Package 'performance'

February 3, 2021

Type Package

Title Assessment of Regression Models Performance

Version 0.7.0

Maintainer Daniel Lüdecke <d.luedecke@uke.de>

Description Utilities for computing measures to assess model quality, which are not directly provided by R's 'base' or 'stats' packages. These include e.g. measures like r-squared, intraclass correlation coefficient (Nakagawa, Johnson & Schielzeth (2017) <doi:10.1098/rsif.2017.0213>), root mean squared error or functions to check models for overdispersion, singularity or zero-inflation and more. Functions apply to a large variety of regression models, including generalized linear models, mixed effects models and Bayesian models.

URL https://easystats.github.io/performance/

BugReports https://github.com/easystats/performance/issues

Depends R (>= 3.4)

Imports insight (>= 0.11.0), bayestestR (>= 0.7.5), stats, utils

Suggests AER, BayesFactor, betareg, bigutilsr, brms, car, CompQuadForm, correlation, covr, cplm, dbscan, fixest, forecast, ggplot2, gridExtra, glmmTMB, ICS, ICSOutlier, lavaan, lme4, lmtest, loo, Matrix, MASS, metafor, mgcv, mlogit, nlme, nonnest2, ordinal, parallel, parameters, pscl, psych, randomForest, rmarkdown, rstanarm, rstantools, sandwich, see, survey, survival, testthat, tweedie, VGAM, spelling

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Language en-US

Config/testthat/edition 3

Config/testthat/parallel true

NeedsCompilation no

Author Daniel Lüdecke [aut, cre] (<https://orcid.org/0000-0002-8895-3206>),
Dominique Makowski [aut, ctb] (<https://orcid.org/0000-0001-5375-9967>),
Philip Waggoner [aut, ctb] (<https://orcid.org/0000-0002-7825-7573>),
Indrajeet Patil [aut, ctb] (<https://orcid.org/0000-0003-1995-6531>),
Mattan S. Ben-Shachar [aut, ctb]
(<https://orcid.org/0000-0002-4287-4801>)

Repository CRAN

Date/Publication 2021-02-03 19:40:06 UTC

R topics documented:

`binned_residuals` *Binned residuals for logistic regression*

Description

Check model quality of logistic regression models.

Usage

```
binned_residuals(model, term = NULL, n_bins = NULL, ...)
```

Arguments

`model`	A `glm`-object with binomial-family.
`term`	Name of independent variable from `x`. If not `NULL`, average residuals for the categories of `term` are plotted; else, average residuals for the estimated probabilities of the response are plotted.
`n_bins`	Numeric, the number of bins to divide the data. If `n_bins = NULL`, the square root of the number of observations is taken.
`...`	Further argument like `size` (for point-size) or `color` (for point-colors).

Details

Binned residual plots are achieved by "dividing the data into categories (bins) based on their fitted values, and then plotting the average residual versus the average fitted value for each bin." *(Gelman, Hill 2007: 97)*. If the model were true, one would expect about 95% of the residuals to fall inside the error bounds.

If `term` is not `NULL`, one can compare the residuals in relation to a specific model predictor. This may be helpful to check if a term would fit better when transformed, e.g. a rising and falling pattern of residuals along the x-axis (the pattern is indicated by a green line) is a signal to consider taking the logarithm of the predictor (cf. Gelman and Hill 2007, pp. 97ff).

Value

A data frame representing the data that is mapped to the plot, which is automatically plotted. In case all residuals are inside the error bounds, points are black. If some of the residuals are outside the error bounds (indicates by the grey-shaded area), blue points indicate residuals that are OK, while red points indicate model under- or overfitting for the related range of estimated probabilities.

Note

Since `binned_residuals()` returns a data frame, the default action for the result is *printing*. However, the 'print()'-method for `binned_residuals()` actually creates a plot. For further modifications of the plot, use 'print()' and add ggplot-layers to the return values, e.g `plot(binned_residuals(model)) + see::scale_color_pizza()`.

References

Gelman, A., & Hill, J. (2007). Data analysis using regression and multilevel/hierarchical models. Cambridge; New York: Cambridge University Press.

Examples

```
if (require("see")) {
  model <- glm(vs ~ wt + mpg, data = mtcars, family = "binomial")
  binned_residuals(model)
}
```

`check_autocorrelation` *Check model for independence of residuals.*

Description

Check model for independence of residuals, i.e. for autocorrelation of error terms.

Usage

```
check_autocorrelation(x, ...)

## Default S3 method:
check_autocorrelation(x, nsim = 1000, ...)
```

Arguments

x	A model object.
...	Currently not used.
nsim	Number of simulations for the Durbin-Watson-Test.

Details

Performs a Durbin-Watson-Test to check for autocorrelated residuals. In case of autocorrelation, robust standard errors return more accurate results for the estimates, or maybe a mixed model with error term for the cluster groups should be used.

Value

Invisibly returns the p-value of the test statistics. A p-value < 0.05 indicates autocorrelated residuals.

Examples

```
m <- lm(mpg ~ wt + cyl + gear + disp, data = mtcars)
check_autocorrelation(m)
```

---

check_collinearity *Check for multicollinearity of model terms*

Description

`check_collinearity()` checks regression models for multicollinearity by calculating the variance inflation factor (VIF). `multicollinearity()` is an alias for `check_collinearity()`.

Usage

```
check_collinearity(x, ...)

multicollinearity(x, ...)

## Default S3 method:
check_collinearity(x, verbose = TRUE, ...)

## S3 method for class 'glmmTMB'
check_collinearity(
  x,
```

```
  component = c("all", "conditional", "count", "zi", "zero_inflated"),
  verbose = TRUE,
  ...
)
```

Arguments

x	A model object (that should at least respond to `vcov()`, and if possible, also to `model.matrix()` - however, it also should work without `model.matrix()`).
...	Currently not used.
verbose	Toggle off warnings or messages.
component	For models with zero-inflation component, multicollinearity can be checked for the conditional model (count component, `component = "conditional"` or `component = "count"`), zero-inflation component (`component = "zero_inflated"` or `component = "zi"`) or both components (`component = "all"`). Following model-classes are currently supported: `hurdle`, `zeroinfl`, `zerocount`, `MixMod` and `glmmTMB`.

Details

Multicollinearity: Multicollinearity should not be confused with a raw strong correlation between predictors. What matters is the association between one or more predictor variables, *conditional on the other variables in the model*. In a nutshell, multicollinearity means that once you know the effect of one predictor, the value of knowing the other predictor is rather low. Thus, one of the predictors doesn't help much in terms of better understanding the model or predicting the outcome. As a consequence, if multicollinearity is a problem, the model seems to suggest that the predictors in question don't seems to be reliably associated with the outcome (low estimates, high standard errors), although these predictors actually are strongly associated with the outcome, i.e. indeed might have strong effect (*McElreath 2020, chapter 6.1*).

Multicollinearity might arise when a third, unobserved variable has a causal effect on each of the two predictors that are associated with the outcome. In such cases, the actual relationship that matters would be the association between the unobserved variable and the outcome.

Remember: "Pairwise correlations are not the problem. It is the conditional associations - not correlations - that matter." (*McElreath 2020, p. 169*)

Interpretation of the Variance Inflation Factor: The variance inflation factor is a measure to analyze the magnitude of multicollinearity of model terms. A VIF less than 5 indicates a low correlation of that predictor with other predictors. A value between 5 and 10 indicates a moderate correlation, while VIF values larger than 10 are a sign for high, not tolerable correlation of model predictors (*James et al. 2013*). The *Increased SE* column in the output indicates how much larger the standard error is due to the association with other predictors conditional on the remaining variables in the model.

Multicollinearity and Interaction Terms: If interaction terms are included in a model, high VIF values are expected. This portion of multicollinearity among the component terms of an interaction is also called "inessential ill-conditioning", which leads to inflated VIF values that are typically seen for models with interaction terms *(Francoeur 2013)*.

Value

A data frame with three columns: The name of the model term, the variance inflation factor and the factor by which the standard error is increased due to possible correlation with other terms.

Note

There is also a `plot()`-method implemented in the **see**-package.

References

- Francoeur, R. B. (2013). Could Sequential Residual Centering Resolve Low Sensitivity in Moderated Regression? Simulations and Cancer Symptom Clusters. Open Journal of Statistics, 03(06), 24–44.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (eds.). (2013). An introduction to statistical learning: with applications in R. New York: Springer.
- McElreath, R. (2020). Statistical rethinking: A Bayesian course with examples in R and Stan. 2nd edition. Chapman and Hall/CRC.
- Vanhove, J. (2019). Collinearity isn't a disease that needs curing. webpage

Examples

```
m <- lm(mpg ~ wt + cyl + gear + disp, data = mtcars)
check_collinearity(m)

# plot results
if (require("see")) {
  x <- check_collinearity(m)
  plot(x)
}
```

---

check_convergence *Convergence test for mixed effects models*

Description

`check_convergence()` provides an alternative convergence test for `merMod`-objects.

Usage

```
check_convergence(x, tolerance = 0.001, ...)
```

Arguments

`x`	A `merMod`-object.
`tolerance`	Indicates up to which value the convergence result is accepted. The smaller `tolerance` is, the stricter the test will be.
`...`	Currently not used.

Details

Convergence and log-likelihood: Convergence problems typically arise when the model hasn't converged to a solution where the log-likelihood has a true maximum. This may result in unreliable and overly complex (or non-estimable) estimates and standard errors.

Inspect model convergence: **lme4** performs a convergence-check (see `?lme4::convergence`), however, as as discussed here and suggested by one of the lme4-authors in this comment, this check can be too strict. `check_convergence()` thus provides an alternative convergence test for `merMod`-objects.

Resolving convergence issues: Convergence issues are not easy to diagnose. The help page on `?lme4::convergence` provides most of the current advice about how to resolve convergence issues. Another clue might be large parameter values, e.g. estimates (on the scale of the linear predictor) larger than 10 in (non-identity link) generalized linear model *might* indicate complete separation. Complete separation can be addressed by regularization, e.g. penalized regression or Bayesian regression with appropriate priors on the fixed effects.

Convergence versus Singularity: Note the different meaning between singularity and convergence: singularity indicates an issue with the "true" best estimate, i.e. whether the maximum likelihood estimation for the variance-covariance matrix of the random effects is positive definite or only semi-definite. Convergence is a question of whether we can assume that the numerical optimization has worked correctly or not.

Value

`TRUE` if convergence is fine and `FALSE` if convergence is suspicious. Additionally, the convergence value is returned as attribute.

Examples

```
if (require("lme4")) {
  data(cbpp)
  set.seed(1)
  cbpp$x <- rnorm(nrow(cbpp))
  cbpp$x2 <- runif(nrow(cbpp))

  model <- glmer(
    cbind(incidence, size - incidence) ~ period + x + x2 + (1 + x | herd),
    data = cbpp,
    family = binomial()
  )

  check_convergence(model)
}
```

`check_distribution` *Classify the distribution of a model-family using machine learning*

Description

Choosing the right distributional family for regression models is essential to get more accurate estimates and standard errors. This function may help to check a models' distributional family and see if the model-family probably should be reconsidered. Since it is difficult to exactly predict the correct model family, consider this function as somewhat experimental.

Usage

```
check_distribution(model)
```

Arguments

`model`	Typically, a model (that should response to `residuals()`). May also be a numeric vector.

Details

This function uses an internal random forest model to classify the distribution from a model-family. Currently, following distributions are trained (i.e. results of `check_distribution()` may be one of the following): `"bernoulli"`, `"beta"`, `"beta-binomial"`, `"binomial"`, `"chi"`, `"exponential"`, `"F"`, `"gamma"`, `"lognormal"`, `"normal"`, `"negative binomial"`, `"negative binomial (zero-inflated)"`, `"pareto"`, `"poisson"`, `"poisson (zero-inflated)"`, `"uniform"` and `"weibull"`.

Note the similarity between certain distributions according to shape, skewness, etc. Thus, the predicted distribution may not be perfectly representing the distributional family of the underlying fitted model, or the response value.

There is a `plot()` method, which shows the probabilities of all predicted distributions, however, only if the probability is greater than zero.

Note

This function is somewhat experimental and might be improved in future releases. The final decision on the model-family should also be based on theoretical aspects and other information about the data and the model.

There is also a `plot()`-method implemented in the **see**-package.

Examples

```
if (require("lme4") && require("parameters") && require("gridExtra")) {
  data(sleepstudy)

  model <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
```

```
  check_distribution(model)
  plot(check_distribution(model))
}
```

`check_heteroscedasticity` *Check model for (non-)constant error variance*

Description

Check model for (non-)constant error variance.

Usage

```
check_heteroscedasticity(x, ...)
```

Arguments

`x`	A model object.
`...`	Currently not used.

Value

Invisibly returns the p-value of the test statistics. A p-value < 0.05 indicates a non-constant variance (heteroskedasticity).

Note

There is also a `plot()`-method implemented in the **see**-package.

