Package: mlumr (via r-universe)

May 20, 2026

Title Multilevel Unanchored Meta-Regression for Indirect Treatment Comparisons

Version 0.1.0

Description Bayesian multilevel unanchored meta-regression (ML-UMR) for indirect treatment comparisons using individual patient data (IPD) and aggregate data (AgD). Implements shared prognostic factor assumption (SPFA) and relaxed SPFA models for binary, continuous, and count outcomes via 'Stan'. Also provides simulated treatment comparison (STC) via parametric G-computation and naive unadjusted benchmarks. ML-UMR is an adaptation of the ML-NMR methodology (Phillippo et al. 2020, <doi:10.1111/rssa.12579>) implemented in the 'multinma' package (GPL-3) to the unanchored two-trial case; the public API deliberately mirrors multinma's so users can move between ML-NMR and ML-UMR with the same workflow.

License GPL-3

URL https://github.com/choxos/mlumr, https://choxos.github.io/mlumr/

BugReports https://github.com/choxos/mlumr/issues

Encoding UTF-8

Biarch true

Depends R (>= 4.1.0)

Imports methods, stats, Rcpp (>= 0.12.0), RcppParallel (>= 5.0.1), rstan (>= 2.26.0), rstantools (>= 2.3.0), randtoolbox, copula

LinkingTo BH (>= 1.66.0), Rcpp (>= 0.12.0), RcppEigen (>= 0.3.3.3.0), RcppParallel (>= 5.0.1), rstan (>= 2.26.0), StanHeaders (>= 2.26.0)

Suggests testthat (>= 3.0.0), knitr, rmarkdown, posterior, bayesplot, loo, Matrix, cmdstanr, withr

Additional_repositories https://stan-dev.r-universe.dev

SystemRequirements GNU make

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Ahmad Sofi-Mahmudi [aut, cre] (ORCID: <https://orcid.org/0000-0001-6829-0823>), Conor Chandler [aut] (ORCID: <https://orcid.org/0000-0002-1365-9002>)

Maintainer Ahmad Sofi-Mahmudi <a.sofimahmudi@gmail.com>

Config/pak/sysreqs make

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

Date/Publication 2026-05-20 08:30:02 UTC

RemoteUrl https://github.com/cran/mlumr

RemoteRef HEAD

RemoteSha 5a8d1ccb0a47949e856f0c2f952361a8f45285c2

Contents

`add_integration` *Add numerical integration points*

Description

Generate quasi-Monte Carlo integration points using Sobol sequences and a Gaussian copula to account for correlations between covariates in the AgD.

Usage

```
add_integration(
  data,
  n_int = 64,
  cor = NULL,
  cor_adjust = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`data`	An `mlumr_data` object from `combine_data()`
`n_int`	Number of integration points (default 64, powers of 2 recommended)
`cor`	Correlation matrix for covariates. If `NULL`, computed from IPD.
`cor_adjust`	Adjustment method: `"spearman"`, `"pearson"`, or `"none"`
`verbose`	Logical; if `FALSE`, suppresses progress messages.
`...`	Distribution specifications for each covariate using `distr()`

Value

An `mlumr_data` object with integration points added

Examples

```
## Not run:
dat <- add_integration(
  dat,
  n_int = 64,
  x1 = distr(qnorm, mean = x1_mean, sd = x1_sd),
  x2 = distr(qbern, prob = x2_mean)
)

## End(Not run)
```

calculate_dic *Calculate DIC for model comparison*

Description

Computes the Deviance Information Criterion using the variance-based effective-parameters formula from Gelman et al. (2004): `pD = 0.5 * Var(D)`. This is more stable than the plug-in alternative for multimodal posteriors.

Usage

```
calculate_dic(object)
```

Arguments

`object`	An `mlumr_fit` object

Details

DIC is retained for backward compatibility and rough comparison. For principled Bayesian model comparison, prefer `calculate_loo()` or `calculate_waic()` (Vehtari, Gelman, Gabry 2017).

Value

A list of class `mlumr_dic` with components `DIC`, `pD`, `D_bar`

Examples

```
## Not run:
dic_spfa <- calculate_dic(fit_spfa)
dic_relaxed <- calculate_dic(fit_relaxed)

## End(Not run)
```

calculate_loo *Calculate LOO-CV for an mlumr_fit*

Description

Computes approximate leave-one-out cross-validation (PSIS-LOO, Vehtari, Gelman, Gabry 2017) using the pointwise log-likelihoods stored by the Stan models. Returns a `loo` object from the `loo` package.

Usage

```
calculate_loo(object, ...)
```

Arguments

`object`	An `mlumr_fit` object.
`...`	Additional arguments passed to `loo::loo.array()`.

Details

Pareto-k diagnostics: values > 0.7 indicate observations for which the PSIS approximation is unreliable; the printed output flags these. Typical remedies are running more iterations, using `moment_match = TRUE`, or (for highly influential AgD rows) refitting without the offending observation to check sensitivity.

Value

An object of class `psis_loo` (see `loo::loo()`).

Note

AgD rows are treated as independent observations. Each AgD row contributes one column to the pointwise `log_lik` matrix. If two or more AgD rows come from the same study (e.g. subgroup summaries within a single trial) the PSIS-LOO approximation does not account for the within-study clustering; effective sample sizes are inflated and Pareto-k warnings are understated. For clustered AgD, corroborate with `prior_sensitivity()` or refit omitting suspect rows to check the influence on the posterior.

Examples

