Package: fairGATE (via r-universe)

October 20, 2026

Title Fair Gated Algorithm for Targeted Equity

Version 1.0.0

Description Fairness-aware correction of a prediction model for binary and continuous outcomes with two or more protected groups and their intersections. A cross-fitted base learner (gradient-boosted trees or a neural network) is fitted without the protected attribute, and a gated mixture of neural experts learns a per-person correction to its score under a fairness penalty (a random-effects or variance penalty on outcome-stratified group means, or a thresholded worst-group penalty on error rates), with a cross-fitted disparity signal, an intervention cost, smoothed early stopping, a per-person export of adjustments with bootstrap stability intervals, and routing analyses of which expert corrects whom. Methods draw on mixture-of-experts neural networks by Jordan and Jacobs (1994) <doi:10.1007/978-1-4471-2097-1_113>, gradient boosting by Chen and Guestrin (2016) <doi:10.1145/2939672.2939785>, cross-fitting by Chernozhukov et al. (2018) <doi:10.1111/ectj.12097>, equalised odds by Hardt, Price and Srebro (2016) <doi:10.48550/arXiv.1610.02413>, the random-effects variance estimator of DerSimonian and Laird (1986) <doi:10.1016/0197-2456(86)90046-2>, and personalised treatment prediction for depression by Iniesta, Stahl and McGuffin (2016) <doi:10.1016/j.jpsychires.2016.03.016>.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

Depends R (>= 4.1.0)

Imports dplyr, ggalluvial, ggplot2, magrittr, pROC, purrr, readr, rlang, stats, tibble, tidyr, tidyselect, utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0), torch, xgboost (>= 3.1.0)

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/rhysholland/FairGATE

BugReports https://github.com/rhysholland/FairGATE/issues

SystemRequirements Optional 'LibTorch' backend for the neural-network components; install via torch::install_torch().

LazyData true

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libicu-dev libx11-dev

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

Date/Publication 2026-10-20 12:55:57 UTC

RemoteUrl https://github.com/rhysholland/fairgate

RemoteRef HEAD

RemoteSha e9598a45e86b5472388c95aae54cc2c85c734b41

Contents

adult_ready_small . . . 3
analyse_experts . . . 3
analyse_gnn_results . . . 5
export_f360_csv . . . 6
fairgate . . . 7
fairgate_base_grid . . . 10
fairgate_base_learner . . . 12
fairgate_crossed_penalty . . . 14
fairgate_crossed_plan . . . 15
fairgate_disparity_target . . . 16
fairgate_draw_configs . . . 17
fairgate_export . . . 18
fairgate_fair_penalty . . . 18
fairgate_gate_map . . . 20
fairgate_group_disparity . . . 21
fairgate_group_gap_penalty . . . 22
fairgate_lambda_ladder . . . 23
fairgate_penalty_from_cells . . . 24
fairgate_penalty_plan . . . 25
fairgate_space . . . 27
fairgate_threshold_penalty . . . 27
fairgate_threshold_plan . . . 28
fairgate_tune_base_learner . . . 29
plot_sankey . . . 31
prepare_data . . . 32
train_gnn . . . 33

Index **41**

`adult_ready_small` *Adult ready (small sample)*

Description

A compact (n = 1000) sample of the pre-cleaned Adult dataset for examples and vignettes. All original columns are retained; only rows are subsampled.

Usage

```
adult_ready_small
```

Format

A data frame with 1000 rows and K columns (same schema as the full dataset).

Details

Created from the full CSV by random sampling with a fixed seed: `adult_ready_small <- dplyr::slice_sample(big, n = 1000)`.

Source

UCI Adult (pre-cleaned in-package).

Examples

```
data("adult_ready_small", package = "fairGATE")
str(adult_ready_small)
head(adult_ready_small)
```

`analyse_experts` *Analyse and Visualise Expert Network Specialisation*

Description

Analyses expert input weights to determine which features are most important per subgroup. For two groups, returns a signed difference plot (GroupB - GroupA). For >2 groups, performs a non-inferential analysis: per-group mean importances, pairwise (B - A) difference tables for all pairs, and a multi-group plot for the features with the largest max-min spread across groups.

Usage

```
analyse_experts(
  gnn_results,
  prepared_data,
  group_mappings = NULL,
  top_n_features = 10,
  verbose = FALSE
)
```

Arguments

gnn_results	A list from train_gnn().
prepared_data	A list from prepare_data() (used to retrieve group mappings if not provided).
group_mappings	Optional named mapping from group codes (names) to labels (values).
top_n_features	Integer; number of top features to visualise.
verbose	Logical; print progress messages.

Value

A list with:

all_weights	Long table of feature importances by group & repeat
means_by_group_wide	Wide table of per-feature mean importance per group
pairwise_differences	Named list of B_vs_A difference tables (descriptive)
difference_plot	ggplot; only when there are exactly 2 groups
multi_group_plot	ggplot; only when there are >2 groups
top_features_multi	Long table used for the multi-group plot

Examples

```
if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  data("adult_ready_small", package = "fairGATE")
  d <- adult_ready_small
  # drop columns with missing values or no variation before scaling
  keep <- vapply(d, function(v) !anyNA(v) && (!is.numeric(v) || stats::sd(v) > 0), TRUE)
  pd <- prepare_data(d[, keep], outcome_var = "income", group_var = "sex")
  fit <- train_gnn(pd, run_tuning = FALSE,
                   best_params = list(lr = 0.01, hidden_dim = 8, dropout_rate = 0.1,
                                      lambda = 1, temperature = 1),
                   num_repeats = 1, epochs = 5, seed = 1)
  ex <- analyse_experts(fit, pd, top_n_features = 5)
  head(ex$means_by_group_wide)
}
```

`analyse_gnn_results` *Analyse and Visualise GNN Results*

Description

Generates ROC/Calibration outputs and subgroup gate analyses.

Usage

```
analyse_gnn_results(
  gnn_results,
  prepared_data,
  group_mappings = NULL,
  create_roc_plot = TRUE,
  create_calibration_plot = TRUE,
  analyse_gate_weights = TRUE,
  analyse_gate_entropy = TRUE,
  verbose = FALSE,
  nonparametric = FALSE
)
```

Arguments

`gnn_results`	List from `train_gnn()` (expects $final_results, $performance_summary, $gate_weights).
`prepared_data`	List from `prepare_data()` (used to retrieve group mappings if not provided).
`group_mappings`	Optional named mapping from group codes (names) to labels (values).
`create_roc_plot`	Logical; return ROC ggplot (default TRUE).
`create_calibration_plot`	Logical; return calibration ggplot (default TRUE).
`analyse_gate_weights`	Logical; analyse gate weights across groups (default TRUE).
`analyse_gate_entropy`	Logical; analyse gate entropy across groups (default TRUE).
`verbose`	Logical; print progress messages.
`nonparametric`	Logical; if TRUE and k>2, use Kruskal-Wallis + Wilcoxon instead of ANOVA + Tukey.

Value

A list of ggplots, test results, and summary tables.

Examples

```
if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  data("adult_ready_small", package = "fairGATE")
  d <- adult_ready_small
  # drop columns with missing values or no variation before scaling
  keep <- vapply(d, function(v) !anyNA(v) && (!is.numeric(v) || stats::sd(v) > 0), TRUE)
  pd <- prepare_data(d[, keep], outcome_var = "income", group_var = "sex")
  fit <- train_gnn(pd, run_tuning = FALSE,
                   best_params = list(lr = 0.01, hidden_dim = 8, dropout_rate = 0.1,
                                      lambda = 1, temperature = 1),
                   num_repeats = 1, epochs = 5, seed = 1)
  res <- analyse_gnn_results(fit, pd)
  names(res)
}
```

---

export_f360_csv *Export predictions for IBM Fairness 360*

Description

Create a CSV (or return a data frame) with columns expected by common IBM Fairness 360 workflows: subjectid, y_true, y_pred, score, group, group_label, plus optional gate columns.

Usage

```
export_f360_csv(
  gnn_results,
  prepared_data,
  path = NULL,
  include_gate_cols = TRUE,
  threshold = 0.5,
  verbose = FALSE
)
```

Arguments

gnn_results	List returned by train_gnn(); must contain $final_results.
prepared_data	List returned by prepare_data(); used to pull group mappings.
path	Optional file path to write the CSV. If NULL, no file is written.
include_gate_cols	Logical; include gate_prob_expert_* and gate_entropy if available. Default TRUE.
threshold	Numeric between 0 and 1 inclusive; classification threshold for y_pred. Default 0.5.
verbose	Logical; if TRUE, prints progress messages. Default FALSE.

Value

Invisibly returns the data.frame that is written (or would be written).