Examples

```
m <- lm(mpg ~ wt + cyl + gear + disp, data = mtcars)
check_heteroscedasticity(m)

# plot results
if (require("see")) {
  x <- check_heteroscedasticity(m)
  plot(x)
}
```

`check_homogeneity` *Check model for homogeneity of variances*

Description

Check model for homogeneity of variances between groups described by independent variables in a model.

Usage

```
check_homogeneity(x, method = c("bartlett", "fligner", "levene", "auto"), ...)
```

Arguments

`x`	A linear model or an ANOVA object.
`method`	Name of the method (underlying test) that should be performed to check the homogeneity of variances. May either be `"levene"` for Levene's Test for Homogeneity of Variance, `"bartlett"` for the Bartlett test (assuming normal distributed samples or groups), `"fligner"` for the Fligner-Killeen test (rank-based, non-parametric test), or `"auto"`. In the latter case, Bartlett test is used if the model response is normal distributed, else Fligner-Killeen test is used.
`...`	Arguments passed down to `car::leveneTest()`.

Value

Invisibly returns the p-value of the test statistics. A p-value < 0.05 indicates a significant difference in the variance between the groups.

Note

There is also a `plot()`-method implemented in the **see**-package.

Examples

```
model <- lm(len ~ supp + dose, data = ToothGrowth)
check_homogeneity(model)

# plot results
if (require("see")) {
  result <- check_homogeneity(model)
  plot(result)
}
```

`check_itemscale` *Describe Properties of Item Scales*

Description

Compute various measures of internal consistencies applied to (sub)scales, which items were extracted using `parameters::principal_components()`.

Usage

```
check_itemscale(x)
```

Arguments

`x`	An object of class `parameters_pca`, as returned by `parameters::principal_components()`.

Details

`check_itemscale()` calculates various measures of internal consistencies, such as Cronbach's alpha, item difficulty or discrimination etc. on subscales which were built from several items. Subscales are retrieved from the results of `parameters::principal_components()`, i.e. based on how many components were extracted from the PCA, `check_itemscale()` retrieves those variables that belong to a component and calculates the above mentioned measures.

Value

A list of data frames, with related measures of internal consistencies of each subscale.

Note

- *Item difficulty* should range between 0.2 and 0.8. Ideal value is `p+(1-p)/2` (which mostly is between 0.5 and 0.8). See `item_difficulty` for details.
- For *item discrimination*, acceptable values are 0.20 or higher; the closer to 1.00 the better. See `item_reliability` for more details.
- In case the total *Cronbach's alpha* value is below the acceptable cut-off of 0.7 (mostly if an index has few items), the *mean inter-item-correlation* is an alternative measure to indicate acceptability. Satisfactory range lies between 0.2 and 0.4. See also `item_intercor`.

References

- Briggs SR, Cheek JM (1986) The role of factor analysis in the development and evaluation of personality scales. Journal of Personality, 54(1), 106-148. doi: 10.1111/j.1467-6494.1986.tb00391.x
- Trochim WMK (2008) Types of Reliability. (web)

Examples

```
# data generation from '?prcomp', slightly modified
C <- chol(S <- toeplitz(.9^(0:15)))
set.seed(17)
X <- matrix(rnorm(16000), 100, 16)
Z <- X %*% C
if (require("parameters") && require("psych")) {
  pca <- principal_components(as.data.frame(Z), rotation = "varimax", n = 3)
  pca
  check_itemscale(pca)
}
```

---

check_model *Visual check of model assumptions*

Description

Visual check of model various assumptions (normality of residuals, normality of random effects, heteroscedasticity, homogeneity of variance, multicollinearity).

Usage

```
check_model(x, ...)

## Default S3 method:
check_model(
  x,
  dot_size = 2,
  line_size = 0.8,
  panel = TRUE,
  check = "all",
  alpha = 0.2,
  ...
)
```

Arguments

x	A model object.
...	Currently not used.
dot_size	Size of dot-geoms.
line_size	Size of line-geoms.
panel	Logical, if TRUE, plots are arranged as panels; else, single plots for each diagnostic are returned.

check	Character vector, indicating which checks for should be performed and plotted. May be one or more of "all","vif","qq","normality","ncv","homogeneity","outliers","reqq' "reqq" is a QQ-plot for random effects and only available for mixed models. "ncv" checks for non-constant variance, i.e. for heteroscedasticity. By default, all possible checks are performed and plotted.
alpha	The alpha level of the confidence bands. Scalar from 0 to 1.

Value

The data frame that is used for plotting.

Note

This function just prepares the data for plotting. To create the plots, **see** needs to be installed. Furthermore, this function suppresses all possible warnings. In case you observe suspicious plots, please refer to the dedicated functions (like `check_collinearity()`, `check_normality()` etc.) to get informative messages and warnings.

Examples

```
## Not run:
m <- lm(mpg ~ wt + cyl + gear + disp, data = mtcars)
check_model(m)

if (require("lme4")) {
  m <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
  check_model(m, panel = FALSE)
}

if (require("rstanarm")) {
  m <- stan_glm(mpg ~ wt + gear, data = mtcars, chains = 2, iter = 200)
  check_model(m)
}

## End(Not run)
```

---

check_normality *Check model for (non-)normality of residuals.*

Description

Check model for (non-)normality of residuals.

Usage

```
check_normality(x, ...)

## S3 method for class 'merMod'
check_normality(x, effects = c("fixed", "random"), ...)
```

Arguments

x	A model object.
...	Currently not used.
effects	Should normality for residuals ("fixed") or random effects ("random") be tested? Only applies to mixed models. May be abbreviated.

Details

check_normality() calls stats::shapiro.test and checks the standardized residuals (or studentized residuals for mixed models) for normal distribution. Note that this formal test almost always yields significant results for the distribution of residuals and visual inspection (e.g. Q-Q plots) are preferable.

Value

Invisibly returns the p-value of the test statistics. A p-value < 0.05 indicates a significant deviation from normal distribution

Note

For mixed models, studentized residuals are used for the test, *not* the standardized residuals. There is also a plot()-method implemented in the **see**-package.

Examples

```
m <- lm(mpg ~ wt + cyl + gear + disp, data = mtcars)
check_normality(m)

# plot results
if (require("see")) {
  x <- check_normality(m)
  plot(x)
}
## Not run:
# QQ-plot
plot(check_normality(m), type = "qq")

# PP-plot
plot(check_normality(m), type = "pp")

## End(Not run)
```

`check_outliers` *Outliers detection (check for influential observations)*

Description

Checks for and locates influential observations (i.e., "outliers") via several distance and/or clustering methods. If several methods are selected, the returned "Outlier" vector will be a composite outlier score, made of the average of the binary (0 or 1) results of each method. It represents the probability of each observation of being classified as an outlier by at least one method. The decision rule used by default is to classify as outliers observations which composite outlier score is superior or equal to 0.5 (i.e., that were classified as outliers by at least half of the methods). See the **Details** section below for a description of the methods.

Usage

```
check_outliers(x, ...)

## Default S3 method:
check_outliers(x, method = c("cook", "pareto"), threshold = NULL, ...)

## S3 method for class 'numeric'
check_outliers(x, method = "zscore", threshold = NULL, ...)

## S3 method for class 'data.frame'
check_outliers(x, method = "mahalanobis", threshold = NULL, ...)
```

Arguments

`x`	A model or a data.frame object.
`...`	When `method = "ics"`, further arguments in `...` are passed down to `ICSOutlier::ics.outlier()`.
`method`	The outlier detection method(s). Can be "all" or some of c("cook", "pareto", "zscore", "iqr", "mahalanobis", "robust", "mcd", "ics", "optics", "lof").
`threshold`	A list containing the threshold values for each method (e.g. `list('mahalanobis' = 7, 'cook' = 1)`), above which an observation is considered as outlier. If `NULL`, default values will be used (see 'Details').

Details

Outliers can be defined as particularly influential observations. Most methods rely on the computation of some distance metric, and the observations greater than a certain threshold are considered outliers. Importantly, outliers detection methods are meant to provide information to the researcher, rather than being an automatized procedure which mindless application is a substitute for thinking.

An **example sentence** for reporting the usage of the composite method could be:

"Based on a composite outlier score (see the 'check_outliers' function in the 'performance' R package; Lüdecke et al., 2019) obtained via the joint application of multiple outliers detection algorithms (Z-scores, Iglewicz, 1993; Interquartile range (IQR); Mahalanobis distance, Cabana, 2019;

Robust Mahalanobis distance, Gnanadesikan & Kettenring, 1972; Minimum Covariance Determinant, Leys et al., 2018; Invariant Coordinate Selection, Archimbaud et al., 2018; OPTICS, Ankerst et al., 1999; Isolation Forest, Liu et al. 2008; and Local Outlier Factor, Breunig et al., 2000), we excluded n participants that were classified as outliers by at least half of the methods used."

Model-specific methods:

- **Cook's Distance**: Among outlier detection methods, Cook's distance and leverage are less common than the basic Mahalanobis distance, but still used. Cook's distance estimates the variations in regression coefficients after removing each observation, one by one (Cook, 1977). Since Cook's distance is in the metric of an F distribution with p and n-p degrees of freedom, the median point of the quantile distribution can be used as a cut-off (Bollen, 1985). A common approximation or heuristic is to use 4 divided by the numbers of observations, which usually corresponds to a lower threshold (i.e., more outliers are detected). This only works for Frequentist models. For Bayesian models, see `pareto`.
- **Pareto**: The reliability and approximate convergence of Bayesian models can be assessed using the estimates for the shape parameter k of the generalized Pareto distribution. If the estimated tail shape parameter k exceeds 0.5, the user should be warned, although in practice the authors of the `loo` package observed good performance for values of k up to 0.7 (the default threshold used by `performance`).

Univariate methods:

- **Z-scores**: The Z-score, or standard score, is a way of describing a data point as deviance from a central value, in terms of standard deviations from the mean or, as it is here the case by default (Iglewicz, 1993), in terms of Median Absolute Deviation (MAD) from the median (which are robust measures of dispersion and centrality). The default threshold to classify outliers is 1.959 (`threshold = list("zscore" = 1.959)`), corresponding to the 2.5% (`qnorm(0.975)`) most extreme observations (assuming the data is normally distributed). Importantly, the Z-score method is univariate: it is computed column by column. If a data.frame is passed, then the maximum distance is kept for each observations. Thus, all observations that are extreme for at least one variable will be detected. However, this method is not suited for high dimensional data (with many columns), returning too liberal results (detecting many outliers).
- **IQR**: Using the IQR (interquartile range) is a robust method developed by John Tukey, which often appears in box-and-whisker plots (e.g., in `geom_boxplot`). The interquartile range is the range between the first and the third quartiles. Tukey considered as outliers any data point that fell outside of either 1.5 times (the default threshold) the IQR below the first or above the third quartile. Similar to the Z-score method, this is a univariate method for outliers detection, returning outliers detected for at least one column, and might thus not be suited to high dimensional data.

Multivariate methods:

- **Mahalanobis Distance**: Mahalanobis distance (Mahalanobis, 1930) is often used for multivariate outliers detection as this distance takes into account the shape of the observations. The default `threshold` is often arbitrarily set to some deviation (in terms of SD or MAD) from the mean (or median) of the Mahalanobis distance. However, as the Mahalanobis distance can be approximated by a Chi squared distribution (Rousseeuw & Van Zomeren, 1990), we can use the alpha quantile of the chi-square distribution with k degrees of freedom (k being the number of columns). By default, the alpha threshold is set to 0.025 (corresponding to the

2.5% most extreme observations; Cabana, 2019). This criterion is a natural extension of the median plus or minus a coefficient times the MAD method (Leys et al., 2013).

- **Robust Mahalanobis Distance**: A robust version of Mahalanobis distance using an Orthogonalized Gnanadesikan-Kettenring pairwise estimator (Gnanadesikan \& Kettenring, 1972). Requires the **bigutilsr** package.
- **Minimum Covariance Determinant (MCD)**: Another robust version of Mahalanobis. Leys et al. (2018) argue that Mahalanobis Distance is not a robust way to determine outliers, as it uses the means and covariances of all the data – including the outliers – to determine individual difference scores. Minimum Covariance Determinant calculates the mean and covariance matrix based on the most central subset of the data (by default, 66%), before computing the Mahalanobis Distance. This is deemed to be a more robust method of identifying and removing outliers than regular Mahalanobis distance.
- **Invariant Coordinate Selection (ICS)**: The outlier are detected using ICS, which by default uses an alpha threshold of 0.025 (corresponding to the 2.5% most extreme observations) as a cut-off value for outliers classification. Refer to the help-file of `ICSOutlier::ics.outlier()` to get more details about this procedure. Note that `method = "ics"` requires both **ICS** and **ICSOutlier** to be installed, and that it takes some time to compute the results.
- **OPTICS**: The Ordering Points To Identify the Clustering Structure (OPTICS) algorithm (Ankerst et al., 1999) is using similar concepts to DBSCAN (an unsupervised clustering technique that can be used for outliers detection). The threshold argument is passed as `minPts`, which corresponds to the minimum size of a cluster. By default, this size is set at 2 times the number of columns (Sander et al., 1998). Compared to the others techniques, that will always detect several outliers (as these are usually defined as a percentage of extreme values), this algorithm functions in a different manner and won't always detect outliers. Note that `method = "optics"` requires the **dbscan** package to be installed, and that it takes some time to compute the results.
- **Isolation Forest**: The outliers are detected using the anomaly score of an isolation forest (a class of random forest). The default threshold of 0.025 will classify as outliers the observations located at `qnorm(1-0.025) * MAD)` (a robust equivalent of SD) of the median (roughly corresponding to the 2.5% most extreme observations). Requires the **solitude** package.
- **Local Outlier Factor**: Based on a K nearest neighbours algorithm, LOF compares the local density of an point to the local densities of its neighbors instead of computing a distance from the center (Breunig et al., 2000). Points that have a substantially lower density than their neighbors are considered outliers. A LOF score of approximately 1 indicates that density around the point is comparable to its neighbors. Scores significantly larger than 1 indicate outliers. The default threshold of 0.025 will classify as outliers the observations located at `qnorm(1-0.025) * SD)` of the log-transformed LOF distance. Requires the **dbscan** package.

