`ml_gamma` *Fit Gamma Model by Maximum Likelihood*

Description

Fit Gamma Model by Maximum Likelihood

Usage

```
ml_gamma(
  value,
```

```
  scale = NULL,
  weights = NULL,
  data,
  subset = NULL,
  noint_value = FALSE,
  noint_scale = FALSE,
  constraints = NULL,
  start = NULL,
  method = "NR",
  control = NULL,
  ...
)
```

Arguments

value	Formula for the conditional log(mean) equation.
scale	Formula for log(nu) equation (shape parameter - optional). If NULL, a homoskedastic (constant shape) model is fitted.
weights	Optional weights variable. It can be either the name of the variable in data, or a vector with the weights.
data	Data frame.
subset	Optional subset expression. Only observations for which this expression evaluates to TRUE are used in the estimation. This can be a logical vector or an expression (e.g. subset = age > 30).
noint_value	Logical. Should the value equation omit the intercept? Default is FALSE.
noint_scale	Logical. Should the scale equation omit the intercept? Default is FALSE.
constraints	Optional constraints on the parameters. Can be a character vector of string constraints, a named list of string constraints, or a raw maxLik constraints list. See **Details**.
start	Numeric vector of starting values for the coefficients. Required if constraints are being supplied. If supplied without constraints they will be ignored. See **Details**.
method	A string with the method used for optimization. See maxLik for options, and see **Details**.
control	A list of control parameters passed to maxLik. If NULL (default), a sensible set of options is chosen automatically depending on whether constraints are used. See maxControl.
...	Additional arguments passed to maxLik.

Details

Important: Do not use the usual R syntax to remove the intercept in the formula (- 1 or + 0) for the value or scale equations. Use the dedicated arguments noint_value and noint_scale instead.

Coefficient names in the fitted object use the prefixes value:: and scale:: to clearly identify to which equation each coefficient belongs to, and to avoid confusion when the same variable(s) appear(s) in both the value and scale equations.

Either inequality or equality linear constraints are accepted, but not both. A constraint cannot have a linear combination of more than two coefficients.

Important: When `constraints` are supplied, `start` cannot be `NULL`. You **must** provide initial values that yield a feasible log-likelihood. If no constraints are used, any supplied `start` is ignored.

When constraints are used, `ml_lm` automatically chooses the optimizer:

- Equality constraints => Nelder-Mead (`"NM"`)
- Inequality constraints => BFGS (`"BFGS"`)

In these cases your supplied `method` argument (if any) is ignored.

The Gamma model requires a strictly positive response variable (`y > 0`). Observations where `y <= 0` are automatically dropped with a warning.

If your data contains zeros or non-positive values, consider using `ml_poisson()` or `ml_negbin()` instead, as they are frequently applied to continuous non-negative outcomes.

Value

An object of class `ml_gamma` that extends `mlmodel`.

Author(s)

Alfonso Sanchez-Penalver

Examples

```
# Homoskedastic gamma regression
data(mroz)
fit_gamma <- ml_gamma(faminc ~ hours + hushrs + age + educ,
                      data = mroz)

summary(fit_gamma, vcov.type = "robust")

# Heteroskedastic gamma regression
fit_gamma_het <- ml_gamma(faminc ~ hours + hushrs + age + educ,
                          scale = ~ kidslt6,
                          data = mroz)

summary(fit_gamma_het, vcov.type = "robust")

# Different predict types
head(predict(fit_gamma, type = "response")$fit)   # Expected value E[y]
head(predict(fit_gamma, type = "variance")$fit)   # Variance of y

# Fitted values and residuals
head(fitted(fit_gamma))
head(residuals(fit_gamma))
head(residuals(fit_gamma, type = "pearson"))

# Comparison with lognormal model (often very similar mean predictions)
fit_lognorm <- ml_lm(log(faminc) ~ hours + hushrs + age + educ,
```

```
                    data = mroz)

head(predict(fit_gamma, type = "response")$fit)
head(predict(fit_lognorm, type = "response")$fit)
```

`ml_lm` *Fit linear model by Maximum Likelihood*

Description

Fit linear model by Maximum Likelihood

Usage

```
ml_lm(
  value,
  scale = NULL,
  weights = NULL,
  data,
  subset = NULL,
  noint_value = FALSE,
  noint_scale = FALSE,
  constraints = NULL,
  start = NULL,
  method = "NR",
  control = NULL,
  ...
)
```

Arguments

`value`	Formula for the conditional mean (value) equation.
`scale`	Formula for log(sigma) (optional). If `NULL`, a homoskedastic model is fitted.
`weights`	Optional weights variable. It can be either the name of the variable in `data`, or a vector with the weights.
`data`	Data frame.
`subset`	Optional subset expression. Only observations for which this expression evaluates to `TRUE` are used in the estimation. This can be a logical vector or an expression (e.g. `subset = age > 30`).
`noint_value`	Logical. Should the value equation omit the intercept? Default is `FALSE`.
`noint_scale`	Logical. Should the scale equation omit the intercept? Default is `FALSE`.
`constraints`	Optional constraints on the parameters. Can be a character vector of string constraints, a named list of string constraints, or a raw maxLik constraints list. See **Details**.

start	Numeric vector of starting values for the coefficients. Required if constraints are being supplied. If supplied without constraints they will be ignored. See **Details**.
method	A string with the method used for optimization. See maxLik for options, and see **Details**.
control	A list of control parameters passed to maxLik. If NULL (default), a sensible set of options is chosen automatically depending on whether constraints are used. See maxControl.
...	Additional arguments passed to maxLik.

Details

Important: Do not use the usual R syntax to remove the intercept in the formula (`- 1` or `+ 0`) for the value or scale equations. Use the dedicated arguments `noint_value` and `noint_scale` instead.

Coefficient names in the fitted object use the prefixes `value::` and `scale::` to clearly identify to which equation each coefficient belongs to, and to avoid confusion when the same variable(s) appear(s) in both the value and scale equations.

Either inequality or equality linear constraints are accepted, but not both. A constraint cannot have a linear combination of more than two coefficients.

Important: When `constraints` are supplied, `start` cannot be `NULL`. You **must** provide initial values that yield a feasible log-likelihood. If no constraints are used, any supplied `start` is ignored.

When constraints are used, `ml_lm` automatically chooses the optimizer:

- Equality constraints => Nelder-Mead (`"NM"`)
- Inequality constraints => BFGS (`"BFGS"`)

In these cases your supplied `method` argument (if any) is ignored.

Value

An object of class `ml_lm` that extends `mlmodel`.

