Package: GLAMMGoF (via r-universe)

May 19, 2026

Title Resampling-Based Predictive Validation for Generalized Linear and Generalized Additive Models

Date 2026-05-14

URL https://github.com/colinpshea/glammgof

Version 1.0.7

Description Provides resampling-based predictive validation for generalized linear and generalized additive models (with or without random effects) fitted using packages such as glmmTMB, mgcv, stats, lme4, and MASS. Predictive performance is assessed using either repeated random holdout (Monte Carlo cross-validation) or bootstrap resampling with out-of-bag evaluation. In each replicate, models are refit to a training dataset and evaluated on separate testing data, generating sampling distributions of in-sample and out-of-sample performance statistics. For continuous or integer response models, supported metrics include relative root mean squared error (RRMSE), relative mean absolute error (RMAE), relative median absolute error (RMedAE), and relative bias (RBIAS). For binary response models, supported metrics include AUC, Brier score, and log loss. All predictive metrics are based on population-level predictions, meaning random effects are excluded when present. Optional residual diagnostics can also be performed using the DHARMa package.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

LazyData true

Imports dplyr ,tidyr ,rms (>= 6.0-0) ,DHARMa (>= 0.4.0) ,glmmTMB (>= 1.1.5) ,lme4 ,MASS ,magrittr ,mgcViz ,mgcv ,rlang

Depends ggplot2, R (>= 3.5)

Suggests knitr, rmarkdown

VignetteBuilder knitr

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev zlib1g-dev

Repository https://colinpshea.r-universe.dev

Date/Publication 2026-05-19 17:27:27 UTC

RemoteUrl https://github.com/colinpshea/GLAMMGoF

RemoteRef HEAD

RemoteSha 53c63557570915baf9e7f1016cbaca15f3029877

Contents

`bias_precision` *Bootstrap or Monte Carlo assessment of RRMSE, RMAE, RMedAE, and RBIAS predictive performance statistics*

Description

Assess in- and out-of-sample predictive performance of generalized linear and generalized additive models with continuous or integer response variables and with or without random effects and zero-inflation, using either repeated random holdout (Monte Carlo cross-validation) or bootstrap resampling with out-of-bag evaluation. Four performance statistics are reported: relative root mean squared error (RRMSE), relative Mean Absolute Error (RMAE), relative median absolute error (RMedAE), and relative bias (RBIAS). Note that all performance measures are based on population-level predictions (i.e., random effects are ignored). For models with a zero-inflation component, predictions account for zero-inflation (e.g., for glmmTMB, the predicted value represents the product of the mean_count and (1 - prob_zero)).

All three accuracy statistics (RRMSE, RMAE, and RMedAE) express prediction error as a percentage of the mean observed value, which makes them interpretable on a common scale regardless of the units or magnitude of the response. To recover the corresponding raw error in the original units of the response, multiply any of these values by the observed mean and divide by 100. For example, an RRMSE of 18% with a sample mean of 50 implies a root mean squared error of 9 in the original units.

RRMSE (Relative Root Mean Squared Error): The square root of average squared prediction errors, expressed as a percentage of the mean observed value. Because errors are squared before averaging, RRMSE penalizes large individual errors more heavily than small ones, making it sensitive to cases where the model produces occasional extreme mispredictions.

RRMSE = sqrt(mean((predicted - observed)^2))/mean(observed)*100

RMAE (Relative Mean Absolute Error): The average absolute prediction error expressed as a percentage of the mean observed value. RMAE treats all errors proportionally to their size without the extra weight that squaring applies, and is often the most intuitive summary of typical prediction accuracy.

RMAE = mean(abs((predicted - observed)))/mean(observed)*100

RMedAE (Relative Median Absolute Error): The median absolute prediction error expressed as a percentage of the mean observed value. Because it is based on the median rather than the mean of the error distribution, RMedAE is the most robust of the three to outlying predictions and gives the best picture of accuracy for a typical observation when the error distribution is skewed.

RMedAE = median(abs((predicted - observed)))/mean(observed)*100

Taken together, these three statistics tell a more complete story than any one alone. If RRMSE is notably larger than RMAE, that signals a handful of high-leverage mispredictions are inflating the squared-error average, which is a pattern RMedAE will often fail to reflect if the bulk of predictions are accurate. Conversely, close agreement among all three suggests errors are roughly symmetric and there are no extreme outliers driving the summary.