Threshold specification: Default thresholds are currently specified as follows:

```
list( zscore = stats::qnorm(p = 1 -0.025),iqr = 1.5,cook = stats::qf(0.5,ncol(x),nrow(x)
-ncol(x)),pareto = 0.7,mahalanobis = stats::qchisq(p = 1 -0.025,df = ncol(x)),robust
= stats::qchisq(p = 1 -0.025,df = ncol(x)),mcd = stats::qchisq(p = 1 -0.025,df = ncol(x)),ics
= 0.025,optics = 2 * ncol(x),iforest = 0.025,lof = 0.025 )
```

Value

A logical vector of the detected outliers with a nice printing method: a check (message) on whether outliers were detected or not. The information on the distance measure and whether or not an observation is considered as outlier can be recovered with the `as.data.frame` function.

Note

There is also a `plot()`-method implemented in the **see**-package.

References

- Archimbaud, A., Nordhausen, K., \& Ruiz-Gazen, A. (2018). ICS for multivariate outlier detection with application to quality control. Computational Statistics & Data Analysis, 128, 184–199. doi: 10.1016/j.csda.2018.06.011
- Gnanadesikan, R., \& Kettenring, J. R. (1972). Robust estimates, residuals, and outlier detection with multiresponse data. Biometrics, 81-124.
- Bollen, K. A., & Jackman, R. W. (1985). Regression diagnostics: An expository treatment of outliers and influential cases. Sociological Methods & Research, 13(4), 510-542.
- Cabana, E., Lillo, R. E., \& Laniado, H. (2019). Multivariate outlier detection based on a robust Mahalanobis distance with shrinkage estimators. arXiv preprint arXiv:1904.02596.
- Cook, R. D. (1977). Detection of influential observation in linear regression. Technometrics, 19(1), 15-18.
- Iglewicz, B., & Hoaglin, D. C. (1993). How to detect and handle outliers (Vol. 16). Asq Press.
- Leys, C., Klein, O., Dominicy, Y., \& Ley, C. (2018). Detecting multivariate outliers: Use a robust variant of Mahalanobis distance. Journal of Experimental Social Psychology, 74, 150-156.
- Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008, December). Isolation forest. In 2008 Eighth IEEE International Conference on Data Mining (pp. 413-422). IEEE.
- Rousseeuw, P. J., \& Van Zomeren, B. C. (1990). Unmasking multivariate outliers and leverage points. Journal of the American Statistical association, 85(411), 633-639.

Examples

```
# Univariate
check_outliers(mtcars$mpg)

# Multivariate
# select only mpg and disp (continuous)
mt1 <- mtcars[, c(1, 3, 4)]
# create some fake outliers and attach outliers to main df
mt2 <- rbind(mt1, data.frame(mpg = c(37, 40), disp = c(300, 400), hp = c(110, 120)))
# fit model with outliers
model <- lm(disp ~ mpg + hp, data = mt2)

ol <- check_outliers(model)
# plot(ol)
insight::get_data(model)[ol, ]

if (require("MASS")) {
  check_outliers(model, method = c("mahalabonis", "mcd"))
}
## Not run:
# This one takes some seconds to finish...
check_outliers(model, method = "ics")
```

```
# For dataframes
check_outliers(mtcars)
check_outliers(mtcars, method = "all")

## End(Not run)
```

`check_overdispersion` *Check overdispersion of GL(M)M's*

Description

`check_overdispersion()` checks generalized linear (mixed) models for overdispersion.

Usage

```
check_overdispersion(x, ...)
```

Arguments

`x`	Fitted model of class `merMod`, `glmmTMB`, `glm`, or `glm.nb` (package **MASS**).
`...`	Currently not used.

Details

Overdispersion occurs when the observed variance is higher than the variance of a theoretical model. For Poisson models, variance increases with the mean, thus, variance usually (roughly) equals the mean value. If the variance is much higher, the data are "overdispersed".

Interpretation of the Dispersion Ratio: If the dispersion ratio is close to one, a Poisson model fits well to the data. Dispersion ratios larger than one indicate overdispersion, thus a negative binomial model or similar might fit better to the data. A p-value < .05 indicates overdispersion.

Overdispersion in Poisson Models: For Poisson models, the overdispersion test is based on the code from *Gelman and Hill (2007), page 115*.

Overdispersion in Mixed Models: For `merMod`- and `glmmTMB`-objects, `check_overdispersion()` is based on the code in the GLMM FAQ, section *How can I deal with overdispersion in GLMMs?*. Note that this function only returns an *approximate* estimate of an overdispersion parameter, and is probably inaccurate for zero-inflated mixed models (fitted with `glmmTMB`).

How to fix Overdispersion: Overdispersion can be fixed by either modeling the dispersion parameter, or by choosing a different distributional family (like Quasi-Poisson, or negative binomial, see *Gelman and Hill (2007), pages 115-116*).

Value

A list with results from the overdispersion test, like chi-squared statistics, p-value or dispersion ratio.

References

- Bolker B et al. (2017): GLMM FAQ.
- Gelman, A., & Hill, J. (2007). Data analysis using regression and multilevel/hierarchical models. Cambridge; New York: Cambridge University Press.

Examples

```
if (require("glmmTMB")) {
  data(Salamanders)
  m <- glm(count ~ spp + mined, family = poisson, data = Salamanders)
  check_overdispersion(m)

  m <- glmmTMB(
    count ~ mined + spp + (1 | site),
    family = poisson,
    data = Salamanders
  )
  check_overdispersion(m)
}
```

---

check_singularity *Check mixed models for boundary fits*

Description

Check mixed models for boundary fits.

Usage

```
check_singularity(x, tolerance = 1e-05, ...)
```

Arguments

x	A mixed model.
tolerance	Indicates up to which value the convergence result is accepted. The larger tolerance is, the stricter the test will be.
...	Currently not used.

Details

If a model is "singular", this means that some dimensions of the variance-covariance matrix have been estimated as exactly zero. This often occurs for mixed models with complex random effects structures.

"While singular models are statistically well defined (it is theoretically sensible for the true maximum likelihood estimate to correspond to a singular fit), there are real concerns that (1) singular fits correspond to overfitted models that may have poor power; (2) chances of numerical problems

and mis-convergence are higher for singular models (e.g. it may be computationally difficult to compute profile confidence intervals for such models); (3) standard inferential procedures such as Wald statistics and likelihood ratio tests may be inappropriate." (*lme4 Reference Manual*)

There is no gold-standard about how to deal with singularity and which random-effects specification to choose. Beside using fully Bayesian methods (with informative priors), proposals in a frequentist framework are:

- avoid fitting overly complex models, such that the variance-covariance matrices can be estimated precisely enough (*Matuschek et al. 2017*)
- use some form of model selection to choose a model that balances predictive accuracy and overfitting/type I error (*Bates et al. 2015*, *Matuschek et al. 2017*)
- "keep it maximal", i.e. fit the most complex model consistent with the experimental design, removing only terms required to allow a non-singular fit (*Barr et al. 2013*)

Note the different meaning between singularity and convergence: singularity indicates an issue with the "true" best estimate, i.e. whether the maximum likelihood estimation for the variance-covariance matrix of the random effects is positive definite or only semi-definite. Convergence is a question of whether we can assume that the numerical optimization has worked correctly or not.

Value

TRUE if the model fit is singular.

References

- Bates D, Kliegl R, Vasishth S, Baayen H. Parsimonious Mixed Models. arXiv:1506.04967, June 2015.
- Barr DJ, Levy R, Scheepers C, Tily HJ. Random effects structure for confirmatory hypothesis testing: Keep it maximal. Journal of Memory and Language, 68(3):255-278, April 2013.
- Matuschek H, Kliegl R, Vasishth S, Baayen H, Bates D. Balancing type I error and power in linear mixed models. Journal of Memory and Language, 94:305-315, 2017.
- lme4 Reference Manual, https://cran.r-project.org/package=lme4

Examples

```
if (require("lme4")) {
  data(sleepstudy)
  set.seed(123)
  sleepstudy$mygrp <- sample(1:5, size = 180, replace = TRUE)
  sleepstudy$mysubgrp <- NA
  for (i in 1:5) {
    filter_group <- sleepstudy$mygrp == i
    sleepstudy$mysubgrp[filter_group] <-
      sample(1:30, size = sum(filter_group), replace = TRUE)
  }

  model <- lmer(
    Reaction ~ Days + (1 | mygrp / mysubgrp) + (1 | Subject),
    data = sleepstudy
```

```
  )

  check_singularity(model)
}
```

---

check_zeroinflation *Check for zero-inflation in count models*

Description

check_zeroinflation() checks whether count models are over- or underfitting zeros in the outcome.

Usage

```
check_zeroinflation(x, tolerance = 0.05)
```

Arguments

x	Fitted model of class merMod, glmmTMB, glm, or glm.nb (package **MASS**).
tolerance	The tolerance for the ratio of observed and predicted zeros to considered as over- or underfitting zeros. A ratio between 1 +/- tolerance is considered as OK, while a ratio beyond or below this threshold would indicate over- or underfitting.

Details

If the amount of observed zeros is larger than the amount of predicted zeros, the model is underfitting zeros, which indicates a zero-inflation in the data. In such cases, it is recommended to use negative binomial or zero-inflated models.

Value

A list with information about the amount of predicted and observed zeros in the outcome, as well as the ratio between these two values.

Examples

```
if (require("glmmTMB")) {
  data(Salamanders)
  m <- glm(count ~ spp + mined, family = poisson, data = Salamanders)
  check_zeroinflation(m)
}
```

classify_distribution *Machine learning model trained to classify distributions*

Description

Mean accuracy and Kappa of 0.86 and 0.85, repsectively.

Usage

```
classify_distribution
```

Format

An object of class `randomForest.formula` (inherits from `randomForest`) of length 8.

compare_performance *Compare performance of different models*

Description

`compare_performance()` computes indices of model performance for different models at once and hence allows comparison of indices across models.

Usage

```
compare_performance(..., metrics = "all", rank = FALSE, verbose = TRUE)
```

Arguments

`...`	Multiple model objects (also of different classes).
`metrics`	Can be `"all"`, `"common"` or a character vector of metrics to be computed. See related `documentation` of object's class for details.
`rank`	Logical, if `TRUE`, models are ranked according to 'best' overall model performance. See 'Details'.
`verbose`	Toggle off warnings.

Details

Ranking Models: When `rank = TRUE`, a new column `Performance_Score` is returned. This score ranges from 0% to 100%, higher values indicating better model performance. Note that all score value do not necessarily sum up to 100%. Rather, calculation is based on normalizing all indices (i.e. rescaling them to a range from 0 to 1), and taking the mean value of all indices for each model. This is a rather quick heuristic, but might be helpful as exploratory index.

In particular when models are of different types (e.g. mixed models, classical linear models,

logistic regression, ...), not all indices will be computed for each model. In case where an index can't be calculated for a specific model type, this model gets an NA value. All indices that have any NAs are excluded from calculating the performance score.

There is a plot()-method for compare_performance(), which creates a "spiderweb" plot, where the different indices are normalized and larger values indicate better model performance. Hence, points closer to the center indicate worse fit indices (see online-documentation for more details).

Value

A data frame (with one row per model) and one column per "index" (see metrics).

Note

There is also a plot()-method implemented in the **see**-package.