Author(s)

Alfonso Sanchez-Penalver

Examples

```
# Homoskedastic linear model
data(mroz)
fit_lin <- ml_lm(faminc ~ age + I(age^2) + huswage + educ + unem,
                 data = mroz)
summary(fit_lin, vcov.type = "robust")

# Heteroskedastic linear model
fit_het <- ml_lm(faminc ~ age + I(age^2) + huswage + educ + unem,
                 scale = ~ educ + exper,
                 data = mroz)
```

```
summary(fit_het, vcov.type = "robust")

# Lognormal (log-linear) model
fit_log <- ml_lm(log(faminc) ~ age + I(age^2) + huswage + educ + unem,
                 data = mroz)
summary(fit_log, vcov.type = "robust")

# Different predict types
head(predict(fit_log, type = "response")$fit)      # Expected value E[y]
head(predict(fit_log, type = "median")$fit)        # Median of y
head(predict(fit_log, type = "variance_y")$fit)    # Variance of y
head(predict(fit_log, type = "var")$fit)           # Variance of log(y)

# Fitted values and residuals
head(fitted(fit_lin))
head(residuals(fit_lin))
head(residuals(fit_lin, type = "pearson"))
```

`ml_logit` *Fit Binary Logit Model by Maximum Likelihood*

Description

Fit Binary Logit Model by Maximum Likelihood

Usage

```
ml_logit(
  value,
  scale = NULL,
  weights = NULL,
  data,
  subset = NULL,
  noint_value = FALSE,
  constraints = NULL,
  start = NULL,
  method = "NR",
  control = NULL,
  ...
)
```

Arguments

`value`	Two-sided formula for the probability equation.
`scale`	Optional one-sided formula for heteroskedasticity.
`weights`	Optional weights variable. It can be either the name of the variable in `data`, or a vector with the weights.

`data`	Data frame.
`subset`	Optional subset expression. Only observations for which this expression evaluates to `TRUE` are used in the estimation. This can be a logical vector or an expression (e.g. `subset = age > 30`).
`noint_value`	Logical. Should the value equation omit the intercept? Default is `FALSE`.
`constraints`	Optional constraints on the parameters. Can be a character vector of string constraints, a named list of string constraints, or a raw maxLik constraints list. See **Details**.
`start`	Numeric vector of starting values for the coefficients. Required if constraints are being supplied. If supplied without constraints they will be ignored. See **Details**.
`method`	A string with the method used for optimization. See maxLik for options, and see **Details**.
`control`	A list of control parameters passed to maxLik. If `NULL` (default), a sensible set of options is chosen automatically depending on whether constraints are used. See maxControl.
`...`	Additional arguments passed to maxLik.

Details

Important: Do not use the usual R syntax to remove the intercept in the formula (`- 1` or `+ 0`) for the value equation. Use the dedicated argument `noint_value` instead. For the scale equation (if modeling heteroskedasticity), the formula must contain only the predictors (right-hand side).

`ml_logit()` handles both strictly binary (`0/1`), and fractional response (`0 <= y <= 1`) outcomes. When using fractional responses, it is recommended to use robust standard errors (`vcov.type = "robust"`).

Coefficient names in the fitted object use the prefixes `value::` and `scale::` (when heteroskedasticity is modeled) to clearly identify which equation each coefficient belongs to.

Either inequality or equality linear constraints are accepted, but not both. A constraint cannot have a linear combination of more than two coefficients.

Important: When `constraints` are supplied, `start` cannot be `NULL`. You **must** provide initial values that yield a feasible log-likelihood. If no constraints are used, any supplied `start` is ignored.

When constraints are used, `ml_logit` automatically chooses `method`:

- Equality constraints => Nelder-Mead (`"NM"`)
- Inequality constraints => BFGS (`"BFGS"`)

In these cases your supplied `method` argument (if any) is ignored.

Value

An object of class `ml_logit` that extends `mlmodel`.

Author(s)

Alfonso Sanchez-Penalver

See Also

ml_probit ml_beta

Examples

```
# Homoskedastic binary logit model
data(smoke)
smoke$smokes <- smoke$cigs > 0

fit_logit <- ml_logit(smokes ~ cigpric + income + age,
                      data = smoke)

summary(fit_logit, vcov.type = "robust")

# Heteroskedastic binary logit model
fit_logit_het <- ml_logit(smokes ~ cigpric + income + age,
                          scale = ~ educ,
                          data = smoke)

summary(fit_logit_het, vcov.type = "robust")

# Different predict types
head(predict(fit_logit, type = "response")$fit)   # Predicted probability
head(predict(fit_logit, type = "link")$fit)       # Linear predictor (log-odds)

# Fitted values and residuals
head(fitted(fit_logit))
head(residuals(fit_logit))
head(residuals(fit_logit, type = "pearson"))
```

---

ml_negbin *Fit negative binomial models by Maximum Likelihood*

Description

Fit negative binomial models by Maximum Likelihood

Usage

```
ml_negbin(
  value,
  scale = NULL,
  dispersion = "NB2",
  weights = NULL,
  data,
  subset = NULL,
  noint_value = FALSE,
  noint_scale = FALSE,
```

```
  constraints = NULL,
  start = NULL,
  method = "NR",
  control = NULL,
  ...
)
```

Arguments

value	Formula for the conditional mean (value) equation.
scale	Formula for the dispersion parameter log(alpha) (optional). If NULL, a constant alpha is used for all observations.
dispersion	Either NB1 (proportional to mean variance), or NB2 (quadratic to mean variance). Defaults to NB2.
weights	Optional weights variable. It can be either the name of the variable in data, or a vector with the weights.
data	Data frame.
subset	Optional subset expression. Only observations for which this expression evaluates to TRUE are used in the estimation. This can be a logical vector or an expression (e.g. subset = age > 30).
noint_value	Logical. Should the value equation omit the intercept? Default is FALSE.
noint_scale	Logical. Should the scale equation omit the intercept? Default is FALSE.
constraints	Optional constraints on the parameters. Can be a character vector of string constraints, a named list of string constraints, or a raw maxLik constraints list. See **Details**.
start	Numeric vector of starting values for the coefficients. Required if constraints are being supplied. If supplied without constraints they will be ignored. See **Details**.
method	A string with the method used for optimization. See maxLik for options, and see **Details**.
control	A list of control parameters passed to maxLik. If NULL (default), a sensible set of options is chosen automatically depending on whether constraints are used. See maxControl.
...	Additional arguments passed to maxLik.

Details

Important: Do not use the usual R syntax to remove the intercept in the formula (- 1 or + 0) for the value or scale equations. Use the dedicated arguments noint_value and noint_scale instead.

Coefficient names in the fitted object use the prefixes value:: and scale:: to clearly identify to which equation each coefficient belongs to, and to avoid confusion when the same variable(s) appear(s) in both the value and scale equations.

Either inequality or equality linear constraints are accepted, but not both. A constraint cannot have a linear combination of more than two coefficients.

Important: When `constraints` are supplied, `start` cannot be `NULL`. You **must** provide initial values that yield a feasible log-likelihood. If no constraints are used, any supplied `start` is ignored.

