Package: vennabers (via r-universe)

November 8, 2026

Title Venn-Abers Classification and Regression Calibration

Version 1.0.0

Description Calibrates the predictions of classification and regression models with Venn-Abers predictors. A Venn-Abers predictor turns the score of any model into a pair of probabilities that contains a calibrated value when the observations are independent and identically distributed (Vovk and Petej, 2014, <doi:10.48550/arXiv.1211.0025>); Inductive and Cross Venn-Abers predictors make this practical for large data sets (Vovk, Petej and Fedorova, 2015, <doi:10.48550/arXiv.1511.00213>). The package serves analysts and researchers who need calibrated probabilities or prediction bounds for binary classification, multiclass classification by one-vs-one coupling (Manokhin, 2017, <https://proceedings.mlr.press/v60/manokhin17a.html>) or one-vs-rest, and regression (Petej and Vovk, 2026, <doi:10.48550/arXiv.2605.06646>). Predictions of an existing model can be calibrated directly, or optional 'tidymodels' workflows fit the model, create the calibration splits and tune the complete procedure. Diagnostics include the Brier score, log loss, plots of fitted calibration maps and reliability diagrams with the CORP (consistent, optimally binned, reproducible and pool-adjacent-violators based) method of Dimitriadis, Gneiting and Jordan (2021) <doi:10.1073/pnas.2016191118>.

License MIT + file LICENSE

URL https://rmkruse.github.io/vennabers/, https://github.com/RMKruse/vennabers

BugReports https://github.com/RMKruse/vennabers/issues

Encoding UTF-8

Language en-US

Depends R (>= 4.5.0)

Suggests knitr, rmarkdown, mgcv, nnet, parsnip (>= 1.6.0), workflows (>= 1.3.0), hardhat (>= 1.4.3), rsample (>= 1.3.2), recipes (>= 1.4.0), rlang (>= 1.2.0), tibble (>= 3.3.1), withr (>= 3.0.3), generics (>= 0.1.4), tidyselect (>= 1.2.1), tailor (>= 0.1.0), probably (>= 1.1.0), tune (>= 2.1.0), dials (>= 1.4.3), yardstick (>= 1.4.0), future (>= 1.67.0), rpart, spelling, testthat (>= 3.0.0)

VignetteBuilder knitr, rmarkdown

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

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

Date/Publication 2026-10-30 11:00:25 UTC

RemoteUrl https://github.com/rmkruse/vennabers

RemoteRef HEAD

RemoteSha 50a64b581ab3599234f760e999196e333005f9bb

Contents

vennabers-package . . . 3
brier_score . . . 4
calibrate_workflow . . . 5
calibrated_workflow . . . 9
calibration_plot . . . 13
corp_plot . . . 15
cross_venn_abers . . . 18
cross_venn_abers_regression . . . 22
local_calibration_splits . . . 25
model_adapter . . . 26
plot.venn_abers . . . 29
predict.cross_venn_abers . . . 32
predict.venn_abers . . . 34
print.cross_venn_abers . . . 36
print.venn_abers . . . 37
regression_plot . . . 37
venn_abers . . . 39
venn_abers_details . . . 41
venn_abers_one_hot . . . 43
venn_abers_regression . . . 44
venn_abers_schema . . . 46
venn_abers_tuning . . . 48
vennabers_examples . . . 52
workflow_postprocessing . . . 53
workflow_training . . . 58

Index **65**

vennabers-package *Venn-Abers Classification and Regression Calibration*

Description

Calibrate binary and multiclass classification probabilities or real-valued regression predictions using Inductive or Cross Venn-Abers methods. Supply predictions from an existing model, or use optional tidymodels workflows to train preprocessing, models and calibrators together.

Start with `vignette("method", package = "vennabers")`: it explains what Venn-Abers predictors compute, which guarantees the literature gives and which it does not, and checks the formulas against the package.

Calibrating supplied predictions

The manual interfaces require no additional R packages. Use `venn_abers` for Inductive classification calibration and `cross_venn_abers` for matched out-of-fold probabilities. Both support binary inputs and multiclass one-vs-one or one-vs-rest calibration. For numeric predictions, use `venn_abers_regression` or `cross_venn_abers_regression`. `venn_abers_one_hot` represents class decisions as a 0/1 matrix with one column per class.

Continue with `vignette("manual-calibration", package = "vennabers")`.

Training and calibrating workflows

Use `calibrated_workflow` with an unfitted tidymodels workflow, or `calibrate_workflow` to calibrate an already fitted workflow on new data. These interfaces require the optional packages in DESCRIPTION. `model_adapter` turns your own fit and predict functions into a workflow model without registering a parsnip engine. Follow `vignette("workflows", package = "vennabers")` for a complete example, then `vignette("tuning", package = "vennabers")` for Grid/Bayes selection and independent assessment.

workflow_training documents splits and prediction models; workflow_postprocessing covers learned postprocessors. Use `venn_abers_details` for prediction diagnostics and `venn_abers_tuning` for search configuration and results.

Example scripts

vennabers_examples indexes the installed standalone examples and their annotated articles. Each collection has a contents list for navigating scripts: `vignette("examples-manual", package = "vennabers")`, `vignette("examples-workflows", package = "vennabers")`, `vignette("examples-postprocessing", package = "vennabers")` and `vignette("examples-tuning", package = "vennabers")`. Use `system.file("examples", package = "vennabers")` to locate the scripts for inspection or execution with `source()`.

Assessment and interpretation

Score final classification probabilities with `brier_score` and `log_loss`, and assess them on held-out pairs with `corp_plot`, `corp_decomposition_plot` or `calibration_plot`; assess regression

predictions with regression_plot. plot() draws the fit diagnosis of a fitted calibrator from its stored state, without new truth values. vignette("diagnostics", package = "vennabers") compares both kinds of graphics. Keep assessment data separate from model training, selection and calibration. Manual Cross calibration requires predictions from the corresponding retained fold models.

Classification bounds are Venn-Abers probability bounds, not confidence intervals. Regression bounds concern regularized conditional expectations, not individual outcome coverage; Cross aggregate bounds are descriptive.

See Also

browseVignettes("vennabers"), help(package = "vennabers").

Cite the package and the method papers with citation("vennabers").

Project: https://github.com/RMKruse/vennabers

Bug reports: https://github.com/RMKruse/vennabers/issues

Examples

```
calibrator <- venn_abers(
  truth = c(FALSE, TRUE, FALSE, TRUE),
  probability = c(0.1, 0.4, 0.35, 0.8)
)
prediction <- predict(calibrator, c(0.2, 0.5, 0.9))
prediction
brier_score(c(FALSE, TRUE, TRUE), prediction$probability)
```

---

brier_score *Score Classification Probabilities*

Description

Computes the Brier Score or Log Loss from final binary or multiclass probabilities, regardless of the loss chosen during calibration prediction.

Usage

```
brier_score(truth, probability)

log_loss(truth, probability)
```

Arguments

truth	A non-empty logical or numeric vector containing only FALSE/0 and TRUE/1. TRUE/1 is the event class. Alternatively a non-empty factor with at least two non-empty levels and no missing values. Assessment subsets may omit observations of some levels; retain the full fitted class schema.

`probability` A numeric vector of regularized event probabilities, the same length as `truth`, with finite values in `[0, 1]`. For factor truth, a numeric class matrix following the class-name, value, and row-sum contract of `venn_abers`.

Value

One unnamed double. `brier_score()` returns the binary mean squared error. `log_loss()` returns the mean negative natural log-likelihood; an incorrect probability of zero or one produces `Inf`. For multiclass, Brier is the mean of the row sums of squared errors against one-hot truth, without division by two or class count. A two-column Brier score is twice the binary vector score for the same probabilities. Multiclass log loss is the mean negative natural logarithm of the observed class probability; zero observed probability gives `Inf`. Neither score clips probabilities.

Examples

```
truth <- c(FALSE, TRUE, FALSE, TRUE)
uncalibrated_probability <- c(0.1, 0.4, 0.35, 0.8)
calibrator <- venn_abers(truth, uncalibrated_probability)
prediction <- predict(calibrator, uncalibrated_probability)

brier_score(truth, prediction$probability)
log_loss(truth, prediction$probability)
```

`calibrate_workflow` *Calibrate a Frozen Workflow*

Description

Adds an Inductive Venn-Abers calibrator to an already fitted classification or regression workflow and fits only that calibrator, on the workflow's final predictions; the complete workflow, including learned preprocessing and any learned postprocessor, is retained unchanged. It needs the fitted workflow and new calibration data that played no part in its training, tuning or selection. Choose it when the workflow is final or expensive to refit; use `calibrated_workflow` when the package should train the workflow and create the calibration split itself.

Usage

```
calibrate_workflow(fitted_workflow, data_calibration, outcome, ...,
  classification = "binary", event_level = "first", precision = NULL,
  loss = "log", m = NULL, epsilon = NULL, schema = NULL)
## S3 method for class 'vennabers_frozen'
predict(object, new_data, type = NULL, ...,
  loss = NULL)
## S3 method for class 'vennabers_frozen'
print(x, ...)
```

Arguments

`fitted_workflow`	A fully fitted classification or regression workflow. Single parsnip fits, extracted engines and unfitted workflows are rejected.
`data_calibration`	A nonempty data frame in the original learned input format, including the outcome. Do not pre-bake the data. Missing predictor values are allowed only if the stored prediction path handles them.
`outcome`	One column name matching the trained outcome. Calibration truth must be a nonmissing factor with exactly the trained class universe. Factor levels may be reordered; the stored workflow fixes the output class order. Binary calibration may observe only one class. OvO/OvR must observe every stored class globally. Regression truth must be a finite numeric vector, including when all values are 0/1; the workflow fixes the mode.
`classification`	Exactly `"binary"` (default), `"ovo"` or `"ovr"`. Choose OvO/OvR explicitly or through the training schema, including for a two-class matrix problem.
`event_level`	Exactly `"first"` or `"second"`, selecting the event by the trained class order. Only accepted explicitly for binary calibration.
`precision`	NULL or a finite numeric whole number. Only calibration probabilities are rounded, following `venn_abers`. New probabilities are never rounded. Changing this option or the event requires a new calibrator, but no new base workflow fit.
`loss`	Constructor: exactly `"log"` or `"brier"`, saved as the prediction default. Predict and `venn_abers_details()`: omit to retain that default, or override without refitting or changing stored state. Explicit NULL is invalid at prediction. Only classification accepts loss or precision. Regression rejects explicit classification, event_level, precision and loss options, including explicit defaults or NULL.
`m`, `epsilon`	Regression moderation. With both NULL, use m = 1. Otherwise supply exactly one non-NULL value: a whole number with 1 <= m <= floor((k-1)/2), or 0 < epsilon < 1 giving m = floor(epsilon*(k+1)/2) within that same range, where k is the calibration size. At least three observations are needed. Requested values and effective m and epsilon = 2*m/(k+1) are retained. Classification rejects these options.
`schema`	NULL for an originally named workflow, or the fixed `venn_abers_schema` used to encode the original training data. With a schema, explicitly encode calibration and prediction data using `venn_abers_encode(data, schema)` and use `schema$encoded_outcome`. Omitted classification and binary event options inherit the schema; explicit options must agree. The workflow mode, input columns, class order and ordered factor status must match. A schema does not rename an existing workflow.
`object`, `x`	A fitted predictor returned by `calibrate_workflow()`.
`new_data`	A data frame in the learned input format. The outcome is not needed and, if present, is removed before the original workflow predicts.
`type`	NULL selects `"prob"` for classification and `"numeric"` for regression. Classification also accepts `"class"`. Regression returns only numeric points; `"conf_int"` and `"pred_int"` are unsupported.

... Must be empty. Constructor argument names must be exact. Unknown, training, split, Cross and full-model options are rejected.

Details

Calibration uses the complete public workflow `predict(type = "prob")` or `predict(type = "numeric")` path, including its existing postprocessor exactly once. Missing, extra, duplicate or unknown class columns, nonfinite/out-of-range probabilities, invalid row sums and changed row counts cause errors. Multiclass matrices are normalized only within the existing 1e-8 row-sum tolerance, by the manual core. OvO zero pair sums, coupling, OvR normalization and calibration-only rounding use those same cores. Regression requires finite final numeric values. Failures retain original class/input names and prediction-stage context. No learned state is renamed, refitted or bypassed through an extracted engine or a raw fallback.

The user is responsible for keeping the new calibration data outside all earlier model, preprocessing, postprocessing and parameter-selection data. Assessment data must also be separate from calibration and selection. This history is not automatically verified. Frozen adoption is an R extension, neither Cross calibration nor full-model transfer. `calibrated_workflow` is the separate path that trains new base workflows on its own split.

Native `fit_resamples()` and `last_fit()` evaluate the base workflow itself. Their fitted workflows, from `tune::extract_workflow()` or saved extracts, can be adopted with calibration rows outside both their analysis and assessment data; the adoption then learns only the calibrator. The returned predictor is not itself a `tune` input.

To choose between the Log and Brier rules after adoption, compare `predict(fitted, validation, type = "prob", loss = ...)` with yardstick metrics on separate, previously unused validation rows, then pass the chosen `loss` explicitly when predicting. Neither the workflow nor the calibrator is refitted, and no further helper is needed. Validation rows remain selection data; final assessment uses other rows. Automatic search with new training belongs to `calibrated_workflow()` tuning.

`saveRDS()` and `readRDS()` preserve prediction and diagnostics without learning for the tested GLM, multinomial and linear workflows, including recipes and Tailor, in fresh R processes and parallel workers. Required packages must be available in the new R process. Engines with external resources retain their own storage contracts; no universal portability is promised. The integration dependencies are optional; manual cores work without them. The installed `examples/frozen-binary.R` demonstrates earlier model/recipe and postprocessor learning, separate calibration, the loss-only choice on validation rows and independent assessment. `examples/frozen-multiclass-regression.R` demonstrates learned multinomial and linear postprocessing with OvO/OvR and IVAR. The original workflow must support the requested batch size: for example, the tested probably 1.2.0 multinomial smooth=FALSE path fails on one row; smooth=TRUE supports it. Original prediction failures propagate without modifying the workflow.

For encoded training, retain the original schema together with the fitted workflow. No schema is learned from later calibration data. Inputs must carry the encoded names; valid reordered input columns and factor levels are matched by identity. Predictions and details carry the schema binding and remain encoded. `venn_abers_decode()` returns the existing roles with original names and classes, including diagnostics; it invents neither outer identities nor Cross aggregates for these Inductive methods. See the installed `examples/frozen-schema.R` for all four paths with learned postprocessing.

Bindings retained on the workflow, its original recipe template, or incoming data are checked, along with learned metadata. Workflows and table operations can discard attributes: matching detached

codes alone cannot prove which schema was used during training. The caller must supply the actually associated schema. An originally named workflow must continue receiving its original inputs with `schema = NULL`. Creating a schema afterward does not authorize recipe rewriting, refitting or encoded input substitution. An old workflow whose complete prediction path fails on reserved names reports the stage, names and underlying cause. The limited binary GLM raw-response probe without a postprocessor is not a product fallback; no engine or postprocessor is bypassed.

Value

An already fitted `vennabers_frozen` predictor, also inheriting `vennabers_fit`. It is not a trainable workflow or a specification for `fit_resamples()`. The object retains complete workflows in `models`, the `calibrator`, `schema`, `class_levels`, `event_class`, input column metadata, calibration size, requested/effective `parameters`, and relevant R, package and engine `versions`. No additional complete copy of new raw calibration data is stored; data already retained by the original workflow remain. Prediction returns a tibble with named `.pred_<class>` probabilities in trained class order, or a factor `.pred_class`, ordered exactly when the trained outcome is ordered. The event wins at probability 0.5 in binary calibration. OvO/OvR ties choose the first stored level. Regression returns one double `.pred` column. `venn_abers_details()` returns the same prediction table and binary lower/upper bounds and points with row, Inductive fold and event identity; OvO/OvR add class-pair or class-against-rest identities. Regression details contain row, fold, prediction, lower and upper. IVAR bounds concern regularized conditional expectations, not confidence or prediction intervals. Zero and one row preserve types and columns; zero rows do not call the base engine. Print reports the adoption without base fits, the one retained workflow, the number of calibration rows and the Inductive fold-to-model mapping. `plot.vennabers_fit` draws the stored calibration map.

See Also

`vignette("workflows", package = "vennabers")` for frozen workflows in context, `vignette("method", package = "vennabers")` for the theory and its guarantees, and vennabers_examples for annotated installed scripts.

`calibrated_workflow`, `venn_abers_details`, `venn_abers_schema`, `venn_abers`, `venn_abers_regression`

Examples

```
if (all(vapply(c("parsnip", "workflows", "hardhat", "rsample", "recipes",
                 "rlang", "tibble", "withr", "generics", "tidyselect"),
               requireNamespace, logical(1), quietly = TRUE))) {
  d <- data.frame(x = sin(1:80), truth = factor(rep(c("yes", "no", "no", "yes"), 20)))
  base <- workflows::workflow() |>
    workflows::add_formula(truth ~ x) |>
    workflows::add_model(parsnip::logistic_reg(engine = "glm"))
  earlier <- generics::fit(base, d[1:40, ])
  calibrated <- calibrate_workflow(earlier, d[41:60, ], "truth")
  predict(calibrated, d[61:80, "x", drop = FALSE])
}
```

`calibrated_workflow` *Train a Calibrated Classification or Regression Workflow*

Description

Wraps an unfitted tidymodels workflow for classification or regression so that `generics::fit()` trains preprocessing, models and a Venn-Abers calibrator on separate local splits of the same data. It needs the workflow and the outcome column; the fitted workflow is then used with `predict()` on raw predictors. Choose it when the package should fit the model and keep the calibration rows out of training; use `calibrate_workflow` for a workflow that is already fitted, or `venn_abers` and `venn_abers_regression` for predictions computed elsewhere.