Examples

```
## Not run:
tmp <- file.path(tempdir(), "fairness360_input.csv")
export_f360_csv(
  gnn_results   = final_fit,
  prepared_data = prepared_data_chd,
  path          = tmp,
  include_gate_cols = TRUE,
  threshold     = 0.5,
  verbose       = TRUE
)

## End(Not run)
```

`fairgate` *Fit fairGATE in One Call*

Description

Fits the cross-fitted base learner (XGBoost or a neural network), the cross-fitted disparity signal, searches the fairGATE configuration space on an inner split under the selection rule of the accompanying study, and fits the selected configuration on every partition with per-person adjustment intervals. The defaults reproduce the fairGATE protocol of the study.

Usage

```
fairgate(
  X,
  y = NULL,
  group = NULL,
  family = c("binomial", "gaussian"),
  reference_group = NULL,
  splits = NULL,
  n_partitions = 1L,
  test_prop = 0.2,
  inner = NULL,
  base_learner = c("xgboost", "nn"),
  base_params = NULL,
  base_tune_args = NULL,
  base_score = NULL,
  signal = NULL,
  signal_learner = c("xgboost", "nn"),
  signal_params = NULL,
  gate_signal = NULL,
```

```
  params = NULL,
  space = fairgate_space(),
  n_draws = space$n_draws %||% 48L,
  perf_tolerance = 0.01,
  crossed_groups = NULL,
  epochs = 300L,
  es_prop = 0.15,
  es_patience = 30L,
  interval_reps = 10L,
  interval_epochs = 10L,
  seed = 4242L,
  nthread = 1L,
  verbose = FALSE
)
```

Arguments

X	Numeric feature matrix or data frame, without the protected attribute; or the list returned by prepare_data(), in which case y and group are taken from it.
y	Outcome: 0/1 (family = "binomial") or numeric ("gaussian").
group	Integer codes of the protected attribute. Used only in the fairness objective, in the target of the disparity signal and in evaluation; never as an input to the base learner, the gate or the experts.
family	"binomial" or "gaussian".
reference_group	Code of the reference (privileged) group. Defaults to the largest group.
splits	Optional list of partitions, each list(train =, test =) of row indices. When NULL, n_partitions stratified partitions with held-out share test_prop are drawn.
n_partitions, test_prop	Number of partitions and held-out share used when splits is NULL. Defaults 1 and 0.2. test_prop is also the nominal held-out share of the minimum-cell rule (see Details), so set it to the held-out share of splits when supplying your own partitions.
inner	Optional list(train =, valid =) of row indices for the configuration search. Defaults to a stratified 75:25 split of the training rows of the first partition.
base_learner	"xgboost" (default, as in the study) or "nn".
base_params	Optional fixed base-learner hyperparameters (see fairgate_base_learner()); when NULL they are tuned on inner.
base_tune_args	Optional list of further arguments to fairgate_tune_base_learner().
base_score	Optional precomputed base scores: a list with element inner (list(train =, test =) for inner$train and inner$valid) and element partitions (a list aligned with splits), on the response scale, training rows out of fold. When supplied, no base learner is fitted.
signal	Logical; use the cross-fitted disparity signal in the gate. Defaults to the fixed setting of space (TRUE).

`signal_learner` Learner for the disparity signal, `"xgboost"` (default, as in the study) or `"nn"`.

`signal_params` Hyperparameters of the signal learner; default 200 rounds, learning rate 0.05, depth 4, minimum child weight 5, subsample and column subsample 0.8, L2 penalty 1.

`gate_signal` Optional precomputed signal in the layout of `base_score`; overrides `signal_learner`.

`params` Optional fixed fairGATE configuration (a named list with the elements of `fairgate_space()$space`); when supplied, no search is run.

`space` Search space and fixed settings, default `fairgate_space()`.

`n_draws` Number of configurations searched. Default 48.

`perf_tolerance` Selection rule: among configurations whose inner-validation AUC or R-squared is within this distance of the best, the one with the smallest worst-group absolute disparity (average odds difference at the prevalence-matched threshold, or conditional mean prediction difference in outcome standard deviations). Default 0.01.

`crossed_groups` Optional two-column matrix of two protected attributes for the crossed penalty (used only when `space$fixed$crossed` is `TRUE` and the penalty is a variance component).

`epochs`, `es_prop`, `es_patience` Epoch cap, early-stopping slice and patience of `train_gnn()`. Defaults 300, 0.15 and 30.

`interval_reps`, `interval_epochs` Bootstrap replicates and epochs per replicate of the per-person adjustment intervals. Defaults 10 and 10; `interval_reps = 0` switches the intervals off.

`seed` Integer seed. Configuration `j` of the search is fitted with `seed + j`, final fits with `seed`, the base learner and signal of partition `r` with `seed + r` (the inner split with `seed`).

`nthread` Threads used by XGBoost. Default 1.

`verbose` Logical; print progress.

Details

Base learner. Fitted without the protected attribute by `fairgate_base_learner()`: 5-fold out-of-fold scores for training rows, a fit on all training rows for held-out rows.

Disparity signal. For training rows a target is computed by `fairgate_disparity_target()` from the outcome, the protected group and the out-of-fold base score; a regression on the features and the link-scale base score predicts it, cross-fitted in the same way, and the prediction is given to the gate only.

Search and selection. `n_draws` configurations drawn by `fairgate_draw_configs()` are fitted on `inner` and scored on its validation rows; the selection rule is described under `perf_tolerance`, with the worst group taken over groups passing a minimum-cell rule (at least 10 expected positives and negatives, or 50 expected people, per held-out set, computed as full-sample counts times `test_prop`). When no group passes the rule (small samples), the worst group is taken over all non-reference groups, with a warning.

Final fit. The selected configuration is fitted on every partition by `train_gnn()` with median imputation, standardisation, early stopping on a stratified 15 per cent slice (smoothed over 5 epochs,

earliest selectable epoch 10, patience 30, cap 300) and per-person adjustment intervals from warm-started bootstrap replicates.

The full protocol is computationally heavy (48 searched configurations of up to 300 epochs each). For a quick look reduce `n_draws`, `epochs` and `interval_reps`.

Value

An object of class `fairgate_fit`: a list with `predictions` (a data frame with `partition`, `row`, `y`, `group`, `base`, `pred` and `adjustment` for every held-out row), `export` (the per-person export: the routing weights, base weight, adjustment and, when intervals are on, `adj_lo`, `adj_hi`, `adj_sd` and `interval_excludes_zero`), `selected` (the selected configuration with its fixed settings), `search` (one row per searched configuration with inner-validation performance and worst-group disparity), `base` (base-learner scores and settings), `signal`, `fit` (the `train_gnn()` result) and `settings`.

Examples

```
if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed() &&
    requireNamespace("xgboost", quietly = TRUE) &&
    utils::packageVersion("xgboost") >= "3.1.0") {
  set.seed(1)
  n <- 800
  g <- rbinom(n, 1, 0.3)
  X <- cbind(x1 = rnorm(n) + 0.5 * g, x2 = rnorm(n), x3 = rnorm(n))
  y <- rbinom(n, 1, stats::plogis(X[, 1] - 0.5 * X[, 2]))
  fg <- fairgate(X, y, g, "binomial", reference_group = 0,
                 base_params = list(nrounds = 50, max_depth = 3),
                 n_draws = 2, epochs = 20, interval_reps = 2,
                 interval_epochs = 2)
  fg
  head(fg$export)
}
```

---

fairgate_base_grid *Default Hyperparameter Grids for the Base Learner*

Description

Returns the candidate values searched by `fairgate_tune_base_learner()` for each base learner. Every element is a vector of candidate values; a single value is held fixed.

Usage

```
fairgate_base_grid(base_learner = c("xgboost", "nn"))
```

Arguments

`base_learner`	`"xgboost"` or `"nn"`.

Details

XGBoost (`"xgboost"`, gradient-boosted trees with the histogram method):

`max_depth` tree depth, 3, 4, 6 or 8

`eta` learning rate, 0.02, 0.05 or 0.1

`nrounds` ceiling on boosting rounds, 1000; the number actually used is chosen by early stopping on the inner validation slice

`early_stopping_rounds` 50 rounds without improvement in validation loss

`subsample` row subsampling per tree, 0.7, 0.9 or 1

`colsample_bytree` column subsampling per tree, 0.7, 0.9 or 1

`min_child_weight` minimum hessian weight per leaf, 1, 5 or 10

`lambda`, `alpha` L2 penalty 0.5, 1 or 5 and L1 penalty 0 or 0.5 on leaf weights

These are the values of the gradient-boosting comparator in the accompanying study, with the number of rounds chosen by early stopping rather than drawn from a list.

Neural network (`"nn"`, a torch multilayer perceptron with rectified linear units and dropout, trained by Adam on mini-batches):

`n_layers` hidden layers, 1, 2 or 3

`width` units per hidden layer, 32, 64, 128 or 256

`dropout` dropout rate, 0, 0.1, 0.2 or 0.3

`lr` Adam learning rate, 3e-4, 1e-3 or 3e-3

`weight_decay` Adam weight decay, 0, 1e-5, 1e-4 or 1e-3

`batch_size` mini-batch size, 256, 512 or 1024

`epochs` ceiling on epochs, 200; the number actually used is chosen by early stopping on the inner validation slice

`patience` 20 epochs without improvement in validation loss

The loss is binary cross-entropy on the logit (binary outcome) or squared error on the standardised outcome (continuous outcome).