When constraints are used, `ml_lm` automatically chooses the optimizer:

- Equality constraints => Nelder-Mead (`"NM"`)
- Inequality constraints => BFGS (`"BFGS"`)

In these cases your supplied `method` argument (if any) is ignored.

Value

An object of class `ml_negbin` that extends `mlmodel.count` and `mlmodel`.

Author(s)

Alfonso Sanchez-Penalver

See Also

ml_poisson

Examples

```
# Homoskedastic NB2 model (default dispersion)
data(docvis)
fit_nb2 <- ml_negbin(docvis ~ age + educyr + totchr,
                     data = docvis)

summary(fit_nb2, vcov.type = "robust")

# Homoskedastic NB1 model
fit_nb1 <- ml_negbin(docvis ~ age + educyr + totchr,
                     dispersion = "NB1",
                     data = docvis)

summary(fit_nb1, vcov.type = "robust")

# Heteroskedastic NB2 model
fit_nb2_het <- ml_negbin(docvis ~ age + educyr + totchr,
                         scale = ~ female + bh,
                         data = docvis)

summary(fit_nb2_het, vcov.type = "robust")

# Different predict types
head(predict(fit_nb2, type = "response")$fit)   # Expected count
head(predict(fit_nb2, type = "var")$fit)       # Variance
head(predict(fit_nb2, type = "alpha")$fit)     # Dispersion parameter

# Fitted values and residuals
head(fitted(fit_nb2))
head(residuals(fit_nb2))
```

```
head(residuals(fit_nb2, type = "pearson"))
```

`ml_poisson` *Fit Poisson model by Maximum Likelihood*

Description

Fit Poisson model by Maximum Likelihood

Usage

```
ml_poisson(
  value,
  weights = NULL,
  data,
  subset = NULL,
  noint_value = FALSE,
  constraints = NULL,
  start = NULL,
  method = "NR",
  control = NULL,
  ...
)
```

Arguments

`value`	Formula for the conditional mean (value) equation.
`weights`	Optional weights variable. It can be either the name of the variable in `data`, or a vector with the weights.
`data`	Data frame.
`subset`	Optional subset expression. Only observations for which this expression evaluates to `TRUE` are used in the estimation. This can be a logical vector or an expression (e.g. `subset = age > 30`).
`noint_value`	Logical. Should the value equation omit the intercept? Default is `FALSE`.
`constraints`	Optional constraints on the parameters. Can be a character vector of string constraints, a named list of string constraints, or a raw maxLik constraints list. See **Details**.
`start`	Numeric vector of starting values for the coefficients. Required if constraints are being supplied. If supplied without constraints they will be ignored. See **Details**.
`method`	A string with the method used for optimization. See maxLik for options, and see **Details**.
`control`	A list of control parameters passed to maxLik. If `NULL` (default), a sensible set of options is chosen automatically depending on whether constraints are used. See maxControl.
`...`	Additional arguments passed to maxLik.

Details

Important: Do not use the usual R syntax to remove the intercept in the formula (`- 1` or `+ 0`) for the value equation. Use the dedicated argument `noint_value` instead.

Coefficient names in the fitted object use the prefixes `value::`. This is for consistency with other `mlmodel` estimators that model the scale (dispersion) as well.

Either inequality or equality linear constraints are accepted, but not both. A constraint cannot have a linear combination of more than two coefficients.

Important: When `constraints` are supplied, `start` cannot be `NULL`. You **must** provide initial values that yield a feasible log-likelihood. If no constraints are used, any supplied `start` is ignored.

When constraints are used, `ml_lm` automatically chooses the optimizer:

- Equality constraints => Nelder-Mead (`"NM"`)
- Inequality constraints => BFGS (`"BFGS"`)

In these cases your supplied `method` argument (if any) is ignored.

The Poisson model assumes equidispersion (mean = variance). When the data show overdispersion (as is common), consider using ml_negbin instead.

Value

An object of class `ml_poisson` that extends `mlmodel.count` and `mlmodel`.

Author(s)

Alfonso Sanchez-Penalver

See Also

ml_negbin

Examples

```
# Poisson model
data(docvis)
fit_pois <- ml_poisson(docvis ~ age + educyr + totchr,
                       data = docvis)

summary(fit_pois, vcov.type = "robust")

# Different predict types
head(predict(fit_pois, type = "response")$fit)   # Expected count
head(predict(fit_pois, type = "P(2,)")$fit)      # Probability of at least 2
head(predict(fit_pois, type = "P(3)")$fit)       # Probability of exactly 3

# Fitted values and residuals
head(fitted(fit_pois))
head(residuals(fit_pois))
head(residuals(fit_pois, type = "pearson"))
```

`ml_probit` *Fit Binary Probit Model by Maximum Likelihood*

Description

Fit Binary Probit Model by Maximum Likelihood

Usage

```
ml_probit(
  value,
  scale = NULL,
  weights = NULL,
  data,
  subset = NULL,
  noint_value = FALSE,
  constraints = NULL,
  start = NULL,
  method = "NR",
  control = NULL,
  ...
)
```

Arguments

`value`	Two-sided formula for the probability equation.
`scale`	Optional one-sided formula for heteroskedasticity.
`weights`	Optional weights variable. It can be either the name of the variable in `data`, or a vector with the weights.
`data`	Data frame.
`subset`	Optional subset expression. Only observations for which this expression evaluates to `TRUE` are used in the estimation. This can be a logical vector or an expression (e.g. `subset = age > 30`).
`noint_value`	Logical. Should the value equation omit the intercept? Default is `FALSE`.
`constraints`	Optional constraints on the parameters. Can be a character vector of string constraints, a named list of string constraints, or a raw maxLik constraints list. See **Details**.
`start`	Numeric vector of starting values for the coefficients. Required if constraints are being supplied. If supplied without constraints they will be ignored. See **Details**.
`method`	A string with the method used for optimization. See maxLik for options, and see **Details**.
`control`	A list of control parameters passed to maxLik. If `NULL` (default), a sensible set of options is chosen automatically depending on whether constraints are used. See maxControl.
`...`	Additional arguments passed to maxLik.

Details

Important: Do not use the usual R syntax to remove the intercept in the formula (`- 1` or `+ 0`) for the value equation. Use the dedicated argument `noint_value` instead. For the scale equation (if modeling heteroskedasticity), the formula must contain only the predictors (right-hand side).

The dependent variable must be binary (0/1 or TRUE/FALSE).

Coefficient names in the fitted object use the prefixes `value::` and `scale::` (when heteroskedasticity is modeled) to clearly identify which equation each coefficient belongs to.

`ml_probit()` handles both strictly binary (`0/1`), and fractional response (`0 <= y <= 1`) outcomes. When using fractional responses, it is recommended to use robust standard errors (`vcov.type = "robust"`).

Either inequality or equality linear constraints are accepted, but not both. A constraint cannot have a linear combination of more than two coefficients.