Examples

```
data(iris)
lm1 <- lm(Sepal.Length ~ Species, data = iris)
lm2 <- lm(Sepal.Length ~ Species + Petal.Length, data = iris)
lm3 <- lm(Sepal.Length ~ Species * Petal.Length, data = iris)
compare_performance(lm1, lm2, lm3)
compare_performance(lm1, lm2, lm3, rank = TRUE)

if (require("lme4")) {
  m1 <- lm(mpg ~ wt + cyl, data = mtcars)
  m2 <- glm(vs ~ wt + mpg, data = mtcars, family = "binomial")
  m3 <- lmer(Petal.Length ~ Sepal.Length + (1 | Species), data = iris)
  compare_performance(m1, m2, m3)
}
```

---

cronbachs_alpha *Cronbach's Alpha for Items or Scales*

Description

Compute various measures of internal consistencies for tests or item-scales of questionnaires.

Usage

```
cronbachs_alpha(x)
```

Arguments

x	A matrix or a data frame.

Details

The Cronbach's Alpha value for x. A value closer to 1 indicates greater internal consistency, where usually following rule of thumb is applied to interpret the results: $\alpha < 0.5$ is unacceptable, $0.5 < \alpha < 0.6$ is poor, $0.6 < \alpha < 0.7$ is questionable, $0.7 < \alpha < 0.8$ is acceptable, and everything > 0.8 is good or excellent.

Value

The Cronbach's Alpha value for x.

References

Bland, J. M., \& Altman, D. G. Statistics notes: Cronbach's alpha. BMJ 1997;314:572. 10.1136/bmj.314.7080.572

Examples

```
data(mtcars)
x <- mtcars[, c("cyl", "gear", "carb", "hp")]
cronbachs_alpha(x)
```

---

display.performance_model

Print tables in different output formats

Description

Prints tables (i.e. data frame) in different output formats. `print_md()` is a alias for `display(format = "markdown")`.

Usage

```
## S3 method for class 'performance_model'
display(object, format = "markdown", digits = 2, ...)

## S3 method for class 'performance_model'
print_md(x, digits = 2, ...)
```

Arguments

object, x	An object returned by model_performance() or compare_performance(). or its summary.
format	String, indicating the output format. Currently, only "markdown" is supported.
digits	Number of decimal places.
...	Currently not used.

Details

`display()` is useful when the table-output from functions, which is usually printed as formatted text-table to console, should be formatted for pretty table-rendering in markdown documents, or if knitted from rmarkdown to PDF or Word files. See vignette for examples.

Value

A character vector. If `format = "markdown"`, the return value will be a character vector in markdown-table format.

Examples

```
model <- lm(mpg ~ wt + cyl, data = mtcars)
mp <- model_performance(model)
display(mp)
```

---

icc *Intraclass Correlation Coefficient (ICC)*

Description

This function calculates the intraclass-correlation coefficient (ICC) - sometimes also called *variance partition coefficient* (VPC) - for mixed effects models. The ICC can be calculated for all models supported by `insight::get_variance()`. For models fitted with the **brms**-package, `icc()` might fail due to the large variety of models and families supported by the **brms**-package. In such cases, an alternative to the ICC is the `variance_decomposition()`, which is based on the posterior predictive distribution (see 'Details').

Usage

```
icc(model, by_group = FALSE)

variance_decomposition(model, re_formula = NULL, robust = TRUE, ci = 0.95, ...)
```

Arguments

`model`	A (Bayesian) mixed effects model.
`by_group`	Logical, if `TRUE`, `icc()` returns the variance components for each random-effects level (if there are multiple levels). See 'Details'.
`re_formula`	Formula containing group-level effects to be considered in the prediction. If `NULL` (default), include all group-level effects. Else, for instance for nested models, name a specific group-level effect to calculate the variance decomposition for this group-level. See 'Details' and `?brms::posterior_predict`.
`robust`	Logical, if `TRUE`, the median instead of mean is used to calculate the central tendency of the variances.
`ci`	Credible interval level.
`...`	Arguments passed down to `brms::posterior_predict()`.

Details

Interpretation: The ICC can be interpreted as "the proportion of the variance explained by the grouping structure in the population". This index goes from 0, if the grouping conveys no information, to 1, if all observations in a group are identical (Gelman \& Hill, 2007, p. 258). In other word, the ICC "can also be interpreted as the expected correlation between two randomly drawn units that are in the same group" *(Hox 2010: 15)*, although this definition might not apply to mixed models with more complex random effects structures.

Calculation: The ICC is calculated by dividing the random effect variance, σ_i^2, by the total variance, i.e. the sum of the random effect variance and the residual variance, σ_ϵ^2.

Adjusted and conditional ICC: `icc()` calculates an adjusted and conditional ICC, which both take all sources of uncertainty (i.e. of *all random effects*) into account. While the *adjusted ICC* only relates to the random effects, the *conditional ICC* also takes the fixed effects variances into account (see *Nakagawa et al. 2017*). Typically, the *adjusted* ICC is of interest when the analysis of random effects is of interest. `icc()` returns a meaningful ICC also for more complex random effects structures, like models with random slopes or nested design (more than two levels) and is applicable for models with other distributions than Gaussian. For more details on the computation of the variances, see `?insight::get_variance`.

ICC for unconditional and conditional models: Usually, the ICC is calculated for the null model ("unconditional model"). However, according to *Raudenbush and Bryk (2002)* or *Rabe-Hesketh and Skrondal (2012)* it is also feasible to compute the ICC for full models with covariates ("conditional models") and compare how much, e.g., a level-2 variable explains the portion of variation in the grouping structure (random intercept).

ICC for specific group-levels: The proportion of variance for specific levels related to the overall model can be computed by setting `by_group = TRUE`. The reported ICC is the variance for each (random effect) group compared to the total variance of the model. For mixed models with a simple random intercept, this is identical to the classical (adjusted) ICC.

Variance decomposition for brms-models: If `model` is of class `brmsfit`, `icc()` might fail due to the large variety of models and families supported by the **brms** package. In such cases, `variance_decomposition()` is an alternative ICC measure. The function calculates a variance decomposition based on the posterior predictive distribution. In this case, first, the draws from the posterior predictive distribution *not conditioned* on group-level terms (`posterior_predict(...,re_formula = NA)`) are calculated as well as draws from this distribution *conditioned* on *all random effects* (by default, unless specified else in `re_formula`) are taken. Then, second, the variances for each of these draws are calculated. The "ICC" is then the ratio between these two variances. This is the recommended way to analyse random-effect-variances for non-Gaussian models. It is then possible to compare variances across models, also by specifying different group-level terms via the `re_formula`-argument.

Sometimes, when the variance of the posterior predictive distribution is very large, the variance ratio in the output makes no sense, e.g. because it is negative. In such cases, it might help to use `robust = TRUE`.

Value

A list with two values, the adjusted and conditional ICC. For `variance_decomposition()`, a list with two values, the decomposed ICC as well as the credible intervals for this ICC.

References

- Hox, J. J. (2010). Multilevel analysis: techniques and applications (2nd ed). New York: Routledge.
- Nakagawa, S., Johnson, P. C. D., & Schielzeth, H. (2017). The coefficient of determination R2 and intra-class correlation coefficient from generalized linear mixed-effects models revisited and expanded. Journal of The Royal Society Interface, 14(134), 20170213. doi: 10.1098/rsif.2017.0213
- Rabe-Hesketh, S., & Skrondal, A. (2012). Multilevel and longitudinal modeling using Stata (3rd ed). College Station, Tex: Stata Press Publication.
- Raudenbush, S. W., & Bryk, A. S. (2002). Hierarchical linear models: applications and data analysis methods (2nd ed). Thousand Oaks: Sage Publications.

Examples

```
if (require("lme4")) {
  model <- lmer(Sepal.Length ~ Petal.Length + (1 | Species), data = iris)
  icc(model)
}

# ICC for specific group-levels
if (require("lme4")) {
  data(sleepstudy)
  set.seed(12345)
  sleepstudy$grp <- sample(1:5, size = 180, replace = TRUE)
  sleepstudy$subgrp <- NA
  for (i in 1:5) {
    filter_group <- sleepstudy$grp == i
    sleepstudy$subgrp[filter_group] <-
      sample(1:30, size = sum(filter_group), replace = TRUE)
  }
  model <- lmer(
    Reaction ~ Days + (1 | grp / subgrp) + (1 | Subject),
    data = sleepstudy
  )
  icc(model, by_group = TRUE)
}
```

---

item_difficulty *Difficulty of Questionnaire Items*

Description

Compute various measures of internal consistencies for tests or item-scales of questionnaires.

Usage

```
item_difficulty(x)
```

Arguments

x	Depending on the function, x may be a matrix as returned by the cor()-function, or a data frame with items (e.g. from a test or questionnaire).

Details

This function calculates the item difficulty, which should range between 0.2 and 0.8. Lower values are a signal for more difficult items, while higher values close to one are a sign for easier items. The ideal value for item difficulty is p + (1 -p) / 2, where p = 1 / max(x). In most cases, the ideal item difficulty lies between 0.5 and 0.8.

Value

A data frame with three columns: The name(s) of the item(s), the item difficulties for each item, and the ideal item difficulty.

Examples

```
data(mtcars)
x <- mtcars[, c("cyl", "gear", "carb", "hp")]
item_difficulty(x)
```

---

item_intercor *Mean Inter-Item-Correlation*

Description

Compute various measures of internal consistencies for tests or item-scales of questionnaires.

Usage

```
item_intercor(x, method = c("pearson", "spearman", "kendall"))
```

Arguments

x	A matrix as returned by the cor()-function, or a data frame with items (e.g. from a test or questionnaire).
method	Correlation computation method. May be one of "spearman" (default), "pearson" or "kendall". You may use initial letter only.

Details

This function calculates a mean inter-item-correlation, i.e. a correlation matrix of `x` will be computed (unless `x` is already a matrix as returned by the `cor()`-function) and the mean of the sum of all item's correlation values is returned. Requires either a data frame or a computed `cor()`-object.

"Ideally, the average inter-item correlation for a set of items should be between .20 and .40, suggesting that while the items are reasonably homogenous, they do contain sufficiently unique variance so as to not be isomorphic with each other. When values are lower than .20, then the items may not be representative of the same content domain. If values are higher than .40, the items may be only capturing a small bandwidth of the construct." *(Piedmont 2014)*

Value

The mean inter-item-correlation value for `x`.

References

Piedmont RL. 2014. Inter-item Correlations. In: Michalos AC (eds) Encyclopedia of Quality of Life and Well-Being Research. Dordrecht: Springer, 3303-3304. doi: 10.1007/9789400707535_1493

Examples

```
data(mtcars)
x <- mtcars[, c("cyl", "gear", "carb", "hp")]
item_intercor(x)
```

`item_reliability` *Reliability Test for Items or Scales*

Description

Compute various measures of internal consistencies for tests or item-scales of questionnaires.

Usage

```
item_reliability(x, standardize = FALSE, digits = 3)
```

Arguments

`x`	A matrix or a data frame.
`standardize`	Logical, if `TRUE`, the data frame's vectors will be standardized. Recommended when the variables have different measures / scales.
`digits`	Amount of digits for returned values.

Details

This function calculates the item discriminations (corrected item-total correlations for each item of x with the remaining items) and the Cronbach's alpha for each item, if it was deleted from the scale. The absolute value of the item discrimination indices should be above 0.1. An index between 0.1 and 0.3 is considered as "fair", while an index above 0.3 (or below -0.3) is "good". Items with low discrimination indices are often ambiguously worded and should be examined. Items with negative indices should be examined to determine why a negative value was obtained (e.g. reversed answer categories regarding positive and negative poles).

Value

A data frame with the corrected item-total correlations (*item discrimination*, column `item_discrimination`) and Cronbach's Alpha (if item deleted, column `alpha_if_deleted`) for each item of the scale, or `NULL` if data frame had too less columns.

Examples

```
data(mtcars)
x <- mtcars[, c("cyl", "gear", "carb", "hp")]
item_reliability(x)
```

---

item_split_half *Split-Half Reliability*

Description

Compute various measures of internal consistencies for tests or item-scales of questionnaires.

Usage

```
item_split_half(x, digits = 3)
```

Arguments

x	A matrix or a data frame.
digits	Amount of digits for returned values.

Details

This function calculates the split-half reliability for items in x, including the Spearman-Brown adjustment. Splitting is done by selecting odd versus even columns in x. A value closer to 1 indicates greater internal consistency.

Value

A list with two elements: the split-half reliability `splithalf` and the Spearman-Brown corrected split-half reliability `spearmanbrown`.

References

Spearman C. 1910. Correlation calculated from faulty data. British Journal of Psychology (3): 271-295. doi: 10.1111/j.20448295.1910.tb00206.x

Brown W. 1910. Some experimental results in the correlation of mental abilities. British Journal of Psychology (3): 296-322. doi: 10.1111/j.20448295.1910.tb00207.x

Examples

```
data(mtcars)
x <- mtcars[, c("cyl", "gear", "carb", "hp")]
item_split_half(x)
```

---

looic *LOO-related Indices for Bayesian regressions.*

Description

Compute LOOIC (leave-one-out cross-validation (LOO) information criterion) and ELPD (expected log predictive density) for Bayesian regressions.

Usage

```
looic(model, verbose = TRUE)
```

Arguments

model	A Bayesian regression model.
verbose	Toggle off warnings.