```
## Not run:
loo_spfa <- calculate_loo(fit_spfa)
print(loo_spfa)

## End(Not run)
```

`calculate_waic` *Calculate WAIC for an mlumr_fit*

Description

Watanabe-Akaike Information Criterion (Watanabe 2010) based on the pointwise log-likelihoods. WAIC is asymptotically equivalent to LOO-CV; prefer `calculate_loo()` when Pareto-k is well-behaved.

Usage

```
calculate_waic(object, ...)
```

Arguments

object	An mlumr_fit object.
...	Additional arguments passed to loo::waic().

Value

An object of class waic (see loo::waic()).

Note

As with calculate_loo(), each AgD row is treated as an independent observation. WAIC will be optimistic for AgD rows that share a study (see the note on calculate_loo()).

Examples

```
## Not run:
waic_spfa <- calculate_waic(fit_spfa)

## End(Not run)
```

---

check_integration *Check integration point adequacy*

Description

Compare integration results at the current n_int against a doubled resolution to assess numerical accuracy. Large discrepancies indicate that n_int should be increased.

Usage

```
check_integration(
  data,
  ...,
  cor = NULL,
  cor_adjust = NULL,
  check_joint = TRUE,
  verbose = TRUE
)
```

Arguments

data	An mlumr_data object with integration points
...	Distribution specifications (same as passed to add_integration())
cor	Correlation matrix (same as passed to add_integration())
cor_adjust	Adjustment method (same as passed to add_integration())

check_joint	If TRUE (default), also compare pairwise correlation matrices between the current and doubled n_int, and the maximum per-AgD-row absolute deviation from the user-supplied cor. The pairwise comparison catches cases where marginals converge but joint dependence structure does not (rare in practice for QMC with sensible cor_adjust but worth flagging when n_int is small).
verbose	Logical; if FALSE, suppresses printed diagnostic messages.

Value

A list with components marginals (the original data frame returned by previous versions) and, if check_joint = TRUE, correlations – a data frame of pairwise covariate correlations at the current and doubled n_int for each AgD row. Printed with a pass/warn verdict.

---

combine_data *Combine IPD and AgD for unanchored comparison*

Description

Combine IPD and AgD for unanchored comparison

Usage

```
combine_data(ipd, agd)
```

Arguments

ipd	An mlumr_ipd object from set_ipd()
agd	An mlumr_agd object from set_agd()

Value

An object of class mlumr_data

Examples

```
## Not run:
dat <- combine_data(ipd, agd)

## End(Not run)
```

compare_models *Compare fitted ML-UMR models*

Description

Compare two or more `mlumr_fit` objects by DIC (default), LOO, or WAIC. For LOO/WAIC, `loo::loo_compare()` is used under the hood; the output is the standard `loo_compare` table. For DIC the return is a data frame ordered by DIC.

Usage

```
compare_models(..., criterion = c("dic", "loo", "waic"))
```

Arguments

`...`	Two or more `mlumr_fit` objects. For DIC, `mlumr_dic` objects are also accepted.
`criterion`	One of `"dic"` (default), `"loo"`, or `"waic"`. LOO and WAIC require the optional `loo` package.

Details

DIC is the default for backward compatibility and because it has no additional package dependencies. For principled Bayesian model comparison, LOO (Vehtari, Gelman, Gabry 2017) is preferred and requires the optional `loo` package.

Value

For `"loo"` / `"waic"`: a `compare.loo` table from `loo::loo_compare()`. For `"dic"`: a data frame (invisibly) with columns `Model`, `DIC`, `pD`, `Delta_DIC`.

conditional_effects *Conditional treatment effects*

Description

Compute conditional (individual-level) treatment effects at specific covariate values from a fitted ML-UMR model. Unlike `marginal_effects()`, which averages over a population's covariate distribution, conditional effects evaluate the treatment effect at a particular covariate profile.

Usage

```
conditional_effects(
  object,
  newdata = NULL,
  effect = "all",
  summary = TRUE,
  probs = c(0.025, 0.5, 0.975)
)
```

Arguments

object	An mlumr_fit object
newdata	Data frame of covariate values at which to compute effects. Each row defines one covariate profile. Column names must match the covariates used in fitting. If NULL (default), uses the covariate means from the IPD as a single reference profile.
effect	Which effect measure. For binomial: "all", "link_effect", "rd", or "rr". For normal: "all" or "md". For Poisson: "all" or "rr". The legacy value "lor" is accepted as an alias for "link_effect" when the fitted link is logit.
summary	Return summary statistics (TRUE) or full posterior draws (FALSE)
probs	Quantiles for summary (default c(0.025, 0.5, 0.975))

Details

For SPFA models, the conditional link-scale treatment effect is constant across all covariate values because the shared beta cancels in the treatment contrast on the fitted link scale. However, risk difference (RD) and risk ratio (RR) still vary with covariates because they depend on absolute probability levels. For relaxed models, all conditional effects vary with covariate values because the index and comparator treatments have different regression coefficients.

The conditional link-scale effect is computed directly as eta_index - eta_comparator. This avoids numerical distortion from transforming extreme response-scale probabilities back through the link function.

Conditional vs marginal on non-identity links. Conditional effects are evaluated at a single covariate profile, so there is no averaging over a population and no Jensen's-inequality gap between the conditional and marginal response. Compare with marginal_effects() and predict.mlumr_fit(), which average over either the IPD individuals (index population) or the AgD integration points (comparator population) and therefore return E[g^{-1}(eta)], not g^{-1}(E[eta]).

Value

A data frame. If summary = TRUE, contains columns profile, effect, mean, sd, and quantile columns. If summary = FALSE, returns a single combined data frame of full posterior draws with a profile column indicating which covariate profile each draw belongs to.

See Also

marginal_effects() for population-averaged treatment effects; conditional_predict() for absolute predictions at specific profiles; predict.mlumr_fit() for population-level predictions.

Examples