RBIAS (Relative Bias): The mean signed prediction error expressed as a percentage of the mean observed value, where positive values indicate systematic over-prediction and negative values indicate systematic under-prediction. RBIAS is independent of the accuracy metrics above: a model can be nearly unbiased on average yet still produce large errors, or it can be highly biased while still ranking observations correctly. Reporting RBIAS alongside RRMSE and RMAE therefore distinguishes random prediction noise from systematic directional error.

RBIAS = mean((predicted - observed))/mean(observed)*100

Usage

```
bias_precision(
  nReps = 100,
  testModel = NULL,
  testData = NULL,
  propTrain = 0.8,
  DHARMaPlot = TRUE,
  testZI = TRUE,
  DHARMaReps = 1000,
  seed = NULL,
  method = c("holdout", "bootstrap")
)
```

Arguments

nReps	Desired number of bootstrap or Monte Carlo replicates. The default value is 100, but this number should be at least 1000 in practice.
testModel	A regression model fit to testData in glmmTMB (with or without random effects), glmer/glmer.nb/lmer (with random effects), glm/glm.nb/lm (without random effects), or gam (with or without random effects). The response variable can be continuous or an integer, and possible statistical distributions include Poisson, negative binomial, gamma, tweedie, and gaussian.
testData	A data frame with a continuous or integer response variable and continuous and/or categorical predictors.
propTrain	The proportion of testData used for model-fitting and in-sample predictive performance when method = holdout (the default value is 0.8). The remaining proportion is used to assess out-of-sample predictive performance. This argument is ignored when method = bootstrap.
DHARMaPlot	Do you want to return a goodness-of-fit plot from the simulateResiduals() function of the DHARMa package? The default is TRUE.
testZI	Logical. If TRUE and DHARMaPlot = TRUE, runs testZeroInflation on the simulated residuals. Most relevant for count models (Poisson, negative binomial, ZIP, hurdle). Default is TRUE.
DHARMaReps	You can also specify DHARMaReps if you want something other than the default of 1000 simulation replicates.
seed	Optional integer seed for reproducibility. If NULL (the default), no seed is set and results will differ across runs.
method	The resampling method to use. The default, holdout, repeatedly splits the data into random training and testing data sets (Monte Carlo cross-validation), whereas bootstrap samples the training data with replacement and evaluates in-sample performance on the bootstrap sample and out-of-sample performance on the out-of-bag observations not selected in the bootstrap sample (approximately 36.8% of observations on average). For well-behaved models and reasonably sized datasets, both methods should produce similar results; differences are most likely to emerge with small datasets, highly overdispersed data, or poorly specified models.

Value

This function returns either three, four, or five objects depending on the values of DHARMaPlot and testZI: a data frame with all bootstrapping or Monte Carlo resampling results (i.e., all nReps values for each performance statistic), a data frame with a summary (mean and 95% confidence intervals) of all replicates for each performance statistic, and a histogram of values for each performance statistic. If DHARMaPlot = TRUE, a fourth object is also returned: a goodness-of-fit plot based on scaled residuals from simulateResiduals(). If testZI = TRUE and DHARMaPlot = TRUE, a fifth object dharmaZI is also returned containing the result of testZeroInflation(). In the histogram, a blue dotted vertical line indicates the mean across replicates.

This package contains an example data set for fitting a negative binomial or Poisson regression called countData. Six example negative binomial regression model objects are also included: countModel_GLM is a GLM with no random effects; countModel_GLMM is a GLMM with one random

effect; countModel_GLMM2 is a GLMM with two random effects; countModel_GAM is a GAM with no random effects; countModel_GAMM is a GAMM with one random effect; and countModel_GAMM2 is a GAMM with two random effects. GLMs and GLMMs were fitted using glmmTMB, whereas GAMs and GAMMs were fitted using mgcv:

countModel_GLM <- glmmTMB(y ~ Season + Temp, family = nbinom2, data = countData)

countModel_GLMM <- glmmTMB(y ~ Season + Temp + (1|Site), family = nbinom2, data = countData)

countModel_GLMM2 <- glmmTMB(y ~ Season + Temp + (1|Site) + (1|Year), family = nbinom2, data = countData)

countModel_GAM <- gam(y ~ Season + s(Temp), family = nb, data = countData)

countModel_GAMM <- gam(y ~ Season + s(Temp) + s(Site, bs = "re"), family = nb, data = countData)

countModel_GAMM2 <- gam(y ~ Season + s(Temp) + s(Site, bs = "re") + s(Year, bs = "re"), family = nb, data = countData)