Important: When `constraints` are supplied, `start` cannot be `NULL`. You **must** provide initial values that yield a feasible log-likelihood. If no constraints are used, any supplied `start` is ignored.

When constraints are used, `ml_probit` automatically chooses `method`:

- Equality constraints => Nelder-Mead (`"NM"`)
- Inequality constraints => BFGS (`"BFGS"`)

In these cases your supplied `method` argument (if any) is ignored.

Value

An object of class `ml_probit` that extends `mlmodel`.

Author(s)

Alfonso Sanchez-Penalver

See Also

ml_logit ml_beta

Examples

```
# Probit model (strictly binary outcome)
data(smoke)
smoke$smokes <- smoke$cigs > 0

fit_probit <- ml_probit(smokes ~ cigpric + income + age,
                        data = smoke)

summary(fit_probit, vcov.type = "robust")

# Heteroskedastic probit model
fit_probit_het <- ml_probit(smokes ~ cigpric + income + age,
                            scale = ~ educ,
                            data = smoke)
```

```
summary(fit_probit_het, vcov.type = "robust")

# Different predict types
head(predict(fit_probit, type = "response")$fit)   # Probability of success
head(predict(fit_probit, type = "prob0")$fit)      # Probability of failure
head(predict(fit_probit, type = "link")$fit)       # Linear predictor (z)

# Fitted values and residuals
head(fitted(fit_probit))
head(residuals(fit_probit))
head(residuals(fit_probit, type = "pearson"))
```

---

mroz *University of Michigan Panel Study of Income Dynamics (PSID)*

Description

Sample of 753 between the ages of 30 and 60 in 1975 (PSID interview year 1976)

Usage

```
mroz
```

Format

A data frame with 753 observations on 22 variables.

inlf Binary: in labor force in 1975

hours Hours worked in 1975

kidslt6 Number of children less than 6 years old

kidsge6 Number of children 6 years old or older

age Woman's age in years

educ Woman's years of education

wage Woman's estimated average hourly earnings

repwage Representative wage in 1976 (interview year)

hushrs Hours worked by husband in 1975

husage Husband's age in years

huseduc Husband's years of education

huswage Husband's estimated average hourly earnings

faminc Total family income

mtr Woman's Federal marginal income tax rate

motheduc Mother's years of education

fatheduc Father's years of education

unem Unemployment rate in county of residence

city Binary: lives in Standard Metropolitan Statistical Area (SMSA)

exper Years of labor market experience

nwifeinc Income from other sources than the wife's labor, in thousands

lwage `log(wage)`

expersq `exper^2`

Source

https://cran.r-project.org/package=wooldridge

References

Wooldridge, J. M. (2020). *Introductory Econometrics: A Modern Approach*. 7th Edition. Boston, MA: Cengage Learning.

Mroz, T. A. (1987). "The Sensitivity of an Empirical Model of Married Women's Hours of Work to Economic and Statistical Assumptions." *Econometrica* 55, 765-799.

`nobs.mlmodel` *Extract the Number of Observations from an mlmodel*

Description

Extract the Number of Observations from an mlmodel

Usage

```
## S3 method for class 'mlmodel'
nobs(object, ...)
```

Arguments

`object`	An object of class `"mlmodel"`.
`...`	Further arguments passed to methods (currently not used).

Value

An integer giving the number of observations used in the estimation (after removing missing values and applying any `subset`).

null-default *Null default operator*

Description

Provides a convenient infix operator for replacing NULL values. `x \%||\% y` is equivalent to `if (is.null(x)) y else x`.

Usage

```
x %||% y
```

Arguments

x, y	Any R objects.

Value

The value of x if it is not NULL, otherwise the value of y.

Examples

```
NULL %||% "fallback"
list(a = 1) %||% list(b = 2)
```

OVDtest *Overdispersion Tests for Count Models*

Description

Performs Cameron and Trivedi's (1990) regression-based tests for overdispersion in count models.

Usage

```
OVDtest(object)
```

Arguments

object	An object of class "mlmodel.count" (fitted with ml_poisson() or ml_negbin()).

Details

These tests evaluate the null hypothesis that the conditional variance equals the conditional mean (the Poisson assumption). Rejection indicates overdispersion and suggests that a negative binomial model may be more appropriate.

When the input object is not a Poisson model, a Poisson regression is fitted internally using the value (mean) equation specification from object in order to perform the test.

Value

A list containing the results of the tests against NB1 and NB2 alternatives, with coefficient estimates, t-statistics, and p-values.

Author(s)

Alfonso Sanchez-Penalver

References

Cameron, A. C., & Trivedi, P. K. (1990). 'Regression-based tests for overdispersion in the Poisson model.' *Journal of Econometrics*, 46(3), 347-364. doi:10.1016/03044076(90)90014K

Cameron, A. C., & Trivedi, P. K. (2013). *Regression Analysis of Count Data* (2nd ed.). Cambridge University Press. doi:10.1017/CBO9781139013567

See Also

`IMtest()`, `GOFtest()`

Examples

```
# Poisson model
fit_pois <- ml_poisson(docvis ~ private + medicaid + age + I(age^2) +
                       educyr + actlim + totchr, data = docvis)
OVDtest(fit_pois)

# Negative binomial model (the test still fits a Poisson internally using
# only the value equation, so results are identical)
fit_nb2 <- ml_negbin(docvis ~ private + medicaid + age + I(age^2) +
                     educyr + actlim + totchr, data = docvis)
OVDtest(fit_nb2)
```

---

predict.ml_beta *Predictions for mlmodel models*

Description

Methods for computing predictions from models fitted with the `mlmodels` package.

Usage

```
## S3 method for class 'ml_beta'
predict(
  object,
  newdata = NULL,
  type = "response",
  se.fit = FALSE,
```

```
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)

## S3 method for class 'ml_gamma'
predict(
  object,
  newdata = NULL,
  type = "response",
  se.fit = FALSE,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)

## S3 method for class 'ml_lm'
predict(
  object,
  newdata = NULL,
  type = "response",
  se.fit = FALSE,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)

## S3 method for class 'ml_logit'
predict(
  object,
  newdata = NULL,
  type = "response",
  se.fit = FALSE,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
```

```
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)

## S3 method for class 'mlmodel'
predict(object, ...)

## S3 method for class 'ml_negbin'
predict(
  object,
  newdata = NULL,
  type = "response",
  se.fit = FALSE,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)

## S3 method for class 'ml_poisson'
predict(
  object,
  newdata = NULL,
  type = "response",
  se.fit = FALSE,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)

## S3 method for class 'ml_probit'
predict(
  object,
  newdata = NULL,
  type = "response",
  se.fit = FALSE,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
```

```
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)
```

Arguments

object	An object from an estimation with one of our models.
newdata	Optional data frame for out-of-sample predictions.
type	Character string indicating what to predict. See **Details**.
se.fit	Logical. If TRUE, also return standard errors (delta method).
vcov	Optional user-supplied variance-covariance matrix.
vcov.type	Type of variance-covariance matrix. See vcov.
cl_var	Clustering variable (name or vector).
repetitions	Number of bootstrap replications when vcov.type = "boot".
seed	Random seed for bootstrapping, for reproducibility.
progress	Logical. Show bootstrap/jackknife progress bar? Default is FALSE in higher-level functions.
...	Additional arguments passed to methods.