```
## Not run:
# Conditional effects at IPD covariate means (default)
conditional_effects(fit)

# At specific covariate values
conditional_effects(fit, newdata = data.frame(age = 60, sex = 1))
```

```
# Multiple profiles
profiles <- data.frame(age = c(50, 60, 70), sex = c(0, 0, 1))
conditional_effects(fit, newdata = profiles)

## End(Not run)
```

`conditional_predict` *Conditional predictions*

Description

Generate absolute predictions at specific covariate values for both treatments.

Usage

```
conditional_predict(
  object,
  newdata = NULL,
  type = c("response", "link"),
  summary = TRUE,
  probs = c(0.025, 0.5, 0.975)
)
```

Arguments

`object`	An `mlumr_fit` object
`newdata`	Data frame of covariate values. If `NULL`, uses IPD covariate means.
`type`	`"response"` for probabilities, means, or rates; `"link"` for the fitted linear-predictor scale.
`summary`	Return summary (`TRUE`) or full draws (`FALSE`)
`probs`	Quantiles for summary

Value

A data frame with predictions for each treatment at each profile

See Also

`conditional_effects()` for covariate-conditional treatment *effects*; `predict.mlumr_fit()` for population-level predictions.

Examples

```
## Not run:
conditional_predict(fit)
conditional_predict(fit, newdata = data.frame(age = 60, sex = 1))

## End(Not run)
```

dbern *Bernoulli PMF*

Description

Bernoulli PMF

Usage

```
dbern(x, prob, log = FALSE)
```

Arguments

x	Vector of values
prob	Success probability
log	Logical; if TRUE, return log-density

Value

Numeric vector

default_priors *Default priors used by* mlumr()

Description

These accessors return the current default priors used by mlumr(), tagged with $default = TRUE and the package version. prior_summary() prints the version so cross-release reproducibility is diagnosable: if a later release changes a default, fits produced with an older version will still carry the correct $version tag.

Usage

```
default_prior_intercept()

default_prior_beta()

default_prior_sigma()
```

Value

A prior list (see prior_normal()).

Examples

```
default_prior_intercept()
default_prior_beta()
default_prior_sigma()
```

distr *Specify a marginal distribution*

Description

Used to specify marginal distributions for covariates when adding integration points. Wraps an inverse CDF (quantile) function with its parameters.

Usage

```
distr(qfun, ...)
```

Arguments

qfun	Inverse CDF function (e.g., qnorm, qbern)
...	Parameters of the distribution, can reference column names in AgD

Value

A list with class "mlumr_distr" containing the distribution specification

Examples

```
# Normal distribution
distr(qnorm, mean = 0, sd = 1)

# Bernoulli distribution with probability 0.3
distr(qbern, prob = 0.3)
```

marginal_effects *Marginal treatment effects*

Description

Extract marginal treatment effects from a fitted ML-UMR model. For binomial: log odds ratio, risk difference, risk ratio. For normal: mean difference. For poisson: rate ratio.

Usage

```
marginal_effects(
  object,
  population = c("both", "index", "comparator"),
  effect = "all",
  summary = TRUE,
  probs = c(0.025, 0.5, 0.975)
)
```

Arguments

object	An mlumr_fit object
population	Which population: "both", "index", or "comparator"
effect	Which effect measure. For binomial: "all", "lor", "rd", or "rr". For normal: "all" or "md" (mean difference). For poisson: "all" or "rr" (rate ratio).
summary	Return summary (TRUE) or full draws (FALSE)
probs	Quantiles for summary

Value

A data frame

See Also

predict.mlumr_fit() for absolute predictions; conditional_effects() for covariate-conditional effects at specific profiles; prior_sensitivity() to check how strongly the marginal effect depends on prior_beta.

Examples

```
## Not run:
# All effect measures for both populations
marginal_effects(fit)

# Only the log odds ratio in the index population
marginal_effects(fit, population = "index", effect = "lor")

# Full posterior draws rather than summary statistics
marginal_effects(fit, summary = FALSE)

## End(Not run)
```

---

mlumr *Fit ML-UMR Model*

Description

Fit a Bayesian multilevel unanchored meta-regression model using individual patient data (IPD) and aggregate data (AgD). Supports binary, continuous, and count outcomes.

Usage