Usage

```
calibrated_workflow(workflow, outcome, ..., predictors = NULL,
  method = "inductive", classification = "binary", event_level = "first",
  calibration_prop = 0.2, training_size = NULL, calibration_size = NULL,
  v = 5L, strata = NULL, splits = NULL, seed = NULL,
  precision = NULL, loss = "log", m = NULL, epsilon = NULL, schema = NULL,
  prediction_model = "matched", model_training = "shared", tuning = NULL,
  postprocessing = NULL)
## S3 method for class '`_vennabers_fit`'
predict(object, new_data, type = NULL,
  opts = list(), ..., loss = NULL)
```

Arguments

`workflow`	An unfitted workflows object with a classification or regression model and recipe, formula, or variable selection. Prepped or partly trained recipes, unresolved tune placeholders without `tuning`, and postprocessors without an explicit matching `postprocessing` request are rejected.
`outcome`	One original outcome column name, or `schema$encoded_outcome` when a schema is supplied. Fit requires a nonmissing factor with exactly two levels for binary or at least two for OvO/OvR. All stored levels must be represented in the development data. Regression requires finite numeric truth; numeric 0/1 outcomes remain regression.
`...`	Must be empty. Unknown, irrelevant and partially matched names are errors.
`predictors`	Optional character vector of raw predictor names, excluding outcome. NULL selects all other supplied columns. Their stored order is canonical; unused selected columns still have to be supplied for prediction.
`method`	Exactly `"inductive"` (default) or `"cross"`.
`classification`	Exactly `"binary"` (default), `"ovo"`, or `"ovr"`. OvO/OvR explicitly use the factor/matrix core, including with two classes. Mode comes from the base model. Regression rejects explicit classification, event_level, precision and loss options, including their defaults; implicit classification defaults do not apply.

event_level
: Exactly "first" or "second", identifying the original factor level whose probability is calibrated. Only for binary; explicitly supplying this option for OvO/OvR is an error.

calibration_prop
: A number strictly between zero and one. rsample::initial_split() uses training proportion 1-calibration_prop. Explicitly supplying this option requires Inductive.

training_size, calibration_size
: Independent Inductive training and calibration sizes, like sklearn's train_size/test_size. Both NULL (default) keep calibration_prop and its draw unchanged. A value strictly between zero and one is a proportion of the local rows; a whole number of at least one is a row count. Proportions round training down and calibration up; row counts are exact. A missing size is the complement. Rows left over when both are supplied form the unassigned set U: the matched source model, its B/P postprocessing and the calibrator never use them, while prediction_model = "full" trains its additional workflow on all local rows, including calibration rows and U. Classification draws are stratified by outcome with exact totals (largest-remainder class allocation, ties in level order) unless strata = FALSE; regression is unstratified. Sizes are checked against the local rows and never shrunk: more rows than available, or an empty training or calibration set, is an error naming both sizes. They cannot be tuned and cannot be combined with method = "cross", explicit calibration_prop or splits; the default calibration_prop is then ignored. With model_training = "pairwise" the sizes resolve on each class pair's local rows, with U per pair; after calibration selection they resolve on the final-fit F. Calibration tuning and joint model tuning fit Venn-Abers candidates that resolve the same sizes on their resample rows, so they accept only proportions; two-stage model tuning also accepts row counts. See 'inductive-sizes.R' in vennabers_examples.

v
: Cross fold count, a whole number of at least two, default five. Automatic rsample::vfold_cv() uses one repetition and never reduces this count; it must not exceed the local row count. Explicitly supplying v requires Cross.

strata
: NULL stratifies classification by outcome with pool=0; regression is unstratified. FALSE disables stratification. Other values are rejected. No retry or automatic split adjustment.

splits
: NULL generates a local split. Otherwise a mixed or ordered holdout rsplit from rsample::initial_split() or rsample::make_splits(), for Inductive; a simple rset from rsample::vfold_cv() or rsample::manual_rset() for Cross; or a serializable function receiving only selected local raw predictors followed by outcome and returning such a plan. Explicit plans cannot be combined with explicit calibration_prop, v or strata, even their defaults. Use local_calibration_splits for prepared assignments. Pairwise accepts a function of the current local pair data, or for direct fits a list named by exactly pair_1, pair_2, etc. A single global multiclass plan is not a pair plan. Automatic holdouts/partitions are generated after filtering. Explicit Cross plans may have different fold counts per pair.

seed
: NULL uses the caller's RNG stream. A finite whole number in R's integer seed range sets a local seed for splitting and base fitting, restoring the caller's state

and RNG kind afterwards, including on error. The scope includes the plan callback and model learning. Seed the outer plan separately. Reproducibility also depends on RNG kind, engine, package versions, thread and parallel configuration.

`precision` NULL or a finite numeric whole number, including negative numbers. Only calibration probabilities are rounded, following `venn_abers()`.

`loss` Constructor: stored prediction default, exactly `"log"` or `"brier"`. Prediction/details: omit to use that default; an explicit choice overrides it without refitting or changing stored state. Explicit NULL is invalid. Regression rejects explicit loss at construction and prediction/details.

`m`, `epsilon` Regression moderation: both NULL means effective `m=1`. Only one non-NULL option may be supplied and only that option reaches the core. `m` is a positive numeric whole number; `epsilon` is strictly between zero and one. Before any inner training, each fold of calibration size k must satisfy `1 <= m <= floor((k-1)/2)`, with `m = floor(epsilon*(k+1)/2)` for an epsilon request. Invalid folds fail the entire fit with fold/size context; there is no clipping or fold-count adjustment. Classification rejects explicit m/epsilon, including NULL.

`schema` NULL retains the direct naming contract. A schema from `venn_abers_schema()` fixes the encoded input names, mode, class order and binary event. Omitted classification/event options inherit the schema; contradictory explicit options fail before learning. Build the base formula/recipe and resamples on `venn_abers_encode()` data. All schema predictors must be selected.

`prediction_model`
Exactly `"matched"` (default) or `"full"`. Available for binary, shared OvO/OvR, pairwise OvO classification and regression. This is a fixed fit option; predict cannot change it. Matched uses each calibration source model for new observations. Full applies the unchanged held-out calibrators to predictions from one additional workflow trained on the entire permitted local fit domain, including calibration rows. For pairwise training there is one such full workflow per local class pair. With postprocessing, full means an additional complete B/P holdout or ensemble on that domain, with its own inner splits; each engine still learns only on B. Matching-model calibration guarantees do not automatically transfer to full models; see workflow_training.

`model_training` Exactly `"shared"` (default) or `"pairwise"`. Explicit values are rejected for binary and regression workflows. A shared model produces one complete class matrix; OvO splits calibration into pairs without training pair models. Pairwise requires `classification="ovo"`, and a workflow that can train on a two-level factor. Each pair has its own models, recipes and holdout or Cross partition. The later global level is its event; no extra shared/OvR model is fitted.

`tuning` NULL keeps fixed settings. Otherwise, a named list with `strategy = "joint"` or `"two_stage"`, and `model` and/or `calibration` Grid or Bayes searches matching the `tune()` placeholders. See `venn_abers_tuning` for the configuration, selection splits, pairwise search rules and diagnostics; start with `vignette("tuning", package = "vennabers")` for examples.

`postprocessing` NULL leaves the unprocessed training path unchanged; it does not remove a supplied postprocessor. Otherwise, a named list selecting `method` (`"sigmoid"`,

"tailor_logistic", "tailor_multinomial", "tailor_linear" or "isotonic_numeric") and strategy ("holdout" or "ensemble"). Holdouts need prop or local splits; ensembles need v or local splits. Sigmoid also requires an explicit sigmoid_input. See workflow_postprocessing for compatible methods, data roles and the complete configuration.

object — A fitted calibrated workflow. The predict method is normally dispatched by predict() on that workflow.

new_data — A data frame with required raw predictors. Normal workflow blueprint and hardhat::forge() checks apply equally to prediction and details. Novel factor levels reach the inner workflow, whose recipe/engine handles them or errors.

type — Classification: "prob" or "class", default class. Regression: "numeric", also the default, returns double .pred in outcome units. Neither "conf_int" nor "pred_int" is registered.

opts — The standard parsnip prediction options list; normally empty.

Value

The constructor returns an unfitted ordinary workflow. After generics::fit(), native probability predictions are tibbles of double .pred_<original class> columns in original level order. Class predictions contain .pred_class, a factor retaining all levels. Binary favors its event at exactly 0.5; OvO/OvR select the first maximum in stored level order from the final probabilities, even with two classes. Empty predictions keep the schema without calling the inner engine. venn_abers_one_hot represents these class decisions as a one-hot matrix.

Regression predictions contain a double .pred column in outcome units. Use venn_abers_details for bounds, fold identities and source models, and venn_abers_tuning for selection diagnostics. The fitted engine, workflows::extract_fit_engine(fitted), is a vennabers_fit: printing it shows source mappings and base-fit counts, and plot.vennabers_fit draws its stored calibration map.

Learning path

Start with the binary example below, then follow vignette("workflows", package = "vennabers") through learned preprocessing, Cross calibration, multiclass, regression, frozen workflows and post-processing. vignette("tuning", package = "vennabers") continues with parameter selection, outer resampling and final assessment.

For individual rules, use these reference topics:

Training and evaluation workflow_training: dependencies, split validation, stored state, pairwise models, full-model transfer and resampling.

Prediction diagnostics venn_abers_details: returned columns, bounds and fold/source identities.

Tuning venn_abers_tuning: search configuration, selection data, local searches and diagnostics.

Postprocessing workflow_postprocessing: supported methods, B/P/C data roles, holdouts, ensembles and full compositions.

For a model that is already fitted, use calibrate_workflow.

Data roles and limitations

The split isolates new base learning from calibration observations. An unfitted workflow does not prove that earlier tuning, feature choices, or external transformations excluded these observations. Keep assessment observations separate from all training, selection and calibration. Calibration is not an independent evaluation. Cross model parameters must be fixed using independent data or selected through the configured local `tuning` searches inside the allowed training portions. The statistical setting assumes IID observations and a predictive procedure fixed independently of its calibration data. The bounds are Venn-Abers probability intervals, not confidence intervals. Regression bounds concern regularized conditional expectations, not individual outcome coverage; Cross aggregate bounds are descriptive. Improved scores on every data set are not promised. The engine cannot validate an outer partition it never receives. Validate outer analysis/assessment disjointness before calling fit_resamples; assessment sets from different outer resamples may overlap.

See Also

For a step-by-step introduction, start with `vignette("workflows", package = "vennabers")`; `vignette("method", package = "vennabers")` explains the theory and its guarantees. vennabers_examples lists annotated installed scripts.

venn_abers, cross_venn_abers, venn_abers_schema, calibrate_workflow, model_adapter.

Examples

```
packages <- c("parsnip", "workflows", "recipes", "rsample", "hardhat",
              "generics", "rlang", "tibble", "tidyselect", "withr")
if (all(vapply(packages, requireNamespace, logical(1), quietly = TRUE))) {
  # Development rows are available for training and calibration.
  # Reserve separate assessment data before choosing this procedure.
  development <- data.frame(
    x = sin(1:80),
    truth = factor(rep(c("yes", "no", "no", "yes"), 20),
                   levels = c("yes", "no"))
  )
  base <- workflows::workflow() |>
    workflows::add_formula(truth ~ x) |>
    workflows::add_model(parsnip::logistic_reg(engine = "glm"))

  specification <- calibrated_workflow(base, outcome = "truth", seed = 42)
  fitted <- generics::fit(specification, development)
  new_data <- data.frame(x = c(-0.5, 0.5))
  predict(fitted, new_data, type = "prob")
  venn_abers_details(fitted, new_data)$details
}
```

---

calibration_plot *Plot Classification Probability Calibration*

Description

Draws a dependency-free reliability diagram of mean regularized probability against observed event rate. Multiclass draws one panel per class in a rectangular layout and restores the previous graphical parameters on exit. Graphical options apply to every panel; by default its title identifies the class. Bins are equal-width, left-closed and right-open, except that the last bin includes one.

Usage

```
calibration_plot(truth, probability, bins = 10, ...)
```

Arguments

`truth`	A non-empty logical or numeric vector containing only `FALSE`/`0` and `TRUE`/`1`. `TRUE`/`1` is the event class. Alternatively, factor truth under the scoring contract in `brier_score`; assessment may omit observations of some levels.
`probability`	A numeric vector of regularized event probabilities, the same length as `truth`, with finite values in `[0, 1]`. For factor truth, the final class matrix under the input contract of `venn_abers`, never the internal pair probabilities.
`bins`	One finite whole number of equal-width bins.
`...`	Base R graphical options passed to `plot()`.

Value

Invisibly, a data frame with columns `mean_probability`, `event_rate`, and `count` for each non-empty bin. Multiclass adds a leading factor column `class` with the truth levels, grouping rows by class order and then increasing bin order. Each class is evaluated against the rest using its final probability. Empty bins are omitted.

See Also

`corp_plot` for CORP reliability diagrams without bins and with uncertainty bands, `regression_plot` for held-out regression predictions, and `plot.venn_abers` for fit diagnoses of the stored calibration map. `vignette("diagnostics", package = "vennabers")` compares both kinds of graphics.

Examples

```
truth <- c(FALSE, TRUE, FALSE, TRUE)
uncalibrated_probability <- c(0.1, 0.4, 0.35, 0.8)
calibrator <- venn_abers(truth, uncalibrated_probability)
prediction <- predict(calibrator, uncalibrated_probability)

calibration_plot(truth, prediction$probability)
```

corp_plot *Assess Calibration with CORP Reliability Diagrams*

Description

Assesses the calibration of probabilities on evaluation pairs of truth and probability, held out from fitting the calibrator, such as an assessment set or a test set. Unlike plot.venn_abers, which shows the stored state of a fit, these graphics need new truth values.

corp_plot() draws the CORP reliability diagram of Dimitriadis, Gneiting and Jordan (2021): the CORP estimate of the conditional event probability as a step over the probabilities, from the smallest to the largest, against the dashed diagonal of perfect calibration on [0, 1]. corp_decomposition_plot() draws the CORP decomposition of the Brier score as bars of miscalibration (MCB), discrimination (DSC), and uncertainty (UNC).

Usage

```
corp_plot(truth, probability, band = "consistency", level = 0.9,
          replicates = 1000, seed = NULL, ...)

corp_decomposition_plot(truth, probability, ...)
```

Arguments

truth	A non-empty logical or numeric vector containing only FALSE/0 and TRUE/1. TRUE/1 is the event class. Alternatively factor truth under the scoring contract in brier_score; assessment may omit observations of some levels.
probability	A numeric vector of regularized event probabilities, the same length as truth, with finite values in [0, 1]. For factor truth, the final class matrix under the input contract of venn_abers, never the internal pair probabilities.
band	Exactly one of "consistency" (default), "confidence", or "none": the uncertainty band drawn with the CORP estimate: the pointwise consistency band around the diagonal, the approximate pointwise confidence band around the estimate, or none. See the section on uncertainty bands.
level	One number strictly between 0 and 1: the pointwise level of the band.
replicates	One finite whole number of at least 1: the resampling replicates of the band. replicates * (1 - level) / 2 must be at least 1, so that each band quantile has a replicate beyond it.
seed	NULL or one integer in R's seed range for the band replicates. With a seed, the band is reproducible and the caller's random state is left unchanged; without one, the band draws from and advances the global random stream.
...	Base R graphical options passed to plot() in corp_plot() and to barplot() in corp_decomposition_plot(), taking precedence over the defaults in every panel.

Details

The CORP estimate is the isotonic (PAV) regression of the events on the probabilities, computed with `isoreg`. Equal probabilities are valid and pooled, so they share one estimate, and the estimate never decreases. Unlike the bins of `calibration_plot`, it needs no tuning.

With the PAV estimate `q` and the event rate `r` on the same pairs, `mcb` is the Brier score of the probabilities minus that of `q`, `dsc` the Brier score of `r` minus that of `q`, and `unc` the Brier score of `r`, so that `brier = mcb - dsc + unc` up to rounding. Lower `mcb` and higher `dsc` are better; `unc` depends on the truth only. The decomposition is deterministic.

Pairs are assumed independent. Both graphics describe the evaluation pairs given, conditional on the fitted calibrator; they make no statement about Venn-Abers validity.

Value

`corp_plot()` returns invisibly a data frame with one row per unique probability in increasing order and the same columns whatever `band` holds: `probability`, `estimate` (the CORP estimate of the conditional event probability), `count` (the pairs with this probability), `band_lower` and `band_upper` (the band limits), `valid_replicates` (the replicates defined at this probability, which the limits rest on), and the controls `band`, `level`, `replicates`, and `seed`, the same in every row. Without a band, `band_lower`, `band_upper`, and `valid_replicates` are `NA`, as they are when the confidence band is omitted; without a seed, `seed` is `NA`. The band limits are named apart from the Venn-Abers values `lower` and `upper`, which they are not.

`corp_decomposition_plot()` returns invisibly a data frame with one row and the columns `brier`, `mcb`, `dsc`, and `unc`.

For factor truth, both add a leading factor column `class` with the truth levels, grouping rows by class order.

Uncertainty bands

Both bands are computed by resampling as in the **reliabilitydiag** package (version 0.2.1). Each of the `replicates` draws the pairs with replacement, draws new events from the drawn probabilities themselves (consistency band) or from the CORP estimate at them (confidence band), fits PAV again with ties pooled, and interpolates that fit linearly at the unique probabilities, leaving it missing outside the drawn range. The band limits at each probability are the `(1 - level) / 2` and `(1 + level) / 2` quantiles of its valid replicates (`quantile`, type 7), so they lie in `[0, 1]` and equal probabilities share them. The band is drawn as a translucent gray area with a legend naming it; the limits are linearly interpolated between the unique probabilities.