Details

ml_beta prediction types:

The type argument controls what quantity is returned.

Type	Description	Notes
"link"	Linear mean predictor (xb)	logit-mean
"response"	Expected proportion (outcome)	Default
"mean"	Alias for "response"	-
"fitted"	Alias for "response"	-
"odds"	Odds ratio	exp(xb)
"zd"	Linear precision predictor	log-phi
"phi"	Dispersion parameter	-
"shape1"	Shape parameter of the beta distribution	mu * phi
"shape2"	Shape parameter of the beta distribution	(1 - mu) * phi
"mode"	Mode prediction (See below)	(shape1 - 1) / (shape1 + shape2 - 2)
"variance"	Variance of the outcome variable	mu * (1 - mu) / (1 + phi)
"var"	Alias for "variance"	-
"sigma"	Standard deviation of outcome variable	sqrt("variance")
"sd"	Alias for "sigma"	-

When se.fit = TRUE, standard errors are computed using the delta method for all supported types.

Mode Indeterminations

The mode is only defined if shape1 > 1 **and** shape2 > 1 **and** shape1 + shape2 != 2. If these conditions are not met the prediction and standard error will be NA.

ml_gamma prediction types:

The `type` argument controls what quantity is returned.

Type	Description	Notes
`"link"`	Linear mean predictor (xb)	log-mean
`"response"`	Expected outcome	Default
`"mean"`	Alias for `"response"`	-
`"fitted"`	Alias for `"response"`	-
`"zd"`	Linear shape predictor	log-nu
`"nu"`	Shape parameter	-
`"variance"`	Variance of the outcome variable	-
`"var"`	Alias for `"variance"`	-
`"sigma"`	Standard deviation of outcome variable	sqrt(`"variance"`)
`"sd"`	Alias for `"sigma"`	-

When `se.fit = TRUE`, standard errors are computed using the delta method for all supported types.

ml_lm prediction types:

The `type` argument controls what quantity is returned. Behavior differs depending on whether the outcome was modeled in logs (`log(y)`).

Type	Normal (linear) case	Lognormal case (`log(y)`)	Notes
`link`	Linear predictor for scale (zd)	Linear predictor on log scale (mu-log)	Scale equation
`fitted`	xb (mean predictor)	xb (original log-scale predictor)	Mean equation
`response`, `mean`, `mu`	xb (E[y])	E[y] = exp(mu-log + sigma^2/2) - shift	Proper expecte
`median`	xb (same as mean)	exp(mu-log) - shift	Median of y
`sigma`, `sd`	sd of y	sd of `log(y)`	On log scale
`sigma_y`, `sd_y`	same as `sigma`	sd of y	Only meaning
`variance`, `var`	sigma^2	sigma^2 (variance of `log(y)`)	On log scale
`variance_y`, `var_y`	same as `variance`	Var(y) = exp(2 mu-log + sigma^2)(exp(sigma^2) - 1)	Only meaning
`zd`	Linear predictor for scale (zd)	Linear predictor for scale (zd)	Alias for `link`

When the outcome is log-transformed, `response` (or `mean`) returns the correct lognormal expected value on the original scale of y. The `median` is the simple exponential back-transform.

ml_logit prediction types:

The `type` argument controls what quantity is returned. Behavior differs depending on whether the model is homoskedastic or heteroskedastic.

Type	Homoskedastic case	Heteroskedastic case	Notes
`"xb"`	Linear predictor xb	Linear predictor xb	Linear predictor for value
`"response"`	P(y=1 | x)	P(y=1 | x)	Prob. of success (default)
`"prob"`	Alias for `"response"`	Alias for `"response"`	-
`"fitted"`	Alias for `"response"`	Alias for `"response"`	-
`"prob0"`	P(y=0 | x)	P(y=0 | x)	Prob. of failure
`"link"`	Linear predictor xb	xb / exp(zd)	Log-odds

"odds"	Odds = exp(xb)	Odds = exp(xb / exp(zd))	-
"sigma"	1 (constant)	Std. Deviation: exp(zd)	Only available if heteroskedastic
"variance"	1 (constant)	Variance: exp(2*zd)	Only available if heteroskedastic
"zd"	0 (constant)	Linear predictor zd	Linear predictor for scale

In binary logit models, the **overall scale** of the latent error term is not identified and is normalized to 1. In the homoskedastic case there is no scale equation, so sigma is fixed at 1. In the heteroskedastic case, the scale equation has no intercept. Therefore, the predicted "sigma" and "variance" represent **individual-level deviations** from the normalized overall scale, not the absolute standard deviation or variance.

When se.fit = TRUE, standard errors are computed using the delta method. Standard errors are not available (and will return NA) for "sigma", "variance", and "zd" in homoskedastic models.

ml_negbin prediction types:

The type argument controls what quantity is returned. In addition to standard types, Negative Binomial models support flexible probability requests using the P(...) syntax.

Type	Description	Notes
"link"	Linear mean predictor (xb)	log-mean
"response"	Expected count (mu = exp(xb))	Default
"mean"	Alias for "response"	-
"fitted"	Alias for "response"	-
"zd"	Linear dispersion predictor	log-alpha
"alpha"	Dispersion parameter	-
"variance"	Variance of the outcome variable	-
"var"	Alias for "variance"	-
"sigma"	Standard deviation of outcome variable	sqrt("variance")
"sd"	Alias for "sigma"	-
P(k)	P(Y = k)	Exact probability, k integer >= 0
P(,k)	P(Y <= k)	Cumulative (lower tail)
P(k,)	P(Y >= k)	Survival (upper tail)
P(a,b)	P(a <= Y <= b)	Interval probability, a <= b, a >= 0

When se.fit = TRUE, standard errors are computed using the delta method for all supported types.

ml_poisson prediction types:

The type argument controls what quantity is returned. In addition to standard types, Poisson models support flexible probability requests using the P(...) syntax.

Type	Description	Notes
"link"	Linear predictor (xb)	log-mean
"response"	Expected count (mu = exp(xb))	Default
"mean"	Alias for "response"	-
"mu"	Alias for "response"	-
"fitted"	Alias for "response"	-
P(k)	P(Y = k)	Exact probability, k integer >= 0

P(,k)	P(Y <= k)	Cumulative (lower tail)
P(k,)	P(Y >= k)	Survival (upper tail)
P(a,b)	P(a <= Y <= b)	Interval probability, a <= b, a >= 0

When se.fit = TRUE, standard errors are computed using the delta method for all supported types.

ml_probit prediction types:

The type argument controls what quantity is returned. Behavior differs depending on whether the model is homoskedastic or heteroskedastic.