```
mlumr(
  data,
  model = c("spfa", "relaxed"),
  link = NULL,
  prior_intercept = default_prior_intercept(),
```

```
  prior_beta = default_prior_beta(),
  prior_sigma = default_prior_sigma(),
  chains = 4,
  iter = 2000,
  warmup = 1000,
  seed = NULL,
  adapt_delta = 0.95,
  max_treedepth = 15,
  refresh = 200,
  engine = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

data	An mlumr_data object with integration points (from add_integration())
model	Model type: "spfa" (shared prognostic factor assumption) or "relaxed" (treatment-specific coefficients). Default "spfa".
link	Link function. For binomial: "logit" (default), "probit", or "cloglog". For normal: "identity" (default) or "log". For poisson: "log" (default, only option). If NULL, uses the canonical default for the family.
prior_intercept	Prior for treatment intercepts. Default from default_prior_intercept() (prior_normal(0, 10), weakly informative on the linear-predictor scale). See prior_normal() for guidance.
prior_beta	Prior for regression coefficients. May be a single prior broadcast to all covariates, or a list of priors of length n_cov for per-coefficient specification. All per-coefficient priors must share the same family and (for Student-t) df. Default from default_prior_beta() (prior_normal(0, 2.5); Gelman et al. 2008; Stan community prior-choice wiki). Set autoscale = TRUE on the prior to divide the scale by each covariate's empirical SD — useful when predictors are on very different scales.
prior_sigma	Prior for residual SD (normal family only). Default from default_prior_sigma() (prior_normal(0, 2.5), half-normal via the Stan <lower=0> constraint). prior_exponential() is also supported for sigma.
chains	Number of MCMC chains (default 4)
iter	Total iterations per chain (default 2000)
warmup	Number of warmup iterations (default 1000)
seed	Random seed for reproducibility
adapt_delta	Target acceptance rate (default 0.95)
max_treedepth	Maximum tree depth for NUTS (default 15)
refresh	How often to print progress (0 = silent, default 200)
engine	Stan backend: "rstan" (default) or "cmdstanr". If NULL, uses the engine set by mlumr_engine(). See mlumr_engine() for setup.

verbose	Logical; if FALSE, suppresses mlumr progress messages. Stan sampler progress is still controlled by refresh.
...	Additional arguments passed to the Stan sampling function (rstan::sampling() or cmdstanr's $sample() method)

Details

The model assumes that all AgD rows come from the same comparator treatment and that, conditional on covariates, there is no between-study heterogeneity. If AgD rows come from multiple studies with different designs or unmeasured confounders, this assumption may not hold. No random effects for study-level heterogeneity are included.

AgD scale assumptions (family = "normal"). The AgD likelihood is y_agd ~ normal(E[exp(eta)], se_agd) under link = "log" and y_agd ~ normal(E[eta], se_agd) under link = "identity". In both cases set_agd() expects outcome_mean and outcome_se on the **arithmetic (original, untransformed) scale**, not log-scale or geometric. Passing log-scale summaries silently misspecifies the likelihood. See set_agd() for details.

Comparator-population weighting is family-dependent. Integrated marginal predictions in the comparator population (*_comparator generated quantities) are weighted by:

- **binomial**: n_agd[k] (AgD sample size), so larger AgD rows contribute more to the marginal mean.
- **normal**: equal weights across AgD rows (/ n_agd_rows), reflecting the normal likelihood's treatment of each row as a single summary statistic.
- **poisson**: E_agd[k] (AgD exposure), matching the rate-based likelihood.

Each weighting is natural for the corresponding likelihood; users comparing marginal effects across families should be aware they are not identically weighted.

Weakly-identified coefficients in the relaxed model — beta_comparator is identified only through AgD, so the relaxed model needs informative priors (or many AgD rows) to estimate effect modification reliably. prior_sensitivity() is the recommended diagnostic.

Value

An object of class mlumr_fit

See Also

prior_sensitivity() for sensitivity of the posterior to prior_beta; set_agd() for AgD scale requirements; prior_summary() for introspection of the priors actually used.

Examples

```
## Not run:
# Binary SPFA model
fit_spfa <- mlumr(dat, model = "spfa")

# Relaxed SPFA (allows effect modification)
fit_relaxed <- mlumr(dat, model = "relaxed")

## End(Not run)
```

`mlumr_engine` *Get or Set the Stan Engine*

Description

Control which Stan backend mlumr uses for model fitting. The default engine is `"rstan"` (compiled C++ models via rstantools). Users who prefer cmdstanr can switch engines after installation.

Usage

```
mlumr_engine(engine = NULL)
```

Arguments

`engine`	Character string: `"rstan"` or `"cmdstanr"`. If `NULL` (default), returns the current engine without changing it.

Details

When switching to `"cmdstanr"`, this function checks whether the cmdstanr package and CmdStan toolchain are installed. If either is missing, it offers to install them interactively.

Engine names must be matched exactly. Partial strings such as `"c"` are not accepted.