The consistency band shows where the CORP estimate of a perfectly calibrated forecast with these probabilities would fall; an estimate outside it points to miscalibration there. It is a reference region, not a test of Venn-Abers validity. The confidence band resamples from the estimate itself and is only approximate: the bootstrap of an isotonic fit is not consistent for continuous probabilities, so the band claims no nominal coverage. It fares better with few distinct probabilities and many pairs at each, as Venn-Abers calibrators often produce. In a simulation study at level 0.9 with 1000 replicates, 1000 runs, and true conditional event probability `x^2`, its mean pointwise coverage at probabilities between 0.05 and 0.95 was 0.92 for 100 and 0.88 for 1000 uniform probabilities, and 0.95 and 0.91 for 100 and 1000 pairs on ten distinct probabilities. It varied along the curve, down to 0.34 just above 0.05 for 1000 uniform probabilities. The consistency band held the CORP estimate of calibrated probabilities with mean coverage between 0.90 and 0.93 in the same settings. At inner

probabilities with an estimate of exactly 0 or 1, the confidence limits are interpolated linearly from the neighboring probabilities, a heuristic of the reference; at the smallest and largest probabilities the band stays narrow there. When the pairs (for factor truth, of a class) hold only one outcome, the confidence band is omitted with a warning and its limits are NA; the consistency band depends on the probabilities only and is always computed.

Both bands are pointwise at `level` for each probability, not simultaneous over the curve, and for factor truth they are computed class by class without joint coverage across classes. They assume independent pairs; dependent data such as time series or clusters need other methods. Small assessment sets give wide bands that say little. The band limits are not Venn-Abers lower and upper probabilities.

The cost grows with `replicates` times the number of pairs, each replicate sorting and fitting them again. The exact quantiles hold the replicate values at every unique probability in memory at once, a matrix of unique probabilities by replicates. With the default 1000 replicates and 100000 continuous probabilities, that matrix takes about 800 MB and the call about 2 GB at its peak, and one call takes minutes, the confidence band several times longer than the consistency band. Few distinct probabilities need far less. For large assessment sets, `band = "none"` skips the resampling.

Multiclass

Every factor, even with two levels, is evaluated class by class: each class against the rest, using its final probability. `corp_plot()` draws one panel per class, titled `"CORP: <class>"`, in its own nearly square grid as in `calibration_plot`, and restores the previous graphical parameters on exit, also after errors. Its bands draw the replicates of the classes in turn from one random stream, set once by `seed`. `corp_decomposition_plot()` draws one panel of the components grouped side by side, one bar and legend entry per class, without a combined value.

Each `brier` is the binary Brier score of its class, `mean((y_k - p_k)^2)`. Their sum over classes is the multiclass value of `brier_score`, which they do not replace.

References

Dimitriadis, T., Gneiting, T. and Jordan, A. I. (2021). Stable reliability diagrams for probabilistic classifiers. *Proceedings of the National Academy of Sciences*, 118(8), e2016191118. doi:10.1073/pnas.2016191118

See Also

`calibration_plot` for binned reliability diagrams, `brier_score` for scores, `regression_plot` for held-out regression predictions, and `plot.venn_abers` for fit diagnoses. `vignette("diagnostics", package = "vennabers")` compares both kinds of graphics.

Examples

```
set.seed(1)
score <- runif(400)
truth <- rbinom(400, 1, score^2)
fitting <- 1:200
calibrator <- venn_abers(truth[fitting], score[fitting])
assessment <- predict(calibrator, score[-fitting])$probability
```

```
corp_plot(truth[-fitting], assessment, seed = 1)
corp_plot(truth[-fitting], assessment, band = "confidence", replicates = 199, seed = 1)
corp_decomposition_plot(truth[-fitting], assessment)
```

`cross_venn_abers` *Fit a Binary or Multiclass Cross Venn-Abers Calibrator*

Description

Calibrates binary or multiclass classifier probabilities with a Cross Venn-Abers predictor (CVAP): every fold calibrates the model that was trained without it, and prediction averages the fold results. It needs the true classes, the out-of-fold probabilities of one retained model per fold and the fold labels; it fits no models itself. Choose it over `venn_abers` when a single held-out calibration set would cost too much training data, or use `calibrated_workflow` with `method = "cross"` to let the package fit the fold models.

Usage

```
cross_venn_abers(truth, probability, fold, precision = NULL, method = "ovo")
```

Arguments

`truth`	A non-empty logical or numeric/integer vector containing only `FALSE/0` and `TRUE/1`. The event class is always `TRUE/1`. Alternatively, a factor with at least two levels for multiclass calibration; every level must be globally observed.
`probability`	A numeric/integer vector of out-of-fold event probabilities, finite and in `[0, 1]`, aligned with `truth`. For factor truth, a numeric matrix with one row per observation and complete, unique class column names matching its levels; see `venn_abers`.
`fold`	An aligned vector of non-empty, non-missing character labels or finite positive whole-number numeric/integer IDs. Character matching is literal, without trimming or case folding. Numeric IDs match their Base R `as.character` representations. Repeated IDs indicate membership in the same fold. Factors and logical IDs are not supported.
`precision`	`NULL` (default) disables rounding. Otherwise, one finite numeric whole number, including zero or negative values, used for native R rounding of calibration probabilities in every fold. See `venn_abers` for validation and rounding semantics.
`method`	For factor truth, exactly `"ovo"` (default) or `"ovr"`. Explicit use with binary vector truth is an error.

Details

The function fits Inductive Venn-Abers calibrators per calibration fold and, for multiclass, per class pair or class against the rest. Prediction combines their results with equal-weight log-loss (default) or Brier aggregation to produce a regularized event probability.

For binary calibration, all three inputs must be vectors of equal, nonzero length with no missing values. Names attached to vector elements are not alignment keys: positions must refer to the same observation. Invalid types, shapes, nonbinary truth, nonfinite values, or out-of-range probabilities cause an error for the entire call. No rows are dropped and probabilities are not clamped or coerced from other representations. Original probabilities are checked before optional rounding; exact ties after rounding are combined. New prediction values are never rounded. Changing precision requires refitting, while loss is chosen at prediction time. R and NumPy rounding can differ at floating-point boundaries.

Supply one non-overlapping partition with at least two represented non-empty folds. Unequal sizes, singleton folds, and single-class folds are supported; each fold receives equal aggregation weight regardless of its size. There is no separate empty-fold declaration. Use `venn_abers()` for one calibration set.

Each fold model must exclude its calibration fold from the entire learned training pipeline, including preprocessing and tuning. That same model must produce both its fold's out-of-fold probabilities and its corresponding named column of new-observation probabilities. Retain those models for subsequent prediction: a single model refitted on all data is not a replacement. Keep evaluation truth outside model fitting and calibration.

Callers ensure each underlying calibration observation appears once and that new-observation rows align across model columns. The supplied arrays cannot prove model provenance, training exclusion, absence of leakage, unique underlying observations, or semantic row alignment.

Cross output defaults to a regularized probability. Classes and individual fold results are available via `predict.cross_venn_abers`. There is no aggregate interval. Geometric aggregate endpoints can cross. Neither those endpoints nor the combined point inherit the IVAP perfect-calibration guarantee. They are not confidence intervals, and score improvement is not guaranteed on every data set.

Value

An S3 object of class `"cross_venn_abers"`: a list retaining fold identities and calibration state for repeated predictions from the matching fold models. For binary truth it records the fold IDs in first-observed order, the total calibration size and one Inductive `venn_abers` calibrator per fold; for factor truth, the class levels, the method, the fold IDs and one Cross calibrator per class pair (OvO) or class (OvR). These list elements are internal and not public API. Use the methods instead:

- `predict.cross_venn_abers` combines the fold calibrators on new probabilities from each fold model into regularized probabilities, class decisions or per-fold details, with the loss chosen at prediction time.
- `plot.cross_venn_abers` draws each fold's stored calibration map and returns the maps invisibly as a data frame.
- `print.cross_venn_abers` summarizes the number of folds and calibration observations, and the classes for multiclass.

Predictive models are retained separately by the caller.

Unsupported inputs and features

Package-managed predictive-model fitting, fold generation, preprocessing, tuning or resampling; repeated or overlapping partitions; regression calibration; separate pairwise model training or pair-specific fold plans; observation weights; alternate event-class selection; automatic coercion or

missing-value handling; fold-size weighting or pooling; public aggregate endpoints; formula or data-frame cross interfaces; tidymodels/yardstick integrations; and new metrics, smoothing or plotting features are not supported. Only Base R and standard R distribution packages are needed. Installation and acceptance require neither Python nor networking; Python reference-fixture regeneration is optional and separate.

Multiclass with a shared fold plan

Factor levels fix the global class order. Supply one out-of-fold class matrix and a shared fold vector. All classes must be observed globally; individual folds may omit classes. Each OvO pair must have at least one observation in every fold. Empty pairs fail with their class names and fold ID; singleton and single-class binary subcalibrations are allowed. OvR uses every row in each fold. No pair-specific training plans are introduced.

Class matrices must have finite entries in `[0, 1]` and row sums within `1e-8` of one; accepted rows are normalized. In OvO, pair normalization precedes calibration rounding, with zero pair sums replaced by an event probability of 0.5 (also subject to fit rounding). The later class in factor order is the event; in OvR, the selected class is the event against the rest. New prediction probabilities remain unrounded.

Predict with a list of class matrices named by fold ID, from the corresponding retained models, for the same new observations in the same order. Binary Cross aggregation happens separately for each subproblem before reciprocal OvO coupling or OvR normalization. Log loss uses the stable equal-weight endpoint merger; Brier uses the equal-weight mean of binary fold points. No pooling, size weighting, or averaging of finished multiclass fold outputs is performed. Scores and plots use the final class matrix.

This manual shared-fold interface is an R extension. Python 1.5.4 has no corresponding public manual Cross-multiclass whole-procedure API; numerical checks use binary reference components and independent combination. Binary diagnostics are not multiclass or aggregate Cross confidence intervals.

See Also

`vignette("method", package = "vennabers")` for the theory and its guarantees, `vignette("manual-calibration", package = "vennabers")` for a worked introduction, and vennabers_examples for annotated installed scripts.

`predict.cross_venn_abers`, `print.cross_venn_abers`, `brier_score`, `log_loss`, `calibration_plot`

Examples

```
# Precomputed probabilities from three retained fold models. Each model
# excluded its corresponding calibration fold from its learned pipeline.
calibrator <- cross_venn_abers(
  truth = c(FALSE, TRUE, FALSE, TRUE, FALSE, TRUE),
  probability = c(.1, .8, .2, .9, .3, .7),
  fold = c("A", "A", "B", "B", "C", "C")
)
print(calibrator)

# Each row is the same new observation across all corresponding fold models.
newdata <- cbind(A = c(.2, .8), B = c(.3, .7), C = c(.1, .9))
```

```
result <- predict(calibrator, newdata)
result
test_truth <- c(FALSE, TRUE)  # Untouched by fitting and calibration.
brier_score(test_truth, result$probability)
log_loss(test_truth, result$probability)
calibration_plot(test_truth, result$probability)

# After installation, find the standalone example with:
# system.file("examples", "cross-multiclass.R", package = "vennabers")
library(vennabers)

# Precomputed out-of-fold probabilities from two retained multiclass models.
# Model A excluded rows 1:2; model B excluded rows 3:6, including preprocessing
# and tuning. Retain both models yourself. No model is trained by this package.
classes <- c("cat", "dog", "bird")
truth <- factor(c("cat", "dog", "cat", "bird", "dog", "bird"), levels = classes)
fold <- c("A", "A", "B", "B", "B", "B")
p <- rbind(c(.7, .2, .1), c(.2, .6, .2), c(.5, .3, .2),
           c(.1, .2, .7), c(.3, .5, .2), c(.2, .3, .5))
colnames(p) <- classes
# The same three untouched assessment observations, in the same row order,
# predicted by each corresponding retained model. List order is immaterial.
new_p <- list(B = rbind(c(.6, .3, .1), c(.1, .4, .5), c(.1, .8, .1)),
              A = rbind(c(.5, .4, .1), c(.2, .2, .6), c(.2, .7, .1)))
new_p <- lapply(new_p, function(x) { colnames(x) <- classes; x })
assessment <- factor(c("cat", "bird", "dog"), levels = classes)

local({
  plot_file <- tempfile(fileext = ".pdf")
  grDevices::pdf(plot_file)
  on.exit({ grDevices::dev.off(); unlink(plot_file) })
  for (method in c("ovo", "ovr")) {
    fit <- cross_venn_abers(truth, p, fold, method = method, precision = 2)
    print(fit)
    for (loss in c("log", "brier")) {
      details <- predict(fit, new_p, loss = loss, type = "details")
      probability <- predict(fit, new_p, loss = loss)
      predicted_class <- predict(fit, new_p, loss = loss, type = "class")
      # Reconstruct from the fold endpoints/points, then combine subproblems.
      parts <- details[[2L]]
      problems <- unique(parts[c(if (method == "ovo") "non_event", "event")])
      merged <- vapply(seq_len(nrow(problems)), function(i) {
        selected <- parts$event == problems$event[i]
      if (method == "ovo") selected <- selected & parts$non_event == problems$non_event[i]
        vapply(seq_len(nrow(probability)), function(row) {
          f <- parts[selected & parts$row == row, ]
          if (loss == "log") {
            plogis(mean(log(f$upper)) - mean(log1p(-f$lower)))
          } else mean(f$probability)
        }, 0)
      }, numeric(nrow(probability)))
      recovered <- if (method == "ovr") merged else vapply(classes, function(class) {
        q <- cbind(1 - merged[, problems$non_event == class, drop = FALSE],
```

```
                  merged[, problems$event == class, drop = FALSE])
        1 / (1 + rowSums((1 - q) / q))
      }, numeric(nrow(probability)))
      recovered <- recovered / rowSums(recovered)
      stopifnot(max(abs(recovered - probability)) <= 1e-10,
                identical(probability, details$prediction),
                identical(predicted_class,
                          factor(classes[max.col(probability, ties.method = "first")],
                                 levels = classes)))
      print(details)
      print(predicted_class)
      print(c(brier = brier_score(assessment, probability),
              log = log_loss(assessment, probability)))
      groups <- calibration_plot(assessment, probability, bins = 1)
      stopifnot(max(abs(groups$mean_probability - colMeans(probability))) <= 1e-10,
                all(groups$count == length(assessment)), all(groups$event_rate == 1 / 3))
      print(groups)
    }
  }
})
# One shared fold plan serves all pairs/classes. Fold A lacks bird but every
# OvO pair remains nonempty. Empty pairs fail with the pair and fold identified.
# Endpoints describe binary subproblems, with no aggregate confidence interval.
# Manual Cross multiclass is an R extension; Python has no matching public
# manual whole-procedure API. Numerical tests reconstruct binary components.
```

`cross_venn_abers_regression` *Manual Cross Venn-Abers Regression with Matched Fold Models*

Description

Calibrates the numeric predictions of regression models with a Cross Venn-Abers regressor (CVAR): one moderated Inductive Venn-Abers regression calibrator per fold, with complete fold results averaged with equal weights. It needs the observed outcomes, the out-of-fold predictions of one retained model per fold and the fold labels; the caller trains and retains the models and chooses all data roles. Choose it over `venn_abers_regression` when a single held-out calibration set would cost too much training data, or use `calibrated_workflow` with `method = "cross"` to let the package fit the fold models.

Usage

```
cross_venn_abers_regression(truth, prediction, fold, ..., m = 1,
                            epsilon = NULL)

## S3 method for class 'cross_venn_abers_regression'
predict(object, newdata, ...,
                                            type = "response")
```

Arguments

`truth`	A nonempty numeric vector of finite real-valued outcomes.
`prediction`	A numeric vector of finite out-of-fold regression predictions, the same length as `truth`. Each observation must have been excluded from the entire learned procedure supplying its prediction.
`fold`	One fold ID per observation: nonempty character labels or finite positive whole-number numeric IDs. At least two represented folds are required; unequal sizes are allowed. Factors, logical IDs, missing values and invalid lengths are errors. Labels match literally, without trimming or case folding; numeric IDs use Base R `as.character()` names.
`m`	One finite positive whole number, default 1. Every fold of size k_f must satisfy $1 \leq m \leq \lfloor (k_f - 1)/2 \rfloor$. Even the default requires at least three observations per fold.
`epsilon`	Alternatively, one finite numeric scalar strictly between 0 and 1. Each fold uses $m_f = \lfloor \epsilon(k_f + 1)/2 \rfloor$, checked against the same domain. Explicitly supplying both `m` and `epsilon` is an error; omitting `m` permits `epsilon` alone. Explicit `NULL` is invalid.
`object`	A fitted `cross_venn_abers_regression` object.
`newdata`	A finite real-valued numeric matrix with exactly one uniquely named column per stored fold. Each column comes from that fold's retained model; every row refers to the same new observation across columns. Columns are matched by name and may be reordered. Missing, additional, duplicated, empty, missing-value or absent column names are errors.
`type`	Exactly `"response"` (default) or `"details"`.
`...`	Must be empty. Unknown options are errors.

Details

Each fold follows the asymmetric moderation, isotonic endpoints and Formula 6 weighted point merger documented in `venn_abers_regression`. The midpoint is used only when both merger weights vanish, including with nonconstant truth; small positive weights retain the weighted merger. Constant truth and coincident endpoints return their common finite value.

For each new observation, first compute the complete point, lower and upper IVAR results for every fold. Then average each of those three quantities separately, with equal fold weights even for unequal fold sizes. No pooling, fold-size weighting, or new point merger on averaged bounds is performed. The three aggregates can be reconstructed by averaging the corresponding columns from `details$folds`. Means use scaling and compensated summation to avoid intermediate overflow and preserve cancellation residuals, including on platforms without extended floating-point precision.

Truth and prediction vectors align by position, ignoring names. Negative values and values outside $[0, 1]$ are valid; numeric 0/1 truth remains regression. Factors, logical values, lists, classed vectors, matrices at fit, nonfinite values and mismatched lengths are rejected. No values are rounded, clipped or silently dropped. Ties and new scores outside calibration ranges are supported. Predict trains nothing and accepts no new moderation options.

The theoretical setting assumes IID observations and a base procedure fixed independently of its calibration observations. Every observation in a fold must have been excluded from its corresponding model's entire learned flow, including preprocessing and tuning. That same retained model supplies the fold's new prediction column. Assessment outcomes stay outside training and calibration. The caller is responsible for model provenance and semantic row identity; matching column names cannot prove these. A single model retrained on all development observations cannot replace the matching fold models.