Value

A named list of candidate values.

Examples

```
str(fairgate_base_grid("xgboost"))
str(fairgate_base_grid("nn"))
```

`fairgate_base_learner` *Cross-Fitted Base Learner (XGBoost or Neural Network)*

Description

Fits the base learner whose scores fairGATE corrects. Training rows receive out-of-fold scores from `n_folds`-fold cross-fitting, so that no training row is scored by a model that saw its outcome; held-out rows receive the prediction of a fit on all training rows. Features are median-imputed and standardised with the training rows' constants. The protected attribute is never an input: pass only the features in `X`.

Usage

```
fairgate_base_learner(
  X,
  y,
  family = c("binomial", "gaussian"),
  train = NULL,
  test = NULL,
  base_learner = c("xgboost", "nn"),
  params = NULL,
  n_folds = 5L,
  folds = NULL,
  seed = 4242L,
  standardise = TRUE,
  nthread = 1L,
  tune_args = NULL,
  verbose = FALSE
)
```

Arguments

`X`	Numeric feature matrix or data frame. It must not contain the protected attribute.
`y`	Outcome: 0/1 for `family = "binomial"`, numeric for `"gaussian"`.
`family`	`"binomial"` or `"gaussian"`.
`train`	Integer row indices of the training rows (default all rows).
`test`	Optional integer row indices of held-out rows to score.
`base_learner`	`"xgboost"` or `"nn"`.
`params`	Named list of fixed hyperparameters. When `NULL` (default) they are tuned first by `fairgate_tune_base_learner()` on an inner split of `train`. Entries not supplied take the learner's defaults (XGBoost: `nrounds = 500`, `eta = 0.05`, `max_depth = 4`, `min_child_weight = 1`, `subsample = 0.8`, `colsample_bytree = 0.8`, `lambda = 1`, `alpha = 0`; neural network: `n_layers = 2`, `width = 64`, `dropout = 0.1`, `lr = 0.001`, `weight_decay = 1e-4`, `batch_size = 512`, `epochs = 100`). With fixed parameters the number of rounds or epochs is used as given (no early stopping).

n_folds	Number of cross-fitting folds. Default 5.
folds	Optional integer vector of fold labels 1..n_folds, one per element of train, to reuse a fold assignment made elsewhere.
seed	Integer seed for the split, the random draws and the model fits. The caller's random number state is restored on exit.
standardise	Logical; centre and scale features with the inner training rows' constants (median imputation of missing values first).
nthread	Threads used by XGBoost. Default 1 (reproducible).
tune_args	Optional list of further arguments to fairgate_tune_base_learner() (for example grid, n_search, valid).
verbose	Logical; print one line per configuration.

Value

An object of class fairgate_base: a list with train and test (scores on the response scale, probabilities for a binary outcome, in the order of train and test), train_link and test_link (the same on the link scale: logit, or standardised by the training outcome mean and standard deviation), train_rows, test_rows, folds, params, tuning (the tuning result or NULL), perf (out-of-fold and, when test is given, held-out AUC or R-squared), family and base_learner. Pass list(list(train = b$train, test = b$test)) as base_score to train_gnn().

Examples

```
if (requireNamespace("xgboost", quietly = TRUE) &&
    utils::packageVersion("xgboost") >= "3.1.0") {
  set.seed(2)
  X <- matrix(rnorm(500 * 4), 500, 4)
  y <- X[, 1] + 0.5 * X[, 2] + rnorm(500)
  b <- fairgate_base_learner(X, y, "gaussian", train = 1:400, test = 401:500,
                             base_learner = "xgboost",
                             params = list(nrounds = 50, max_depth = 3))
  b$perf
}

if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  set.seed(3)
  X <- matrix(rnorm(400 * 4), 400, 4)
  y <- rbinom(400, 1, stats::plogis(X[, 1]))
  b <- fairgate_base_learner(X, y, "binomial", train = 1:300, test = 301:400,
                             base_learner = "nn",
                             params = list(epochs = 10, width = 16))
  b$perf
}
```

`fairgate_crossed_penalty` *Crossed Variance-Component Penalty for Intersectional Groups*

Description

Decomposes, within each outcome stratum, the between-cell dispersion of the mean predictions of the (a, b) intersectional cells into two main effects and an interaction, each estimated with a correction for the sampling noise in the cell means, and returns their sum over strata.

Usage

```
fairgate_crossed_penalty(
  pred,
  plan,
  penalty = c("random_effects", "bias_corrected", "variance"),
  eps = 1e-12
)
```

Arguments

`pred`	Predictions (tensor), on the penalty scale.
`plan`	From `fairgate_crossed_plan()`.
`penalty`	`"random_effects"`, `"bias_corrected"` or `"variance"`, applied to the main effects; the interaction is bias corrected unless `"variance"`.
`eps`	Floor on sampling variances.

Details

For stratum q write m_ab for the cell means and v_ab for their detached sampling variances. The main effect of A is the between-level variance component (as in `fairgate_penalty_from_cells()`) of the unweighted marginal means m_a. = mean_b m_ab, with sampling variances sum_b v_ab / B_a^2; likewise for B. The interaction is computed on the largest complete sub-grid A' x B': with residuals r_ab = m_ab - m_a. - m_.b + m_.., E(sum r^2) = (A'-1)(B'-1) sigma^2_int + (1 - 1/A')(1 - 1/B') sum v_ab, because the residual projection has constant diagonal, so sigma^2_int is estimated by max(0, (sum r^2 - (1 - 1/A')(1 - 1/B') sum v) / ((A'-1)(B'-1))). With a single level of B the penalty reduces to the one-way component of A.

Value

A scalar tensor.

Examples

```
if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  set.seed(1)
  A <- sample(0:1, 400, replace = TRUE); B <- sample(0:2, 400, replace = TRUE)
  strata <- sample(1:2, 400, replace = TRUE)
  pred <- torch::torch_tensor(stats::plogis(rnorm(400) + 0.3 * A * B))
  plan <- fairgate_crossed_plan(A, B, strata)
  fairgate_crossed_penalty(pred, plan, penalty = "random_effects")
}
```

---

fairgate_crossed_plan *Cell Structure for Crossed (Intersectional) Variance Components*

Description

Cell Structure for Crossed (Intersectional) Variance Components

Usage

```
fairgate_crossed_plan(
  A,
  B,
  strata,
  Q = max(as.integer(strata)),
  min_cell = 2L,
  dtype = torch::torch_float()
)
```

Arguments

A, B	Codes of the two protected attributes, one per row.
strata	Outcome stratum identifiers in 1:Q.
Q	Number of strata.
min_cell	Smallest (a, b, stratum) cell that may contribute.
dtype	torch dtype.

Value

A list with one element per stratum holding the indicator matrix of the contributing (a, b) cells, their sizes and codes.

Examples

```
if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  set.seed(1)
  A <- sample(0:1, 400, replace = TRUE); B <- sample(0:2, 400, replace = TRUE)
  strata <- sample(1:2, 400, replace = TRUE)
  plan <- fairgate_crossed_plan(A, B, strata)
  length(plan)
}
```

`fairgate_disparity_target` *Per-Person Disparity-Contribution Target*

Description

The quantity whose cross-fitted prediction fairGATE feeds to the gate: for each training individual, the signed direction in which moving that individual's score would close its group's gap relative to the reference group, times its leverage on the decision.

Usage

```
fairgate_disparity_target(
  y,
  group,
  base,
  reference,
  family = c("binomial", "gaussian"),
  prevalence = NULL,
  n_strata = 5L,
  bandwidth = 1
)
```

Arguments

`y`	Outcome.
`group`	Protected group codes.
`base`	Base-learner scores on the response scale, out of fold.
`reference`	Reference group code.
`family`	`"binomial"` or `"gaussian"`.
`prevalence`	Share flagged positive (binomial); defaults to `mean(y)`.
`n_strata`	Outcome strata for `"gaussian"`.
`bandwidth`	Bandwidth, on the logit scale, of the Gaussian leverage kernel centred on the threshold (binomial).

Details

Binomial: with t the prevalence-matched threshold of base and gaps d_kc = r_kc - r_ref,c in the rate of positive decisions of group k in outcome class c, the target is -d_{g_i y_i} * phi((logit b_i - logit t)/h) for a non-reference individual and +sum_k w_k d_{k y_i} * phi(.) for a reference one, w_k the non-reference group shares. Gaussian: the same with the stratum-conditional mean base prediction gap, in outcome standard deviations, and unit leverage. The target uses the outcome and the protected attribute, so it must be cross-fitted from the features alone before it is given to the gate; the protected attribute itself never is.

Value

A numeric vector, one value per row.