The engine preference is stored as `options(mlumr.stan_engine = ...)` and persists for the current R session. To set a permanent default, add to your `.Rprofile`:

```
options(mlumr.stan_engine = "cmdstanr")
```

Value

The current engine (character), returned invisibly when setting.

Examples

```
# Check current engine
mlumr_engine()

# Switch to cmdstanr (interactive)
## Not run:
mlumr_engine("cmdstanr")

## End(Not run)
```

naive *Naive unadjusted indirect comparison*

Description

Compute an unadjusted (naive) indirect treatment comparison by comparing crude outcomes from the IPD and AgD without any covariate adjustment. Returns treatment effect on the link scale plus absolute predictions (event probabilities, risk difference, risk ratio) for both populations.

Usage

```
naive(data, link = NULL, conf_level = 0.95)
```

Arguments

data	An mlumr_data object from combine_data()
link	Link function. For binomial: "logit" (default), "probit", or "cloglog". For normal/poisson: ignored (identity/log always used). If NULL, uses the canonical default.
conf_level	Confidence level for the interval (default 0.95)

Details

For binomial outcomes, event-probability intervals use Wald standard errors and are bounded to [0, 1]. For Poisson outcomes, the log-rate contrast uses a 0.5 continuity correction when an observed event count is zero.

Value

An object of class mlumr_naive

Examples

```
## Not run:
result <- naive(dat)
print(result)

## End(Not run)
```

pbern *Bernoulli CDF*

Description

Bernoulli CDF

Usage

```
pbern(q, prob, lower.tail = TRUE, log.p = FALSE)
```

Arguments

q	Vector of quantiles
prob	Success probability
lower.tail	Logical
log.p	Logical

Value

Numeric vector

predict.mlumr_fit *Predictions from ML-UMR model*

Description

Generate population-average absolute-outcome predictions in the index and comparator populations.

Usage

```
## S3 method for class 'mlumr_fit'
predict(
  object,
  population = c("both", "index", "comparator"),
  type = c("response", "link"),
  summary = TRUE,
  probs = c(0.025, 0.5, 0.975),
  ...
)
```

Arguments

object	An mlumr_fit object
population	Which population: "both", "index", or "comparator"
type	Prediction type: "response" or "link". For "response": probabilities (binomial), means (normal), or rates (poisson). For "link": mean linear predictor on the fitted link scale (logit, probit, cloglog, log, or identity). The link-scale values are computed directly from parameter draws as E[eta], not as link(E[g^{-1}(eta)]), to avoid Jensen's inequality bias.
summary	Return summary statistics (TRUE) or full posterior draws (FALSE)
probs	Quantiles for summary (default c(0.025, 0.5, 0.975))
...	Additional arguments (unused)

Details

Marginalization on non-identity links. For type = "response" the reported values are E[g^{-1}(eta)] — the posterior expectation of the inverse-link-transformed linear predictor — *not* g^{-1}(E[eta]). The two differ whenever g is non-linear (logit, probit, cloglog, log) by Jensen's inequality. In the index population the expectation is taken over IPD individuals; in the comparator population it is taken over the integration points constructed by add_integration() from the AgD moments. This is the correct population-average prediction for an individual randomly drawn from that population, and it matches what the Stan generated quantities block computes. For the link scale (type = "link") the reported value is E[eta], a linear functional, and the two interpretations coincide.

Value

A data frame with predictions. When type = "link", values are mean linear predictors computed directly from parameter draws (avoiding Jensen's inequality bias).

See Also

marginal_effects() for treatment-effect summaries; conditional_predict() and conditional_effects() for predictions at specific covariate profiles.

---

prior_cauchy *Specify a Cauchy prior*

Description

Cauchy is Student-t with df = 1; this constructor is a convenience wrapper around prior_student_t(). It has very heavy tails and should be used with care — modern recommendations generally prefer prior_student_t(df in 3:7, ...) over Cauchy for regression coefficients to keep sampling well-behaved (see Piironen & Vehtari on the horseshoe; Ghosh et al. 2015).

Usage

```
prior_cauchy(mean = 0, sd = 2.5, autoscale = FALSE)
```

Arguments

mean	Prior location (default 0).
sd	Prior scale (default 2.5).
autoscale	See prior_normal(). Default FALSE.

Value

A list with components distribution = "student_t", df = 1, mean, sd, autoscale.

Examples

```
prior_cauchy(mean = 0, sd = 2.5)
```

---

prior_exponential *Specify an exponential prior*

Description

Exponential prior on a positive scalar. Currently supported for prior_sigma (normal-family residual SD) only; rejected for unconstrained intercepts and regression coefficients. prior_exponential(rate = 1) has mean 1 and is a reasonable weakly-informative choice when the outcome is standardized.

Usage

```
prior_exponential(rate = 1)
```

Arguments

rate	Rate parameter (default 1). Larger rate = tighter prior concentrated near zero.

Value

A list with components distribution = "exponential", rate, and placeholders (mean = 0, sd = 1 / rate) so the same Stan-field translation works.

Examples

```
prior_exponential(rate = 1)
```

prior_normal *Specify a normal prior*

Description

Construct a normal prior for passing to `mlumr()` via `prior_intercept`, `prior_beta`, or `prior_sigma`. The resulting list is consumed by the Stan models.