Value

A list with four elements, the ELPD, LOOIC and their standard errors.

Examples

```
if (require("rstanarm")) {
  model <- stan_glm(mpg ~ wt + cyl, data = mtcars, chains = 1, iter = 500, refresh = 0)
  looic(model)
}
```

`model_performance` *Model Performance*

Description

See the documentation for your object's class:

- Frequentist Regressions
- Instrumental Variables Regressions
- Mixed models
- Bayesian models
- CFA / SEM lavaan models
- Meta-analysis models

Usage

```
model_performance(model, ...)

performance(model, ...)
```

Arguments

`model`	Statistical model.
`...`	Arguments passed to or from other methods, resp. for `compare_performance()`, one or multiple model objects (also of different classes).

Value

A data frame (with one row) and one column per "index" (see `metrics`).

See Also

`compare_performance()` to compare performance of many different models.

Examples

```
model <- lm(mpg ~ wt + cyl, data = mtcars)
model_performance(model)

model <- glm(vs ~ wt + mpg, data = mtcars, family = "binomial")
model_performance(model)
```

model_performance.ivreg

Performance of instrumental variable regression models

Description

Performance of instrumental variable regression models

Usage

```
## S3 method for class 'ivreg'
model_performance(model, metrics = "all", verbose = TRUE, ...)
```

Arguments

model	A model.
metrics	Can be "all", "common" or a character vector of metrics to be computed (some of c("AIC","BIC","R2","RMSE","SIGMA","Sargan","Wu_Hausman")). "common" will compute AIC, BIC, R2 and RMSE.
verbose	Toggle off warnings.
...	Arguments passed to or from other methods.

model_performance.lavaan

Performance of lavaan SEM / CFA Models

Description

Compute indices of model performance for SEM or CFA models from the **lavaan** package.

Usage

```
## S3 method for class 'lavaan'
model_performance(model, metrics = "all", verbose = TRUE, ...)
```

Arguments

model	A **lavaan** model.
metrics	Can be "all" or a character vector of metrics to be computed (some of c("Chisq","Chisq_DoF","Chisq
verbose	Toggle off warnings.
...	Arguments passed to or from other methods.

Details

Indices of fit:

- **Chisq**: The model Chi-squared assesses overall fit and the discrepancy between the sample and fitted covariance matrices. Its p-value should be > .05 (i.e., the hypothesis of a perfect fit cannot be rejected). However, it is quite sensitive to sample size.
- **GFI/AGFI**: The (Adjusted) Goodness of Fit is the proportion of variance accounted for by the estimated population covariance. Analogous to R2. The GFI and the AGFI should be > .95 and > .90, respectively.
- **NFI/NNFI/TLI**: The (Non) Normed Fit Index. An NFI of 0.95, indicates the model of interest improves the fit by 95% relative to the null model. The NNFI (also called the Tucker Lewis index; TLI) is preferable for smaller samples. They should be > .90 (Byrne, 1994) or > .95 (Schumacker & Lomax, 2004).
- **CFI**: The Comparative Fit Index is a revised form of NFI. Not very sensitive to sample size (Fan, Thompson, & Wang, 1999). Compares the fit of a target model to the fit of an independent, or null, model. It should be > .90.
- **RMSEA**: The Root Mean Square Error of Approximation is a parsimony-adjusted index. Values closer to 0 represent a good fit. It should be < .08 or < .05. The p-value printed with it tests the hypothesis that RMSEA is less than or equal to .05 (a cutoff sometimes used for good fit), and thus should be not significant.
- **RMR/SRMR**: the (Standardized) Root Mean Square Residual represents the square-root of the difference between the residuals of the sample covariance matrix and the hypothesized model. As the RMR can be sometimes hard to interpret, better to use SRMR. Should be < .08.
- **RFI**: the Relative Fit Index, also known as RHO1, is not guaranteed to vary from 0 to 1. However, RFI close to 1 indicates a good fit.
- **IFI**: the Incremental Fit Index (IFI) adjusts the Normed Fit Index (NFI) for sample size and degrees of freedom (Bollen's, 1989). Over 0.90 is a good fit, but the index can exceed 1.
- **PNFI**: the Parsimony-Adjusted Measures Index. There is no commonly agreed-upon cutoff value for an acceptable model for this index. Should be > 0.50.

See the documentation for `?lavaan::fitmeasures`.

What to report: Kline (2015) suggests that at a minimum the following indices should be reported: The model **chi-square**, the **RMSEA**, the **CFI** and the **SRMR**.

Value

A data frame (with one row) and one column per "index" (see `metrics`).

References

- Byrne, B. M. (1994). Structural equation modeling with EQS and EQS/Windows. Thousand Oaks, CA: Sage Publications.
- Tucker, L. R., \& Lewis, C. (1973). The reliability coefficient for maximum likelihood factor analysis. Psychometrika, 38, 1-10.
- Schumacker, R. E., \& Lomax, R. G. (2004). A beginner's guide to structural equation modeling, Second edition. Mahwah, NJ: Lawrence Erlbaum Associates.

- Fan, X., B. Thompson, \& L. Wang (1999). Effects of sample size, estimation method, and model specification on structural equation modeling fit indexes. Structural Equation Modeling, 6, 56-83.
- Kline, R. B. (2015). Principles and practice of structural equation modeling. Guilford publications.

Examples

```
# Confirmatory Factor Analysis (CFA) ---------
if (require("lavaan")) {
  structure <- " visual  =~ x1 + x2 + x3
                 textual =~ x4 + x5 + x6
                 speed   =~ x7 + x8 + x9 "
  model <- lavaan::cfa(structure, data = HolzingerSwineford1939)
  model_performance(model)
}
```

---

model_performance.lm *Performance of Regression Models*

Description

Compute indices of model performance for regression models.

Usage

```
## S3 method for class 'lm'
model_performance(model, metrics = "all", verbose = TRUE, ...)
```

Arguments

model	A model.
metrics	Can be "all", "common" or a character vector of metrics to be computed (some of c("AIC","BIC","R2","RMSE","SIGMA","LOGLOSS","PCP","SCORE")). "common" will compute AIC, BIC, R2 and RMSE.
verbose	Toggle off warnings.
...	Arguments passed to or from other methods.

Details

Depending on model, following indices are computed:

- **AIC** Akaike's Information Criterion, see ?stats::AIC
- **BIC** Bayesian Information Criterion, see ?stats::BIC
- **R2** r-squared value, see r2
- **R2_adj** adjusted r-squared, see r2

- **RMSE** root mean squared error, see performance_rmse
- **SIGMA** residual standard deviation, see get_sigma()
- **LOGLOSS** Log-loss, see performance_logloss
- **SCORE_LOG** score of logarithmic proper scoring rule, see performance_score
- **SCORE_SPHERICAL** score of spherical proper scoring rule, see performance_score
- **PCP** percentage of correct predictions, see performance_pcp

Value

A data frame (with one row) and one column per "index" (see metrics).

Examples

```
model <- lm(mpg ~ wt + cyl, data = mtcars)
model_performance(model)

model <- glm(vs ~ wt + mpg, data = mtcars, family = "binomial")
model_performance(model)
```

---

model_performance.merMod

Performance of Mixed Models

Description

Compute indices of model performance for mixed models.

Usage

```
## S3 method for class 'merMod'
model_performance(model, metrics = "all", verbose = TRUE, ...)
```

Arguments

model	A mixed effects model.
metrics	Can be "all", "common" or a character vector of metrics to be computed (some of c("AIC","BIC","R2","ICC","RMSE","SIGMA","LOGLOSS","SCORE")). "common" will compute AIC, BIC, R2, ICC and RMSE.
verbose	Toggle off warnings.
...	Arguments passed to or from other methods.

Details

This method returns the *adjusted ICC* only, as this is typically of interest when judging the variance attributed to the random effects part of the model (see also icc).

Furthermore, see 'Details' in model_performance.lm for more details on returned indices.

Value

A data frame (with one row) and one column per "index" (see `metrics`).

Examples

```
if (require("lme4")) {
  model <- lmer(Petal.Length ~ Sepal.Length + (1 | Species), data = iris)
  model_performance(model)
}
```

`model_performance.rma` *Performance of Meta-Analysis Models*

Description

Compute indices of model performance for meta-analysis model from the **metafor** package.

Usage

```
## S3 method for class 'rma'
model_performance(model, metrics = "all", verbose = TRUE, ...)
```

Arguments

`model`	A `rma` object as returned by `metafor::rma()`.
`metrics`	Can be `"all"` or a character vector of metrics to be computed (some of `c("AIC","BIC","I2","H2","TAU`
`verbose`	Toggle off warnings.
`...`	Arguments passed to or from other methods.

Details

Indices of fit:

- **AIC** Akaike's Information Criterion, see `?stats::AIC`
- **BIC** Bayesian Information Criterion, see `?stats::BIC`
- **I2**: For a random effects model, `I2` estimates (in percent) how much of the total variability in the effect size estimates can be attributed to heterogeneity among the true effects. For a mixed-effects model, `I2` estimates how much of the unaccounted variability can be attributed to residual heterogeneity.
- **H2**: For a random-effects model, `H2` estimates the ratio of the total amount of variability in the effect size estimates to the amount of sampling variability. For a mixed-effects model, `H2` estimates the ratio of the unaccounted variability in the effect size estimates to the amount of sampling variability.
- **TAU2**: The amount of (residual) heterogeneity in the random or mixed effects model.
- **CochransQ (QE)**: Test for (residual) Heterogeneity. Without moderators in the model, this is simply Cochran's Q-test.

- **Omnibus (QM)**: Omnibus test of parameters.
- **R2**: Pseudo-R2-statistic, which indicates the amount of heterogeneity accounted for by the moderators included in a fixed-effects model.

See the documentation for `?metafor::fitstats`.

Value

A data frame (with one row) and one column per "index" (see `metrics`).

Examples

```
if (require("metafor")) {
  data(dat.bcg)
  dat <- escalc(measure = "RR", ai = tpos, bi = tneg, ci = cpos, di = cneg, data = dat.bcg)
  model <- rma(yi, vi, data = dat, method = "REML")
  model_performance(model)
}
```

---

model_performance.stanreg

Performance of Bayesian Models

Description

Compute indices of model performance for (general) linear models.

Usage

```
## S3 method for class 'stanreg'
model_performance(model, metrics = "all", verbose = TRUE, ...)

## S3 method for class 'BFBayesFactor'
model_performance(
  model,
  metrics = "all",
  verbose = TRUE,
  average = FALSE,
  prior_odds = NULL,
  ...
)
```

Arguments

`model`	Object of class `stanreg` or `brmsfit`.
`metrics`	Can be `"all"`, `"common"` or a character vector of metrics to be computed (some of `c("LOOIC","WAIC","R2","R2_adj","RMSE","SIGMA","LOGLOSS","SCORE")`). `"common"` will compute LOOIC, WAIC, R2 and RMSE.

verbose	Toggle off warnings.
...	Arguments passed to or from other methods.
average	Compute model-averaged index? See bayestestR::weighted_posteriors().
prior_odds	Optional vector of prior odds for the models compared to the first model (or the denominator, for BFBayesFactor objects). For data.frames, this will be used as the basis of weighting.

Details

Depending on model, following indices are computed:

- **ELPD** expected log predictive density, see looic
- **LOOIC** leave-one-out cross-validation (LOO) information criterion, see looic
- **WAIC** widely applicable information criterion, see ?loo::waic
- **R2** r-squared value, see r2
- **R2_LOO_adjusted** adjusted r-squared, see r2
- **RMSE** root mean squared error, see performance_rmse
- **SIGMA** residual standard deviation, see get_sigma()
- **LOGLOSS** Log-loss, see performance_logloss
- **SCORE_LOG** score of logarithmic proper scoring rule, see performance_score
- **SCORE_SPHERICAL** score of spherical proper scoring rule, see performance_score
- **PCP** percentage of correct predictions, see performance_pcp

Value

A data frame (with one row) and one column per "index" (see metrics).

References

Gelman, A., Goodrich, B., Gabry, J., & Vehtari, A. (2018). R-squared for Bayesian regression models. The American Statistician, The American Statistician, 1-6.

See Also

r2_bayes

Examples

```
## Not run:
if (require("rstanarm") && require("rstantools")) {
  model <- stan_glm(mpg ~ wt + cyl, data = mtcars, chains = 1, iter = 500, refresh = 0)
  model_performance(model)

  model <- stan_glmer(
    mpg ~ wt + cyl + (1 | gear),
    data = mtcars,
    chains = 1,
```

```
    iter = 500,
    refresh = 0
  )
  model_performance(model)
}

if (require("BayesFactor") && require("rstantools")) {
  model <- generalTestBF(carb ~ am + mpg, mtcars)

  model_performance(model)
  model_performance(model[3])

  model_performance(model, average = TRUE)
}

## End(Not run)
```

`performance_accuracy` *Accuracy of predictions from model fit*

Description

This function calculates the predictive accuracy of linear or logistic regression models.