The aggregated regression bounds are descriptive summaries of fold bounds, without any additional coverage or selector guarantee. They are not confidence intervals or prediction intervals for individual future outcomes. Neither the Cross point nor these bounds inherit a general selector/autocalibration claim, and improvement in RMSE or MAE is not promised. Epsilon is a moderation level, not a prediction-interval error rate.

This manual Cross entry point is an R extension. Offline reference fixtures combine valid IVAR components from Python venn-abers 1.5.4 and independently computed arithmetic means; no public manual Python CVAR wrapper is claimed. See 'tests/testthat/extdata/cross-regression-reference.md' in the source package for provenance, retained reference defects and fixed scale-aware tolerances.

Value

Fit returns an object of class `cross_venn_abers_regression`. Documented fields are `folds` (character IDs in first-observed order), `observations` (total calibration size), and `calibrators`, a named list of complete `venn_abers_regression` fits, one per fold. Each records `observations`, `requested` (empty for the implicit default, otherwise the explicit `m` or `epsilon`), effective `m`, effective `epsilon` equal to $2m_f/(k_f + 1)$, and outer bounds `a`, `b`. Unequal folds can have different effective moderation levels. Any invalid fold fails the entire fit with fold context; nothing is capped or omitted. Fits can be saved with `saveRDS()` and restored with `readRDS()` in a fresh R process with the installed package loaded.

Predict returns a data frame with double columns `prediction`, `lower`, `upper`, in supplied row order. With `type = "details"`, it returns a list containing `prediction` (that standard data frame) and `folds` (a named list of fold data frames with the same three columns, in stored fold order). Zero and one input row retain these schemas; empty details retain every fold ID. Outputs are finite and the point lies between the bounds up to checked numerical roundoff. Row names do not align observations. `plot.cross_venn_abers_regression` draws each fold's stored calibration map and returns the maps invisibly as a data frame.

See Also

`vignette("method", package = "vennabers")` for the theory and its limits, `vignette("manual-calibration", package = "vennabers")` for a worked introduction, and vennabers_examples for annotated installed scripts.

`venn_abers_regression`

Examples

```
# Fits retained lm() fold models with the held-out observations excluded,
# reconstructs the aggregates, and scores on separate assessment data.
source(system.file("examples", "cross-regression.R", package = "vennabers"))
```

```
fit <- cross_venn_abers_regression(
  c(-10, 0, 2, 6, 20, 10, 10, 10), rep(0, 8),
  rep(c("A", "B"), c(5, 3))
)
predict(fit, cbind(B = c(1, 0), A = c(-1, 0)), type = "details")
```

`local_calibration_splits` *Resolve Prepared Local Calibration Splits*

Description

Validate all prepared assignments and return a serializable function mapping selected local raw data to one mixed or ordered Inductive holdout or Cross partition.

Usage

```
local_calibration_splits(splits, columns)
```

Arguments

`splits`	A nonempty list of data-bound holdouts from `rsample::initial_split()` or `rsample::make_splits()`, or simple Cross `rset` partitions from `rsample::vfold_cv()` or `rsample::manual_rset()`. Each split's stored data is its local fitting domain, typically one outer analysis set.
`columns`	Unique selected raw predictor names in carrier order followed by the outcome name. These columns, including outcome, define data identity. They must exactly match the canonical columns received by the resolver; an omitted outcome or selected predictor is an error.

Details

Every prepared split is validated before the function is returned. Identity includes column values, types, factor levels and row order. Container class and row names are ignored. Extra unselected columns are ignored; duplicate observations remain distinct positions. Inductive uses the fixed single fold ID `Inductive1`; Cross retains the ordered unique fold IDs. Repeated, overlapping, incomplete and noncomplementary inner Cross partitions are rejected before fitting. Identical data require the same method, ordered fold IDs and ordered analysis/assessment indices. Equivalent rset containers normalize to the same plan. Duplicate identical plans are allowed; conflicting entries are errors identifying their positions. A lookup without a matching snapshot is an error. No call counter, first conflicting match, or outer assessment access is used. Construction compares pairs of prepared entries; lookup scans the saved snapshots. Large collections may need a more efficient user resolver.

The resolver supports `saveRDS()` and `readRDS()` and arbitrary lookup order. An arbitrary user callback is trusted R code and must obey the same data and plan contract; this helper does not prove the absence of hidden state in other callbacks. Keep model selection and all other learning outside assessment data. An ordered holdout can be constructed with `rsample::make_splits()`:

reserve `ceiling(n * q)` trailing calibration rows and the preceding `n - ceiling(n * q)` training rows, requiring both parts to be nonempty. At `n = 11, q = .25`, these are training positions 1–8 and calibration positions 9–11, with no sorting, mixing or stratification. The order is local to each outer analysis set. Cross accepts mixed or non-shuffled ordinary or classification-stratified complete partitions. Exact training complements are required even for non-shuffled folds; these are not forward validation. See the executable installed `examples/calibrated-resampling.R` and `examples/reference-splits.R`.

Value

A function of local raw `data`, suitable for the `splits` argument of `calibrated_workflow()`. It returns a data-bound `rsplit` or Cross `rset`.

See Also

`calibrated_workflow`

Examples

```
if (all(vapply(c("parsnip", "workflows", "hardhat", "rsample", "recipes",
                 "rlang", "tibble", "withr", "generics", "tidyselect"),
               requireNamespace, logical(1), quietly = TRUE))) {
  data <- data.frame(x = 1:20, truth = factor(rep(c("yes", "no"), 10)))
  split <- withr::with_seed(42, rsample::initial_split(data))
  resolve <- local_calibration_splits(list(split, split), c("x", "truth"))
  stopifnot(identical(as.integer(resolve(data)), as.integer(split)))
}
```

`model_adapter` *Own Fit and Predict Functions as a Workflow Model*

Description

Turn a pair of your own functions into an unfitted parsnip model specification for `workflows` and `calibrated_workflow`. You do not register a parsnip engine. The workflow keeps preprocessing, data roles, calibration splits and calibration, and calls your functions on each local training and prediction step.

Usage

```
model_adapter(mode, fit, predict)
```

Arguments

`mode`	`"classification"` or `"regression"`.

`fit` A `function(x, y)` that trains a fresh model. `x` is a base data frame of the predictors after the workflow's preprocessing, for the local training rows only. `y` is the matching outcome: a factor with all stored levels for classification, or a numeric vector for regression. Return any object that holds everything `predict` needs.

`predict` A `function(object, new_data)`. `object` is the value returned by `fit`. `new_data` is a base data frame with the same preprocessing applied. For classification, return a numeric matrix with one column per level of `y`, named by the original class names. Binary classification also needs both columns. Rows must keep the input order. Values must be finite, in [0, 1] and sum to one within 1e-8 in every row. For regression, return a plain numeric vector with one finite value per row.

Value

An unfitted `parsnip` model specification of class `vennabers_adapter` with engine `"callbacks"`. Add it to a workflow with `workflows::add_model()`. After fitting, `workflows::extract_fit_engine()` returns a `vennabers_adapter_fit` whose `state` element is the object your `fit` returned.

Lifecycle

Every local fit calls `fit` once, on the rows of that fit, after the workflow learned its preprocessing on the same rows: each Inductive or Cross source model, the `Full1` refit of `prediction_model = "full"`, and each analysis set of an outer resample. Calibration rows never reach `fit`. A fitted adapter workflow is an ordinary fitted workflow and can be adopted frozen with `calibrate_workflow`.

Supported tasks are binary classification, shared multiclass OvO/OvR and regression, with Inductive or Cross calibration, matched or full prediction models and outer resampling. For shared multiclass, the callbacks must learn and predict all local classes. Encodings and transformations belong in the workflow; the adapter does not guess model-specific preprocessing.

The existing compositions work unchanged:

- `model_training = "pairwise"`: a binary callback is fitted on each class pair's rows, with `y` holding that pair's two original levels. This does not make it a shared multiclass model.
- `postprocessing`: `fit` learns on B only, for a holdout and for every inner ensemble member; P and C never reach it. Sigmoid on probabilities, Tailor logistic and multinomial on class probabilities, and Tailor linear or numeric isotonic on regression predictions keep their task, method and convergence rules.
- `tuning`: Grid and Bayes searches tune calibration and declared recipe parameters, each local search fitting fresh callbacks.

The callbacks receive no case weights, additional data or tuning parameters. Bind fixed model options inside the functions. An adapter has no `tune()` parameters: a search grid column for it is rejected, and so is `update()` with any argument. `postprocessing$sigmoid_input = "glm_link"` requires the native GLM engine, also when a callback calls `stats::glm()`.

Predictions and errors

Class probability columns are aligned to the stored class order by name, never by position, and returned as `.pred_<class>`. Class predictions of a plain adapter workflow take the first maximum

in level order. Calibrated workflows apply their own class rules. Regression returns `.pred`. Empty `new_data` returns an empty result without calling `predict`.

Malformed output is an error, never repaired: a non-matrix or unnamed result, missing, duplicate or unknown class columns, a wrong row count, or invalid probabilities. For regression, a matrix, a wrong length or non-finite values are errors. Errors from your functions start with `model_adapter() `fit` callback:` or `model_adapter() `predict` callback:`. Calibrated workflows add the fold or refit, for example `Inductive1`, `Fold2` or `Full1`, and the class pair (`pair_2 Fold1`) or inner postprocessing fold (`Inductive1 / Fold2 sigmoid`).

Reproducibility and saving

The local seed of `calibrated_workflow` also applies to callbacks that use R's random number generator. Callbacks must seed other native generators themselves.

A saved fit stores the functions themselves, not their names. To use it in a fresh R process, define the functions in a factory that binds their helpers and fixed settings locally, and call package functions as `pkg::fun`. Values looked up later in the global environment, or state shared and changed between fits, are not supported. Models whose state needs their own save and restore procedure, such as external pointers, are not covered.

See Also

'`model-adapter.R`' in vennabers_examples for GLM, multinomial and linear regression adapters with saved fits, pairwise OvO, Sigmoid postprocessing and a grid search.

`calibrated_workflow`, `calibrate_workflow`, `workflow_training`.

Examples

```
packages <- c("parsnip", "workflows", "recipes", "rsample", "withr")
if (all(vapply(packages, requireNamespace, logical(1), quietly = TRUE))) {
  glm_adapter <- model_adapter("classification",
    fit = function(x, y) list(classes = levels(y),
      model = stats::glm(y ~ ., data = cbind(x, y = y), family = stats::binomial())),
    predict = function(object, new_data) {
      p <- stats::predict(object$model, new_data, type = "response")
      matrix(c(1 - p, p), ncol = 2L, dimnames = list(NULL, object$classes))
    })
  development <- data.frame(
    x = sin(1:80) + rep(c(1, 0, 0, 1), 20),
    truth = factor(rep(c("yes", "no", "no", "yes"), 20), levels = c("yes", "no"))
  )
  base <- workflows::workflow() |>
    workflows::add_formula(truth ~ x) |>
    workflows::add_model(glm_adapter)
 fitted <- generics::fit(calibrated_workflow(base, outcome = "truth", seed = 42), development)
  predict(fitted, data.frame(x = c(-0.5, 0.5)), type = "prob")
}
```

plot.venn_abers *Plot the Calibration Map of a Fitted Venn-Abers Calibrator*

Description

Draws a fit diagnosis from the stored state of a fitted calibrator, without new truth values. For classification, the calibration map shows the lower and upper Venn-Abers values as steps and the regularized probability as a dashed line over the uncalibrated probability on `[0, 1]`. The width diagram shows `upper - lower` over the same scores; it is neither a confidence level nor a nominal coverage.

Cross fits overlay the map of every fold calibrator in one panel, distinguished by color and legend. Comparing fold maps at common scores does not describe the spread of an individual Cross prediction.

Choosing both diagrams draws them in their own grid of panels and restores the previous graphical parameters on exit, also after errors. A single diagram uses the current layout. Fits without a stored calibration map (`knots`, `lower`, `upper`) are errors; refit them with vennabers 1.0.0.

Usage

```
## S3 method for class 'venn_abers'
plot(x, which = "map", loss = "log", ...)

## S3 method for class 'cross_venn_abers'
plot(x, which = "map", loss = "log", ...)

## S3 method for class 'venn_abers_regression'
plot(x, which = "map", ...)

## S3 method for class 'cross_venn_abers_regression'
plot(x, which = "map", ...)

## S3 method for class 'vennabers_fit'
plot(x, which = "map", loss = x$loss, ...)
```

Arguments

`x`	A fitted binary or multiclass `venn_abers` or `cross_venn_abers` object, or a fitted `venn_abers_regression` or `cross_venn_abers_regression` object. For `plot.vennabers_fit`, a frozen fit from `calibrate_workflow` or the engine fit of a trained `calibrated_workflow`, extracted with `workflows::extract_fit_engine()`; see the Workflows section.
`which`	A character vector of distinct diagrams from `"map"` (default), the calibration map, and `"width"`, the Venn-Abers width `upper - lower`. Both draw the map before the width, whatever their order.

loss	Exactly "log" (default) or "brier": the regularization of the drawn and returned probability, as in predict.venn_abers. Classification only: regression methods take no loss, and passing one is an error. For a workflow fit, the default is the loss stored at fitting.
...	Base R graphical options passed to plot(), taking precedence over the defaults in every panel. For regression, xlim also sets the range of the returned table.

Value

Invisibly, a data frame with the same columns whatever which holds: score, lower, upper, probability, and width. Rows hold each stored probability knot plus the range ends 0 and 1 in increasing order. Values are those of predict() at score with the chosen loss, and width is upper - lower.

For cross_venn_abers, a leading character column fold holds the stored fold IDs. Rows group by fold in stored order, each with its own knots and the values of its own fold calibrator, not the aggregated Cross probability.

For multiclass fits, the tables of all subproblems are stacked in stored order, with identity columns named as in venn_abers_details after fold and before the values: class for one-vs-rest, class_1 and class_2 for one-vs-one. Within a Cross subproblem, folds follow the stored fold IDs of the fit.

For regression, the columns are score, lower, upper, prediction, and width, after fold for cross_venn_abers_regression, with the values of predict() at score. The range ends replace 0 and 1; see the Regression section.

For a workflow fit, the table is that of its stored Venn-Abers calibrator. A pairwise fit stacks the tables of its pair components in stored order, led by the identity columns pair_id, then fold for Cross fits, then class_1 and class_2, as in venn_abers_details.

Between neighboring rows, the map holds the lower value of the left row and the upper value of the right row, so the width between knots exceeds the width at them. The drawn curves follow these values exactly.

Multiclass

A multiclass fit draws one panel per subproblem and diagram, titled "Calibration map: <class>" for one-vs-rest and "Calibration map: <class_1> vs <class_2>" for one-vs-one (likewise "VA width"). Each panel shows the map of the subproblem's own binary calibrator over its score: the probability of the class for one-vs-rest, and the probability of class_2 renormalized within the pair for one-vs-one. The final coupling into class probabilities is not drawn. Cross fits overlay the fold maps of each subproblem in its panel, each fold in the same color across panels.

The panels fill their own grid row by row, nearly square as in calibration_plot. With both diagrams, each row of maps is followed by the widths of the same subproblems. A single panel, as for one one-vs-one pair, uses the current layout. Options in ... apply to every panel.

Regression

A regression fit draws the lower and upper Venn-Abers values as steps and the point prediction as a dashed line over the uncalibrated prediction, with gray dashed reference lines at the moderated truth values a and b that bound every Venn-Abers value. The width diagram shows upper - lower in units of the outcome.

The drawn range spans the stored knots, widened on each side by 5 % of their span; a single knot widens by 5 % of its absolute value, or 0.05 at zero. A Cross fit spans the knots of all folds, overlays its fold maps with a reference line at each fold's `a` and `b`, and returns every fold map over this shared range. A finite `xlim` in `...` replaces the range: the table then holds its ends and the knots within.

Workflows

`plot()` on a frozen fit from `calibrate_workflow`, and on `workflows::extract_fit_engine(fit)` for a trained `calibrated_workflow`, draws the fit diagnosis of the stored Venn-Abers calibrator: binary, multiclass, or regression, inductive or Cross, as described above for that calibrator, with the same table. There is no method for the workflow itself. Postprocessing models (sigmoid, tailor, isotonic, or their ensembles) are source models before the calibrator: the map shows the calibrator over their output, not the postprocessing step.

A pairwise fit (`model_training = "pairwise"`) draws one panel per pair component and diagram, titled `"Calibration map: <class_1> vs <class_2>"`, in a grid as for multiclass fits. Each panel shows the binary calibrator of its pair over the probability of `class_2` from that pair's own model.

A fit without a stored Venn-Abers calibrator is an error.

See Also

Assessment graphics need held-out truth values instead of the stored state: `corp_plot` and `calibration_plot` for classification probabilities, `regression_plot` for regression predictions. `vignette("diagnostics", package = "vennabers")` compares both kinds of graphics.