Type	Homoskedastic case	Heteroskedastic case	Notes
"xb"	Linear predictor xb	Linear predictor xb	Linear predictor for value
"response"	P(y=1 | x)	P(y=1 | x)	Prob. of success (default)
"prob"	Alias for "response"	Alias for "response"	-
"fitted"	Alias for "response"	Alias for "response"	-
"prob0"	P(y=0 | x)	P(y=0 | x)	Prob. of failure
"link"	Linear predictor xb	xb / exp(zd)	Probit index
"odds"	Odds = prob / prob0	Odds = prob / prob0.	-
"sigma"	1 (constant)	Std. Deviation: exp(zd)	Only available if heteroskedastic
"variance"	1 (constant)	Variance: exp(2*zd)	Only available if heteroskedastic
"zd"	0 (constant)	Linear predictor zd	Linear predictor for scale

In binary probit models, the **overall scale** of the latent error term is not identified and is normalized to 1. In the homoskedastic case there is no scale equation, so sigma is fixed at 1. In the heteroskedastic case, the scale equation has no intercept. Therefore, the predicted "sigma" and "variance" represent **individual-level deviations** from the normalized overall scale, not the absolute standard deviation or variance.

The "link" type returns the value on the **probit scale**, which is the inverse of the standard normal cumulative distribution function (p = Phi^(-1)(p)). This is the linear prediction (p = xb) ih homoskedastic models, and the standardized linear predictor (p = xb / sigma) in heteroskedastic models.

When se.fit = TRUE, standard errors are computed using the delta method. Standard errors are not available (and will return NA) for "sigma", "variance", and "zd" in homoskedastic models.

Value

An object that inherits from predict.mlmodel and has two elements:

fit Vector with the predictions.

se.fit If se.fit is TRUE a vector with the delta-method standard errors, using analytical gradients. If se.fit is FALSE, it is set to NULL.

Author(s)

Alfonso Sanchez-Penalver

Examples

```
# Basic usage and different predict types
data(docvis)
fit_pois <- ml_poisson(docvis ~ age + educyr + totchr, data = docvis)

head(predict(fit_pois, type = "response")$fit)     # Expected count
head(predict(fit_pois, type = "P(3)")$fit)         # Prob of exactly 3

# Prediction at the mean (typical case)
typical <- data.frame(age = mean(docvis$age),
                      educyr = mean(docvis$educyr),
                      totchr = mean(docvis$totchr))
predict(fit_pois, newdata = typical, type = "response")

# In-sample vs full-data prediction with subset / boundary dropping
data(pw401k)
fit_beta <- ml_beta(prate ~ mrate + I(mrate^2) + log(totemp) +
                    I(log(totemp)^2) + age + I(age^2) + sole,
                    data = pw401k,
                    subset = prate < 1)

# In-sample prediction (NAs for dropped observations)
head(predict(fit_beta, type = "response")$fit)

# Full-data prediction (predicts for all rows, including dropped ones)
head(predict(fit_beta, newdata = pw401k, type = "response")$fit)
```

---

pw401k *401(k) Participation Rates*

Description

A dataset containing participation rates and firm characteristics used in Papke and Wooldridge (1996).

Usage

```
pw401k
```

Format

A data frame with 4734 rows and 5 variables:

prate participation rate, proportion

mrate matching rate, proportion

totemp total number of employees

age years the plan has been in place

sole binary indicating if it's sole plan offered by employer

Source

Papke, L. E. and Wooldridge, J. M. (1996).

References

Papke, L. E., & Wooldridge, J. M. (1996). "Econometric methods for fractional response variables with an application to 401(k) plan participation rates." *Journal of Applied Econometrics*, 11(6), 619-632. doi:10.1002/(SICI)10991255(199611)11:6<619::AIDJAE418>3.0.CO;21

`residuals.mlmodel` *Extract Model Residuals*

Description

Extract Model Residuals

Usage

```
## S3 method for class 'mlmodel'
residuals(object, type = c("response", "pearson"), ...)
```

Arguments

`object`	An `mlmodel` object.
`type`	Character string. Type of residuals to return. Currently supported: `"response"` (default) or `"pearson"`.
`...`	Further arguments passed to methods (currently not used).

Details

`"response"` residuals are the raw residuals: observed minus fitted values.

`"pearson"` residuals are standardized by the model-implied standard deviation: $(y - \hat{y})/\sqrt{\text{Var}(y)}$. For Poisson models they use $\sqrt{\hat{\mu}}$, for binary models $\sqrt{\hat{p}(1-\hat{p})}$, and for other models the appropriate variance from `predict(object, type = "var")` or `type = "var_y"`.

Value

A numeric vector of residuals aligned to the original data frame. Observations dropped during estimation (due to `NA`s or `subset`) return `NA`.

se *Extract Standard Errors from mlmodel Objects*

Description

Extract Standard Errors from mlmodel Objects

Extracts standard errors for an `mlmodel` object.

Usage

```
se(object, ...)

## S3 method for class 'mlmodel'
se(
  object,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)
```

Arguments

`object`	An object of class `"mlmodel"`.
`...`	Further arguments passed to methods (currently not used).
`vcov`	An optional user-supplied variance-covariance matrix.
`vcov.type`	Character string specifying the type of variance-covariance matrix to use. One of `"oim"` (default), `"opg"`, `"robust"`, `"boot"`, or `"jack"`. See `vcov.mlmodel()` for details.
`cl_var`	Character string or vector. Name of the clustering variable (or the vector itself) when `vcov.type = "robust"` (or its alias `"cluster"`).
`repetitions`	Integer. Number of bootstrap replications when `vcov.type = "boot"`. Default is 999.
`seed`	Integer. Random seed for reproducibility when bootstrapping. If `NULL`, a random seed is generated internally.
`progress`	Logical. Should a progress bar be shown during bootstrapping or jackknifing? Default is `FALSE`.

Value

A named numeric vector of standard errors, with the same names as `coef(object)`.

See Also

vcov.mlmodel, summary.mlmodel, confint.mlmodel

`smoke` *1979 National Health Interview Survey*

Description

Sample of males in 1979 and early 1980 from the smoking supplement.

Usage

```
smoke
```

Format

A data frame with 807 observations and 10 variables.

educ Years of education

cigpric State's cigarette price (cents per pack)

white Binary: white

age Age in years

income Annual income in dollars

cigs Number of cigarettes smoked per day

restaurn Binary: restaurant restrictions on smoking

lincome `log(income)`

agesq `age^2`

lcigpric `log(cigprice)`

Source

https://cran.r-project.org/package=wooldridge

References

Wooldridge, J. M. (2020). *Introductory Econometrics: A Modern Approach*. 7th Edition. Boston, MA: Cengage Learning.