Usage

```
performance_accuracy(model, method = c("cv", "boot"), k = 5, n = 1000)
```

Arguments

`model`	A linear or logistic regression model. May also be a mixed model.
`method`	Character string, indicating whether crossvalidation (`method = "cv"`) or bootstrapping (`method = "boot"`) is used to compute the accuracy values.
`k`	The number of folds for the kfold-crossvalidation.
`n`	Number of bootstrap-samples.

Details

For linar models, the accuracy is the correlation coefficient between the actual and the predicted value of the outcome. For logistic regression models, the accuracy corresponds to the AUC-value, calculated with the `bayestestR::auc()`-function.

The accuracy is the mean value of multiple correlation resp. AUC-values, which are either computed with crossvalidation or non-parametric bootstrapping (see argument `method`). The standard error is the standard deviation of the computed correlation resp. AUC-values.

Value

A list with three values: The `Accuracy` of the model predictions, i.e. the proportion of accurately predicted values from the model, its standard error, `SE`, and the `Method` used to compute the accuracy.

Examples

```
model <- lm(mpg ~ wt + cyl, data = mtcars)
performance_accuracy(model)

model <- glm(vs ~ wt + mpg, data = mtcars, family = "binomial")
performance_accuracy(model)
```

`performance_aicc` *Compute AIC and second order AIC*

Description

Compute the second-order Akaike's information criterion (AICc). The second-order (or small sample) is a AIC with a correction for small sample sizes. `performance_aic()` is a small wrapper that returns the AIC. It is a generic function that also works for some models that don't have a AIC method (like Tweedie models).

Usage

```
performance_aicc(x, ...)

performance_aic(x, ...)
```

Arguments

`x`	A model object.
`...`	Currently not used.

Value

Numeric, the AIC or AICc value.

References

- Akaike, H. (1973) Information theory as an extension of the maximum likelihood principle. In: Second International Symposium on Information Theory, pp. 267–281. Petrov, B.N., Csaki, F., Eds, Akademiai Kiado, Budapest.
- Hurvich, C. M., Tsai, C.-L. (1991) Bias of the corrected AIC criterion for underfitted regression and time series models. Biometrika 78, 499–509.

Examples

```
m <- lm(mpg ~ wt + cyl + gear + disp, data = mtcars)
AIC(m)
performance_aicc(m)
```

`performance_hosmer` *Hosmer-Lemeshow goodness-of-fit test*

Description

Check model quality of logistic regression models.

Usage

```
performance_hosmer(model, n_bins = 10)
```

Arguments

`model`	A `glm`-object with binomial-family.
`n_bins`	Numeric, the number of bins to divide the data.

Details

A well-fitting model shows *no* significant difference between the model and the observed data, i.e. the reported p-value should be greater than 0.05.

Value

An object of class `hoslem_test` with following values: `chisq`, the Hosmer-Lemeshow chi-squared statistic; `df`, degrees of freedom and `p.value` the p-value for the goodness-of-fit test.

References

Hosmer, D. W., & Lemeshow, S. (2000). Applied Logistic Regression. Hoboken, NJ, USA: John Wiley & Sons, Inc. doi: 10.1002/0471722146

Examples

```
model <- glm(vs ~ wt + mpg, data = mtcars, family = "binomial")
performance_hosmer(model)
```

`performance_logloss` *Log Loss*

Description

Compute the log loss for models with binary outcome.

Usage

```
performance_logloss(model, verbose = TRUE, ...)
```

Arguments

`model`	Model with binary outcome.
`verbose`	Toggle off warnings.
`...`	Currently not used.

Details

Logistic regression models predict the probability of an outcome of being a "success" or "failure" (or 1 and 0 etc.). `performance_logloss()` evaluates how good or bad the predicted probabilities are. High values indicate bad predictions, while low values indicate good predictions. The lower the log-loss, the better the model predicts the outcome.

Value

Numeric, the log loss of `model`.

See Also

`performance_score()`

Examples

```
data(mtcars)
m <- glm(formula = vs ~ hp + wt, family = binomial, data = mtcars)
performance_logloss(m)
```

performance_mse *Mean Square Error of Linear Models*

Description

Compute mean square error of linear models.

Usage

```
performance_mse(model, ...)

mse(model, ...)
```

Arguments

model	A model.
...	Arguments passed to or from other methods.

Details

The mean square error is the mean of the sum of squared residuals, i.e. it measures the average of the squares of the errors. Less technically speaking, the mean square error can be considered as the variance of the residuals, i.e. the variation in the outcome the model doesn't explain. Lower values (closer to zero) indicate better fit.

Value

Numeric, the mean square error of model.

Examples

```
data(mtcars)
m <- lm(mpg ~ hp + gear, data = mtcars)
performance_mse(m)
```

performance_pcp *Percentage of Correct Predictions*

Description

Percentage of correct predictions (PCP) for models with binary outcome.

Usage

```
performance_pcp(model, ci = 0.95, method = "Herron", verbose = TRUE)
```

Arguments

model	Model with binary outcome.
ci	The level of the confidence interval.
method	Name of the method to calculate the PCP (see 'Details'). Default is "Herron". May be abbreviated.
verbose	Toggle off warnings.

Details

method = "Gelman-Hill" (or "gelman_hill") computes the PCP based on the proposal from *Gelman and Hill 2017, 99*, which is defined as the proportion of cases for which the deterministic prediction is wrong, i.e. the proportion where the predicted probability is above 0.5, although y=0 (and vice versa) (see also *Herron 1999, 90*).

method = "Herron" (or "herron") computes a modified version of the PCP (*Herron 1999, 90-92*), which is the sum of predicted probabilities, where y=1, plus the sum of 1 - predicted probabilities, where y=0, divided by the number of observations. This approach is said to be more accurate.

The PCP ranges from 0 to 1, where values closer to 1 mean that the model predicts the outcome better than models with an PCP closer to 0. In general, the PCP should be above 0.5 (i.e. 50%), the closer to one, the better. Furthermore, the PCP of the full model should be considerably above the null model's PCP.

The likelihood-ratio test indicates whether the model has a significantly better fit than the null-model (in such cases, $p < 0.05$).

Value

A list with several elements: the percentage of correct predictions of the full and the null model, their confidence intervals, as well as the chi-squared and p-value from the Likelihood-Ratio-Test between the full and null model.

References

- Herron, M. (1999). Postestimation Uncertainty in Limited Dependent Variable Models. Political Analysis, 8, 83–98.
- Gelman, A., & Hill, J. (2007). Data analysis using regression and multilevel/hierarchical models. Cambridge; New York: Cambridge University Press, 99

Examples

```
data(mtcars)
m <- glm(formula = vs ~ hp + wt, family = binomial, data = mtcars)
performance_pcp(m)
performance_pcp(m, method = "Gelman-Hill")
```

`performance_rmse` *Root Mean Squared Error*

Description

Compute root mean squared error for (mixed effects) models, including Bayesian regression models.

Usage

```
performance_rmse(model, normalized = FALSE, verbose = TRUE)

rmse(model, normalized = FALSE, verbose = TRUE)
```

Arguments

`model`	A model.
`normalized`	Logical, use `TRUE` if normalized rmse should be returned.
`verbose`	Toggle off warnings.

Details

The RMSE is the square root of the variance of the residuals and indicates the absolute fit of the model to the data (difference between observed data to model's predicted values). It can be interpreted as the standard deviation of the unexplained variance, and is in the same units as the response variable. Lower values indicate better model fit.

The normalized RMSE is the proportion of the RMSE related to the range of the response variable. Hence, lower values indicate less residual variance.

Value

Numeric, the root mean squared error.

Examples

```
if (require("nlme")) {
  m <- lme(distance ~ age, data = Orthodont)

  # RMSE
  performance_rmse(m, normalized = FALSE)

  # normalized RMSE
  performance_rmse(m, normalized = TRUE)
}
```

`performance_roc` *Simple ROC curve*

Description

This function calculates a simple ROC curves of x/y coordinates based on response and predictions of a binomial model.

Usage

```
performance_roc(x, ..., predictions, new_data)
```

Arguments

`x`	A numeric vector, representing the outcome (0/1), or a model with binomial outcome.
`...`	One or more models with binomial outcome. In this case, `new_data` is ignored.
`predictions`	If `x` is numeric, a numeric vector of same length as `x`, representing the actual predicted values.
`new_data`	If `x` is a model, a data frame that is passed to `predict()` as `newdata`-argument. If `NULL`, the ROC for the full model is calculated.

Value

A data frame with three columns, the x/y-coordinate pairs for the ROC curve (`Sensivity` and `Specifity`), and a column with the model name.

Note

There is also a `plot()`-method implemented in the **see**-package.

Examples

```
library(bayestestR)
data(iris)

set.seed(123)
iris$y <- rbinom(nrow(iris), size = 1, .3)
folds <- sample(nrow(iris), size = nrow(iris) / 8, replace = FALSE)
test_data <- iris[folds, ]
train_data <- iris[-folds, ]

model <- glm(y ~ Sepal.Length + Sepal.Width, data = train_data, family = "binomial")
performance_roc(model, new_data = test_data)

roc <- performance_roc(model, new_data = test_data)
area_under_curve(roc$Specifity, roc$Sensivity)
```

```
m1 <- glm(y ~ Sepal.Length + Sepal.Width, data = iris, family = "binomial")
m2 <- glm(y ~ Sepal.Length + Petal.Width, data = iris, family = "binomial")
m3 <- glm(y ~ Sepal.Length + Species, data = iris, family = "binomial")
performance_roc(m1, m2, m3)
```

---

performance_rse *Residual Standard Error for Linear Models*

Description

Compute residual standard error of linear models.

Usage

```
performance_rse(model)
```

Arguments

model	A model.

Details

The residual standard error is the square root of the residual sum of squares divided by the residual degrees of freedom.

Value

Numeric, the residual standard error of `model`.

Examples

```
data(mtcars)
m <- lm(mpg ~ hp + gear, data = mtcars)
performance_rse(m)
```

---

performance_score *Proper Scoring Rules*

Description

Calculates the logarithmic, quadratic/Brier and spherical score from a model with binary or count outcome.

Usage

```
performance_score(model, verbose = TRUE)
```

Arguments

model	Model with binary or count outcome.
verbose	Toggle off warnings.

Details

Proper scoring rules can be used to evaluate the quality of model predictions and model fit. performance_score() calculates the logarithmic, quadratic/Brier and spherical scoring rules. The spherical rule takes values in the interval [0,1], with values closer to 1 indicating a more accurate model, and the logarithmic rule in the interval [-Inf,0], with values closer to 0 indicating a more accurate model.

For stan_lmer() and stan_glmer() models, the predicted values are based on posterior_predict(), instead of predict(). Thus, results may differ more than expected from their non-Bayesian counterparts in **lme4**.

Value

A list with three elements, the logarithmic, quadratic/Brier and spherical score.

Note

Code is partially based on GLMMadaptive::scoring_rules().

References

Carvalho, A. (2016). An overview of applications of proper scoring rules. Decision Analysis 13, 223–242. doi: 10.1287/deca.2016.0337

See Also

performance_logloss()

Examples

```
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12)
outcome <- gl(3, 1, 9)
treatment <- gl(3, 3)
model <- glm(counts ~ outcome + treatment, family = poisson())

performance_score(model)
## Not run:
if (require("glmmTMB")) {
  data(Salamanders)
  model <- glmmTMB(
    count ~ spp + mined + (1 | site),
    zi =  ~ spp + mined,
    family = nbinom2(),
    data = Salamanders
  )
```

```
  performance_score(model)
}

## End(Not run)
```

---

pp_check *Posterior predictive checks for frequentist models*

Description

Posterior predictive checks for frequentist models.

Usage

```
pp_check(object, ...)

## S3 method for class 'lm'
pp_check(object, iterations = 50, check_range = FALSE, re_formula = NULL, ...)

posterior_predictive_check(
  object,
  iterations = 50,
  check_range = FALSE,
  re_formula = NULL,
  ...
)
```

Arguments

object	A statistical model.
...	Passed down to simulate().
iterations	The number of draws to simulate/bootstrap.
check_range	Logical, if TRUE, includes a plot with the minimum value of the original response against the minimum values of the replicated responses, and the same for the maximum value. This plot helps judging whether the variation in the original data is captured by the model or not (*Gelman et al. 2020, pp.163*). The minimum and maximum values of y should be inside the range of the related minimum and maximum values of yrep.
re_formula	Formula containing group-level effects (random effects) to be considered in the simulated data. If NULL (default), condition on all random effects. If NA or ~0, condition on no random effects. See simulate() in **lme4**.

Details

Posterior predictive checks means "simulating replicated data under the fitted model and then comparing these to the observed data" *(Gelman and Hill, 2007, p. 158)*. Posterior predictive checks can be used to "look for systematic discrepancies between real and simulated data" *(Gelman et al. 2014, p. 169)*.