Examples

```
truth <- c(FALSE, TRUE, FALSE, TRUE, TRUE, FALSE, TRUE, TRUE)
uncalibrated_probability <- c(0.1, 0.2, 0.3, 0.4, 0.4, 0.6, 0.7, 0.9)
calibrator <- venn_abers(truth, uncalibrated_probability)

map <- plot(calibrator, which = c("map", "width"))
head(map)

cross <- cross_venn_abers(truth, uncalibrated_probability, fold = rep(1:2, 4))
plot(cross, loss = "brier")

classes <- c("a", "b", "c")
class_truth <- factor(rep(classes, 4), levels = classes)
class_probability <- prop.table(matrix(c(
  6, 3, 1, 2, 5, 3, 1, 2, 7, 5, 1, 4, 3, 4, 3, 2, 2, 6,
  4, 4, 2, 1, 6, 3, 3, 3, 4, 5, 3, 2, 2, 7, 1, 2, 2, 6
), 12, 3, byrow = TRUE, dimnames = list(NULL, classes)), 1)
ovr <- venn_abers(class_truth, class_probability, method = "ovr")
head(plot(ovr, which = c("map", "width")))

outcome <- c(-4, 1, 0, 3, 2, 6, 5, 9, 7, 12)
uncalibrated_prediction <- c(-2, 0, 0, 1, 2, 3, 3, 5, 6, 8)
regression <- venn_abers_regression(outcome, uncalibrated_prediction)
head(plot(regression, which = c("map", "width")))
plot(regression, xlim = c(0, 5))
if (all(vapply(c("parsnip", "workflows", "hardhat", "rsample", "recipes",
```

```
               "rlang", "tibble", "withr", "generics", "tidyselect"),
             requireNamespace, logical(1), quietly = TRUE))) {
  d <- data.frame(x = sin(1:80), truth = factor(rep(c("yes", "no", "no", "yes"), 20)))
  base <- workflows::workflow() |>
    workflows::add_formula(truth ~ x) |>
    workflows::add_model(parsnip::logistic_reg(engine = "glm"))
  frozen <- calibrate_workflow(generics::fit(base, d[1:40, ]), d[41:80, ], "truth")
  print(head(plot(frozen)))

  trained <- generics::fit(calibrated_workflow(base, "truth", seed = 42), d)
  plot(workflows::extract_fit_engine(trained), which = "width")
}
```

`predict.cross_venn_abers` *Predict with Matched Fold-Model Probabilities*

Description

Applies a fitted `cross_venn_abers` calibrator to new observations and returns calibrated probabilities, predicted classes or per-fold details. It needs the new probabilities from every retained fold model, as one named column per fold (binary) or one named class matrix per fold (multiclass), with the same observation in each row. Use `predict.venn_abers` for an Inductive calibrator, which needs only the probabilities of a single model.

Usage

```
## S3 method for class 'cross_venn_abers'
predict(object, newdata, ..., loss = "log", type = "response")
```

Arguments

`object`	A fitted `cross_venn_abers` object.
`newdata`	A numeric/integer matrix of finite event probabilities in `[0, 1]`, one row per new observation and one column per corresponding fold model. Column names must be complete, non-empty, non-missing, unique, and match the fitted fold identities exactly. Any column order is accepted. A correctly named zero-row matrix is accepted. For multiclass, a list named uniquely and completely by fitted fold IDs, each element a numeric class matrix with the same number of new rows and all fitted class columns. Classes and folds are aligned by name. Zero-row matrices are accepted.
`...`	Must be empty; unsupported arguments are errors.
`loss`	Exactly `"log"` (default) or `"brier"`.
`type`	Exactly `"response"` (default), `"class"`, or `"details"`. Supply both options by their full argument names.

Details

New-observation columns are matched to their fold calibrators by identity, and the resulting IVAP results are combined using the chosen equal-weight aggregation.

Character fold identities match literally. Numeric fitting IDs match their Base R character representations. Missing, extra, duplicate, empty, or absent column names fail the entire call, as do invalid probability values and unsupported binary types or shapes (including vectors, lists, data frames, logical matrices and character matrices). A column for an unobserved fold is an error. Values are never dropped, clamped or rounded. For multiclass, invalid list names, unequal row counts, or an invalid class matrix in any fold fail the whole call. The class value, name, and row-sum rules match fit time; accepted row sums within tolerance are normalized. Row names are not alignment keys.

Use each retained fold model that supplied that fold's calibration probabilities; predictions from one all-data refit cannot substitute for the matched columns. The caller ensures the same observation occupies a given row in every column. Row names do not establish that alignment, and valid column names do not prove model provenance or exclusion of calibration folds from training. See `cross_venn_abers` for pipeline responsibilities.

For log loss, let p_0^k, p_1^k be the IVAP endpoints from fold k. If A is the geometric mean of $1-p_0^k$ and B is the geometric mean of p_1^k, the result is $B/(A+B)$. Evaluation uses `plogis(mean(log(upper)) - mean(log1p(-lower)))` in log space, with equal fold weight and no endpoint clipping. It does not pool calibration rows, weight folds by size, or average log-loss fold point probabilities.

For Brier, each fold probability is `upper + (lower^2 - upper^2) / 2`. The combined probability is the arithmetic mean of these fold probabilities, with equal weights even for unequal fold sizes. Both aggregates can be reconstructed from `type = "details"`. Loss changes neither the fold endpoints nor stored learning state and requires no new fit.

No aggregate endpoints are returned. The geometric pair $(1 - A, B)$ can cross; it is not an ordered interval or confidence interval. Neither that pair nor the combined point inherits the IVAP perfect-calibration guarantee. There is no universal score-improvement guarantee. The output feeds the existing scores and Base R calibration plot directly. For multiclass, apply this binary merger within each pair/class first, then OvO coupling or OvR normalization as described in `cross_venn_abers`. Reconstruct the final matrix from these per-fold diagnostics; there is no average of already coupled/normalized multiclass fold predictions.

Value

For binary calibration, `type = "response"` returns a data frame with exactly one double column, `probability`, containing one finite regularized event probability in `[0, 1]` per input row, in input row order. A zero-row input returns the same empty schema.

For `type = "class"`, an unnamed logical vector: `TRUE` when the combined event probability is at least 0.5, otherwise `FALSE`. This uses the combined probability, not a vote among folds. Zero rows return `logical(0)`.

For `type = "details"`, a list with `prediction` (the standard data frame) and `folds` (a data frame with columns `row`, `fold`, `lower`, `upper`, `probability`). The integer `row` is the one-based position in this call's new data, independent of input row names. The character `fold` is the stored fold ID. The three double columns contain that fold's IVAP endpoints and its probability for the chosen loss. Rows are grouped by stored fold order, then new-row order. There is exactly one record per new row and fold: zero input rows give an empty fold table, and one input row gives one record per fold. These are new-observation results, not out-of-fold calibration probabilities.

For multiclass, the standard output is a double class-probability matrix in stored factor-level order, with row names from the first stored fold's matrix. Class output is a factor with those levels, choosing the highest final probability and the first level on exact ties (also for two classes). Details contain `prediction` plus `pairs` for OvO or `classes` for OvR. The table columns are integer `row`, character `non_event` (OvO only), character `event`, character `fold`, and double `lower`, `upper`, `probability`. Events are the later pair class for OvO or the literal class against the rest for OvR. Records follow pair/class order, then fold first-appearance order within that subproblem's calibration rows, then new-row order. Each row/subproblem/fold appears once. The points use the chosen loss; endpoints do not change with loss. Zero or one new row retains matrix, factor, and detail-table types and mappings.

See Also

`vignette("method", package = "vennabers")` explains the Cross merger and the limits of its guarantees; `vignette("manual-calibration", package = "vennabers")` shows matched fold-model columns in a worked example.

`cross_venn_abers`, `venn_abers_one_hot`, `brier_score`, `log_loss`

Examples

```
calibrator <- cross_venn_abers(c(0, 1, 0, 1), c(.1, .8, .2, .9), c(1, 1, 2, 2))
newdata <- cbind("1" = c(.2, .8), "2" = c(.3, .7))
predict(calibrator, newdata)
stopifnot(identical(predict(calibrator, newdata), predict(calibrator, newdata[, 2:1])))
predict(calibrator, newdata[FALSE, ])
details <- predict(calibrator, newdata, loss = "brier", type = "details")
details$folds
reconstructed <- vapply(seq_len(nrow(newdata)), function(i) {
  mean(details$folds$probability[details$folds$row == i])
}, 0)
stopifnot(max(abs(reconstructed - details$prediction$probability)) < 1e-10)
predict(calibrator, newdata, loss = "brier", type = "class")
```

`predict.venn_abers` *Predict with a Venn-Abers Calibrator*

Description

Applies a fitted `venn_abers` calibrator to new uncalibrated probabilities from the same model and returns calibrated probabilities or predicted classes, preserving the order of `newdata`. For binary calibration it returns the Venn-Abers interval of two probabilities together with the regularized event probability that merges them according to `loss`; multiclass calibration returns a class-probability matrix. Use `predict.cross_venn_abers` for a Cross calibrator, which needs one probability column per retained fold model.

Usage

```
## S3 method for class 'venn_abers'
predict(object, newdata, ..., loss = "log", type = "response")
```

Arguments

object	A fitted venn_abers object.
newdata	A numeric vector of uncalibrated event probabilities with finite values in [0, 1]. numeric(0) is accepted for binary objects. For multiclass, a numeric matrix under the fit-time class, name, range, and row-sum contract; zero rows are accepted.
...	Must be empty; unsupported arguments are errors.
loss	Exactly "log" (default) or "brier".
type	Exactly "response" (default) or "class". Multiclass also accepts "details". Supply both options by their full argument names.

Details

Log-loss regularization is upper / (1 - lower + upper). Brier regularization is upper + (lower^2 - upper^2) / 2. Changing loss does not change the interval endpoints or mutate the fitted calibrator. New probabilities are never rounded, even if calibration used precision. Input names are not alignment keys. The endpoints are Venn-Abers bounds, not confidence intervals. The probability column can be passed to the existing scores and calibration plot with either loss choice.

For OvO, class and pair normalization precedes prediction, including the zero-pair replacement described in venn_abers. New pair probabilities remain unrounded. After binary regularization, for class j let q[j,i] be its probability in its pair with class i. Reciprocal coupling is w[j] = 1 / (sum(1 / q[j,i]) - (K - 2)), summed over the other K-1 classes. The response is w / sum(w). For each diagnostic record, probability is the event's q and 1 - probability is the non-event's q.

For OvR, input class normalization precedes prediction. Each class probability enters its binary calibrator unrounded. After log or Brier regularization, divide the class points by their sum for each new row to obtain the response. The classes table supplies those pre-normalization points, so details reconstruct the final matrix for either method and loss. The endpoints describe only the binary subproblems, not multiclass confidence intervals. Binary Brier regularization does not imply optimality for the final multiclass Brier score or guarantee improvement in assessment scores.

Value

For type = "response", a data frame with one row per value in newdata and three double columns: lower and upper are the Venn-Abers interval bounds, and probability is the regularized event probability using the chosen loss. For type = "class", an unnamed logical vector, TRUE when the final event probability is at least 0.5 and FALSE otherwise. Zero observations return the same empty data-frame schema or logical(0); one observation retains its data-frame/vector shape.

For multiclass, "response" returns a numeric class-probability matrix in fitted factor-level order, retaining new-row order and matrix row names. "class" returns an unnamed factor with the stored levels; the first stored class wins exact ties. This also applies with exactly two classes, whereas binary vector ties select the event. Classes use the final probabilities of the fitted method. venn_abers_one_hot represents binary or multiclass class decisions as a one-hot matrix. For OvO, "details" returns a list with prediction (the response matrix) and pairs, a data frame with row, non_event, event, lower, upper, and probability. Row is the integer one-based new-row position; the two class names are character columns. The remaining columns are doubles. Pairs follow

factor-level combination order, then new-row order. For OvR, the list contains `prediction` and `classes`, a data frame with integer `row`, character `event`, and double `lower`, `upper`, `probability`. The event is the named class against all other classes; no artificial rest label is introduced. Records follow stored class order, then new-row order. Zero or one new rows retain these matrix, factor, list, and table schemas.

See Also

`vignette("method", package = "vennabers")` explains the interval and the merger into one probability; `vignette("manual-calibration", package = "vennabers")` shows prediction in a worked example.

`venn_abers`, `venn_abers_one_hot`, `brier_score`, `log_loss`

Examples

```
calibrator <- venn_abers(c(0, 1, 0), c(0.1, 0.2, 0.3))
predict(calibrator, c(0.05, 0.2, 0.4))
predict(calibrator, c(0.05, 0.2, 0.4), loss = "brier")
predict(calibrator, c(0.05, 0.2, 0.4), loss = "brier", type = "class")
```

---

print.cross_venn_abers

Print a Cross Venn-Abers Calibrator

Description

Prints three lines: `<cross_venn_abers>`, `Folds: K`, and `Calibration observations: N`, with the actual counts substituted. For multiclass, also identifies the method and class names in stored order. It exposes neither calibration arrays nor predictive models.

Usage

```
## S3 method for class 'cross_venn_abers'
print(x, ...)
```

Arguments

`x`	A fitted `cross_venn_abers` object.
`...`	Unused.

Value

`x`, unchanged and invisibly.

Examples

```
calibrator <- cross_venn_abers(c(0, 1, 0, 1, 0, 1), c(.1, .8, .2, .9, .3, .7),
                               c("A", "A", "B", "B", "C", "C"))
print(calibrator)
```

print.venn_abers *Print a Venn-Abers Calibrator*

Description

Prints the calibrator class and counts of calibration observations, event observations, and unique probability knots. For multiclass objects, prints the method, calibration count, class names in stored order, and number of class pairs or classes against rest instead.

Usage

```
## S3 method for class 'venn_abers'
print(x, ...)
```

Arguments

x	A fitted venn_abers object.
...	Unused.

Value

x, invisibly.

Examples

```
venn_abers(c(0, 1, 0), c(0.1, 0.2, 0.3))
```

regression_plot *Assess Regression Predictions on Held-Out Pairs*

Description

Assesses regression predictions on evaluation pairs of truth and prediction, held out from fitting the calibrator, such as an assessment set or a test set. Unlike plot.venn_abers_regression, which shows the stored state of a fit, this graphic needs new truth values.

which = "observed" draws the observed against the predicted values with the dashed diagonal of equality, titled "Observed vs predicted". which = "residual" draws the residuals, observed minus predicted, against the predicted values with a dashed horizontal line at zero, titled "Residuals vs predicted".

Usage

```
regression_plot(truth, prediction, which = "observed", ...)
```

Arguments

truth	A non-empty numeric vector of finite observed values.
prediction	A numeric vector of finite predicted values, the same length as truth, such as the prediction column of predict.venn_abers_regression.
which	One or more distinct values of "observed" (default) and "residual": the diagrams to draw.
...	Base R graphical options passed to plot(), taking precedence over the defaults in every panel.

Details

The pairs are drawn as points, without a smoother. Choosing both diagrams draws them in this order, one above the other over the same predictions, in their own grid, and restores the previous graphical parameters on exit, also after errors.

Invalid pairs are errors; there is no na.rm. The graphic describes the evaluation pairs given and makes no statement about the Venn-Abers regression bounds lower and upper, which it does not use.

Value

Invisibly, a data frame with one row per pair in input order and the columns prediction, truth, and residual (truth - prediction), the same whatever which holds.

See Also

venn_abers_regression for regression calibrators, plot.venn_abers_regression for their fit diagnoses, and corp_plot and calibration_plot for held-out classification probabilities. vignette("diagnostics", package = "vennabers") compares both kinds of graphics.

Examples

```
set.seed(1)
score <- runif(300, 0, 10)
truth <- score + sin(score) + rnorm(300, sd = .5)
fitting <- 1:150
calibrator <- venn_abers_regression(truth[fitting], score[fitting])
assessment <- predict(calibrator, score[-fitting])$prediction

regression_plot(truth[-fitting], assessment)
regression_plot(truth[-fitting], assessment, which = c("observed", "residual"))
```

venn_abers *Fit an Inductive Venn-Abers Calibrator*

Description

Turns the uncalibrated probabilities of an already fitted binary or multiclass classifier into calibrated probabilities with an Inductive Venn-Abers predictor (IVAP). It needs the true classes and the model's probabilities for a separate calibration set; it creates neither predictive models nor data splits. Choose it when one held-out calibration set is affordable; use `cross_venn_abers` when data are scarce and one model per fold can be retained, or `calibrated_workflow` to let the package fit the model and split the data.

Usage

```
venn_abers(truth, probability, precision = NULL, method = "ovo")
```

Arguments

`truth`	A non-empty logical or numeric vector containing only `FALSE`/`0` and `TRUE`/`1`. `TRUE`/`1` is the event class. For multiclass, a non-empty factor without missing values, with at least two non-empty levels. Levels fix class order; every level must be observed in the calibration set.
`probability`	A numeric vector of uncalibrated event probabilities, the same length as `truth`, with finite values in `[0, 1]`. For factor truth, a numeric matrix with one row per truth and complete, unique, non-empty column names matching the factor levels. Columns are reordered by name. Entries must be finite and in `[0, 1]`; row sums must be within `1e-8` of one and are normalized within that tolerance. No clipping, name repair, or dropped rows.
`precision`	`NULL` (default) disables rounding. Otherwise, one finite numeric whole number, including zero or negative values, passed to native R `round()` for calibration probabilities only. Logical, missing, nonfinite, fractional, and multiple values are errors.
`method`	Exactly `"ovo"` (default) or `"ovr"`. Explicitly supplying this argument requires factor truth and a class matrix.

Details

The calibration set must be separate from the fixed predictive model's training, including learned preprocessing and tuning. Single-class and singleton binary calibration sets are valid. Rows align by position, not names.

One-vs-One (`method = "ovo"`) uses each pair in factor-level order and only that pair's calibration observations. The later class is the binary event. Its probability is divided by the pair sum before optional rounding. An exactly zero pair sum uses event probability 0.5, including during prediction. This fit replacement is also rounded: at `precision = 0`, native R rounds 0.5 to zero. Only the event probability is rounded, not its complement independently. There are no separate trained pair

models. All global classes must occur, so an Inductive OvO pair always has both observed classes. The underlying binary calibrator still accepts single-class subsets.

One-vs-Rest (`method = "ovr"`) uses all calibration observations for each class. That class is the binary event; its class probability is the calibration probability, optionally rounded at fit time. The binary points are normalized across classes only after the chosen regularization.

Original probabilities are validated before rounding; an out-of-range input cannot become valid through rounding. Exact ties after rounding are combined. New probabilities supplied to `predict()` remain unrounded, and changing precision requires a new fit. Native R rounding can differ from NumPy at floating-point boundaries; NumPy rounding is not emulated.

Choose log-loss (default) or Brier regularization at prediction time without refitting. See `predict.venn_abers` for probabilities, classes, pair coupling or class normalization, and details. Manual Brier for both multiclass methods is a deliberate R extension over the examined Python 1.5.4 manual path, which supports log regularization and precision. Exactly two factor levels also select the multiclass matrix contract, with its distinct tie rule and Brier score scale.