Examples

```
set.seed(1)
n <- 400
g <- sample(0:2, n, replace = TRUE)
y <- rbinom(n, 1, 0.4)
base <- stats::plogis(rnorm(n) + y - 0.3 * g)   # out-of-fold base scores
tgt <- fairgate_disparity_target(y, g, base, reference = 0, family = "binomial")
tapply(tgt, list(g, y), mean)
```

---

fairgate_draw_configs *Draw fairGATE Configurations*

Description

Draws distinct random configurations from a search space with the generator settings of the accompanying study.

Usage

```
fairgate_draw_configs(space, n = 48L, seed = 4242L)
```

Arguments

space	Named list of candidate values, as fairgate_space()$space.
n	Number of distinct configurations.
seed	Integer seed.

Value

A list of configurations (named lists). With the defaults of fairgate_space() this is the configuration list searched in the study.

Examples

```
g <- fairgate_draw_configs(fairgate_space()$space, 3)
str(g[[1]])
```

`fairgate_export` *Per-Person Export of a fairGATE Fit*

Description

Returns, and optionally writes, the per-person table of a `fairgate()` fit: partition, row, group, outcome, base score, fairGATE prediction, adjustment, base weight, routing weights and, when intervals were fitted, the adjustment interval (`adj_lo`, `adj_hi`) and whether it excludes zero.

Usage

```
fairgate_export(x, file = NULL)
```

Arguments

`x`	A `fairgate_fit` object.
`file`	Optional path of a CSV file to write. Nothing is written when `NULL` (default).

Value

A data frame, invisibly when `file` is given.

Examples

```
# fg <- fairgate(...)
# tab <- fairgate_export(fg)
# fairgate_export(fg, file.path(tempdir(), "fairgate_export.csv"))
```

`fairgate_fair_penalty` *fairGATE Fairness Penalty from Individual Predictions*

Description

Evaluates the total fairness penalty `sum_q L_q` on a vector of individual predictions, on the scale on which the penalty is defined: probabilities for a binary outcome, the response scale for a continuous one.

Usage

```
fairgate_fair_penalty(
  pred,
  group = NULL,
  strata = NULL,
  Q = NULL,
  penalty = c("random_effects", "bias_corrected", "variance"),
  eps = 1e-12,
  plan = NULL
)
```

Arguments

pred	Individual predictions: a torch_tensor of length n, or a numeric vector. A tensor carrying a gradient gives a differentiable result.
group	Protected group codes, one per row, or NULL when plan is supplied.
strata	Outcome stratum identifiers in 1:Q, one per row, or NULL when plan is supplied.
Q	Number of outcome strata. Defaults to max(strata).
penalty	One of "random_effects" (default), "bias_corrected" or "variance"; see fairgate_penalty_from_cells().
eps	Lower bound applied to the cell sampling variances before they are inverted. Default 1e-12.
plan	Optional precomputed cell structure from fairgate_penalty_plan(). Supplying it avoids rebuilding the indicator matrices, which is what the training loop does.

Details

Per cell, the mean prediction is differentiable and the sampling variance is not: the within-cell variance is computed from detached predictions, so the weights of the random-effects rung cannot be gamed by inflating within-cell spread. Cells with fewer than two rows carry no estimable within-cell variance and are dropped for every rung except "variance", which needs none and admits cells of one row.

Value

A scalar of the same kind as pred (tensor in, tensor out), zero when no stratum contributes.

Examples

```
if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  set.seed(1)
  group <- sample(0:2, 200, replace = TRUE)
  y <- rbinom(200, 1, 0.4)
 pred <- torch::torch_tensor(stats::plogis(rnorm(200) + 0.3 * group), requires_grad = TRUE)
  pen <- fairgate_fair_penalty(pred, group, y + 1L, penalty = "random_effects")
  pen$backward()
  as.numeric(pen)
```

```
}
```

`fairgate_gate_map` *Gate Maps: softmax, sparsemax and entmax-1.5*

Description

Maps each row of a matrix of gate logits to a probability vector. The sparse maps return exact zeros, so a gate built on them can place all of an individual's weight on a single component, in particular on the base learner, in which case the fairGATE adjustment for that individual is exactly zero.

Usage

```
fairgate_gate_map(z, type = c("softmax", "sparsemax", "entmax15"))
```

Arguments

`z`	A two-dimensional `torch_tensor` of logits (rows by components), already divided by the gate temperature.
`type`	One of `"softmax"`, `"sparsemax"` or `"entmax15"`.

Value

A tensor of the same shape whose rows are non-negative and sum to one. Differentiable; the sparse maps use their closed-form Jacobians.

Examples

```
if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  z <- torch::torch_tensor(matrix(c(2, 1, -1, 0.5, 0.4, 0.3), 2, byrow = TRUE))
  fairgate_gate_map(z, "softmax")
  fairgate_gate_map(z, "sparsemax")
  fairgate_gate_map(z, "entmax15")
}
```

`fairgate_group_disparity` *Multi-Group Separation Disparity: Per Group, Worst Group and Joint*

Description

Computes, for every non-reference group, the average odds difference (binary, at a supplied threshold) or the conditional mean prediction difference (continuous, in outcome standard deviations) relative to the reference group, and summarises them.

Usage

```
fairgate_group_disparity(
  y,
  pred,
  group,
  reference,
  family = c("binomial", "gaussian"),
  threshold = NULL,
  edges = NULL,
  scale = NULL,
  eligible = NULL
)
```

Arguments

`y`	Outcome.
`pred`	Predictions (probabilities for a binary outcome, the outcome scale for a continuous one).
`group`	Protected group codes.
`reference`	Reference group code.
`family`	`"binomial"` or `"gaussian"`.
`threshold`	Decision threshold (binomial). Use the prevalence-matched threshold `quantile(pred, 1 - prevalence)`.
`edges`	Stratum edges for the continuous outcome; default the equal-frequency quintiles of `y`.
`scale`	Outcome standard deviation used to express CMPD.
`eligible`	Optional vector of group codes over which the worst-group maximum and the joint summaries are taken (a minimum-cell rule). Groups outside it are still reported in `per_group`, flagged `eligible = FALSE`. `NULL` uses every group.

Value

A list: `per_group` (data frame of group, n, gap), `worst` (the largest absolute gap), `worst_group`, `signed_worst` (the gap of that group), `joint_mean` (size-weighted mean absolute gap), `joint_pairwise` (largest absolute gap between any two groups, reference included) and `pooled` (every non-reference group pooled against the reference).

Examples

```
set.seed(1)
n <- 600
g <- sample(0:2, n, replace = TRUE)
y <- rbinom(n, 1, 0.4)
pred <- stats::plogis(rnorm(n) + y - 0.4 * g)
thr <- stats::quantile(pred, 1 - mean(y), names = FALSE)   # prevalence-matched
d <- fairgate_group_disparity(y, pred, g, reference = 0, family = "binomial",
                              threshold = thr)
d$per_group
d$worst
```

---

fairgate_group_gap_penalty

Worst-Group Conditional Mean Gap Penalty (continuous outcomes)

Description

The continuous counterpart of `fairgate_threshold_penalty()`: for each non-reference group the squared mean over outcome strata of the difference between that group's and the reference group's mean prediction (the squared CMPD of that group), aggregated by the maximum or the mean.

Usage

```
fairgate_group_gap_penalty(
  pred,
  plan,
  reference = NULL,
  agg = c("max", "mean")
)
```

Arguments

`pred`	Predictions on the scale the penalty is defined on.
`plan`	From `fairgate_penalty_plan()` with `min_cell = 1`.
`reference`	Reference group code, or `NULL` for the pooled stratum mean.
`agg`	`"max"` or `"mean"`.

Value

A scalar tensor.

Examples

```
if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  set.seed(1)
  group <- sample(0:2, 300, replace = TRUE)
  y <- rnorm(300)
  strata <- as.integer(cut(y, stats::quantile(y, 0:5 / 5), include.lowest = TRUE))
  pred <- torch::torch_tensor(y + 0.2 * group)
  plan <- fairgate_penalty_plan(group, strata, min_cell = 1L)
  fairgate_group_gap_penalty(pred, plan, reference = 0)
}
```

---

fairgate_lambda_ladder

Frontier Ladder of Expert Fairness Weights

Description

The auxiliary fairness weights carried by the individual experts of the frontier-anchored model.

Usage

```
fairgate_lambda_ladder(K, lambda_max)
```

Arguments

K	Number of experts, which is the number of protected groups.
lambda_max	Fairness weight of the most heavily constrained expert.

Details

Expert 1 is the accuracy anchor and carries $\lambda_1 = 0$. The remaining experts are spaced geometrically, $\lambda_k = \lambda_{\max} 10^{-(K-k)/(K-1)}$ for $k \geq 2$, so that $\lambda_K = \lambda_{\max}$. The experts therefore sit at different points of the fairness and accuracy trade-off by construction, rather than having to discover a division of labour from a symmetric starting point that the gate gradient cannot leave.