An example how posterior predictive checks can also be used for model comparison is following plot (from *Gabry et al. 2019, Figure 6*):

The model shown in the right panel (b) can simulate new data that are more similar to the observed outcome than the model in the left panel (a). Thus, model (b) is likely to be preferred over model (a).

Value

A data frame of simulated responses and the original response vector.

Note

The default-method, `pp_check.default()` is in package **bayesplot**. Thus, **performance** adds `pp_check()`-methods for different classes and packages (like `lm`, `merMod`, `glmmTMB`, ...). However, since it might be that not all model objects that have a `simulate()` function are covered, and those objects probably can't be passed down to the default-method, there is also a "generic" `posterior_predictive_check()` function, which just calls `pp_check.lm()`. Thus, every model object that has a `simulate()`-method should work with `posterior_predictive_check()`.

References

- Gabry, J., Simpson, D., Vehtari, A., Betancourt, M., & Gelman, A. (2019). Visualization in Bayesian workflow. Journal of the Royal Statistical Society: Series A (Statistics in Society), 182(2), 389–402. https://doi.org/10.1111/rssa.12378
- Gelman, A., & Hill, J. (2007). Data analysis using regression and multilevel/hierarchical models. Cambridge; New York: Cambridge University Press.
- Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2014). Bayesian data analysis. (Third edition). CRC Press.
- Gelman, A., Hill, J., & Vehtari, A. (2020). Regression and Other Stories. Cambridge University Press.

Examples

```
## Not run:
library(performance)
model <- lm(Sepal.Length ~ Species * Petal.Width + Petal.Length, data = iris)
if (require("ggplot2") && require("see")) {
  pp_check(model)
}
## End(Not run)
```

r2 *Compute the model's R2*

Description

Calculate the R2 value for different model objects. Depending on the model, R2, pseudo-R2 or marginal / adjusted R2 values are returned.

Usage

```
r2(model, ...)

## S3 method for class 'merMod'
r2(model, tolerance = 1e-05, ...)
```

Arguments

model	A statistical model.
...	Arguments passed down to the related r2-methods.
tolerance	Tolerance for singularity check of random effects, to decide whether to compute random effect variances for the conditional r-squared or not. Indicates up to which value the convergence result is accepted. When r2_nakagawa() returns a warning, stating that random effect variances can't be computed (and thus, the conditional r-squared is NA), decrease the tolerance-level. See also check_singularity.

Value

Returns a list containing values related to the most appropriate R2 for the given model. See the list below:

- Logistic models: Tjur's R2
- General linear models: Nagelkerke's R2
- Multinomial Logit: McFadden's R2
- Models with zero-inflation: R2 for zero-inflated models
- Mixed models: Nakagawa's R2
- Bayesian models: R2 bayes

See Also

r2_bayes, r2_coxsnell, r2_kullback, r2_loo, r2_mcfadden, r2_nagelkerke, r2_nakagawa, r2_tjur, r2_xu and r2_zeroinflated.

Examples

```
model <- glm(vs ~ wt + mpg, data = mtcars, family = "binomial")
r2(model)

if (require("lme4")) {
  model <- lmer(Sepal.Length ~ Petal.Length + (1 | Species), data = iris)
  r2(model)
}
```

---

r2_bayes *Bayesian R2*

Description

Compute R2 for Bayesian models. For mixed models (including a random part), it additionally computes the R2 related to the fixed effects only (marginal R2). While `r2_bayes()` returns a single R2 value, `r2_posterior()` returns a posterior sample of Bayesian R2 values.

Usage

```
r2_bayes(model, robust = TRUE, ci = 0.89, verbose = TRUE, ...)

r2_posterior(model, ...)

## S3 method for class 'brmsfit'
r2_posterior(model, verbose = TRUE, ...)

## S3 method for class 'stanreg'
r2_posterior(model, verbose = TRUE, ...)

## S3 method for class 'BFBayesFactor'
r2_posterior(model, average = FALSE, prior_odds = NULL, ...)
```

Arguments

`model`	A Bayesian regression model (from **brms**, **rstanarm**, **BayesFactor**, etc).
`robust`	Logical, if `TRUE`, the median instead of mean is used to calculate the central tendency of the variances.
`ci`	Value or vector of probability of the CI (between 0 and 1) to be estimated.
`verbose`	Toggle off warnings.
`...`	Arguments passed to `r2_posterior()`.
`average`	Compute model-averaged index? See `bayestestR::weighted_posteriors()`.
`prior_odds`	Optional vector of prior odds for the models compared to the first model (or the denominator, for `BFBayesFactor` objects). For `data.frames`, this will be used as the basis of weighting.

Details

`r2_bayes()` returns an "unadjusted" R2 value. See `r2_loo` to calculate a LOO-adjusted R2, which comes conceptionally closer to an adjusted R2 measure.

For mixed models, the conditional and marginal R2 are returned. The marginal R2 considers only the variance of the fixed effects, while the conditional R2 takes both the fixed and random effects into account.

`r2_posterior()` is the actual workhorse for `r2_bayes()` and returns a posterior sample of Bayesian R2 values.

Value

A list with the Bayesian R2 value. For mixed models, a list with the Bayesian R2 value and the marginal Bayesian R2 value. The standard errors and credible intervals for the R2 values are saved as attributes.

References

Gelman, A., Goodrich, B., Gabry, J., & Vehtari, A. (2018). R-squared for Bayesian regression models. The American Statistician, 1–6. doi: 10.1080/00031305.2018.1549100

Examples

```
library(performance)
if (require("rstanarm") && require("rstantools")) {
  model <- stan_glm(mpg ~ wt + cyl, data = mtcars, chains = 1, iter = 500, refresh = 0)
  r2_bayes(model)

  model <- stan_lmer(
    Petal.Length ~ Petal.Width + (1 | Species),
    data = iris,
    chains = 1,
    iter = 500,
    refresh = 0
  )
  r2_bayes(model)
}


## Not run:
if (require("BayesFactor")) {
  data(mtcars)

  BFM <- generalTestBF(mpg ~ qsec + gear, data = mtcars, progress = FALSE)
  FM <- lm(mpg ~ qsec + gear, data = mtcars)

  r2_bayes(FM)
  r2_bayes(BFM[3])
  r2_bayes(BFM, average = TRUE) # across all models
```

```
  # with random effects:
  mtcars$gear <- factor(mtcars$gear)
  model <- lmBF(
    mpg ~ hp + cyl + gear + gear:wt,
    mtcars,
    progress = FALSE,
    whichRandom = c("gear", "gear:wt")
  )
  r2_bayes(model)
}

if (require("brms")) {
  model <- brms::brm(mpg ~ wt + cyl, data = mtcars)
  r2_bayes(model)

  model <- brms::brm(Petal.Length ~ Petal.Width + (1 | Species), data = iris)
  r2_bayes(model)
}

## End(Not run)
```

`r2_coxsnell` *Cox & Snell's R2*

Description

Calculates the pseudo-R2 value based on the proposal from *Cox & Snell (1989)*.

Usage

```
r2_coxsnell(model)
```

Arguments

`model`	Model with binary outcome.

Details

This index was proposed by *Cox & Snell (1989, pp. 208-9)* and, apparently independently, by *Magee (1990)*; but had been suggested earlier for binary response models by *Maddala (1983)*. However, this index achieves a maximum of less than 1 for discrete models (i.e. models whose likelihood is a product of probabilities) which have a maximum of 1, instead of densities, which can become infinite *(Nagelkerke, 1991)*.

Value

A named vector with the R2 value.

References

- Cox, D. R., Snell, E. J. (1989). Analysis of binary data (Vol. 32). Monographs on Statistics and Applied Probability.
- Magee, L. (1990). R 2 measures based on Wald and likelihood ratio joint significance tests. The American Statistician, 44(3), 250-253.
- Maddala, G. S. (1986). Limited-dependent and qualitative variables in econometrics (No. 3). Cambridge university press.
- Nagelkerke, N. J. (1991). A note on a general definition of the coefficient of determination. Biometrika, 78(3), 691-692.

Examples

```
model <- glm(vs ~ wt + mpg, data = mtcars, family = "binomial")
r2_coxsnell(model)
```

---

r2_efron *Efron's R2*

Description

Calculates Efron's pseudo R2.

Usage

```
r2_efron(model)
```

Arguments

model	Generalized linear model.

Details

Efron's R2 is calculated by taking the sum of the squared model residuals, divided by the total variability in the dependent variable. This R2 equals the squared correlation between the predicted values and actual values, however, note that model residuals from generalized linear models are not generally comparable to those of OLS.

Value

The R2 value.

References

- Efron, B. (1978). Regression and ANOVA with zero-one data: Measures of residual variation. Journal of the American Statistical Association, 73, 113-121.

Examples

```
## Dobson (1990) Page 93: Randomized Controlled Trial:
counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12) #
outcome <- gl(3, 1, 9)
treatment <- gl(3, 3)
model <- glm(counts ~ outcome + treatment, family = poisson())

r2_efron(model)
```

`r2_kullback` *Kullback-Leibler R2*

Description

Calculates the Kullback-Leibler-divergence-based R2 for generalized linear models.

Usage

```
r2_kullback(model, adjust = TRUE)
```

Arguments

`model`	A generalized linear model.
`adjust`	Logical, if `TRUE` (the default), the adjusted R2 value is returned.

Value

A named vector with the R2 value.

References

Cameron, A. C. and Windmeijer, A. G. (1997) An R-squared measure of goodness of fit for some common nonlinear regression models. Journal of Econometrics, 77: 329-342.

Examples

```
model <- glm(vs ~ wt + mpg, data = mtcars, family = "binomial")
r2_kullback(model)
```

r2_loo *LOO-adjusted R2*

Description

Compute LOO-adjusted R2.

Usage

```
r2_loo(model, verbose = TRUE)
```

Arguments

model	A Bayesian regression model.
verbose	Toggle off warnings.

Details

Unlike r2_bayes, which returns an "unadjusted" R2 value, r2_loo() calculates a LOO-adjusted R2, which comes conceptionally closer to an "adjusted" R2 measure.

Value

The LOO-adjusted R2 for model, as numeric value.

Examples

```
if (require("rstanarm")) {
  model <- stan_glm(mpg ~ wt + cyl, data = mtcars, chains = 1, iter = 500, refresh = 0)
  r2_loo(model)
}
```

r2_mcfadden *McFadden's R2*

Description

Calculates McFadden's pseudo R2.

Usage

```
r2_mcfadden(model)
```

Arguments

model	Generalized linear or multinomial logit (mlogit) model.

Value

For most models, a list with McFadden's R2 and adjusted McFadden's R2 value. For some models, only McFadden's R2 is available.

References

- McFadden, D. (1987). Regression-based specification tests for the multinomial logit model. Journal of econometrics, 34(1-2), 63-82.
- McFadden, D. (1973). Conditional logit analysis of qualitative choice behavior.

Examples

```
if (require("mlogit")) {
  data("Fishing", package = "mlogit")
  Fish <- mlogit.data(Fishing, varying = c(2:9), shape = "wide", choice = "mode")

  model <- mlogit(mode ~ price + catch, data = Fish)
  r2_mcfadden(model)
}
```

---

r2_mckelvey *McKelvey & Zavoinas R2*

Description

Calculates McKelvey & Zavoinas pseudo R2.

Usage

```
r2_mckelvey(model)
```

Arguments

model	Generalized linear model.

Details

McKelvey & Zavoinas R2 is based on the explained variance, where the variance of the predicted response is divided by the sum of the variance of the predicted response and residual variance. For binomial models, the residual variance is either `pi^2/3` for logit-link and 1 for probit-link. For poisson-models, the residual variance is based on log-normal approximation, similar to the *distribution-specific variance* as described in `?insight::get_variance`.

Value

The R2 value.

References

- McKelvey, R., Zavoina, W. (1975), "A Statistical Model for the Analysis of Ordinal Level Dependent Variables", Journal of Mathematical Sociology 4, S. 103–120.

Examples

```
## Dobson (1990) Page 93: Randomized Controlled Trial:
counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12) #
outcome <- gl(3, 1, 9)
treatment <- gl(3, 3)
model <- glm(counts ~ outcome + treatment, family = poisson())

r2_mckelvey(model)
```

---

r2_nagelkerke *Nagelkerke's R2*

Description

Calculate Nagelkerke's pseudo-R2.

Usage

```
r2_nagelkerke(model)
```

Arguments

model	A generalized linear model, including cumulative links resp. multinomial models.

Value

A named vector with the R2 value.

References

Nagelkerke, N. J. (1991). A note on a general definition of the coefficient of determination. Biometrika, 78(3), 691-692.

Examples

```
model <- glm(vs ~ wt + mpg, data = mtcars, family = "binomial")
r2_nagelkerke(model)
```

r2_nakagawa *Nakagawa's R2 for mixed models*

Description

Compute the marginal and conditional r-squared value for mixed effects models with complex random effects structures.