Value

An S3 object of class `"venn_abers"`: a list holding the fitted calibration state, from which new probabilities are calibrated without refitting. For binary truth it records the sorted unique calibration probabilities with their lower and upper Venn-Abers values and the calibration and event counts; for factor truth, the class levels, the method and one binary calibrator per class pair (OvO) or class (OvR). These list elements are internal and not public API. Use the methods instead:

- `predict.venn_abers` returns the lower and upper values with the regularized event probability (binary), class probabilities (multiclass), class decisions or binary subproblem details, with the loss chosen at prediction time.
- `plot.venn_abers` draws the stored calibration map and returns it invisibly as a data frame.
- `print.venn_abers` summarizes the calibration size, events and probability knots, or the method, calibration size, classes and number of subproblems.

No predictive model or new data are stored.

See Also

`vignette("method", package = "vennabers")` for the theory and its guarantees, `vignette("manual-calibration", package = "vennabers")` for a worked introduction, and vennabers_examples for annotated installed scripts.

`predict.venn_abers`, `cross_venn_abers`, `venn_abers_regression`

Examples

```
calibrator <- venn_abers(
  truth = c(FALSE, TRUE, FALSE, TRUE),
  probability = c(0.1, 0.4, 0.35, 0.8)
)
calibrator

predict(calibrator, c(0.2, 0.5, 0.9))
rounded <- venn_abers(c(0, 1, 0, 1), c(.501, .502, .509, .510), precision = 2)
```

```
predict(rounded, c(.501, .505, .51), loss = "brier")

# Fixed-model probabilities on a separate calibration set.
classes <- c("cat", "dog", "bird")
truth <- factor(rep(classes, 2), levels = classes)
p <- rbind(c(.7, .2, .1), c(.2, .6, .2), c(.1, .2, .7),
           c(.5, .3, .2), c(.3, .5, .2), c(.2, .3, .5))
colnames(p) <- classes
fit <- venn_abers(truth, p, precision = 2)
new_p <- rbind(c(.6, .3, .1), c(.1, .4, .5))
colnames(new_p) <- classes
assessment <- factor(c("cat", "bird"), levels = classes)
details <- predict(fit, new_p, loss = "brier", type = "details")
details$pairs
predict(fit, new_p, loss = "brier", type = "class")
brier_score(assessment, details$prediction)
log_loss(assessment, details$prediction)
calibration_plot(assessment, details$prediction)

ovr <- venn_abers(truth, p, method = "ovr", precision = 2)
predict(ovr, new_p)
ovr_details <- predict(ovr, new_p, loss = "brier", type = "details")
ovr_details$classes
predict(ovr, new_p, loss = "brier", type = "class")
brier_score(assessment, ovr_details$prediction)
log_loss(assessment, ovr_details$prediction)
calibration_plot(assessment, ovr_details$prediction)

# Installed standalone examples for binary options (precision, loss, event
# selection, Cross) and both multiclass methods:
system.file("examples", c("binary-options.R", "inductive-ovo.R", "inductive-ovr.R"),
  package = "vennabers")
```

`venn_abers_details` *Inspect Calibrated Workflow Predictions*

Description

Return native predictions together with calibration bounds, fold identities and source-model diagnostics from a fitted trainable or frozen workflow. Prediction uses the retained models and calibrators without learning again.

Usage

```
venn_abers_details(object, new_data, ..., loss = NULL)
```

Arguments

`object`	A fitted workflow returned by `calibrated_workflow` and `generics::fit()`, or by `calibrate_workflow`.

new_data	A data frame with required raw predictors. The same blueprint, schema and inner recipe/engine checks apply as for predict().
...	Must be empty. Unknown, irrelevant and partially matched names are errors.
loss	For classification, omit to use the stored default, or supply "log" or "brier" to override it without refitting or changing stored state. Explicit NULL is invalid. Regression rejects explicit loss.

Value

venn_abers_details() returns a list with prediction (the same native probability table) and details: integer row, character fold ("Inductive1" for Inductive; the supplied fold IDs for Cross), character event_class, and double lower, upper, probability. Cross details contain one row per new observation and fold; their probability is the fold point, while prediction contains the final aggregate. Trainable workflow details also contain character prediction_model and model_source. The source is the fold ID in matched mode, or "Full1" for every fold in full mode. These are model metadata, separate from original class identities and the fold identifying the held-out calibrator; venn_abers_decode() preserves them without decoding them as classes. Trainable OvO/OvR details include model_training. OvO/OvR details additionally contain character method and either class_1/class_2 (OvO) or class (OvR). Each row identifies one new observation, fold and binary subproblem. event_class is the later class in an OvO pair or the considered OvR class. These binary point probabilities must first be merged across folds for Cross (Log/Brier), then coupled (OvO) or normalized (OvR) to reconstruct prediction. Regression returns the native .pred table as prediction, and details with integer row, character fold, and double prediction, lower, upper. Cross additionally returns aggregate with row, lower, upper: descriptive separate equal-weight means of fold bounds. Trainable regression aggregates also contain character prediction_model and a model_source list column: each row lists all contributing source model IDs (only "Full1" in full mode). Zero/one rows retain these schemas. Frozen regression outputs retain their existing schema.

Postprocessing diagnostics

For learned postprocessing, the additional postprocessing element records local data memberships, fitted mappings and fit counts. Pairwise OvO uses unique pair_N/VA_fold keys; ensemble prediction details retain inner-fold names and sizes in list-columns. See workflow_postprocessing for the complete record layout and method-specific diagnostics.

Interpreting bounds

Classification bounds are Venn-Abers probability intervals, not confidence intervals. Multiclass details describe binary subproblems, not multiclass intervals. Regression bounds concern regularized conditional expectations, not individual outcome coverage; Cross aggregate bounds are descriptive.

See Also

calibrated_workflow, calibrate_workflow, venn_abers_decode, venn_abers_tuning, and workflow_training for the fitted engine's stored state. For worked diagnostics, see vignette("workflows", package = "vennabers").

Examples

```
packages <- c("parsnip", "workflows", "recipes", "rsample", "hardhat",
              "generics", "rlang", "tibble", "tidyselect", "withr")
if (all(vapply(packages, requireNamespace, logical(1), quietly = TRUE))) {
  development <- data.frame(
    x = sin(1:80),
    truth = factor(rep(c("yes", "no", "no", "yes"), 20),
                   levels = c("yes", "no"))
  )
  base <- workflows::workflow() |>
    workflows::add_formula(truth ~ x) |>
    workflows::add_model(parsnip::logistic_reg(engine = "glm"))
  specification <- calibrated_workflow(base, outcome = "truth",
                                       method = "cross", v = 3, seed = 42)
  fitted <- generics::fit(specification, development)
  new_data <- data.frame(x = c(-0.5, 0.5))

  # The final probabilities, then one record per new row and Cross fold.
  result <- venn_abers_details(fitted, new_data)
  print(result$prediction)
  result$details
}
```

`venn_abers_one_hot` *One-Hot Matrices of Class Decisions*

Description

Represent existing class decisions as a one-hot matrix with one column per class. The helper converts the decision a predictor has already made; it never chooses a class from probabilities.

Usage

```
venn_abers_one_hot(x)
```

Arguments

`x`	A class decision: the factor returned by multiclass `predict(..., type = "class")` or by `venn_abers_decode()$class`; the logical vector returned by binary `predict(..., type = "class")`; or a data frame with a factor `.pred_class` column, as returned by workflow, Grid, parsnip and frozen `predict(..., type = "class")`. Other columns of a data frame are ignored.

Details

The ones follow the existing tie rules: a binary probability of exactly 0.5 is the event, and a multiclass tie keeps the first stored level, also with exactly two classes. The Python reference facade breaks a binary tie towards the first sorted class instead.

Ordered factors only set the column order. Schema-encoded `.pred_class` columns keep their encoded classes as column names; pass `venn_abers_decode(prediction, schema)$class` to obtain original class names. Probability matrices or tables, numeric or character vectors, data frames without a factor `.pred_class`, and factors without levels are errors.

Value

A base `matrix` of type double with values 0 and 1, one row per element of `x` in the same order and no row names. Column names are the factor levels in their stored order, or `"FALSE"` and `"TRUE"` for a logical vector, so the binary event is the `TRUE` column. Every class keeps its column, also when no row predicts it. Zero rows give a 0 x K matrix and one row a 1 x K matrix; the result is never reduced to a vector. A missing decision gives a row of `NA`.

See Also

`predict.venn_abers`, `calibrated_workflow`, `venn_abers_decode`

Examples

```
source(system.file("examples", "one-hot-classes.R", package = "vennabers"))
```

---

venn_abers_regression *Moderated Inductive Venn-Abers Regression*

Description

Calibrates the numeric predictions of an already fitted regression model with an Inductive Venn-Abers regressor (IVAR), returning a calibrated point prediction between a lower and an upper bound. It needs the observed outcomes and the model's predictions for a separate calibration set; the caller trains and retains the model and chooses all data roles. Choose it when one held-out calibration set is affordable; use `cross_venn_abers_regression` to calibrate with retained fold models, or `calibrated_workflow` to let the package fit the model and split the data.

Usage

```
venn_abers_regression(truth, prediction, ..., m = 1, epsilon = NULL)

## S3 method for class 'venn_abers_regression'
predict(object, newdata, ...)
```

Arguments

`truth`	A nonempty numeric vector of finite real-valued outcomes.
`prediction`	A numeric vector of finite uncalibrated regression predictions, the same length as `truth`, from an independently fixed model.
`m`	One finite numeric whole number satisfying $1 \leq m \leq \lfloor (k-1)/2 \rfloor$, for calibration size k. The default is 1, requiring at least three observations.

epsilon	Alternatively, one finite numeric scalar strictly between 0 and 1. Computes $m = \lfloor \epsilon(k+1)/2 \rfloor$, then checks the same domain. Supplying both m and epsilon explicitly is an error; omitting m permits epsilon alone. Explicit NULL is invalid.
object	A fitted venn_abers_regression object.
newdata	A numeric vector of finite new predictions from the same fixed model. Zero or one observation is allowed.
...	Must be empty. Unknown options are errors.

Details

The calibrator applies asymmetric moderation and isotonic regression to the supplied real-valued predictions, then combines the resulting bounds into a point estimate.

All vectors align by position, regardless of names. Negative values and values outside $[0, 1]$ are valid; numeric 0/1 truth remains regression. Factors, logical values, matrices, lists, classed vectors, missing/nonfinite values, length mismatches and invalid options are rejected. Inputs are never rounded, clipped or silently removed. Tied predictions and new predictions beyond the calibration prediction range are supported.

For sorted outcomes $y_{(1)}, \ldots, y_{(k)}$, set $a = y_{(m)}$ and $b = y_{(k-m+1)}$. The lower calibration replaces outcomes below a with a and those above $y_{(k-m)}$ with that value. The upper calibration replaces outcomes below $y_{(m+1)}$ with that value and those above b with b. Every observation is retained. Lower and upper isotonic endpoints correspond to inserting the new score with outcome a and b, respectively; tied scores are pooled.

For endpoints L, U, use weights $w_L = L - a$, $w_U = b - U$ and point $(w_L L + w_U U)/(w_L + w_U)$. Only when both weights are exactly zero is the midpoint used, including for nonconstant outcomes. Small positive weights use the weighted merger. Constant outcomes and coincident endpoints return their common finite value. Arithmetic is scaled to avoid avoidable overflow.

The theoretical setting assumes IID observations and a base model determined independently of calibration observations, including preprocessing and tuning. Assessment outcomes must be excluded from both model training and calibration. The bounds concern a regularized conditional expectation; they are not confidence or prediction intervals for individual future outcomes. No individual coverage, universal improvement in RMSE/MAE, or general selector guarantee for this particular point merger is asserted.

This manual R entry point is an R extension, not a public manual Python wrapper API. Offline fixtures use valid regression components from Python venn-abers 1.5.4, commit 0e7b4bc575ce230134576ba1c76688d46bf99a
Reference lower-bound defects are checked with reflected upper components and independent isotonic calculations; undefined reference point results use the explicit contract above. See 'tests/testthat/extdata/regress
in the source package for provenance. Package-managed training, automatic splits, new regression score/plot functions, exact minimax, observation weights and rounding options are not provided.

Value

Fit returns an object of class venn_abers_regression. Its documented metadata are observations, requested (an empty list for the implicit default, otherwise a named list containing m or epsilon), m (effective integer-valued moderation), epsilon (effective level $2m/(k+1)$), and a, b (outer bounds in outcome units). Other fields are internal. The fit can be saved with saveRDS() and restored with readRDS() in a fresh R process with this package loaded.

Predict returns a data frame with double columns `prediction`, `lower`, `upper`, in input order, including for zero or one row. All results are finite and the point lies between the bounds up to numerical roundoff. Prediction only looks up fitted endpoints and merges them; it trains nothing. `plot.venn_abers_regression` draws the stored calibration map and returns it invisibly as a data frame.

See Also

`vignette("method", package = "vennabers")` for the theory and its limits, `vignette("manual-calibration", package = "vennabers")` for a worked introduction, and vennabers_examples for annotated installed scripts.

`cross_venn_abers_regression`

Examples

```
# The installed example fixes disjoint training/calibration/assessment roles,
# fits lm() only on training data, and reports Base R RMSE and MAE.
source(system.file("examples", "inductive-regression.R", package = "vennabers"))

fit <- venn_abers_regression(c(-10, 0, 2, 6, 20), rep(0, 5))
fit$requested
fit$m
fit$epsilon
predict(fit, c(-1, 0, 1))
```

---

venn_abers_schema *Reversible Names for Calibrated Workflows*

Description

Explicitly encode input columns and outcome classes before constructing a base workflow, and decode native results into separate roles. These helpers learn no preprocessing, statistics, or calibration.

Usage

```
venn_abers_schema(data, outcome, classification = "binary", event_level = "first")
venn_abers_encode(data, schema)
venn_abers_decode(x, schema)
```

Arguments

`data`	A data frame containing exactly the selected input columns. At schema construction the outcome must be a factor with exactly two declared levels for `"binary"`, at least two for `"ovo"`/`"ovr"`, or a finite numeric vector for regression (including numeric 0/1). Encoding accepts new data without the outcome, but requires every predictor.
`outcome`	One exact original outcome name, including reserved or nonsyntactic names.

classification	Exactly "binary", "ovo", or "ovr" for factor outcomes. Must be omitted for numeric regression outcomes.
event_level	Exactly "first" or "second", selecting the event from the original factor levels in their stored order. Only binary classification accepts an explicit event option; omit it for OvO/OvR and regression.
schema	The fixed object returned by venn_abers_schema(). Pass the same schema to encoding, calibrated_workflow() or calibrate_workflow, and decoding.
x	A native probability, class or numeric prediction data frame, an unsummarized tune::collect_predictions() table, or a venn_abers_details() result.

Details

Select the intended columns before creating the schema. All selected names and all declared outcome levels get deterministic, collision-free technical codes. Mapping uses exact names, never name repair, Unicode normalization, case folding, or frequency sorting. Even original names that resemble codes remain distinct. Empty, missing or duplicate names, unknown columns or classes, missing predictors, and damaged or incomplete schema mappings are errors. The schema is fixed across folds, prediction batches and processes; do not edit it after construction.

Build formulas, recipes, variable selections, and outer resamples explicitly on the encoded data. Use schema$encoded_outcome in the workflow constructor. Existing recipes are not rewritten. To retain prescribed outer memberships, rebuild their splits on the encoded data with the same row positions and IDs. Inner plan functions receive local encoded data; the exact local plan/data comparison and all training, calibration and assessment separation rules remain.

For a frozen workflow originally trained on encoded data, pass that original schema to calibrate_workflow(), with encoded calibration data and schema$encoded_outcome. Encode new prediction data with the same schema. The stored workflow is never renamed or refitted. Without encoded training, keep original input names and schema = NULL; a new schema cannot repair an old workflow's failed complete prediction path. See examples/frozen-schema.R for the four frozen Inductive methods.

Native predictions remain encoded. Separate decoded roles avoid collisions between an original class class, outcome .pred_class, and external id/id2. The decoder does not flatten or repair these names. Probability columns must cover every schema class exactly once; named columns are reordered to schema order. Unknown encoded output names or class codes fail.

Encoded data, native predictions, and details carry a vennabers_schema attribute. Existing foreign bindings are rejected. Ordinary table operations or collectors may drop attributes: detached tables therefore require the actually associated schema supplied by the caller. Matching codes alone cannot establish provenance. Decode unsummarized predictions to preserve repetitions; the decoder cannot recover rows already summarized or dropped. Collect notes and unsummarized metrics separately, audit completeness, and explicitly extract fitted states when inner models or diagnostics are needed. Match control_resamples()'s event_level to binary schemas; no control is silently changed.

Schema helpers use base R and do not load optional integration packages. The schema and fitted pure R workflow state survive RDS restoration in a fresh process with the required engine namespaces loaded. Outer workers require pkgs = c("vennabers", "stats") for GLM/LM examples (add "nnet" for the installed multinomial example), parallel_over = "resamples", and consistent seeds and thread settings.

Value

`venn_abers_schema()` returns a `vennabers_schema` metadata list. `version` is the format version; `outcome` and `encoded_outcome` are the original and technical target names. `columns` maps original column names to codes, in `original_columns` order. `classes` maps original class names to codes, in `class_levels` order. `ordered` records whether the original target is ordered. `mode`, `classification`, `event_level`, and original `event_class` define the task. Multiclass schemas have NULL event fields; regression has NULL class levels, classification and event fields, and an empty class map. No observation values or additional calibration data are retained in the schema.

`venn_abers_encode()` returns a data frame in schema column order with encoded column names and, when present, encoded target levels. Predictor values, types, attributes, factor levels, row order, and row names are preserved. The target is aligned to the saved class order even if incoming levels are reordered or known levels are absent. Missing factor target values stay missing; numeric targets must be finite; the fitting boundary rejects missing truth.