The gate gradient is the reason the anchoring is structural rather than cosmetic. With $\hat{y}_i = \sum_k \pi_{ik} f_{ik}$ and $\pi_i = \mathrm{softmax}(z_i/\tau)$, $\partial \hat{y}_i / \partial z_{il} = \tau^{-1} \pi_{il}(f_{il} - \hat{y}_i)$, so the gradient reaching a gate logit is proportional to that expert's departure from the mixture. Identical experts make it exactly zero: the gate cannot specialise and the experts never differentiate.

Value

A numeric vector of length K, increasing, with first element zero and last element lambda_max. Length-one input returns 0.

Examples

```
fairgate_lambda_ladder(K = 4, lambda_max = 16)
```

`fairgate_penalty_from_cells` *Between-Group Variance Components for the fairGATE Fairness Penalty*

Description

Computes one stratum's contribution to the fairGATE fairness penalty from the group-conditional mean predictions of that stratum and their sampling variances. This is the lowest level of the penalty and is exported so that the algebra can be exercised directly on constructed inputs, independently of any model fit.

Usage

```
fairgate_penalty_from_cells(
  mhat,
  vhat = NULL,
  penalty = c("random_effects", "bias_corrected", "variance"),
  eps = 1e-12
)
```

Arguments

`mhat`	Group-conditional mean predictions for one outcome stratum: a `torch_tensor` of length `K_q`, or a numeric vector of the same length. When a tensor carrying a gradient is supplied, the returned value is differentiable with respect to it. A numeric vector is evaluated in base R and does not need torch.
`vhat`	Sampling variances of the entries of `mhat`, that is `s2_kq / n_kq` where `s2_kq` is the within-cell sample variance of the individual predictions (denominator `n_kq - 1`). A numeric vector or a `torch_tensor` of the same length as `mhat`. Ignored, and may be `NULL`, when `penalty = "variance"`. Always used detached from the autograd graph; see Details.
`penalty`	One of `"random_effects"` (default), `"bias_corrected"` or `"variance"`. See Details.
`eps`	Lower bound applied to `vhat` before it is inverted. Default `1e-12`.

Details

Write the cell (k, q) for the individuals in protected group k and outcome stratum q, of size n_{kq}, with mean prediction $\hat{m}_{kq}$, within-cell sample variance s^2_{kq} and sampling variance $\hat{v}_{kq} = s^2_{kq}/n_{kq}$. Let K_q be the number of groups contributing to stratum q. The three rungs are

`"variance"` $L_q = \mathrm{Var}_k(\hat{m}_{kq})$, the unbiased sample variance over the K_q group means. This is the penalty of fairGATE 0.2.0 and earlier.

`"bias_corrected"` $L_q = \max(0, \mathrm{Var}_k(\hat{m}_{kq}) - K_q^{-1} \sum_k \hat{v}_{kq})$.

`"random_effects"` the DerSimonian and Laird between-group variance component. With $w_{kq} = 1/\hat{v}_{kq}$, $\bar{m}_{wq} = \sum_k w_{kq}\hat{m}_{kq} / \sum_k w_{kq}$, $Q_q = \sum_k w_{kq}(\hat{m}_{kq} - \bar{m}_{wq})^2$ and $C_q = \sum_k w_{kq} - (\sum_k w_{kq}^2)/(\sum_k w_{kq})$, $L_q = \max(0, (Q_q - (K_q - 1))/C_q)$.

The correction terms follow from the bias of the sample variance as an estimator of the between-group dispersion. Writing $\hat{m}_k = \mu_k + e_k$ with e_k independent, mean zero and variance v_k, the unbiased sample variance S^2 over the K estimates satisfies $E[S^2] = S^2(\mu) + K^{-1} \sum_k v_k$. The second term is a floor that no improvement in fairness can remove: an optimiser can only reduce it by distorting predictions.

`vhat`, and therefore the weights and the constant C_q, are always detached from the autograd graph. This is load-bearing rather than an optimisation. If they were differentiable, the optimiser could lower the penalty by inflating the within-cell prediction variance, which raises $\hat{v}_{kq}$ and shrinks the weighted departure from the pooled mean without making any group's mean prediction closer to any other's. Gradients therefore flow only through `mhat`.

The rungs nest exactly. If every $\hat{v}_{kq}$ equals a common v, then $\sum w = K/v$, $\sum w^2 = K/v^2$ and $C = (K-1)/v$, while $Q = (K-1)S^2/v$, so the random-effects rung collapses to $\max(0, S^2 - v)$, which is the bias-corrected rung; and as $v \to 0$ that collapses in turn to S^2, the 0.2.0 penalty.

Value

A scalar: a `torch_tensor` if `mhat` was a tensor, otherwise a numeric value. Zero when fewer than two groups are supplied.

See Also

`fairgate_fair_penalty()` for the same quantity computed from individual predictions, and `train_gnn()` for its use in fitting.

Examples

```
# Three groups' mean predictions in one outcome stratum, with their
# sampling variances
mhat <- c(0.30, 0.35, 0.42)
vhat <- c(0.002, 0.004, 0.010)
fairgate_penalty_from_cells(mhat, penalty = "variance")
fairgate_penalty_from_cells(mhat, vhat, penalty = "bias_corrected")
fairgate_penalty_from_cells(mhat, vhat, penalty = "random_effects")
```

---

fairgate_penalty_plan *Cell Structure for the fairGATE Fairness Penalty*

Description

Builds the fixed part of the fairness penalty for one set of rows: which group-by-stratum cells contribute, and the indicator matrices and cell sizes needed to form the group-conditional means. The result depends only on the protected attribute and the outcome strata, so it is computed once per partition and reused at every epoch.

Usage

```
fairgate_penalty_plan(
  group,
  strata,
  Q = max(as.integer(strata)),
  min_cell = 2L,
  dtype = torch::torch_float()
)
```

Arguments

group	Integer or numeric vector of protected group codes, one per row.
strata	Integer vector of outcome stratum identifiers in 1:Q, one per row. For a binary outcome these are the two outcome classes.
Q	Number of outcome strata. Defaults to max(strata).
min_cell	Smallest cell size that may contribute. Cells smaller than this are dropped. Use 2 whenever a within-cell variance is needed, that is for every rung except "variance".
dtype	A torch dtype for the indicator matrices. Defaults to torch::torch_float().

Details

A group contributes to stratum q only if its cell in that stratum has at least min_cell rows, and a stratum contributes only if at least two groups contribute to it. Strata are treated independently, so a group absent from one stratum still contributes to the others. Note that the 0.2.0 penalty used a stricter global rule in which a group had to be present in every stratum to contribute to any of them; train_gnn(penalty = "variance") retains that rule for exact backward compatibility.

Value

A list with one element per contributing stratum, each a list with the stratum index q, the contributing group codes, the indicator matrix M (rows by contributing groups), the cell sizes n, and K the number of contributing groups. An empty list when no stratum contributes.

Examples

```
if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  set.seed(1)
  group <- sample(0:2, 60, replace = TRUE)
  strata <- sample(1:2, 60, replace = TRUE)
  plan <- fairgate_penalty_plan(group, strata)
  length(plan)
}
```

fairgate_space *fairGATE Search Space and Fixed Settings*

Description

The configuration space searched by `fairgate()` and the settings held fixed, as used in the accompanying study.

Usage

```
fairgate_space()
```

Value

A list with `space` (candidate values: learning rate `lr`, hidden width `hidden_dim`, `dropout_rate`, gate `temperature`, fairness weight `lambda`, `penalty` type, thresholded-penalty weight `lambda_thr`, `intervention_cost` and `coupling`), `fixed` (gate, routing balance, adversary, smoothing window and earliest selectable epoch of early stopping, threshold temperature, target and aggregation, whether the disparity signal and crossed groups are used), `n_draws` (48 distinct random configurations) and `seed` (4242; draws use the Mersenne-Twister generator with Inversion and Rejection sampling, `lambda_thr` is set to 0 when `penalty = "thresholded"`, and duplicate draws are rejected).

Examples

```
str(fairgate_space()$space)
```

fairgate_threshold_penalty *Differentiable Penalty on Thresholded Group Error Rates*

Description

Replaces the hard decision `1[p_i >= t]` by `sigmoid((p_i - t) / s)`, with `t` the prevalence-matched threshold of the current batch (the `1 - prevalence` quantile of the predictions, treated as a constant) and `s` a temperature, and penalises the worst group's gap in the resulting soft true- and false-positive rates relative to the reference group.

Usage

```
fairgate_threshold_penalty(
  pred,
  plan,
  prevalence,
  temp = 0.05,
  target = c("eo", "aod"),
  agg = c("max", "mean")
)
```

Arguments

pred	Predicted probabilities, a torch_tensor of length n.
plan	From fairgate_threshold_plan().
prevalence	Share of rows to flag positive, in (0, 1): the training prevalence.
temp	Temperature s of the smooth indicator, on the probability scale.
target	"eo" penalises 0.5 (dTPR^2 + dFPR^2) per group, an upper bound on the squared average odds difference; "aod" penalises AOD^2 = (0.5 (dTPR + dFPR))^2.
agg	"max" (worst group) or "mean" over non-reference groups.