Bootstrapping or Monte Carlo resampling of the performance statistics requires specifying the data and model being tested, the desired number of replicates (the default is 100 but should be at least 1000 in practice), the proportion of data used for training when `method = "holdout"` (the default is 0.8), whether to use `DHARMa` residual diagnostics (the default is `TRUE`), whether to use `DHARMa` to test for zero-inflation (the default is `TRUE`), the number of `DHARMa` simulation replicates (the default is 1000), and an optional integer seed for reproducibility, and the resampling method `holdout` or `bootstrap`:

bias_precision(nReps = 100, testModel = countModel_GLMM, testData = countData, propTrain = 0.8, DHARMaPlot = TRUE, testZI = TRUE, DHARMaReps = 1000, seed = 123, method = "holdout")

Note

This function does not currently support binomial models with cbind() or proportion responses, and for binary 0/1 responses, use brier_auc(). This function also supports models with spatial random effects (e.g, in glmmTMB), but it is much slower than for more conventional GLM(M)s and GAM(M)s.

References

Hyndman, R.J. and Koehler, A.B. (2006) Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22, 679–688.

`brier_auc` *Bootstrap or Monte Carlo assessment of AUC, Brier score, and log loss predictive performance statistics*

Description

Assess in- and out-of-sample predictive performance of generalized linear and generalized additive models with binary response variables and with or without random effects, using either repeated random holdout (Monte Carlo cross-validation) or bootstrap resampling with out-of-bag evaluation. Three performance statistics are reported: Brier scores (see the `rms` package documentation for details), which range from 0 to 1 with values closer to 0 indicating a better-predicting model and where sqrt(Brier score) is the average difference between the predicted probability and the observed value (0 or 1); AUC, an aggregated metric that evaluates how well a model classifies positive and negative outcomes at all possible probability cutoffs, ranging from 0 to 1 with values closer to 1 indicating a better classifier and where an AUC of 0.5 suggests performance no better than random guessing; and log loss (cross-entropy), which penalizes confident wrong predictions logarithmically and is particularly sensitive to miscalibration at the extremes of the predicted probability distribution, with lower values indicating better performance and no upper bound. Note that all performance measures are based on population-level predictions (i.e., random effects are ignored when present).

The three binary metrics each capture a different aspect of predictive performance and are most informative when interpreted together.
`AUC (Area Under the ROC Curve)`: The probability that a randomly chosen positive case receives a higher predicted probability than a randomly chosen negative case, ranging from 0 (no discrimination) to 1 (perfect discrimination), and where an AUC of 0.5 indicates that a model predicts no better than random chance (and AUC <0.5 is worse than random). AUC is purely a measure of discrimination (the model's ability to rank cases by their probability of the outcome) and is insensitive to how well the predicted probabilities are calibrated in absolute terms.
`Brier Score`: The mean squared difference between predicted probabilities and observed binary outcomes (0/1), ranging from 0 (perfect) to 1 (worst). The Brier score jointly rewards good discrimination and good calibration, penalizing confident but wrong predictions more heavily than uncertain ones. A model can have high AUC but a poor Brier score if its predicted probabilities are systematically too high or too low even while correctly ordering cases.
`Log Loss (Binary Cross-Entropy)`: The negative mean log-likelihood of the observed outcomes under the predicted probabilities, where lower values indicate better performance. Like the Brier score, log loss rewards calibration, but its logarithmic scale means it penalizes extreme, confident mispredictions far more aggressively. Log loss is therefore the most sensitive of the three to cases where the model assigns near-zero probability to an outcome that actually occurs.

Reporting all three together is recommended: AUC reflects whether the model correctly separates cases from non-cases; Brier score provides an overall summary of probability accuracy on a squared-error scale; and log loss highlights whether the model is making any dangerously overconfident errors that the Brier score might underweight.