Mullahy, J. (1997). "Instrumental-Variable Estimation of Count Data Models: Applications to Models of Cigarette Smoking and Infant Health." *Review of Economics and Statistics* 79, 586-593.

summary.ml_beta *Summary for mlmodel objects*

Description

Summary for mlmodel objects

Usage

```
## S3 method for class 'ml_beta'
summary(
  object,
  correlation = FALSE,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)

## S3 method for class 'ml_gamma'
summary(
  object,
  correlation = FALSE,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)

## S3 method for class 'ml_lm'
summary(
  object,
  correlation = FALSE,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)
```

```
## S3 method for class 'ml_logit'
summary(
  object,
  correlation = FALSE,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)

## S3 method for class 'mlmodel'
summary(
  object,
  correlation = FALSE,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)

## S3 method for class 'ml_negbin'
summary(
  object,
  correlation = FALSE,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)

## S3 method for class 'ml_poisson'
summary(
  object,
  correlation = FALSE,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
```

```
  seed = NULL,
  progress = FALSE,
  ...
)

## S3 method for class 'ml_probit'
summary(
  object,
  correlation = FALSE,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)
```

Arguments

`object`	A fitted model object of class `"mlmodel"`.
`correlation`	Logical. Should the correlation matrix of the estimated parameters be included in the output? Default is `FALSE`. If `TRUE` the correlation matrix will be computed, and stored in the `'summary.mlmodel'` object the function returns.
`vcov`	Optional user-supplied variance-covariance matrix. If provided, it will be used instead of computing one internally.
`vcov.type`	Character string specifying the type of variance-covariance matrix to use. See vcov.
`cl_var`	Character string or vector. Name of the clustering variable or the vector itself. See vcov.
`repetitions`	Integer. Number of bootstrap replications when `vcov.type = "boot"`. Default is 999.
`seed`	Integer. Random seed for reproducibility when `vcov.type = "boot"`. If `NULL`, a random seed is generated.
`progress`	Logical. Should a progress bar be displayed during bootstrapping or jackknifing? Default is `FALSE` (silent).
`...`	Further arguments passed to methods.

Details

Coefficient names in the fitted object use the prefixes `value::` and `scale::` to identify to which equation they belong to, and to avoid confusion when the same variable(s) appear(s) in both the value and scale equations.

Value

An object of class `summary.mlmodel` and any other class that extends it.

Author(s)

Alfonso Sanchez-Penalver

Examples

```
data(mroz)
mroz$incthou <- mroz$faminc / 1000

fit <- ml_lm(incthou ~ age + I(age^2) + huswage + educ + unem,
             data = mroz)

# Default: observed information matrix
summary(fit)

# Different variance types
summary(fit, vcov.type = "opg")                    # Outer product of gradients
summary(fit, vcov.type = "robust")                 # Robust/sandwich estimator

# Clustered robust standard errors
summary(fit, vcov.type = "robust", cl_var = "age")

# Using a pre-computed variance matrix (e.g. bootstrap)
v_boot <- vcov(fit, type = "boot", repetitions = 100, seed = 123)
summary(fit, vcov = v_boot)
```

`update.mlmodel` *Update an mlmodel Call*

Description

Update an mlmodel Call

Usage

```
## S3 method for class 'mlmodel'
update(
  object,
  formula. = NULL,
  scale. = NULL,
  data = NULL,
  weights = NULL,
  subset = NULL,
  ...,
  evaluate = TRUE
)

## S3 method for class 'ml_poisson'
```

```
update(
  object,
  formula. = NULL,
  data = NULL,
  weights = NULL,
  subset = NULL,
  ...,
  evaluate = TRUE
)
```

Arguments

object	An mlmodel object.
formula.	An updated formula for the value (location/mean) equation.
scale.	An updated formula for the scale equation (if supported by the model).
data	A data frame to be used when re-fitting the model.
weights	Optional case weights.
subset	An expression or logical vector to subset, or a vector of indices to resample (sandwich uses it for that).
...	Further arguments passed to methods (currently ignored).
evaluate	Logical. If TRUE (default), the updated call is evaluated and the new fitted model is returned. If FALSE, the updated call (as a language object) is returned without evaluation.

Details

This method re-evaluates the original model call after modifying selected arguments. It is used internally by IMtest() and serves as a fallback mechanism in bootstrap and jackknife variance estimation when a model-specific implementation is not available.

sandwich **package compatibility**

The functions sandwich::vcovBS() and sandwich::vcovJK() are supported through this update() method. They produce numerically equivalent results to our own vcov(object, type = "boot") and vcov(object, type = "jack") when all bootstrap/jackknife replications converge, taking longer to compute them.

Important difference: When some replications fail to converge, sandwich includes those failed iterations in the variance calculation, while our vcov() implementation uses **only successful replications**. The latter is statistically more appropriate.

We therefore strongly recommend using the native vcov() methods provided by **mlmodels** for bootstrap and jackknife variance-covariance matrices.

Value

And updated mlmodel object (or the class of the estimator that extends it) with the modified formula/call and refitted parameters.

vcov.mlmodel *Variance-Covariance Matrix for mlmodel Objects*

Description

Returns the variance-covariance matrix of the estimated parameters using different methods.

Usage

```
## S3 method for class 'mlmodel'
vcov(
  object,
  type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = TRUE,
  ...
)
```

Arguments

object	An object of class "mlmodel" or a model inheriting from it (e.g. "ml_lm").
type	Character string specifying the type of variance-covariance matrix. One of "oim" (default), "robust", "opg", "cluster", "boot", and "jack" or "jackknife".
cl_var	Character string or vector. Name of the clustering variable in the data, or the vector itself.
repetitions	Integer. Number of bootstrap replications to use when type = "boot". Default is 999.
seed	Integer. Random seed for reproducibility when type = "boot". If NULL, a random seed is generated.
progress	Logical. Should a progress bar be displayed? Default is TRUE when type is "boot" or "jack"/"jackknife". Ignored for other types.
...	Further arguments passed to methods.

Details

The package provides several variance-covariance estimators through the type argument:

- "oim" - Observed Information Matrix (default)
- "opg" - Outer Product of Gradients (BHHH)
- "robust" - Robust (sandwich) estimator
- "cluster" - Alias for "robust" when clustering the variance but requires cl_var to be set.
- "boot" - Bootstrap (with optional clustering)

- "jack" - Jackknife (with optional clustering)
- "jackknife" - alias for "jack"

Clustered standard errors are obtained by setting cl_var when using "robust"/"cluster", "boot", or "jack".

Value

A symmetric variance-covariance matrix with coefficient names on the rows and columns.