Usage

```
prior_normal(mean = 0, sd = 10, autoscale = FALSE)
```

Arguments

`mean`	Prior mean (default 0).
`sd`	Prior standard deviation (default 10). The default matches the historical "very weak" scale; explicit tighter values are recommended for regression coefficients (see Details).
`autoscale`	If `TRUE` and this prior is passed as `prior_beta`, the scale is divided by each covariate's empirical SD so the prior is weakly-informative regardless of predictor scaling. Default `FALSE` to preserve backward-compatible behavior; set to `TRUE` explicitly when passing unstandardized predictors. Ignored for `prior_intercept` and `prior_sigma`.

Value

A list with components `distribution`, `mean`, `sd`, `df`, `autoscale`.

Choosing a scale

The Stan community's prior-choice wiki (Vehtari et al., 2025) describes five broad categories, from least to most informative:

1. Flat prior — not recommended.
2. Super-vague proper prior, e.g., `normal(0, 1e6)` — not recommended.
3. Weakly informative, **very weak**, e.g., `normal(0, 10)`.
4. Generic weakly informative, e.g., `normal(0, 1)`.
5. Specific informative, e.g., `normal(0.4, 0.2)`.

Those scales assume parameters are on roughly unit scale. In ML-UMR models the natural scales are:

Treatment intercepts On the linear-predictor (link) scale. For a binary outcome with logit link, the intercept is a baseline log odds; `normal(0, 10)` spans ±20 log-odds at 95 percent and is "very weak". It is the default because the data usually constrain the intercept strongly. Tightening to `normal(0, 5)` is reasonable when the expected event rate is far from the extremes.

Regression coefficients (`beta`) On the link scale, per unit of covariate. For logistic regression with predictors on unit scale, Gelman et al. (2008) and the Stan wiki recommend `student_t(df, 0, 2.5)` with `df` in `3:7`, or — as a practical approximation — `normal(0, 2.5)`. That is the default used by `mlumr()`. Use `normal(0, 1)` if you expect small effects (e.g., standardized predictors in a normal-outcome model). If predictors are on very different scales, set `autoscale = TRUE` so the scale is divided by each covariate's SD.

Residual SD (`sigma`, **normal family only**) `prior_sigma` is interpreted as a half-normal via the Stan `<lower=0>` constraint. The default `normal(0, 2.5)` (i.e., `half-normal(0, 2.5)`) is weakly informative for residual SDs on the scale of the outcome. Scale to the outcome if it is far from unit scale, or use `prior_exponential()`.

Prior sensitivity is especially important for the relaxed model, where `beta_comparator` is identified only by the AgD likelihood. Run `prior_sensitivity()` to quantify how much conclusions move under alternative scales; see `vignette("mlumr-models")`.

References

Gelman, A., Jakulin, A., Pittau, M. G., & Su, Y.-S. (2008). A weakly informative default prior distribution for logistic and other regression models. *Annals of Applied Statistics*, 2(4), 1360–1383.

Vehtari, A. et al. Prior Choice Recommendations (Stan wiki): https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations.

Examples

```
# Default weakly-very-weak intercept prior
prior_normal(mean = 0, sd = 10)

# Gelman 2008 default for logistic-regression coefficients
prior_normal(mean = 0, sd = 2.5)

# Autoscaled coefficient prior (dividing 2.5 by each covariate's SD)
prior_normal(mean = 0, sd = 2.5, autoscale = TRUE)
```

---

prior_sensitivity *Prior sensitivity analysis for an ML-UMR fit*

Description

Refit an `mlumr()` model across a grid of `prior_beta` scales (keeping the family, mean, and df fixed) and summarize how the posterior for the marginal treatment effects (`delta_index`, `delta_comparator`) moves. This is the workflow recommended by Vehtari et al.'s prior-choice wiki for judging how much of the posterior is driven by the data versus the prior.

Usage

```
prior_sensitivity(
  fit,
  prior_beta_scales = c(0.5, 1, 2.5, 5, 10),
  probs = c(0.025, 0.5, 0.975),
  verbose = TRUE,
  ...
)
```

Arguments

fit	A fitted mlumr_fit object to re-fit under alternative priors.
prior_beta_scales	Numeric vector of scales for prior_beta. Default c(0.5, 1, 2.5, 5, 10).
probs	Quantiles for summarizing each posterior (default c(0.025, 0.5, 0.975)).
verbose	Logical; if FALSE, suppresses progress messages and final printed summary table.
...	Additional arguments forwarded to mlumr() on each refit (e.g. chains, iter, refresh). Sampling defaults otherwise inherit from the original fit.

Details

Only the scale of the prior_beta family is varied; its distribution (normal / student_t) and mean are preserved so comparisons are apples to apples. prior_intercept and prior_sigma are carried through unchanged from the original fit. Each value in prior_beta_scales is used as the **absolute** scale for every coefficient at that refit — if the original fit used per-coefficient priors, all coefficients are set to the same scale (the sweep is deliberately homogeneous so the grid reflects a single level of prior informativeness per refit, not a rescaling of existing relative differences). If the original prior_beta used an exponential family, it is swapped for a prior_normal(0, scale) at each grid point since exponential has no scale parameter to vary.