Usage

```
brier_auc(
  nReps = 100,
  testModel = NULL,
  testData = NULL,
  propTrain = 0.8,
  DHARMaPlot = TRUE,
  DHARMaReps = 1000,
  seed = NULL,
  method = c("holdout", "bootstrap")
```

```
)
```

Arguments

nReps	Desired number of bootstrap or Monte Carlo replicates. The default value is 100, but this number should be at least 1000 in practice.
testModel	A logistic regression model fitted to testData using glmmTMB (with or without random effects), glmer (with random effects), glm (without random effects), or gam (with or without random effects).
testData	A data frame with a binary response variable and continuous and/or categorical predictor variables.
propTrain	The proportion of testData used for model-fitting and in-sample predictive performance when method = holdout (the default value is 0.8). The remaining proportion is used to assess out-of-sample predictive performance. This argument is ignored when method = bootstrap.
DHARMaPlot	Do you want to return a goodness-of-fit plot from the simulateResiduals() function of the DHARMa package? The default is TRUE.
DHARMaReps	If DHARMaPlot is TRUE, you can also specify DHARMaReps if you want something other than the default of 1000 simulation replicates.
seed	Optional integer seed for reproducibility. If NULL (the default), no seed is set and results will differ across runs.
method	The resampling method to use. The default, holdout, repeatedly splits the data into random training and testing data sets (Monte Carlo cross-validation), whereas bootstrap samples the training data with replacement and evaluates in-sample performance on the bootstrap sample and out-of-sample performance on the out-of-bag observations not selected in the bootstrap sample (approximately 36.8% of observations on average). For well-behaved models and reasonably sized datasets, both methods should produce similar results; differences are most likely to emerge with small datasets, highly overdispersed data, or poorly specified models.

Value

This function returns four objects: a data frame with all of the bootstrapping or Monte Carlo resampling results (i.e., all nReps values for each performance statistic), a data frame with a summary (mean and 95% confidence intervals) of all replicates for each performance statistic, a histogram of values for each performance statistic, and a goodness-of-fit plot based on scaled residuals from the simulateResiduals() function of the DHARMa package. If DHARMaPlot = FALSE, then simulateResiduals() is not used to assess the model residuals and only three of the four objects are returned. This function also returns the null model (i.e., intercept-only) log loss, Brier score, and AUC statistic for reference (i.e., the best predictive performance achievable without any predictors), which allows for comparison to the summarized log loss, Brier, and AUC metrics. The null log loss is computed as `-(p_bar * log(p_bar) + (1 - p_bar) * log(1 - p_bar))`, where p_bar is the observed event rate. The null Brier score is computed as `p_bar * (1 - p_bar)`. The null AUC is 0.5, the expected value for a model with no discriminatory ability, equivalent to random classification. The null log loss and Brier score are based on the arithmetic mean of the response, which approximates but is not identical to `plogis(intercept)` from a fitted intercept-only logistic model; the

difference is negligible in practice and avoids the overhead of model fitting. In the histogram, a blue dotted vertical line indicates the mean of each metric across replicates, and a red dashed vertical line indicates the null model baseline for each metric: null log loss, null Brier score, and AUC = 0.5.

This package contains an example data set to fit a logistic regression called logitData. Six example logistic regression model objects are also included: logitModel_GLM is a GLM with no random effects; logitModel_GLMM is a GLMM with one random effect; logitModel_GLMM2 is a GLMM with two random effects; logitModel_GAM is a GAM with no random effects; logitModel_GAMM is a GAMM with one random effect; and logitModel_GAMM2 is a GAMM with two random effects. GLMs and GLMMs were fitted using glmmTMB, whereas GAMs and GAMMs were fitted using mgcv:

logitModel_GLM <- glmmTMB(y ~ Season + Temp, family = binomial, data = logitData)

logitModel_GLMM <- glmmTMB(y ~ Season + Temp + (1|Site), family = binomial, data = logitData)

logitModel_GLMM2 <- glmmTMB(y ~ Season + Temp + (1|Site) + (1|Year), family = binomial, data = logitData)

logitModel_GAM <- gam(y ~ Season + s(Temp), family = binomial, data = logitData)

logitModel_GAMM <- gam(y ~ Season + s(Temp) + s(Site, bs = "re"), family = binomial, data = logitData)

logitModel_GAMM2 <- gam(y ~ Season + s(Temp) + s(Site, bs = "re") + s(Year, bs = "re"), family = binomial, data = logitData)

Bootstrapping or Monte Carlo resampling of the performance statistics requires specifying the data and model being tested, the desired number of replicates (the default is 100 but should be at least 1000 in practice), the resampling method `holdout` or `bootstrap`, the proportion of data used for training when `method = "holdout"` (the default is 0.8), whether to use DHARMa residual diagnostics (the default is TRUE), the number of DHARMa simulation replicates (the default is 1000), and an optional integer seed for reproducibility:

brier_auc(nReps = 100, testModel = logitModel_GLMM, testData = logitData, propTrain = 0.8, DHARMaPlot = TRUE, DHARMaReps = 1000, seed = 123, method = "holdout")

Note

This function only supports binary 0/1 responses and does not currently support binomial models with cbind() or proportion responses. This function also supports models with spatial random effects (e.g, in glmmTMB), but it is much slower than for more conventional GLM(M)s and GAM(M)s.