Author(s)

Alfonso Sanchez-Penalver

Examples

```
data(mroz)
mroz$incthou <- mroz$faminc / 1000

fit <- ml_lm(incthou ~ age + I(age^2) + huswage + educ + unem,
             data = mroz)

# Different variance-covariance estimators
v_oim   <- vcov(fit, type = "oim")      # Observed Information Matrix (default)
v_opg   <- vcov(fit, type = "opg")      # Outer Product of Gradients (BHHH)
v_robust <- vcov(fit, type = "robust")   # Robust / Sandwich estimator

# Clustered robust standard errors
v_clust <- vcov(fit, type = "robust", cl_var = "age")

# Bootstrap variance-covariance matrix
v_boot  <- vcov(fit, type = "boot", repetitions = 100, seed = 123)

# Jackknife variance-covariance matrix
v_jack  <- vcov(fit, type = "jack")

# Compare standard errors across methods
sterrors <- data.frame(
  oim = sqrt(diag(v_oim)),
  opg = sqrt(diag(v_opg)),
  robust = sqrt(diag(v_robust)),
  cluster = sqrt(diag(v_clust)),
  bootstrap = sqrt(diag(v_boot)),
  jackknife = sqrt(diag(v_jack))
)

sterrors
```

vuongtest *Vuong's Test for Non-Nested Models*

Description

Performs Vuong's (1989) test for comparing two non-nested models fitted via maximum likelihood with the `mlmodels` package.

Usage

```
vuongtest(object_1, object_2, ...)

## S3 method for class 'mlmodel'
vuongtest(object_1, object_2, ...)
```

Arguments

`object_1`	A fitted model object inheriting from `"mlmodel"`.
`object_2`	A fitted model object inheriting from `"mlmodel"`.
`...`	Further arguments passed to methods (currently not used).

Details

Vuong's test compares two non-nested models by testing the null hypothesis that the two models are equally close to the true data generating process.

The test statistic is based on the difference in the per-observation log-likelihood contributions between the two models. A positive significant value favors `object_1`, a negative significant value favors `object_2`, and a non-significant value leads to an "inconclusive" result.

Both models must be estimated on exactly the same sample.

Value

An object of class `"vuongtest.mlmodel"` containing the test statistic, p-value, and a conclusion (which model is preferred or "inconclusive").

References

Vuong, Q. H. (1989). 'Likelihood Ratio Tests for Model Selection and Non-Nested Hypotheses.' *Econometrica*, 57(2), 307-333. doi:10.2307/1912557

See Also

`lrtest()`, `waldtest()`, `IMtest()`

Examples

```
# Linear models example (lognormal vs gamma)
data(mroz)
mroz$incthou <- mroz$faminc / 1000

fit_lognormal <- ml_lm(log(incthou) ~ age + I(age^2) + huswage + educ + unem,
                       data = mroz)

fit_gamma <- ml_gamma(incthou ~ age + I(age^2) + huswage + educ + unem,
                      data = mroz)


vuongtest(fit_lognormal, fit_gamma)

# Count models example

fit_poi <- ml_poisson(docvis ~ private + medicaid + age + I(age^2) + educyr +
                          actlim + totchr,
                      data = docvis)

fit_nb1 <- ml_negbin(docvis ~ private + medicaid + age + I(age^2) + educyr +
                          actlim + totchr,
                      data = docvis,
                      dispersion = "NB1")

fit_nb2 <- ml_negbin(docvis ~ private + medicaid + age + I(age^2) + educyr +
                          actlim + totchr,
                      data = docvis)

# Poisson vs. NB1
vuongtest(fit_poi, fit_nb1)

# NB1 vs. NB2
vuongtest(fit_nb1, fit_nb2)

# Binary models example
data(smoke)
smoke$smokes <- smoke$cigs > 0

fit_logit <- ml_logit(smokes ~ cigpric + income + age, data = smoke)
fit_probit <- ml_probit(smokes ~ cigpric + income + age, data = smoke)

vuongtest(fit_logit, fit_probit)
```

`waldtest` *Wald Test for Linear Restrictions*

Description

Performs a Wald test of linear restrictions on the parameters of an `mlmodel` object.

Usage

```
waldtest(object, ...)

## S3 method for class 'mlmodel'
waldtest(
  object,
  indices = NULL,
  coef_names = NULL,
  rest_matrix = NULL,
  rhs = 0,
  vcov = NULL,
  vcov.type = "oim",
  cl_var = NULL,
  repetitions = 999,
  seed = NULL,
  progress = FALSE,
  ...
)
```

Arguments

`object`	An object of class `"mlmodel"`.
`...`	Further arguments passed to methods.
`indices`	Integer vector. Positions of the coefficients to be tested.
`coef_names`	Character vector. Names of the coefficients to test.
`rest_matrix`	Numeric matrix. A q × k restriction matrix (advanced use).
`rhs`	Numeric vector. Value(s) the linear combination(s) should equal. Default is 0.
`vcov`	Optional user-supplied variance-covariance matrix.
`vcov.type`	Character string. Type of variance-covariance matrix to use. One of `"oim"` (default), `"opg"`, `"robust"`, `"boot"`, or `"jack"`. See `vcov.mlmodel()` for details.
`cl_var`	Character string or vector. Clustering variable when `vcov.type = "robust"` or `"boot"`.
`repetitions`	Integer. Number of bootstrap replications when `vcov.type = "boot"`. Default is 999.
`seed`	Integer. Random seed for bootstrap.
`progress`	Logical. Show progress bar during bootstrapping? Default `FALSE`.

Details

The Wald test evaluates linear restrictions of the form $R\beta = r$.

Three convenient interfaces are provided:

- `indices` or `coef_names`: Test individual coefficients (or groups of coefficients) against the value(s) in `rhs` (defaults to 0, which is useful for joint significance tests).
- `rest_matrix` + `rhs`: Test general linear combinations of coefficients (advanced use).

The test statistic follows a χ^2 distribution with degrees of freedom equal to the number of restrictions under the null hypothesis.

Value

An object of class `"waldtest.mlmodel"`.

Author(s)

Alfonso Sanchez-Penalver

See Also

`lrtest()`, `IMtest()`, `confint.mlmodel()`

Examples

```
data(mroz)
mroz$incthou <- mroz$faminc / 1000

fit <- ml_lm(incthou ~ age + I(age^2) + huswage + educ + unem,
             data = mroz)

# 1. Test single coefficients using indices (default OIM)
waldtest(fit, indices = c(2, 5))

# 2. Test using coefficient names and robust standard errors
waldtest(fit, coef_names = c("value::educ", "value::unem"),
         vcov.type = "robust")

# 3. Test explicit constraints
waldtest(fit, coef_names = "value::educ", rhs = 1, vcov.type = "robust")

# 4. Test a linear combination of two coefficients using a restriction matrix
# H0: educ + huswage = 3
R <- matrix(c(0, 0, 0, 1, 1, 0, 0), nrow = 1)
waldtest(fit, rest_matrix = R, rhs = 3, vcov.type = "boot",
         repetitions = 100, seed = 123)
```

Index