Value

A data frame (tibble-style) with one row per (scale, population, quantile) combination and columns scale, parameter, mean, sd, and the requested quantiles. Side effect: prints a summary table at the end.

See Also

prior_summary() for a one-shot description of the priors on a fit; marginal_effects() for the posterior summary quantities this sweep tracks.

Examples

```
## Not run:
sens <- prior_sensitivity(fit_spfa, prior_beta_scales = c(1, 2.5, 5))

## End(Not run)
```

prior_student_t *Specify a Student-t prior*

Description

Heavier-tailed alternative to `prior_normal()`. For logistic-regression coefficients with unit-scale predictors, the Stan community prior-choice recommendations suggest Student-t with `df` between 3 and 7 as a robust weakly informative prior (Gelman et al., 2008).

Usage

```
prior_student_t(df = 5, mean = 0, sd = 2.5, autoscale = FALSE)
```

Arguments

`df`	Degrees of freedom (must be positive). Values in `3:7` are recommended for logistic-regression coefficients.
`mean`	Prior location (default 0).
`sd`	Prior scale (default 2.5).
`autoscale`	See `prior_normal()`. Default `FALSE`.

Value

A list with components `distribution = "student_t"`, `df`, `mean`, `sd`, `autoscale`.

Examples

```
# Gelman et al. 2008 recommendation for logistic-regression coefficients
prior_student_t(df = 5, mean = 0, sd = 2.5)
```

prior_summary *Summary of priors used by a fitted ML-UMR model*

Description

Print a human-readable summary of every prior that was used to fit an `mlumr()` model, including the effective per-coefficient scales after autoscaling. Mirrors the spirit of `rstanarm::prior_summary()`.

Usage

```
prior_summary(object, ...)

## Default S3 method:
prior_summary(object, ...)

## S3 method for class 'mlumr_fit'
prior_summary(object, digits = 3, ...)
```

Arguments

object	An mlumr_fit object.
...	Unused.
digits	Number of significant digits for numeric values (default 3).

Value

Invisibly returns a list describing the priors; the side effect is printing a formatted summary.

See Also

prior_sensitivity() to quantify how much the posterior moves under alternative prior_beta scales; prior_normal(), prior_student_t(), prior_cauchy(), prior_exponential() for the prior constructors themselves.

Examples

```
## Not run:
fit <- mlumr(dat)
prior_summary(fit)

## End(Not run)
```

---

qbern *Bernoulli quantile function*

Description

Bernoulli quantile function

Usage

```
qbern(p, prob, lower.tail = TRUE, log.p = FALSE)
```

Arguments

p	Vector of probabilities
prob	Success probability
lower.tail	Logical; if TRUE, probabilities are P(X <= x)
log.p	Logical; if TRUE, probabilities are given as log(p)

Value

Integer vector of 0s and 1s

`set_agd` *Set up aggregate data (AgD)*

Description

Prepare AgD from the comparator treatment for an unanchored indirect comparison.

Usage

```
set_agd(
  data,
  treatment,
  family = c("binomial", "normal", "poisson"),
  outcome_n = NULL,
  outcome_r = NULL,
  outcome_mean = NULL,
  outcome_se = NULL,
  outcome_E = NULL,
  cov_means,
  cov_sds = NULL,
  cov_types = NULL,
  study = NULL
)
```

Arguments

`data`	Data frame containing AgD summary statistics
`treatment`	Column name for treatment variable
`family`	Outcome family: `"binomial"`, `"normal"`, or `"poisson"`
`outcome_n`	Column name for sample size (required for binomial, optional for normal)
`outcome_r`	Column name for number of events (required for binomial and poisson)
`outcome_mean`	Column name for mean outcome (required for normal)
`outcome_se`	Column name for standard error of outcome (required for normal)
`outcome_E`	Column name for total exposure (required for poisson)
`cov_means`	Character vector of column names for covariate means/proportions
`cov_sds`	Character vector of column names for covariate SDs (`NA` for binary covariates)
`cov_types`	Character vector specifying `"continuous"` or `"binary"` for each covariate. If `NULL`, inferred from presence of SD.
`study`	Column name for study identifier (optional)

Details

Scale assumptions for `family = "normal"`**.** The AgD likelihood is `y_agd ~ normal(E[exp(eta)], se_agd)` under `link = "log"` and `y_agd ~ normal(E[eta], se_agd)` under `link = "identity"`. In both cases `outcome_mean` and `outcome_se` must be on the **arithmetic (original, untransformed) scale**. If a publication reports only a log-scale mean / SD or a geometric mean, back-transform before calling `set_agd()` and propagate uncertainty via the delta method; passing log-scale or geometric summaries silently misspecifies the likelihood and biases the posterior.

Scale assumptions for `family = "poisson"`**.** `outcome_r` is the total count in each AgD row and `outcome_E` is the total person-time (or other exposure). The Stan likelihood uses `log(E_agd)` as an offset, so rates are modeled on the log scale regardless of how `outcome_r` is tabulated.

Scale assumptions for `family = "binomial"`**.** `outcome_r` / `outcome_n` are counts of events and trials. The log-odds (or probit / cloglog under alternative links) are formed from `outcome_r` / `outcome_n`, so no scale conversion is required.

