

Package ‘coldstart’

October 7, 2026

Title Calibrating Generated Items Before Pretesting with Predicted Priors

Version 0.1.0

Description Predicts Rasch item difficulty from item features (text embeddings from any model, template family, content metadata) by ridge regression (Hoerl and Kennard, 1970, <[doi:10.1080/00401706.1970.10488634](https://doi.org/10.1080/00401706.1970.10488634)>) with honest out-of-sample uncertainty, uses the predictions as robust informative priors to calibrate new items from small pretest samples, in the spirit of using collateral information in calibration (Mislevy, Sheehan and Wingersky, 1993, <[doi:10.1111/j.1745-3984.1993.tb00422.x](https://doi.org/10.1111/j.1745-3984.1993.tb00422.x)>), plans the pretest sample size needed for a target precision, and diagnoses item families whose predictions cannot be trusted.

License MIT + file LICENSE

URL <https://github.com/edidatasolutions/coldstart>,
<https://edidatasolutions.github.io/coldstart/>

BugReports <https://github.com/edidatasolutions/coldstart/issues>

Encoding UTF-8

Depends R (>= 4.1)

Imports stats

Suggests knitr, markdown

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Daniel Edi [aut, cre, cph] (ORCID:
<<https://orcid.org/0000-0001-5475-819X>>)

Maintainer Daniel Edi <danieledi2026@gmail.com>

Repository CRAN

Date/Publication 2026-10-07 09:50:35 UTC

Contents

cs_calibrate	2
cs_check	3
cs_distrust	4
cs_plan	5
cs_predictor	5
cs_responses	6
cs_simulate	7
predict.cs_predictor	8

Index	10
-------	----

cs_calibrate	<i>Bayesian calibration of new items with predicted priors</i>
--------------	--

Description

Grid posterior for each item’s Rasch difficulty given pretest responses from examinees with known ability (from operational scoring). Each item is calibrated twice: with the predicted prior, and with a vague baseline prior $N(0, \text{‘baseline_sd’}^2)$, which stands in for conventional calibration.

Usage

```
cs_calibrate(  
  responses,  
  prior = NULL,  
  prior_df = 4,  
  baseline_sd = 3,  
  grid = seq(-7, 7, by = 0.02)  
)
```

Arguments

responses	Long data frame: ‘item’, ‘theta’, ‘x’ (0/1).
prior	Output of ‘predict()’ on a ‘cs_predictor’ (‘item’, ‘mean’, ‘sd’), or ‘NULL’ for baseline only.
prior_df	Degrees of freedom of the t prior (> 2, or ‘Inf’).
baseline_sd	SD of the vague baseline prior.
grid	Difficulty grid.

Details

The predicted prior is a Student-t with ‘prior_df’ degrees of freedom by default. When the prediction is badly wrong (e.g. a template changed), the heavy tail lets the data override it instead of being dragged toward it. ‘prior_df = Inf’ gives a normal prior.

Value

A 'cs_calibration' data frame: 'item', 'n', 'post_mean', 'post_sd', 'base_mean', 'base_sd', 'prior_mean', 'prior_sd', and 'conflict_z' (baseline estimate vs prior, standardized by their combined SD: a prior-data conflict check).

Examples

```
sim <- cs_simulate(n_train = 200, n_new = 60, seed = 1)
it <- sim$items; tr <- it$set == "train"
pr <- cs_predictor(it$b_legacy[tr], sim$features[tr, ], it$family[tr], seed = 1)
pred <- predict(pr, sim$features[!tr, ], it$family[!tr])
cal <- cs_calibrate(cs_responses(sim, 25, seed = 2), pred)
truth <- it$b_true[match(cal$item, it$item)]
c(baseline = sqrt(mean((cal$base_mean - truth)^2)),
  predicted_prior = sqrt(mean((cal$post_mean - truth)^2)))
```

cs_check	<i>Which item families' predictions can be trusted?</i>
----------	---

Description

Compares each item's baseline (data-driven) estimate with its prediction, using 'conflict_z' from [cs_calibrate()]. For each family it reports the mean z (bias direction), coverage of the nominal predictive interval, and a chi-square test of 'sum(z^2)'. Families with p below 'alpha' are marked untrustworthy: their predictions should not be used as priors until the predictor is retrained on their calibrated items.