Value

A scalar tensor. As temp goes to zero the soft rates converge to the hard rates at threshold t.

Examples

```
if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  set.seed(1)
  group <- sample(0:2, 300, replace = TRUE)
  y <- rbinom(300, 1, 0.4)
  p <- torch::torch_tensor(stats::plogis(rnorm(300) + y + 0.4 * group))
  plan <- fairgate_threshold_plan(group, y, reference = 0)
  fairgate_threshold_penalty(p, plan, prevalence = mean(y))
}
```

---

fairgate_threshold_plan

Cell Structure for the Thresholded Error-Rate Penalty

Description

Cell Structure for the Thresholded Error-Rate Penalty

Usage

```
fairgate_threshold_plan(
  group,
  y,
  reference = NULL,
  min_cell = 2L,
  dtype = torch::torch_float()
)
```

Arguments

group	Protected group codes, one per row.
y	Binary outcome (0/1) for family = "binomial"; ignored otherwise.
reference	Code of the reference group. NULL compares every group with the pooled rate over all rows.
min_cell	Smallest (group, outcome class) cell that may contribute.
dtype	torch dtype of the indicator matrices.

Value

A list with indicator matrices M1, M0 (rows by groups) for the positive and negative classes, their sizes, the group codes and the column of the reference group (0 for the pooled comparison).

Examples

```
if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  set.seed(1)
  group <- sample(0:2, 100, replace = TRUE)
  y <- rbinom(100, 1, 0.4)
  plan <- fairgate_threshold_plan(group, y, reference = 0)
  names(plan)
}
```

---

fairgate_tune_base_learner

Tune the Base Learner on an Inner Split

Description

Evaluates candidate hyperparameters of the XGBoost or neural-network base learner on an inner training/validation split of the training rows, with early stopping on the validation slice, and selects the configuration with the smallest validation loss (log loss for a binary outcome, mean squared error for a continuous one). The AUC or R-squared is reported beside it.

Usage

```
fairgate_tune_base_learner(
  X,
  y,
  family = c("binomial", "gaussian"),
  train = NULL,
  valid = NULL,
  base_learner = c("xgboost", "nn"),
  grid = NULL,
  n_search = 24L,
```

```
  search = c("random", "grid"),
  inner_prop = 0.25,
  seed = 4242L,
  standardise = TRUE,
  nthread = 1L,
  verbose = FALSE
)
```

Arguments

X	Numeric feature matrix or data frame. It must not contain the protected attribute.
y	Outcome: 0/1 for `family = "binomial"`, numeric for `"gaussian"`.
family	`"binomial"` or `"gaussian"`.
train	Integer row indices to tune on (default all rows).
valid	Optional integer row indices of the validation slice. When `NULL`, `train` is split at random into an inner training part and a validation share `inner_prop` (stratified on the outcome when binary).
base_learner	`"xgboost"` or `"nn"`.
grid	Named list of candidate values; defaults to `fairgate_base_grid()`. Elements not supplied take the defaults of the chosen learner.
n_search	Number of configurations to evaluate by seeded random search; when the grid has no more points than this, every point is evaluated.
search	`"random"` (default) or `"grid"` (every point of `grid`).
inner_prop	Validation share when `valid` is `NULL`. Default 0.25.
seed	Integer seed for the split, the random draws and the model fits. The caller's random number state is restored on exit.
standardise	Logical; centre and scale features with the inner training rows' constants (median imputation of missing values first).
nthread	Threads used by XGBoost. Default 1 (reproducible).
verbose	Logical; print one line per configuration.

Value

A list with `best` (the selected hyperparameters, with `nrounds` or `epochs` set to the early-stopped value, ready to pass as `params` to `fairgate_base_learner()`), `results` (a data frame with one row per configuration: its hyperparameters, `val_loss`, `val_perf` (AUC or R-squared), `best_iteration`, `secs` and `selected`), and `inner` (the training and validation row indices used).

Examples

```
if (requireNamespace("xgboost", quietly = TRUE) &&
    utils::packageVersion("xgboost") >= "3.1.0") {
  set.seed(1)
  X <- matrix(rnorm(600 * 5), 600, 5)
  y <- rbinom(600, 1, stats::plogis(X[, 1] - X[, 2]))
  tu <- fairgate_tune_base_learner(X, y, "binomial", base_learner = "xgboost",
```

```
                                    n_search = 3, seed = 1)
  tu$results[, c("max_depth", "eta", "val_loss", "val_perf", "best_iteration")]
}
```

---

plot_sankey *Create a Sankey Plot (robust; aggregates to one row per subject)*

Description

Visualises routing from Actual Group -> Assigned Expert (2-axis), or Actual Group -> Learned Feature Profile -> Assigned Expert (3-axis) when feature mapping is available.

Usage

```
plot_sankey(
  prepared_data,
  gnn_results,
  expert_results = NULL,
  top_n_per_side = 2,
  use_profiles = TRUE,
  verbose = FALSE
)
```

Arguments

prepared_data	List from prepare_data(); used for group labels and (optionally) feature mapping. Needs feature_names for 3-axis; subject_ids to align subjects to X.
gnn_results	List from train_gnn(); uses $final_results and $gate_weights.
expert_results	Optional list from analyse_experts(); used only for picking opposed features in 3-axis.
top_n_per_side	Integer; number of features per side to define Profile A/B (default 2) for 3-axis.
use_profiles	Logical; try 3-axis when possible (default TRUE). If FALSE, always do 2-axis.
verbose	Logical; print progress.

Value

A ggplot object.

Examples

```
if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  data("adult_ready_small", package = "fairGATE")
  d <- adult_ready_small
  # drop columns with missing values or no variation before scaling
  keep <- vapply(d, function(v) !anyNA(v) && (!is.numeric(v) || stats::sd(v) > 0), TRUE)
  pd <- prepare_data(d[, keep], outcome_var = "income", group_var = "sex")
  fit <- train_gnn(pd, run_tuning = FALSE,
```

```
                  best_params = list(lr = 0.01, hidden_dim = 8, dropout_rate = 0.1,
                                     lambda = 1, temperature = 1),
                  num_repeats = 1, epochs = 5, seed = 1)
  p <- plot_sankey(pd, fit, use_profiles = FALSE)
  print(p)
}
```

`prepare_data` *Prepare Data for GNN Training*

Description

This function takes a raw dataframe, cleans it, defines the outcome and group variables, and scales the feature matrix. If no `group_mappings` are provided, they are automatically generated from the unique values (or factor levels) of `group_var`.

Usage

```
prepare_data(
  data,
  outcome_var,
  group_var,
  group_mappings = NULL,
  cols_to_remove = NULL
)
```

Arguments

`data`	A dataframe containing the raw data.
`outcome_var`	A string with the column name of the binary outcome (must be 0 or 1).
`group_var`	A string with the column name of the sensitive attribute.
`group_mappings`	Optional named list mapping values in `group_var` to numeric codes (e.g., `list("Male" = 0, "Female" = 1)`).
`cols_to_remove`	A character vector of column names to exclude from the feature matrix (e.g., IDs, highly collinear vars).

Value

A list containing:

`X`	The scaled feature matrix.
`y`	The numeric outcome vector.
`group`	The numeric group vector.
`feature_names`	The names of the features used.
`subject_ids`	A vector of subject IDs, if a 'subjectid' column exists.
`group_mappings`	Added as an attribute for downstream use.

Examples

```
my_data <- data.frame(
  subjectid = 1:10,
  remission = sample(0:1, 10, replace = TRUE),
  gender = sample(c("M", "F"), 10, replace = TRUE),
  feature1 = rnorm(10),
  feature2 = rnorm(10)
)

prepared_data <- prepare_data(
  data = my_data,
  outcome_var = "remission",
  group_var = "gender",
  cols_to_remove = c("subjectid")
)
```

`train_gnn` *Train and Evaluate the Gated Neural Network (robust splits + safe ROC)*

Description

Trains a subgroup-aware gated neural network with a fairness-constrained loss, optionally performs hyperparameter tuning with lightweight budgets, and returns predictions, gate/expert weights, and summary metrics. Designed to be CRAN-safe:

- No background installs, no saving unless requested
- CPU-only torch, capped threads to avoid oversubscription

Usage

```
train_gnn(
  prepared_data,
  hyper_grid,
  num_repeats = 20,
  epochs = 300,
  output_dir = tempdir(),
  run_tuning = TRUE,
  best_params = NULL,
  save_outputs = FALSE,
  seed = NULL,
  verbose = FALSE,
  tune_repeats = NULL,
  tune_epochs = NULL,
  splits = NULL,
  family = c("binomial", "gaussian"),
  n_strata = 2L,
  standardise = TRUE,
```

```
  impute = c("none", "median"),
  penalty = c("random_effects", "bias_corrected", "variance", "thresholded"),
  lambda_max = NULL,
  alpha = 0,
  early_stopping = FALSE,
  es_prop = 0.2,
  es_min_epochs = 1L,
  es_patience = NULL,
  base_score = NULL,
  coupling = c("none", "offset", "expert", "residual"),
  gate_bias_init = NULL,
  gate = c("softmax", "sparsemax", "entmax15"),
  intervention_cost = 0,
  lambda_thr = 0,
  thr_temp = 0.05,
  thr_target = c("eo", "aod"),
  thr_agg = c("max", "mean"),
  reference_group = NULL,
  gate_signal = NULL,
  balance = 0,
  crossed_groups = NULL,
  adversary = 0,
  adversary_hidden = 16L,
  es_smooth = NULL,
  interval_reps = 0L,
  interval_epochs = 10L,
  interval_level = 0.9
)
```