`venn_abers_decode()` returns a list with `outcome` (the original outcome name as metadata), and only the result roles available in `x`: `probability` (a numeric matrix with original class columns in schema order), `class` (an original-level factor), `truth` (an original-level factor or numeric regression truth), `prediction` (numeric regression points), `identity` (available `.row`, `id`, `id2`, further numbered `id` columns, and `.config`), `details` (existing binary, multiclass or regression diagnostics), and `aggregate` (existing Cross regression mean bounds). Multiclass details decode `class_1`/`class_2` for OvO, `class` for OvR, and `event_class`; fold, row and subproblem identities remain unchanged. It preserves numbers, dimensions, row order, and identities, including zero and one row. No class is inferred from probabilities, and no truth or details are inferred from a prediction table. Regression never adds class or probability roles. Multiclass ties favor the first maximum in the stored class order, also with two classes. The decoder preserves decisions already made by the predictor; it does not recompute them.

See Also

`calibrated_workflow`, `local_calibration_splits`

Examples

```
source(system.file("examples", "encoded-binary.R", package = "vennabers"))
# Shared OvO/OvR and regression with Inductive and Cross calibration:
script <- system.file("examples", "encoded-multiclass-regression.R", package = "vennabers")
# source(script)
```

`venn_abers_tuning` *Inspect Classification or Regression Grid or Bayes Selection*

Description

Returns the stored configuration and selection diagnostics without refitting. This page also documents the `tuning` configuration passed to `calibrated_workflow`. For a worked progression from Grid to Bayes and outer assessment, see `vignette("tuning", package = "vennabers")`.

Usage

```
venn_abers_tuning(object)
```

Arguments

object	A fitted binary, shared multiclass, pairwise OvO, or regression calibrated_workflow() with grid or Bayes tuning.

Value

A list with config, calibration, models, seed, class/schema and method metadata, final fit roles and model source mappings. Each search records candidate IDs and grid values, eligibility and exclusion reasons, per-resample main and report scores, native tune_grid() or tune_bayes() results (including prediction rows and notes), the stable candidate-order tie break, selected values, local plan positions, gross base-fit budget estimate, attempted evaluation count and observed base-fit attempts, including failures. Bayes diagnostics additionally retain each public one-step continuation, its stop reason, elapsed time, and the global no-improvement count. The estimate is unavailable when a provider function determines a nested plan size at fit time; its count factors remain visible. In Bayes candidate tables, parameter IDs that collide with status, estimate or reason receive unique numeric suffixes separated by an underscore (for example, status_1). Selected values and native results keep the original parameter IDs. The ID id itself is unsupported because native tune uses it for resample identities. Calibration selection also records the disjoint selection (S) and final fit (F) row positions. In full-model mode, models$Full1 is its separate local search. With postprocessing, each models entry is a list of local B searches named by inner fold, with fold_id, inner_fold, B positions and search_seed (one draw per B from the fit stream, so each search has its own RNG stream); budgets report inner_pairs. For regression calibration searches, saved native predictions also carry the requested m or epsilon value (va_m or va_epsilon when the outcome itself is named m or epsilon). Actual candidate values, including locally generated grids, are stored separately in grid. Effective per-fold moderation is recorded in fit_roles[[fold]]$moderation; epsilon may imply different integer m values on different fold sizes. These are moderation parameters, not confidence or prediction interval levels.

Tuning configuration

NULL keeps fixed settings. For binary, shared OvO/OvR or pairwise OvO classification and regression, a named list with strategy = "joint" or "two_stage" and model and/or calibration searches matching the present tune() placeholders. Each search has method = "grid" or "bayes". Grid uses a concrete grid data frame with columns named by placeholder IDs or a positive integer grid size with dials param_info. Bayes uses positive whole initial and iter budgets plus param_info; initial must exceed the number of tuned parameters. Its control_bayes() must save predictions, use uncertain = Inf, and supplies the seed, no_improve, and optional time limit. Local model searches and calibration searches require resamples = function(data) ... (or a bound rsample rset); pairwise joint model grids use the global plan described below. Optional primary_metric defaults to yardstick::mn_log_loss for classification or yardstick::rmse for regression; report_metrics are independent yardstick metrics. Explicit control must be the matching tune::control_grid() or tune::control_bayes() with save_pred=TRUE. Bayes calls the public tune::tune_bayes() one iteration at a time, filtering incomplete candidates before the next call. Failed, dropped and repeated proposals spend one step of the iter budget and count toward no_improve; the time limit is checked between complete steps. Too few complete initial

candidates stop the fit with their reasons. Inner parallelism is disabled, and a custom `extract` callback is unavailable because selection uses extraction for diagnostics. Model/recipe and calibration IDs must be unique. The ID `id` is unsupported because native tune uses it for resample identities. Calibration search requires exactly one of `selection_prop` (the proportion reserved for selection S) or `selection_split` (rsplit or function, with analysis=final fit F and assessment=selection S). S is never included in F. Automatic calibration selection requires `splits=NULL` or a local plan provider function. In pairwise OvO `two_stage`, model searches and resamples run on each local class pair. Calibration selection scores the coupled complete Multiclass output on S; its final fit uses only F. Pair plans must be local provider functions when calibration is tuned. For pairwise `joint` Grid model tuning, candidate choices form a Cartesian grid over class pairs (and separate full-model choices when requested). Bayes model searches instead advance independently on each local pair and source/full domain; they require local `model$resamples`. Global calibration candidates use complete coupled predictions. Model-only joint Bayes evaluates the complete locally selected procedure on `joint_resamples`; model-only joint Grid uses that plan to select its Cartesian candidates. Omit `model$resamples` only for the Cartesian Grid route. Global plans are never used as pair plans. A joint grid above 4096 combinations is rejected before fitting; reduce the grids. With `postprocessing`, pairwise `joint` Grid follows the Bayes route: each local pair B searches itself and requires `model$resamples`; there is no Cartesian grid. Loss candidates are `"log"`/`"brier"`; precision candidates are `"none"`, `"0"`, `"2"`, and `"-2"`. Regression searches exactly one of `m = tune::tune("moderation")` or `epsilon = tune::tune("moderation")`; classification options are rejected. Every inner calibration fold must satisfy `1 <= m <= floor((k-1)/2)`, with `m = floor(epsilon*(k+1)/2)` when epsilon is supplied. Invalid local candidates are excluded without clipping. Joint model selection scores complete IVAR/CVAR predictions; two-stage selection scores the uncalibrated numeric predictions first. Fixed model/recipe or moderation options stay fixed. Each Cross source and Full1 searches only its own training domain; Full1 never averages fold choices. The return value above describes selections and diagnostics.

Tuned postprocessing

With `postprocessing`, every engine/recipe search runs inside its own B: each VA source Q, each inner ensemble pair and each B of a full F composition, per class pair for pairwise OvO. The search resamples, and for `joint` the VA and B/P splits of every candidate composition, are formed within that B. Only then are the chosen settings fitted on B and the postprocessor on its P; P, C, outer assessment A and a reserved selection share S never enter a search, so mutating P leaves its search, recipe and engine unchanged. Inner pairs, VA folds and F may choose different settings; none are averaged or copied.

`joint` scores complete calibrated compositions with fixed candidate settings (no nested search). The first `two_stage` step scores the raw engine/recipe without postprocessor or VA; calibration options are then chosen on S as before, and S never returns to final B/P fits. A candidate is eligible only with valid finite main scores on every expected resample, inner pair and row; a failed final composition fit aborts without a fallback candidate.

A static `postprocessing$splits` plan cannot bind to candidate data, so joint model selection and calibration selection require a plan function or prop/v. `venn_abers_tuning(object)$models[[fold]]` (and, for pairwise, `$models[[pair]][[fold]]`) is a list of local searches named by inner fold (`Holdout1` or the ensemble fold IDs); each records `fold_id`, `inner_fold`, its B positions and the usual search diagnostics. Each postprocessing record also stores its `selected` values. Budgets multiply by the inner pair count J when it is known in advance; `cost$observed_base_fits` counts every search and final B fit. See installed `examples/postprocessing-grid.R`.

Bayes searches follow the same B-local rule. Only candidates complete over every resample, inner pair and row with a valid main score enter the next GP; a failed inner model/postprocessor pair, a missing row or an invalid main score excludes the whole candidate. Excluded candidates, including proposals that `tune::tune_bayes()` drops after failing on every resample, keep their global candidate IDs and reasons; `steps`, `notes`, `stop_reason`, `attempts`, `bayes_seed` and `search_seed` record each local search. There is no switch to Grid or another method. See installed `examples/postprocessing-bayes.R`.

Related training rules

See workflow_training for local split validation and full-model transfer, and workflow_postprocessing for B/P data roles, methods and compositions. These choices remain fixed while the configured parameters are searched.

See Also

vennabers_examples for annotated installed scripts.

`calibrated_workflow`. Installed Grid examples `binary-grid.R`, `shared-multiclass-grid.R`, `pairwise-grid.R`, `regression-grid.R` and `postprocessing-grid.R`, and Bayes examples `shared-multiclass-bayes.R`, `pairwise-bayes.R`, `regression-bayes.R` and `postprocessing-bayes.R`, are under `system.file("examples", package = "vennabers")`.

Examples

```
packages <- c("parsnip", "workflows", "recipes", "rsample", "hardhat",
              "generics", "rlang", "tibble", "tidyselect", "withr",
              "tune", "dials", "yardstick")
if (all(vapply(packages, requireNamespace, logical(1), quietly = TRUE))) {
  development <- data.frame(
    x = sin(1:80),
    truth = factor(rep(c("yes", "no", "no", "yes"), 20),
                   levels = c("yes", "no"))
  )
  base <- workflows::workflow() |>
    workflows::add_formula(truth ~ x) |>
    workflows::add_model(parsnip::logistic_reg(engine = "glm"))

  # Choose the regularization loss on a reserved selection share S; the
  # final calibrator is fitted on the remaining rows F.
  calibration <- list(method = "grid",
                      grid = data.frame(va_loss = c("log", "brier")),
                      resamples = function(d) rsample::vfold_cv(d, v = 2))
  tuning <- list(strategy = "two_stage", selection_prop = 0.3,
                 calibration = calibration)
  specification <- calibrated_workflow(base, outcome = "truth", seed = 42,
                                       loss = tune::tune("va_loss"),
                                       tuning = tuning)
  fitted <- generics::fit(specification, development)

  choices <- venn_abers_tuning(fitted)
  print(choices$calibration$candidates)
```

```
  print(choices$calibration$selected)
  # Selection rows never enter the final calibration fit.
  length(intersect(choices$calibration$selection_rows,
                   choices$calibration$final_rows))
}
```

---

vennabers_examples *Find and Run Venn-Abers Example Scripts*

Description

Annotated example collections cover manual calibration with its fit diagnoses and assessment graphics, training and frozen workflows, learned postprocessing, and Grid/Bayes selection. Each collection has a contents list with a separate entry for every example script.

Details

Open the collections with `vignette()` or list all guides and examples with `browseVignettes("vennabers")`. Vignettes are available after installing a built source package. The original scripts are installed under `system.file("examples", package = "vennabers")` even when installing directly from a checkout without building vignettes.

The articles display the complete scripts without running their fits during documentation builds. Use `file.show()` to read an installed script and `source()` to run it, including its checks. Workflow, postprocessing and tuning scripts use optional packages named in their code and DESCRIPTION; install them explicitly before running those examples. Resampling, tuning and composition matrices can take longer than the introductory guides.

manual calibration

`vignette("examples-manual", package = "vennabers")`.

Scripts: ‘`binary-options.R`’, ‘`cross-venn-abers.R`’, ‘`inductive-ovo.R`’, ‘`inductive-ovr.R`’, ‘`cross-multiclass.R`’, ‘`one-hot-classes.R`’, ‘`inductive-regression.R`’, ‘`cross-regression.R`’, ‘`calibration-graphics.R`’.

training and frozen workflows

`vignette("examples-workflows", package = "vennabers")`.

Scripts: ‘`calibrated-workflow.R`’, ‘`encoded-binary.R`’, ‘`encoded-multiclass-regression.R`’, ‘`reference-splits.R`’, ‘`inductive-sizes.R`’, ‘`model-adapter.R`’, ‘`engine-options.R`’, ‘`full-model-workflow.R`’, ‘`full-multiclass-workflow.R`’, ‘`full-regression-workflow.R`’, ‘`pairwise-inductive.R`’, ‘`pairwise-cross.R`’, ‘`frozen-binary.R`’, ‘`frozen-multiclass-regression.R`’, ‘`frozen-schema.R`’.

learned postprocessing

`vignette("examples-postprocessing", package = "vennabers")`.

Scripts: ‘`sigmoid-holdout.R`’, ‘`tailor-holdout.R`’, ‘`tailor-multiclass-holdout.R`’, ‘`tailor-regression-holdout`’ ‘`isotonic-regression-holdout.R`’, ‘`binary-ensemble.R`’, ‘`multiclass-ensemble.R`’, ‘`regression-ensemble.R`’, ‘`pairwise-postprocessing.R`’, ‘`full-postprocessing.R`’.

Grid and Bayes selection

`vignette("examples-tuning", package = "vennabers")`.

Scripts: 'binary-grid.R', 'shared-multiclass-grid.R', 'shared-multiclass-bayes.R', 'pairwise-grid.R', 'pairwise-bayes.R', 'regression-grid.R', 'regression-bayes.R', 'postprocessing-grid.R', 'postprocessing-bayes.R'.

See Also

vennabers-package, venn_abers, calibrated_workflow, calibrate_workflow, workflow_postprocessing, venn_abers_tuning.

Examples

```
# Locate an installed script without running its fits:
script <- system.file("examples", "binary-options.R", package = "vennabers")
script
# file.show(script)
# source(script)
```

---

workflow_postprocessing

Postprocessing Before Workflow Calibration

Description

Configuration and reference rules for the `postprocessing` argument of `calibrated_workflow`. Start with the worked Sigmoid example in `vignette("workflows", package = "vennabers")` before using this reference. These options learn a postprocessor before the Venn-Abers calibrator.

Data roles

Each VA source training portion Q is split into B for recipe and engine learning and P for postprocessor learning. Held-out C fits the VA calibrator. Outer assessment A is excluded from all learning. When calibration is tuned, selection rows S are excluded from the final fit domain F. With `prediction_model = "full"`, F gets its own complete B/P composition. An inner pair is a model/postprocessor pair; an OvO class pair is a separate two-class training domain.

Configuration

NULL preserves the existing unprocessed training path and does not remove a supplied postprocessor. Otherwise a named list with explicit `method = "tailor_logistic"`, `"tailor_multinomial"`, `"tailor_linear"`, `"sigmoid"`, or `"isotonic_numeric"`, `strategy = "holdout"`, and either `prop` (the P fraction, strictly between zero and one) or `splits` (a local rsplit or serializable function of local Q data). Analysis is B; assessment is P. A supplied plan cannot be combined with prop or v. Unknown, duplicate, irrelevant and conflicting fields are errors. Currently binary, shared OvO/OvR and pairwise OvO Inductive/Cross VA and regression IVAR/CVAR with matched or full models are implemented, with fixed engine/recipe settings or local selection (see venn_abers_tuning);

Binary and pairwise OvO `tailor_logistic/sigmoid`, shared OvO/OvR `tailor_logistic` (two classes)/`tailor_multinomial` (three or more), and regression `tailor_linear/isotonic_numeric` support `strategy = "ensemble"`, with an explicit integer `v >= 2` or `splits` (a local unrepeated rset or serializable function of Q). Ensembles reject prop; explicit splits reject v. Method, Sigmoid input, strategy, prop/v and prepared plans stay fixed per run.

Supported compositions and costs

The postprocessing scope is a finite matrix of 80 fixed compositions: method x holdout/inner ensemble x Inductive/Cross x matched/full over binary (Sigmoid, Tailor logistic), pairwise OvO (Sigmoid, Tailor logistic), shared OvO/OvR with three or more classes (Tailor multinomial), shared two-class OvO/OvR (Tailor logistic) and regression (reference isotonic, Tailor linear). Every composition also accepts joint or two-stage Grid/Bayes selection of engine/recipe settings inside each B. B trains recipe and engine, P the postprocessor, C the unchanged VA calibrator; F is the permitted final fit data of a full composition. Without tuning, J inner pairs cost J (Inductive matched), 2J (Inductive full), KJ (Cross matched with K folds) or (K+1)J (Cross full) engine and postprocessor fits per domain, retaining J, J, KJ or J pairs; class pairs add up. Out of scope: out-of-fold postprocessing followed by a complete engine refit, foreign engines beyond `model_adapter` callbacks, Tailor step sequences, further postprocessing methods, and other new public constructors. Frozen workflows (`calibrate_workflow`) keep their existing contract.

Binary Sigmoid holdouts

For binary matched or full Inductive/Cross workflows with fixed parameters, set `postprocessing = list(method = "sigmoid", strategy = "holdout", prop = .3, sigmoid_input = "probability")` or explicitly select `sigmoid_input = "glm_link"`. The base workflow must have no postprocessor. Recipe and engine learn only on B; Sigmoid learns only on P; the unchanged VA core learns on held-out C. Both B and P must contain both factor levels, and P needs at least two rows. Constant P inputs are valid; coefficients are not identifiable in that case.

`probability` uses the final named probability of the second stored factor level from the complete public base workflow. `glm_link` requires a `logistic_reg(engine = "glm")` binomial model with logit link and calls `predict(workflow, new_data, type = "raw", opts = list(type = "link"))`, including recipe processing. It never reconstructs links from probabilities. Both VA event choices are supported by named probability selection after Sigmoid.