`countData` *Simulated count data*

Description

A simulated dataset with an integer count response and predictor variables for demonstrating the bias_precision function.

Usage

```
countData
```

Format

A data frame with 1000 rows and 5 variables:

y Integer count response variable

Season Factor with 4 levels: Autumn, Spring, Summer, Winter

Temp Continuous temperature predictor

Site Factor with 10 levels representing sampling sites

Year Factor with 8 levels representing sampling years (2015-2022)

---

countModel_GAM *Simulated count GAM example model*

Description

A mgcv GAM fit to countData with fixed effects only. Formula: y ~ Season + s(Temp)

Usage

```
countModel_GAM
```

Format

A gam model object

---

countModel_GAMM *Simulated count GAMM example model*

Description

A mgcv GAM fit to countData with one random effect. Formula: y ~ Season + s(Temp) + s(Site, bs = "re")

Usage

```
countModel_GAMM
```

Format

A gam model object

countModel_GAMM2 *Simulated count GAMM2 example model*

Description

A mgcv GAM fit to countData with two random effects. Formula: y ~ Season + s(Temp) + s(Site, bs = "re") + s(Year, bs = "re")

Usage

```
countModel_GAMM2
```

Format

A gam model object

countModel_GLM *Simulated count GLM example model*

Description

A glmmTMB model fit to countData with fixed effects only. Formula: y ~ Season + Temp

Usage

```
countModel_GLM
```

Format

A glmmTMB model object

countModel_GLMM *Simulated count GLMM example model*

Description

A glmmTMB model fit to countData with one random effect. Formula: y ~ Season + Temp + (1 | Site)

Usage

```
countModel_GLMM
```

Format

A glmmTMB model object

countModel_GLMM2 *Simulated count GLMM2 example model*

Description

A glmmTMB model fit to countData with two random effects. Formula: y ~ Season + Temp + (1 | Site) + (1 | Year)

Usage

```
countModel_GLMM2
```

Format

A glmmTMB model object

logitData *Simulated binary data*

Description

A simulated dataset with a binary response and predictor variables for demonstrating the brier_auc function.

Usage

```
logitData
```

Format

A data frame with 1000 rows and 5 variables:

y Binary response variable (0 or 1)

Season Factor with 4 levels: Autumn, Spring, Summer, Winter

Temp Continuous temperature predictor

Site Factor with 10 levels representing sampling sites

Year Factor with 8 levels representing sampling years (2015-2022)

logitModel_GAM *Simulated binary GAM example model*

Description

A mgcv GAM fit to logitData with fixed effects only. Formula: y ~ Season + s(Temp)

Usage

```
logitModel_GAM
```

Format

A gam model object

logitModel_GAMM *Simulated binary GAMM example model*

Description

A mgcv GAM fit to logitData with one random effect. Formula: y ~ Season + s(Temp) + s(Site, bs = "re")

Usage

```
logitModel_GAMM
```

Format

A gam model object

logitModel_GAMM2 *Simulated binary GAMM2 example model*

Description

A mgcv GAM fit to logitData with two random effects. Formula: y ~ Season + s(Temp) + s(Site, bs = "re") + s(Year, bs = "re")

Usage

```
logitModel_GAMM2
```

Format

A gam model object

logitModel_GLM *Simulated binary GLM example model*

Description

A glmmTMB model fit to logitData with fixed effects only. Formula: y ~ Season + Temp

Usage

```
logitModel_GLM
```

Format

A glmmTMB model object

logitModel_GLMM *Simulated binary GLMM example model*

Description

A glmmTMB model fit to logitData with one random effect. Formula: y ~ Season + Temp + (1 | Site)

Usage

```
logitModel_GLMM
```

Format

A glmmTMB model object

logitModel_GLMM2 *Simulated binary GLMM2 example model*

Description

A glmmTMB model fit to logitData with two random effects. Formula: y ~ Season + Temp + (1 | Site) + (1 | Year)

Usage

```
logitModel_GLMM2
```

Format

A glmmTMB model object

Index