Usage

```
cs_check(calibration, family, level = 0.9, alpha = 0.01)
```

Arguments

calibration	A 'cs_calibration' computed with a prior.
family	Family per item, named by item id (or a data frame with 'item' and 'family').
level	Nominal coverage level for the interval check.
alpha	Significance level for flagging a family.

Value

Data frame, one row per family: 'family', 'n_items', 'mean_z', 'rms_z', 'coverage', 'p_value', 'trustworthy'.

Examples

```
sim <- cs_simulate(n_train = 150, n_new = 80, seed = 1)
it <- sim$items; tr <- it$set == "train"
pr <- cs_predictor(it$b_legacy[tr], sim$features[tr, ], it$family[tr], seed = 1)
pred <- predict(pr, sim$features[!tr, ], it$family[!tr])
cal <- cs_calibrate(cs_responses(sim, 60, seed = 2), pred)
cs_check(cal, setNames(it$family[!tr], it$item[!tr]))
unique(it$family[it$rogue]) # the family whose template drifted
```

cs_distrust

Withdraw priors for untrustworthy families

Description

Replaces the predicted prior of every item in a family that failed [cs_check()] with a vague prior (mean 0, SD 'vague_sd'), so the final calibration of those items rests on their responses alone. Recalibrate with [cs_calibrate()] afterwards.

Usage

```
cs_distrust(prior, check, vague_sd = 3)
```

Arguments

prior	Prior data frame from 'predict()' on a 'cs_predictor'.
check	Output of [cs_check()].
vague_sd	SD of the replacement prior.

Details

Note that the check and the final calibration use the same responses. This is an empirical-Bayes style decision; in simulation it restores baseline-level accuracy for a drifted family while keeping the prior's benefit elsewhere.

Value

'prior' with modified 'mean'/'sd' for untrusted families and a 'trusted' column.

Examples

```
sim <- cs_simulate(n_train = 150, n_new = 80, seed = 1)
it <- sim$items; tr <- it$set == "train"
pr <- cs_predictor(it$b_legacy[tr], sim$features[tr, ], it$family[tr], seed = 1)
pred <- predict(pr, sim$features[!tr, ], it$family[!tr])
resp <- cs_responses(sim, 60, seed = 2)
chk <- cs_check(cs_calibrate(resp, pred), setNames(it$family[!tr], it$item[!tr]))
final <- cs_calibrate(resp, cs_distrust(pred, chk))
head(final)
```

cs_plan	<i>Plan pretest sample sizes for a target precision</i>
---------	---

Description

Uses the normal approximation ‘posterior precision = $1 / \text{prior_sd}^2 + n * I(b)$ ’, where ‘ $I(b)$ ’ is the average information of one response about the item’s difficulty in the pretest population, evaluated at the predicted difficulty. Compares the responses needed with the predicted prior against the conventional (no-prior) requirement ‘ $1 / (\text{target_sd}^2 * I(b))$ ’.

Usage

```
cs_plan(prior, target_sd = 0.2, theta_mean = 0, theta_sd = 1)
```

Arguments

prior Output of ‘predict()’ on a ‘cs_predictor’.

target_sd Desired posterior SD of each difficulty.

theta_mean, theta_sd Pretest population.

Value

Data frame: ‘item’, ‘prior_sd’, ‘info’, ‘n_with_prior’, ‘n_without_prior’, ‘saved’.

Examples

```
sim <- cs_simulate(n_train = 200, n_new = 60, seed = 1)
it <- sim$items; tr <- it$set == "train"
pr <- cs_predictor(it$b_legacy[tr], sim$features[tr, ], it$family[tr], seed = 1)
pred <- predict(pr, sim$features[!tr, ], it$family[!tr])
plan <- cs_plan(pred, target_sd = 0.3)
summary(plan[c("n_with_prior", "n_without_prior")])
```

cs_predictor	<i>Predict item difficulty from item features, with honest uncertainty</i>
--------------	--

Description

Ridge regression of calibrated difficulties on standardized features plus template-family indicators, with the penalty chosen by cross-validation. Predictive uncertainty is estimated out of sample, separately for the two situations a new item can be in:

seen family RMSE of out-of-fold predictions under random K-fold CV.

unseen family RMSE under leave-one-family-out CV, where the held-out family’s effect is unknown. This is usually much larger, and it is the honest number for a new template.