Usage

```
r2_nakagawa(model, by_group = FALSE, tolerance = 1e-05)
```

Arguments

model	A mixed effects model.
by_group	Logical, if TRUE, returns the explained variance at different levels (if there are multiple levels). This is essentially similar to the variance reduction approach by *Hox (2010), pp. 69-78.*
tolerance	Tolerance for singularity check of random effects, to decide whether to compute random effect variances for the conditional r-squared or not. Indicates up to which value the convergence result is accepted. When r2_nakagawa() returns a warning, stating that random effect variances can't be computed (and thus, the conditional r-squared is NA), decrease the tolerance-level. See also check_singularity.

Details

Marginal and conditional r-squared values for mixed models are calculated based on *Nakagawa et al. 2017*. For more details on the computation of the variances, see ?insight::get_variance.

The marginal r-squared considers only the variance of the fixed effects, while the conditional r-squared takes both the fixed and random effects into account. The random effect variances are actually the mean random effect variances, thus the r-squared value is also appropriate for mixed models with random slopes or nested random effects (see *Johnson 2014*).

Value

A list with the conditional and marginal R2 values.

References

- Hox, J. J. (2010). Multilevel analysis: techniques and applications (2nd ed). New York: Routledge.
- Johnson, P. C. D. (2014). Extension of Nakagawa & Schielzeth's R2 GLMM to random slopes models. Methods in Ecology and Evolution, 5(9), 944–946. doi: 10.1111/2041210X.12225

- Nakagawa, S., & Schielzeth, H. (2013). A general and simple method for obtaining R2 from generalized linear mixed-effects models. Methods in Ecology and Evolution, 4(2), 133–142. doi: 10.1111/j.2041210x.2012.00261.x
- Nakagawa, S., Johnson, P. C. D., & Schielzeth, H. (2017). The coefficient of determination R2 and intra-class correlation coefficient from generalized linear mixed-effects models revisited and expanded. Journal of The Royal Society Interface, 14(134), 20170213. doi: 10.1098/rsif.2017.0213

Examples

```
if (require("lme4")) {
  model <- lmer(Sepal.Length ~ Petal.Length + (1 | Species), data = iris)
  r2_nakagawa(model)
  r2_nakagawa(model, by_group = TRUE)
}
```

---

r2_somers *Somers' Dxy rank correlation for binary outcomes*

Description

Calculates the Somers' Dxy rank correlation for logistic regression models.

Usage

```
r2_somers(model)
```

Arguments

model	A logistic regression model.

Value

A named vector with the R2 value.

References

Somers, R. H. (1962). A new asymmetric measure of association for ordinal variables. American Sociological Review. 27 (6).

Examples

```
## Not run:
if (require("correlation")) {
  model <- glm(vs ~ wt + mpg, data = mtcars, family = "binomial")
  r2_somers(model)
}
## End(Not run)
```

r2_tjur *Tjur's R2 - coefficient of determination (D)*

Description

This method calculates the Coefficient of Discrimination D (also known as Tjur's R2; *Tjur, 2009*) for generalized linear (mixed) models for binary outcomes. It is an alternative to other pseudo-R2 values like Nagelkerke's R2 or Cox-Snell R2. The Coefficient of Discrimination D can be read like any other (pseudo-)R2 value.

Usage

```
r2_tjur(model)
```

Arguments

model	Binomial Model.

Value

A named vector with the R2 value.

References

Tjur, T. (2009). Coefficients of determination in logistic regression models - A new proposal: The coefficient of discrimination. The American Statistician, 63(4), 366-372.

Examples

```
model <- glm(vs ~ wt + mpg, data = mtcars, family = "binomial")
r2_tjur(model)
```

r2_xu *Xu' R2 (Omega-squared)*

Description

Calculates Xu' Omega-squared value, a simple R2 equivalent for linear mixed models.

Usage

```
r2_xu(model)
```

Arguments

model	A linear (mixed) model.

Details

`r2_xu()` is a crude measure for the explained variance from linear (mixed) effects models, which is originally denoted as Ω^2.

Value

The R2 value.

References

Xu, R. (2003). Measuring explained variation in linear mixed effects models. Statistics in Medicine, 22(22), 3527–3541. doi: 10.1002/sim.1572

Examples

```
model <- lm(Sepal.Length ~ Petal.Length + Species, data = iris)
r2_xu(model)
```

---

r2_zeroinflated *R2 for models with zero-inflation*

Description

Calculates R2 for models with zero-inflation component, including mixed effects models.

Usage

```
r2_zeroinflated(model, method = c("default", "correlation"))
```

Arguments

`model`	A model.
`method`	Indicates the method to calculate R2. See 'Details'. May be abbreviated.

Details

The default-method calculates an R2 value based on the residual variance divided by the total variance. For `method = "correlation"`, R2 is a correlation-based measure, which is rather crude. It simply computes the squared correlation between the model's actual and predicted response.

Value

For the default-method, a list with the R2 and adjusted R2 values. For `method = "correlation"`, a named numeric vector with the correlation-based R2 value.

Examples

```
if (require("pscl")) {
  data(bioChemists)
  model <- zeroinfl(
    art ~ fem + mar + kid5 + ment | kid5 + phd,
    data = bioChemists
  )

  r2_zeroinflated(model)
}
```

---

test_bf *Test if Models are Different*

Description

Testing whether models are different is a delicate and often complex procedure, with many limitations and requisites. Moreover, several tests are available, each coming with its own interpretation, and set of strengths and weaknesses.

The `test_performance()` function is available to run the most relevant and appropriate tests based on the input (for instance, whether the models are *nested* or not). That said, it still requires the user to understand what the tests are and what they do in order to prevent their misinterpretation. See the **details** section for more information regarding the different tests and their interpretation.

Usage

```
test_bf(...)

test_likelihoodratio(..., estimator = "ML")

performance_lrt(..., estimator = "ML")

test_performance(..., reference = 1)

test_vuong(...)

test_wald(...)
```

Arguments

`...`	Multiple model objects.
`estimator`	Applied when comparing regression models using `test_likelihoodratio()`. Corresponds to the different estimators for the standard deviation of the errors. If `estimator="OLS"` (default), then it uses the same method as `anova(...,test="LRT")`

implemented in base R, i.e., scaling by n-k (the unbiased OLS estimator) and using this estimator under the alternative hypothesis. If `estimator="ML"`, which is for instance used by `lrtest(...)` in package **lmtest**, the scaling is done by n (the biased ML estimator) and the estimator under the null hypothesis. In moderately large samples, the differences should be negligible, but it is possible that OLS would perform slightly better in small samples with Gaussian errors.

`reference` This only applies when models are non-nested, and determines which model should be taken as a reference, against which all the other models are tested.

Details

Nested vs. Non-nested Models: Model's "nesting" is an important concept of models comparison. Indeed, many tests only make sense when the models are *"nested"*, i.e., when their predictors are nested. This means that all the predictors of a model are contained within the predictors of a larger model (sometimes referred to as the encompassing model). For instance, `model1 (y ~ x1 + x2)` is "nested" within `model2 (y ~ x1 + x2 + x3)`. Usually, people have a list of nested models, for instance `m1 (y ~ x1 + x2 + x3)`, `m2 (y ~ x1 + x2)`, `m3 (y ~ x1)`, `m4 (y ~ 1)`, and they are "ordered" from the largest to smallest on vice versa, to test whether a more parsimonious model, or whether adding a predictor, results in a significant difference in the model's performance. In this case, models are usually compared *sequentially*: m2 is tested against m1, m3 against m2, m4 against m3, etc.

Two models are considered as *"non-nested"* if their predictors are different. For instance, `model1 (y ~ x1 + x2)` and `model2 (y ~ x3 + x4)`. In the case of non-nested models, all models are usually compared against the same *reference* model (by default, the first of the list).

Nesting is detected via the `insight::is_nested_models()` function. Aside of the nesting, note also that in order for the tests to be valid, other requirements have often to be the fulfilled. For instance, outcome variables (the response) must be the same. You cannot meaningfully test whether apples are significantly different from oranges!

Tests Description:

- **Bayes factor for Model Comparison** - `test_bf()`: If all models were fit from the same data, the returned `BF` shows the Bayes Factor (see `bayestestR::bayesfactor_models()`) for each model against the reference model (which depends on whether the models are nested or not). Check out this vignette for more details.
- **Wald's F-Test** - `test_wald()`: The Wald test is a rough approximation of the Likelihood Ratio Test. However, it is more applicable than the LRT: you can often run a Wald test in situations where no other test can be run. Importantly, this test only makes statistical sense if the models are nested.

  This test is also available in base R through the `anova()` function. It returns an `F-value` column as a statistic and its associated `p-value`.
- **Likelihood Ratio Test (LRT)** - `test_likelihoodratio()`: The LRT tests which model is a better (more likely) explanation of the data. Likelihood-Ratio-Test (LRT) gives usually somewhat close results (if not equivalent) to the Wald test and, similarly, only makes sense for nested models. However, Maximum likelihood tests make stronger assumptions than method of moments tests like the F-test, and in turn are more efficient. Agresti (1990) suggests that

you should use the LRT instead of the Wald test for small sample sizes (under or about 30) or if the parameters are large.

For regression models, this is similar to `anova(...,test="LRT")` or `lmtest::lrtest(...)`, depending on the `estimator` argument. For `lavaan` models (SEM, CFA), the function calls `lavaan::lavTestLRT()`.

- **Vuong's Test** - `test_vuong()`: Vuong's (1989) test can be used both for nested and non-nested models, and actually consists of two tests.
  - The **Test of Distinguishability** (the `Omega2` column and its associated p-value) indicates whether or not the models can possibly be distinguished on the basis of the observed data. If its p-value is significant, it means the models are distinguishable.
  - The **Robust Likelihood Test** (the `LR` column and its associated p-value) indicates whether each model fits better than the reference model. If the models are nested, then the test works as a robust LRT. The code for this function is adapted from the `nonnest2` package, and all credit go to their authors.

Value

A data frame containing the relevant indices.

References

- Vuong, Q. H. (1989). Likelihood ratio tests for model selection and non-nested hypotheses. Econometrica, 57, 307-333.
- Merkle, E. C., You, D., & Preacher, K. (2016). Testing non-nested structural equation models. Psychological Methods, 21, 151-163.

See Also

`compare_performance()` to compare the performance indices of many different models.

Examples

```
# Nested Models
# -------------
m1 <- lm(Sepal.Length ~ Petal.Width * Species, data = iris)
m2 <- lm(Sepal.Length ~ Petal.Width + Species, data = iris)
m3 <- lm(Sepal.Length ~ Petal.Width, data = iris)

test_performance(m1, m2, m3)

test_bf(m1, m2, m3)
test_wald(m1, m2, m3) # Equivalent to anova(m1, m2, m3)

# Equivalent to lmtest::lrtest(m1, m2, m3)
test_likelihoodratio(m1, m2, m3, estimator = "ML")

# Equivalent to anova(m1, m2, m3, test='LRT')
test_likelihoodratio(m1, m2, m3, estimator = "OLS")
```

```
test_vuong(m1, m2, m3) # nonnest2::vuongtest(m1, m2, nested=TRUE)

# Non-nested Models
# -----------------
m1 <- lm(Sepal.Length ~ Petal.Width, data = iris)
m2 <- lm(Sepal.Length ~ Petal.Length, data = iris)
m3 <- lm(Sepal.Length ~ Species, data = iris)

test_performance(m1, m2, m3)
test_bf(m1, m2, m3)
test_vuong(m1, m2, m3) # nonnest2::vuongtest(m1, m2)

# Tweak the output
# ----------------
test_performance(m1, m2, m3, include_formula = TRUE)


# SEM / CFA (lavaan objects)
# --------------------------
# Lavaan Models
if (require("lavaan")) {
  structure <- " visual  =~ x1 + x2 + x3
                 textual =~ x4 + x5 + x6
                 speed   =~ x7 + x8 + x9

                  visual ~~ textual + speed "
  m1 <- lavaan::cfa(structure, data = HolzingerSwineford1939)

  structure <- " visual  =~ x1 + x2 + x3
                 textual =~ x4 + x5 + x6
                 speed   =~ x7 + x8 + x9

                  visual ~~ 0 * textual + speed "
  m2 <- lavaan::cfa(structure, data = HolzingerSwineford1939)

  structure <- " visual  =~ x1 + x2 + x3
                 textual =~ x4 + x5 + x6
                 speed   =~ x7 + x8 + x9

                  visual ~~ 0 * textual + 0 * speed "
  m3 <- lavaan::cfa(structure, data = HolzingerSwineford1939)

  test_likelihoodratio(m1, m2, m3)

  # Different Model Types
  # ---------------------
  if (require("lme4") && require("mgcv")) {
    m1 <- lm(Sepal.Length ~ Petal.Length + Species, data = iris)
    m2 <- lmer(Sepal.Length ~ Petal.Length + (1 | Species), data = iris)
    m3 <- gam(Sepal.Length ~ s(Petal.Length, by = Species) + Species, data = iris)

    test_performance(m1, m2, m3)
  }
```

```
}
```

Index