Arguments

prepared_data	List from prepare_data() containing: • X (matrix/data.frame of numeric features) • y (numeric 0/1) • group (numeric codes for sensitive subgroup) • feature_names (character vector; optional) • subject_ids (vector; optional)
hyper_grid	data.frame with columns: lr, hidden_dim, dropout_rate, lambda, temperature.
num_repeats	Integer (>=1). Repeated train/test splits for the **final** model (and for tuning if tune_repeats is not set).
epochs	Integer (>=1). Training epochs per run for the **final** model (and for tuning if tune_epochs is not set).
output_dir	Directory to write csv/rds if save_outputs = TRUE. Defaults to tempdir().
run_tuning	Logical. If TRUE, runs a grid search using hyper_grid and picks best by mean AUC.
best_params	data.frame/list with lr, hidden_dim, dropout_rate, lambda, temperature if run_tuning = FALSE.

save_outputs Logical. If TRUE, writes CSV/RDS outputs to output_dir. Default FALSE.

seed Optional integer seed to make data splits reproducible. If NULL, current RNG state is respected.

verbose Logical. Print progress messages. Default FALSE.

tune_repeats Integer (>=1). Repeats per combo **during tuning only**. Defaults to min(5, num_repeats).

tune_epochs Integer (>=1). Epochs per run **during tuning only**. Defaults to min(epochs, 100).

splits Optional list of pre-computed train/test partitions, each a list with integer elements train and test indexing rows of prepared_data$X. When supplied, num_repeats is set to length(splits) and the internal stratified sampler is bypassed, so several models can be evaluated on byte-identical partitions. Tuning reuses the same partitions, truncated to tune_repeats.

family Either "binomial" (default; binary outcome, sigmoid output, binary cross-entropy, equalised-odds penalty on soft true- and false-positive rates) or "gaussian" (continuous outcome, identity output, squared-error loss, outcome-stratified conditional-mean penalty). See n_strata.

n_strata Integer (>=2). Number of outcome strata used by the "gaussian" fairness penalty; strata are equal-frequency bins of y computed within each training split. Ignored when family = "binomial". With n_strata = 2 and a binary outcome the penalty is algebraically identical to the binomial penalty.

standardise Logical. If TRUE (default), columns of X are centred and scaled using the training rows of each split only, and the same constants are applied to the held-out rows. Columns with zero training variance are centred but not scaled. Set FALSE to use X as supplied.

impute Either "none" (default) or "median". With "median", missing values in X are filled with the column median of the training rows of each split, leaving the held-out rows untouched by their own medians. With "none", rows containing missing values are dropped as before.

penalty Which fairness penalty to minimise: "random_effects" (default), "bias_corrected" or "variance". "variance" is the penalty of version 0.2.0 and earlier and reproduces it exactly, including its requirement that a group be present in every stratum to contribute. The other two rungs correct that penalty for the sampling noise in the cell means and treat each stratum independently. See Details and fairgate_penalty_from_cells().

lambda_max Fairness weight of the most heavily constrained expert in the frontier-anchored ladder. Defaults to NULL, which uses the mixture fairness weight lambda of the selected configuration. Only the ladder shape depends on it; it has no effect when alpha = 0, beyond setting the scale of the exported correction intensity.

alpha Weight on the auxiliary per-expert objective. 0 (the default) reproduces the unanchored model of version 0.2.0 exactly: no auxiliary term is formed and the experts are trained only through the mixture. Positive values anchor each expert at its own point on the ladder. See Details.

early_stopping Logical. If TRUE, a stratified slice of the training rows of each partition is held out, the validation objective is evaluated at every epoch, and the parameters of

the best epoch are restored before prediction. This replaces tuning one global epoch budget: epochs becomes a ceiling rather than the budget actually used. Default FALSE, which trains for exactly epochs epochs on all training rows as before.

es_prop
: Proportion of each partition's training rows held out for early stopping, stratified jointly on outcome stratum and protected group. Default 0.2. Ignored when early_stopping = FALSE.

es_min_epochs
: Minimum number of epochs before the best epoch may be selected. Default 1. Ignored when early_stopping = FALSE.

es_patience
: Optional integer. When early stopping is enabled, stop once this many consecutive epochs have passed without the validation objective improving on its best value, and restore the best epoch. NULL (the default) never stops early, so the loop runs to epochs and the best epoch is restored, exactly as in 0.3.0. The fairGATE protocol of the accompanying study uses es_patience = 30 with epochs = 300. Ignored when early_stopping = FALSE.

base_score
: Optional list aligned with splits, one element per partition, each list(train = numeric, test = numeric): the scores of an independently fitted base learner for that partition's training rows and test rows, in the same row order as splits[[r]]$train and splits[[r]]$test, on the RESPONSE scale (probabilities for family = "binomial", the outcome scale for "gaussian"). Training-row scores should be out-of-fold (cross-fitted), so that no training row is scored by a model that saw it. Requires splits and a coupling other than "none". See Details.

coupling
: How the base score enters the model: "none" (the default, and the only value allowed when base_score is NULL) is the gated mixture of earlier versions; "offset" adds the mixture to the base score on the link scale; "expert" makes the base score one component of the mixture, weighted per individual by the gate. When base_score is supplied, coupling must be given explicitly. See Details.

gate_bias_init
: Optional numeric vector added to the gate's output bias at initialisation, one entry per gate output (K for "none" and "offset", K + 1 for "expert", entry 1 being the base component). A diagnostic device: a very large first entry with coupling = "expert" forces all routing onto the base score, which must then be reproduced. NULL (the default) leaves the initialisation untouched.

gate
: "softmax" (default, 0.4.0), "sparsemax" or "entmax15"; the sparse gates need coupling = "expert" or "residual". See fairgate_gate_map().

intervention_cost
: Weight mu of mean |eta - b|, the link-scale distance from the base learner. Default 0 (off).

lambda_thr, thr_temp, thr_target, thr_agg
: Weight, temperature, per-group target ("eo" or "aod") and aggregation ("max" or "mean") of the thresholded error-rate penalty added to the variance-component penalty. lambda_thr = 0 (default) is off. penalty = "thresholded" uses that penalty alone, with weight lambda.

reference_group
: Reference group code for the thresholded penalty; NULL compares each group with the pooled rate.

`gate_signal` Optional list aligned with `splits`, each `list(train =, test =)` of numeric vectors or matrices: cross-fitted per-person signals given to the gate only. See `fairgate_disparity_target()`.

`balance` Weight of the routing-diversity regulariser (negative mutual information between individual and expert). Default 0.

`crossed_groups` Optional two-column matrix of protected attribute codes; the variance-component penalty becomes the crossed decomposition of `fairgate_crossed_penalty()`. Objective only.

`adversary`, `adversary_hidden` Weight of the gradient reversal on an adversary predicting the protected group from the routing weights, and its hidden width. `adversary = 0` (default) is off.

`es_smooth` Window of the trailing moving average used for early stopping; `NULL` (default) or 1 is the unsmoothed rule.

`interval_reps`, `interval_epochs`, `interval_level` Warm-started bootstrap replicates, epochs per replicate and level of the per-person adjustment intervals. `interval_reps = 0` (default) is off; the point predictions are unchanged either way.