Usage

```
cs_predictor(
  b,
  features,
  family = NULL,
  lambdas = 10^seq(-2, 3, length.out = 26),
  folds = 10,
  seed = NULL
)
```

Arguments

b	Calibrated difficulties of legacy items.
features	Numeric matrix of item features (e.g. text embeddings from any model, cognitive-attribute codes), one row per legacy item.
family	Optional template family / content code per item.
lambdas	Candidate ridge penalties.
folds	Number of CV folds.
seed	Optional seed for fold assignment.

Value

A ‘cs_predictor‘.

Examples

```
sim <- cs_simulate(n_train = 200, n_new = 60, seed = 1)
it <- sim$items; tr <- it$set == "train"
pr <- cs_predictor(it$b_legacy[tr], sim$features[tr, ], it$family[tr], seed = 1)
pr # predictive SD for seen vs unseen template families
```

cs_responses	<i>Simulate pretest responses to new items</i>
--------------	--

Description

Simulate pretest responses to new items

Usage

```
cs_responses(sim, n_per_item = 50, theta_mean = 0, theta_sd = 1, seed = NULL)
```

Arguments

`sim` A 'cs_sim'.
`n_per_item` Responses per new item (scalar, or vector named by item).
`theta_mean, theta_sd`
 Ability distribution of pretest examinees.
`seed` Optional seed.

Value

Long data frame: 'item', 'theta' (examinee ability, treated as known from operational scoring), 'x' (0/1).

Examples

```

sim <- cs_simulate(n_train = 200, n_new = 60, seed = 1)
resp <- cs_responses(sim, n_per_item = 30, seed = 2)
head(resp)

```

cs_simulate

Simulate an item bank with features and known difficulties

Description

Difficulty is ' $b = Xw + \text{family effect} + \text{noise}$ '. Legacy ('train') items have calibrated difficulties from large samples (true value plus 'legacy_se' error). New generated items ('new') come from the same families plus one family never seen in training. Optionally, one seen family is "rogue": its new items are 'rogue_shift' harder than its history implies, as when a generation template changes.

Usage

```

cs_simulate(
  n_train = 400,
  n_new = 150,
  n_families = 10,
  n_features = 32,
  signal_sd = 0.8,
  family_sd = 0.4,
  resid_sd = 0.5,
  legacy_se = 0.1,
  rogue_shift = 1.2,
  seed = NULL
)

```

Arguments

n_train, n_new	Numbers of legacy and new items.
n_families	Number of template families seen in training; one more family appears only among new items.
n_features	Feature (embedding) dimension.
signal_sd, family_sd, resid_sd	SDs of the feature signal, family effects and item-specific residual.
legacy_se	Calibration error of legacy difficulties.
rogue_shift	Shift for the rogue family's new items (0 for none).
seed	Optional seed.

Value

A 'cs_sim': '\$items' ('item', 'family', 'set', 'b_true', 'b_legacy', 'rogue') and '\$features' (matrix, rownames = item ids).

Examples

```
sim <- cs_simulate(n_train = 200, n_new = 60, seed = 1)
table(sim$items$set, sim$items$rogue)
dim(sim$features)
```

predict.cs_predictor	<i>Predicted difficulty and predictive SD for new items</i>
----------------------	---

Description

Predicted difficulty and predictive SD for new items

Usage

```
## S3 method for class 'cs_predictor'
predict(object, features, family = NULL, item = rownames(features), ...)
```

Arguments

object	A 'cs_predictor'.
features	Feature matrix for new items (same columns as training).
family	Family per new item (families not seen in training get the larger unseen-family SD).
item	Optional item ids (default: rownames of 'features').
...	Unused.

Value

Data frame: 'item', 'family', 'mean', 'sd', 'family_seen'.

Examples

```
sim <- cs_simulate(n_train = 200, n_new = 60, seed = 1)
it <- sim$items; tr <- it$set == "train"
pr <- cs_predictor(it$b_legacy[tr], sim$features[tr, ], it$family[tr], seed = 1)
pred <- predict(pr, sim$features[!tr, ], it$family[!tr])
head(pred)
```

Index

`cs_calibrate`, [2](#)
`cs_check`, [3](#)
`cs_distrust`, [4](#)
`cs_plan`, [5](#)
`cs_predictor`, [5](#)
`cs_responses`, [6](#)
`cs_simulate`, [7](#)

`predict.cs_predictor`, [8](#)