Value

An object of class `mlumr_agd`

Examples

```
## Not run:
# Binary outcome
agd <- set_agd(
  data = trial_b,
  treatment = "trt",
  outcome_n = "n_total",
  outcome_r = "n_events",
  cov_means = c("age_mean", "sex_prop"),
  cov_sds = c("age_sd", NA),
  cov_types = c("continuous", "binary")
)

# Continuous outcome
agd <- set_agd(
  data = trial_b,
  treatment = "trt",
  family = "normal",
  outcome_mean = "mean_score",
  outcome_se = "se_score",
  outcome_n = "n_total",
  cov_means = c("age_mean", "sex_prop")
)

# Count outcome
agd <- set_agd(
  data = trial_b,
  treatment = "trt",
  family = "poisson",
  outcome_r = "n_events",
  outcome_E = "person_years",
```

```
  cov_means = c("age_mean", "sex_prop")
)

## End(Not run)
```

`set_ipd` *Set up individual patient data (IPD)*

Description

Prepare IPD from the index treatment for an unanchored indirect comparison.

Usage

```
set_ipd(
  data,
  treatment,
  outcome,
  covariates,
  family = c("binomial", "normal", "poisson"),
  exposure = NULL,
  study = NULL
)
```

Arguments

`data`	Data frame containing IPD
`treatment`	Column name for treatment variable
`outcome`	Column name for outcome variable. For `family = "binomial"`, must be binary (0/1). For `family = "normal"`, any numeric. For `family = "poisson"`, non-negative integer counts.
`covariates`	Character vector of covariate column names
`family`	Outcome family: `"binomial"`, `"normal"`, or `"poisson"`
`exposure`	Column name for exposure/time-at-risk (required when `family = "poisson"`)
`study`	Column name for study identifier (optional)

Value

An object of class `mlumr_ipd`

Examples

```
## Not run:
# Binary outcome
ipd <- set_ipd(
  data = trial_a,
  treatment = "trt",
  outcome = "response",
  covariates = c("age", "sex")
)

# Continuous outcome
ipd <- set_ipd(
  data = trial_a,
  treatment = "trt",
  outcome = "score",
  covariates = c("age", "sex"),
  family = "normal"
)

# Count outcome with exposure
ipd <- set_ipd(
  data = trial_a,
  treatment = "trt",
  outcome = "events",
  covariates = c("age", "sex"),
  family = "poisson",
  exposure = "person_years"
)

## End(Not run)
```

---

stc *Simulated treatment comparison via G-computation*

Description

Perform an unanchored simulated treatment comparison (STC) using parametric G-computation. Fits a regression on IPD, predicts counterfactual outcomes in both the index and comparator populations, and computes marginal treatment effects with delta-method standard errors.

Usage

```
stc(data, link = NULL, conf_level = 0.95)
```

Arguments

`data`	An `mlumr_data` object from `combine_data()`. Integration points are not required for STC but covariate information from the AgD is used.

link	Link function. For binomial: "logit" (default), "probit", or "cloglog". For normal: "identity" (default) or "log". For poisson: "log" (default). If NULL, uses the canonical default.
conf_level	Confidence level for the interval (default 0.95)

Details

For binomial outcomes, returns the treatment effect on the link scale plus event probabilities, risk difference, and log risk ratio with SEs and CIs for both populations. Event-probability intervals use Wald standard errors and are bounded to [0, 1]. For Poisson outcomes, the comparator log rate uses a 0.5 continuity correction when the observed event count is zero.

The STC procedure is:

1. Fit a GLM on IPD (binomial/gaussian/poisson as appropriate).
2. Predict on comparator-population covariates (from integration points or AgD covariate means).
3. Marginalize predictions over the comparator population.
4. Predict on index-population covariates (IPD).
5. Compute treatment effects and SEs via the delta method.

STC is a parametric G-computation benchmark. It relies on the IPD outcome model being correctly specified and transportable to the comparator population. It does not model posterior uncertainty in population covariate distributions or relax treatment-specific covariate effects. When clinically meaningful effect modification is plausible, prefer `mlumr(..., model = "relaxed")` as the primary analysis and use STC as a sensitivity or benchmarking analysis.

For binomial outcomes, the comparator-population treatment contrast is transported to the index population **assuming the treatment contrast is constant on the fitted link scale** (i.e., no effect modification on that scale). Under this assumption, the index-population comparator probability is computed as `inv_link(link(p_A_index) - (link(p_A_comp) - link(p_B)))`, and its uncertainty is propagated through the delta method. If effect modification is expected, fit a Bayesian relaxed model with `mlumr(..., model = "relaxed")` and use `predict(..., population = "index")` instead, which does not require this assumption.

Value

An object of class `mlumr_stc`

Examples

```
## Not run:
result <- stc(dat)
print(result)

## End(Not run)
```

`unnest_integration` *Expand integration points into a long-format data frame*

Description

Expand integration points into a long-format data frame

Usage

```
unnest_integration(data)
```

Arguments

`data`	An `mlumr_data` object with integration points

Value

A data frame with columns for each covariate plus `.int_id` and `.agd_row`

Index