Value

A list with:

- `final_results` (tibble: subjectid, true, prob, group, iteration)
- `gate_weights` (tibble with gate probabilities & entropy per subject/iteration)
- `attribution` (tibble: routing weights, correction intensity `correction_intensity`, anchored-expert prediction `f_anchor`, and fairness adjustment `delta_fair`, per held-out individual and partition; with a base score also `base`, `base_link`, `adjustment` (`pred - base`) and `base_weight` (π_{i0}, `NA` unless `coupling = "expert"`). For the coupled models the ladder quantities `f_anchor`, `delta_fair` and `correction_intensity` are `NA`, because the ladder is not defined there.)
- `expert_weights` (list of expert input-layer weight matrices per repeat)
- `performance_summary` (tibble with one row per metric, AUC for a binary outcome or R-squared and RMSE for a continuous one, and columns `Metric`, `Value` (mean across partitions), `SD`, `Lower`, `Upper` (2.5th and 97.5th percentiles) and `Splits`)
- `performance_by_split` (tibble of each metric per partition)
- `early_stopping` (tibble of the epoch selected per partition, or a message when early stopping is disabled)
- `aif360_data` (tibble for fairness metric tooling)
- `tuning_results` (tibble or message when tuning skipped)

Coupling to a base learner

Write b_i for the base score of individual i on the link scale: $b_i = \mathrm{logit}(\mathrm{clip}(p_i, 10^{-6}, 1 - 10^{-6}))$ for a binomial outcome, and $b_i = (\hat{y}_i - \bar{y}_{tr})/s_{tr}$ for a gaussian one, where $\bar{y}_{tr}$ and s_{tr} are the mean and standard deviation of the outcome over the rows the model is fitted on, the same constants the package uses to standardise the outcome, so that the base score and the outcome share one scale. b_i is appended as an extra input column, after the features have been standardised, to the gate and to

every expert, so the gate is $\pi_i = \mathrm{softmax}(z(x_i, b_i)/\tau)$ and the experts are $f_k(x_i, b_i)$, $k = 1, \ldots, K$. The linear predictor is

$$\eta_i = b_i + \sum_{k=1}^{K} \pi_{ik} f_k(x_i, b_i)$$

for `coupling = "offset"`, with the last layer of every expert initialised at zero so that the untrained model is exactly the base learner, and

$$\eta_i = \pi_{i0} b_i + \sum_{k=1}^{K} \pi_{ik} f_k(x_i, b_i)$$

for `coupling = "expert"`, where the gate has $K + 1$ outputs and component 0 weights the base score, so π_{i0} is the share of the decision delegated to the unconstrained base learner. The task loss, the fairness penalty (on the response scale, $\mathrm{sigmoid}(\eta_i)$ or the standardised η_i), early stopping and the protected attribute's role (objective only) are exactly as for the uncoupled model. When early stopping is enabled the held-out slice keeps its own out-of-fold training base scores.

Held-out predictions are assembled in double precision as $\hat{y}_i = \tilde{b}_i + [h(\eta_i) - h(b_i)]$, with h the inverse link and $\tilde{b}_i$ the supplied base score (clipped as above for a binomial outcome). This equals $h(\eta_i)$ algebraically, and makes $\hat{y}_i = \tilde{b}_i$ hold exactly, not merely to rounding, whenever the network leaves an individual's link-scale prediction unchanged. The attribution export then reports $\tilde{b}_i$, $\hat{y}_i$, the adjustment $\hat{y}_i - \tilde{b}_i$, π_{i0} and the full routing vector.

The fairness penalty

Individual `i` has prediction $\hat{y}_i$ (a probability when `family = "binomial"`, the response scale when `"gaussian"`), protected group g_i in `1:K` and outcome stratum s_i in `1:Q`. For a binary outcome the strata are the two outcome classes; for a continuous one they are `n_strata` equal-frequency bins of `y` computed on the training rows. In cell (k, q) of size n_{kq} the mean prediction is $\hat{m}_{kq}$ and its sampling variance is $\hat{v}_{kq} = s^2_{kq}/n_{kq}$, with s^2_{kq} the within-cell sample variance. The total penalty is $\sum_q L_q$ with

- `"variance"`: $L_q = \mathrm{Var}_k(\hat{m}_{kq})$
- `"bias_corrected"`: $L_q = \max(0, \mathrm{Var}_k(\hat{m}_{kq}) - K_q^{-1} \sum_k \hat{v}_{kq})$
- `"random_effects"`: the DerSimonian and Laird between-group variance component, $L_q = \max(0, (Q_q - (K_q - 1))/C_q)$

The correction matters because $E[S^2] = S^2(\mu) + K^{-1} \sum_k v_k$: the sample variance of the cell means carries a floor set by how precisely those means are estimated, which no genuine improvement in fairness can remove. The rungs nest exactly, `"random_effects"` reducing to `"bias_corrected"` when the cell sampling variances are equal and both reducing to `"variance"` as they go to zero, so the earlier method is a special case rather than a different one. The sampling variances are always detached from the autograd graph, so the optimiser cannot lower the penalty by inflating within-cell spread.

Frontier-anchored experts

With $\hat{y}_i = \sum_k \pi_{ik} f_{ik}$ and $\pi_i = \mathrm{softmax}(z_i/\tau)$, the gradient at gate logit l is $\tau^{-1}\pi_{il}(f_{il} - \hat{y}_i)$, which vanishes when the experts agree. Identical experts are therefore a fixed point: the gate never specialises and the experts never differentiate. Each expert is instead given its own fairness weight on the ladder of `fairgate_lambda_ladder()`, $\lambda_1 = 0$ and $\lambda_k = \lambda_{\max} 10^{-(K-k)/(K-1)}$, and the objective becomes

$$L = L_{task}(\hat{y}) + \lambda L_{fair}(\hat{y}) + \alpha K^{-1} \sum_k [L_{task}(f_k) + \lambda_k L_{fair}(f_k)].$$

The auxiliary terms reuse the expert outputs already formed in the forward pass, so they cost almost nothing. With $\alpha = 0$ the objective is exactly the earlier one.

Per-individual attribution

For each held-out individual the fitted object reports the routing weights π_i, the correction intensity $c_i = \sum_k \pi_{ik}\lambda_k/\lambda_{\max}$ in $[0, 1]$, the prediction of the accuracy-anchored expert f_{i1}, and the fairness adjustment $\delta_i = f_{i1} - \hat{y}_i$, the amount by which the fairness objective moved that individual. The intensity is interpretable only when the ladder is active, that is when `alpha > 0`.

Examples

```
if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  data("adult_ready_small", package = "fairGATE")
  d <- adult_ready_small
  # drop columns with missing values or no variation before scaling
  keep <- vapply(d, function(v) !anyNA(v) && (!is.numeric(v) || stats::sd(v) > 0), TRUE)
  pd <- prepare_data(d[, keep], outcome_var = "income", group_var = "sex")
  fit <- train_gnn(pd, run_tuning = FALSE,
                   best_params = list(lr = 0.01, hidden_dim = 8, dropout_rate = 0.1,
                                      lambda = 1, temperature = 1),
                   num_repeats = 1, epochs = 5, seed = 1)
  fit$performance_summary

  # Correcting an independently fitted base learner (here a logistic
  # regression, cross-fitted by hand) with the residual coupling
  X <- pd$X; y <- pd$y
  sp <- list(list(train = 1:240, test = 241:300))
  folds <- rep_len(1:5, 240)
  oof <- numeric(240)
  for (k in 1:5) {
    m <- stats::glm.fit(cbind(1, X[sp[[1]]$train[folds != k], 1:5]),
                        y[sp[[1]]$train[folds != k]], family = stats::binomial())
   oof[folds == k] <- stats::plogis(cbind(1, X[sp[[1]]$train[folds == k], 1:5]) %*% m$coefficients)
  }
  m <- stats::glm.fit(cbind(1, X[1:240, 1:5]), y[1:240], family = stats::binomial())
  te <- as.numeric(stats::plogis(cbind(1, X[241:300, 1:5]) %*% m$coefficients))
  fit2 <- train_gnn(pd, run_tuning = FALSE,
                    best_params = list(lr = 0.01, hidden_dim = 8, dropout_rate = 0.1,
                                       lambda = 1, temperature = 1),
                    splits = sp, base_score = list(list(train = oof, test = te)),
```

```
                      coupling = "residual", epochs = 5, seed = 1)
  head(fit2$attribution[, c("base", "pred", "adjustment", "base_weight")])
}
```

Index

∗ **datasets**
adult_ready_small, 3

adult_ready_small, 3
analyse_experts, 3
analyse_gnn_results, 5

export_f360_csv, 6

fairgate, 7
fairgate(), *18*, *27*
fairgate_base_grid, 10
fairgate_base_grid(), *30*
fairgate_base_learner, 12
fairgate_base_learner(), *8, 9*, *30*
fairgate_crossed_penalty, 14
fairgate_crossed_penalty(), *37*
fairgate_crossed_plan, 15
fairgate_crossed_plan(), *14*
fairgate_disparity_target, 16
fairgate_disparity_target(), *9*, *37*
fairgate_draw_configs, 17
fairgate_draw_configs(), *9*
fairgate_export, 18
fairgate_fair_penalty, 18
fairgate_fair_penalty(), *25*
fairgate_gate_map, 20
fairgate_gate_map(), *36*
fairgate_group_disparity, 21
fairgate_group_gap_penalty, 22
fairgate_lambda_ladder, 23
fairgate_lambda_ladder(), *39*
fairgate_penalty_from_cells, 24
fairgate_penalty_from_cells(), *14*, *19*, *35*
fairgate_penalty_plan, 25
fairgate_penalty_plan(), *19*, *22*
fairgate_space, 27
fairgate_space(), *9*, *17*
fairgate_threshold_penalty, 27
fairgate_threshold_penalty(), *22*
fairgate_threshold_plan, 28
fairgate_threshold_plan(), *28*
fairgate_tune_base_learner, 29
fairgate_tune_base_learner(), *8*, *10*, *12*, *13*

plot_sankey, 31
prepare_data, 32
prepare_data(), *8*

train_gnn, 33
train_gnn(), *9, 10*, *13*, *25*