The mapping minimizes unregularized logistic loss with targets $(N_1 + 1)/(N_1 + 2)$ for the second class and $1/(N_0 + 2)$ for the first. It follows scikit-learn's Platt calibration, including its start values, L-BFGS-B tolerances (relative loss reduction of 64 machine epsilons, gradient 1e-6) and SciPy's ten correction pairs. These tolerances stop short of the exact optimum, so the optimizer path determines the result. Stored `coefficients` are `a, b` on the stored input `scale`: the second-class output is `plogis(-(a * (x / scale) + b))`. The scale is the maximum absolute P input when at least 30, otherwise one. Constant inputs use the exact zero-slope solution (the mean target). This is an R choice, not Python parity: scikit-learn's iterative estimate stops within its gradient tolerance and, for very small P, can differ from it by more than 1e-7. A line-search stop is kept only when its mean gradient per P row is at most 1e-6 in each coefficient, as at a machine-precision optimum. Nonfinite or incomplete inputs, every other unsuccessful optimization, and incomplete probability outputs fail with method, role and local fold context.

The fitted state retains the complete base workflow, mapping and package versions. Details expose B/P/C memberships, `input`, `sigmoid_class`, scale, coefficients, convergence, warnings and fit

counts. Empty batches retain their schema without engine calls. No Tailor, probably or Python installation is required for Sigmoid. Python is used only by the separate checked-in reference generator; regular tests read its fixed CSV outputs. See `system.file("examples", "sigmoid-holdout.R",package = "vennabers")` for both inputs and native outer evaluation.

Tailor classification holdouts

Each VA source training portion Q is partitioned into disjoint B and P. The recipe and engine learn only on B through the public workflow fit. A package-owned composition validates its P predictions and fits an unchanged canonical Tailor through its public fit, using `tailor::adjust_probability_calibration(method = "logistic", smooth = FALSE)` for two classes, or `method = "multinomial", smooth = FALSE` for three or more classes. The requested method must match the class count; no automatic exchange is performed. Shared two-class OvO/OvR retains multiclass class and tie rules, not the binary event contract. One exactly matching untrained Tailor adjustment may already be configured; it is adopted once. Foreign, trained or multiple adjustments are rejected. The stored composition predicts held-out VA data C and new predictors through the base workflow, input validation and the public Tailor prediction path. The VA core, event alignment, loss and rounding rules are unchanged.

Both B and P must contain all K model classes. P needs at least K+1 rows and a full-rank design: the logistic intercept and one probability for two classes, or the multinomial intercept and K-1 probabilities for K classes. Rank and convergence are checked on the actual learned model after the necessary B predictions. Failed fits, absent learned state and method switches to none are errors. Tailor multinomial uses the `nnet::multinom()` defaults of **probably** (at most 100 iterations), and the fixed Tailor arguments offer no iteration setting: weakly informative class probabilities can stop there unconverged even when the B engine converges. Setting `maxit` on the B engine does not change the P postprocessor's limit. The composition fails with fold, inner-pair and B/P context, without retry, method switch or partial ensemble. This is an accepted limit of the 1.x contract, not a pending option. This R composition limit does not change numerical VA parity on identical valid inputs. Warnings from successful fits are retained with source/fold context. The existing VA rules for missing C classes remain unchanged (including valid one-class Cross subproblems); B/P requirements do not apply to C. Use `venn_abers_schema()` for nonsyntactic outcome or class names that the underlying Tailor/probably formula interface cannot use.

Tailor linear holdouts

Linear regression postprocessing requires `method = "tailor_linear"` explicitly and uses `tailor::adjust_numeric_calib = "linear", smooth = FALSE)` through the same package-owned composition. Recipe/engine learning uses B; the intercept and slope are learned only on P. Numeric 0/1 targets remain regression. Automatic B/P splits are unstratified. P needs at least three rows, finite predictions and truth, full actual design rank two, finite coefficients and convergence; constant predictions, absent fits and switches to none fail. The complete composition's numeric output enters the unchanged IVAR/CVAR core with the requested valid `m` or `epsilon`. CVAR averages fold points and both bounds separately. No confidence or prediction intervals are added. See installed `examples/tailor-regression-holdout.R` for both paths, encoded names, complete native evaluation and data-role checks.

Numeric isotonic holdouts

`method = "isotonic_numeric"` accepts a regression base workflow without a postprocessor. The recipe and engine learn on B, and unweighted increasing least-squares isotonic regression learns on finite P predictions and truths. At least two P rows are needed; tied predictions retain their

frequencies. Fitted support values are linearly interpolated, extended constantly beyond the support, and are not restricted to [0,1]. A single support predicts the mean P truth everywhere. No full-rank or Ridge condition applies. The complete saved B workflow and isotonic fit enter the same IVAR/CVAR cores. See installed `examples/isotonic-regression-holdout.R` for both native evaluation paths.

Local plans and seeds

All source B/P plans are validated before any source is trained. Static plans must match local selected columns in order, values, types, levels and row order. Automatic classification splits use the local outcome with `pool = 0`, without retries or size adjustments; ordered valid explicit holdouts are allowed. The enclosing seed scope covers VA splitting, B/P splitting and fitting in stored fold order; NULL consumes the caller's stream and an explicit seed restores it.

Inner ensembles

For inner ensembles, P folds partition Q exactly once and each B is its own P complement. B/P disjointness is local to a pair: a P row can belong to another pair's B. Gaps, overlap, bootstrap duplicates, repeated folds and incorrect local data bindings fail before learning. Each pair uses the same complete holdout fitting rules. Aligned final probability vectors are averaged with equal pair weights before VA, without inner VA calibrators or size weights. Shared OvO/OvR averages complete class vectors in stored class order before binary VA aggregation and subsequent OvO coupling or OvR normalization. Regression averages finished numeric predictions from the retained complete model/postprocessor pairs before IVAR/CVAR. It does not average coefficients, isotonic supports or separately VA-calibrated inner results. Each pair retains its method's P-size and rank rules, including valid constant isotonic inputs. CVAR's outer fold bounds remain descriptive VA bounds, not confidence or prediction intervals. See installed `examples/regression-ensemble.R` for both numeric methods, native evaluation, schema and held-out data checks. All plans are resolved first, followed by fits in stored VA/inner-fold order in one RNG stream; the seed is never restarted for each inner B/P pair. Local plan functions must be pure functions of their supplied data, not call counters or global row positions. Different Q domains may have different inner fold counts.

Pairwise OvO compositions

For pairwise OvO, restrict the data and outcome levels to each class pair before resolving any VA or B/P plan. Each pair supports `tailor_logistic` or `sigmoid` with either explicit input type; no foreign class reaches its recipe, model, postprocessor or plan function. Pair-local complete predictions are averaged before the binary VA core, and only the resulting pair points enter global OvO coupling. Different pairs and VA folds can have different local sizes and inner fold counts. Explicit B/P plans must bind to their own Q, using a pure local function or `local_calibration_splits()` for multiple prepared domains. Global position plans are rejected. Existing isolated VA pair-plan seeds remain unchanged; all B/P plans are then resolved in class-pair/VA-fold order, followed by fits in class-pair/VA-fold/inner-fold order in the enclosing RNG stream.

Pairwise diagnostics are stored at `extract_fit_engine(fitted)$pair_models[[pair_id]]$postprocessing`. `venn_abers_details()$postprocessing` exposes these records under unique `pair_N/VA_fold` keys. Each record includes `pair_id` and `pair_rows` (positions in the enclosing fit data); Q/B/P/C positions are relative to that class pair. Inner-fold names identify model/postprocessor pairs, not class pairs. The decoder preserves the two-class outcome domain and translates its names. Costs sum the actual inner counts across every class pair and VA fold; all required paths remain stored.

Missing members and failed fits/predictions abort the complete result with pair/fold context. See installed `examples/pairwise-postprocessing.R` for native resampling and last_fit.

Shared diagnostics and serialization

For shared models, `extract_fit_engine(fitted)$postprocessing` and the `postprocessing` element of `venn_abers_details()` store actual Q/B/P/C row memberships, Q-local analysis/assessment positions, sizes, class/event and schema bindings, RNG settings, method-specific fit state, warnings and observed fit counts. Prediction details add the inner fold, method, input and B/P sizes. Package versions are stored in the fitted engine's `versions`. Each nonempty prediction calls each retained base workflow and Tailor once; empty batches call none. Predict needs no truth or plan resolution and never learns. RDS retains the full trained state. Ensemble records contain a named `inner_folds` list of complete pair diagnostics and summed observed fit counts. Their prediction details use list-columns for `inner_fold`, `B_size` and `P_size`, retaining one VA result row and the ordered identities/sizes of every contributing pair. An Inductive ensemble fits J engines and J postprocessors; a K-fold Cross ensemble with J pairs per fold fits K*J of each. Missing or failed pairs abort the whole composition with fold context. See installed `examples/binary-ensemble.R` and `examples/multiclass-ensemble.R` for complete native evaluation. Tailor and probably are optional and required only for this path. See installed `examples/tailor-holdout.R` for native fit, complete repeated outer resampling, last_fit, diagnostics and checks of held-out data roles. `examples/tailor-multiclass-holdout.R` adds shared two-, three- and four-class OvO/OvR, encoded names and the yardstick Brier factor of one half. Single-row multinomial predictions use a duplicated prediction row at the composition boundary, then discard that extra result. The base workflow is predicted once; the canonical Tailor and its learned nnet model remain unchanged. Learned-state compatibility checks are confined to the package's Tailor boundary because Tailor has no public learned-model extractor.

Full postprocessing compositions

All fixed-parameter holdout and inner-ensemble methods also accept `prediction_model = "full"`. First every original Q composition predicts its held-out C and builds its VA calibrator. Only then does each permitted F domain resolve its own B/P plan and fit an additional complete composition. This ordering preserves the source RNG stream, C values and calibrators between matched and full fits with the same seed. F excludes outer assessment A and any reserved selection S; for pairwise OvO it contains only the local class pair. C can influence this extra composition, never its own original Q composition. Full refers to the whole composition: no inner engine is trained on all F rows.

After successful fitting and prediction validation, only the F composition remains in `models$Full1`. Source workflows are discarded, while their held C values, VA calibrators, plans and postprocessing diagnostics remain. The additional `postprocessing$Full1` record uses `Q` for the F row membership, has empty `C`, and records its own B/P memberships and fit counts. Pairwise details expose it as `pair_N/Full1`. Prediction-row inner-fold names and sizes describe the retained F composition; source diagnostic records remain available without claiming new predictions from discarded workflows. If a caller's VA fold already uses `Full1`, the final ID gets the next free numeric suffix (`Full1.1`, etc.); `sources` records the actual ID.

For constant inner count J (J=1 for holdout), full Inductive performs 2J engine and postprocessor fits; full Cross with K VA folds performs (K+1)J. Only J pairs are retained per domain. Different local counts are summed, including F's own J. Three class pairs with K=5 and J=3 perform 54

engine and 54 postprocessor fits, retaining nine pairs. Each nonempty batch predicts those retained pairs once, then reuses the composed F prediction for every original VA calibrator. Empty batches call no engine. A failed final plan, fit or prediction aborts the result; there is no fallback, retention switch or prediction-time model-mode switch.

RDS retains complete F workflows and all diagnostics. Prediction needs neither truth, plan functions nor additional learning. See installed `examples/full-postprocessing.R` for all 80 fixed matched/full combinations, including shared two-class OvO/OvR, native outer evaluation and last_fit. The full-model transfer limitations in workflow_training apply to these complete compositions as well.

See Also

vennabers_examples for annotated installed scripts.

`calibrated_workflow` for the constructor, `venn_abers_details` for returned diagnostics, `venn_abers_tuning` for searches inside each B, and workflow_training for VA splits and full-model transfer.

`workflow_training` *Calibrated Workflow Training Reference*

Description

Split validation, stored models, outer resampling, pairwise training and full-model transfer for `calibrated_workflow`. For a worked introduction, start with `vignette("workflows", package = "vennabers")`. These are reference rules for the trainable workflow, not a separate constructor.

Supported scope and dependencies

All eight combinations of binary, OvO, OvR and regression with Inductive or Cross training are supported, including complete outer resampling. The integration packages in DESCRIPTION are optional Suggests and must be installed explicitly; manual calibration needs none of them. Use the declared minimum versions or newer supported versions. Tuning additionally needs `tune`, `dials` and `yardstick`; Tailor postprocessing needs `tailor` and `probably`. Each path checks its packages and minimum versions at construction and again at fit, and stops with the missing package. Basis engines also need their own packages: the examples use `stats` for GLM/LM, `nnet` for a shared multinomial model and `rpart` for decision trees. Fresh parallel workers must load `vennabers` and those engine namespaces, for example through `control_resamples(pkgs = ...)`. Reversible name encoding for binary, OvO/OvR and regression is available through `venn_abers_schema()`. Frozen classification/regression workflow adoption is available through `calibrate_workflow`. Own fit and predict functions enter as a workflow model through `model_adapter`. Ordered reference holdouts and binary/shared multiclass full-model transfer are supported. Pairwise OvO training supports matched/full Inductive and Cross calibration. Binary, shared OvO/OvR, pairwise OvO and regression paths support joint or two-stage Grid and Bayes selection, and learned postprocessing. Shared multiclass training does not establish complete Python training parity. The installed `NEWS.md` states the 1.x compatibility promise.

Preprocessing and calibration splits

All learned preprocessing belongs in the base workflow; the outer Variables preprocessor only carries raw data. Training and calibration positions must form a complete nonempty disjoint partition. Binary allows singleton and one-class calibration sets. Inductive OvO/OvR require every stored class in the holdout. Cross requires every class globally in the out-of-fold calibration; individual folds may omit classes. Nonempty one-class or singleton binary subcalibrations remain valid. An empty OvO pair in any fold fails before any base training, naming the pair and fold. Splits are never repaired. Trainability of the base engine is a separate requirement, and engine errors identify the training or probability-prediction stage.

Engine options and model-internal validation

Fixed, data-independent model options belong in the base workflow's model specification: main arguments and `parsnip::set_engine()` arguments. They travel with the base workflow, so every local fit uses them on its own rows. `calibrated_workflow()` and `fit()` accept no extra fit arguments; there is no interface for separately supplied validation data. An engine's internal validation, such as native XGBoost early stopping, splits its share from the prepared rows of each local fit, after the base recipe was learned on all of them. It is not an independent assessment of the procedure.

Local fit	Rows for learning and internal validation
Inductive/Cross source	its analysis rows, or their B with postprocessing; never C
Class pair	the analysis rows of that pair with its two original levels
Full1	all local fit rows (F), or its own B with postprocessing
Search candidate	its local training rows; held-out rows stay out
Frozen workflow	none: no base training, only calibration

P trains only the postprocessor and outer assessment rows never reach a fit. Invalid engine settings fail the whole fit with its fold, pair or B/P context; there is no fallback. Validation of engine-specific values stays with the engine. No support is promised for observation weights, offsets or an external `eval_set`. Not run, because xgboost is not a vennabers dependency:

```
spec <- parsnip::boost_tree(mode = "classification",
  trees = 100L, stop_iter = 10L) |>
  parsnip::set_engine("xgboost", validation = 0.2)
base <- workflows::workflow() |>
  workflows::add_formula(truth ~ .) |>
  workflows::add_model(spec)
fitted <- calibrated_workflow(base, outcome = "truth", seed = 42) |>
  generics::fit(data = training_data)
```

Shared multiclass probabilities

With `model_training = "shared"`, multiclass uses one trained base workflow per split complement. Its probability columns must be exactly `.pred_` plus each original class name, unique and complete; column order is matched by name. Values must be finite in [0,1], with row sums within 1e-8 of one. The existing matrix core normalizes within this tolerance. OvO normalizes each pair and uses a neutral 0.5 score for zero-sum pairs before optional native fit rounding (so 0.5 becomes

0 at precision 0). OvR uses all calibration observations for each class against the rest. New probabilities remain unrounded. Coupling, normalization and Log/Brier regularization are the same as in the manual core; no separate pair models are trained.

Names and schemas

Input and class names must be unique, nonmissing and nonempty. Class `class` and names starting with `.pred_` are reserved. Outcome names `.pred`, `.pred_*`, `.row`, `.config`, and `id` or `id` followed by a positive integer are reserved. Without a schema, names are never repaired; distinct names such as `a b` and `a.b` require a compatible base engine. Raw predictor `id` remains valid. Details do not join raw observations to predictions.

Stored state and prediction models

`workflows::extract_fit_engine()` returns a `vennabers_fit` with: `schema` (NULL or the explicitly supplied metadata), `parameters` (`requested`, logical `explicit` flags and `effective`), `outcome`, `predictors`, `columns` (predictors then outcome), `class_levels`, `event_class`, `loss`, `folds` (by fold ID, with integer `analysis`/`assessment` positions and `training_size`/`calibration_size`; with requested independent sizes, `Inductive1` also stores integer `unassigned` positions, and `effective` holds the resolved `training_size`, `calibration_size` and `unassigned_size` row counts; fits without the field have no unassigned rows), `models` (complete fitted prediction sources named by the same IDs, or only `Full1` in full mode; learned postprocessing uses package-owned compositions), `calibrator`, `calibration` (for binary: a list named by fold ID, each with original local integer `row`, logical event `truth` and unrounded double held-out `probability`; for multiclass: one list per fold with integer `row`, factor `truth` retaining all levels, and the complete unrounded named class matrix `probability`; for regression: a data frame per fold with integer `row`, numeric `truth` and held-out double `prediction`), `versions`, and `rng_kind`. Full mode fits the source recipes/models only on each analysis portion, then fits a fresh complete recipe/model on all local fit rows. No source recipe statistics are reused. Only after success are temporary source workflows discarded. Held-out values and the separate calibrators never change; there is no pooled OOF calibration or in-sample replacement. Training failure invalidates the whole fit; there is no fallback. A one-row full-model prediction validates its output contract during fit and is discarded without calibration. Print identifies the mode and fold-to-model mapping, and the training, calibration and unassigned row counts when independent sizes were requested. Saved RDS objects retain the complete prediction procedure. The cost is one/two base fits for matched/full Inductive and K/K+1 for Cross. Full prediction calls its one model once per nonempty batch, reusing that output across calibrators within the call. There is no persistent prediction cache. The existing Log/Brier mergers and prediction-time loss override are unchanged. For regression, `parameters$requested$mode` is `"regression"`; `effective$m` and `effective$epsilon` are numeric vectors named by fold ID. Each `folds[[id]]$moderation` stores `requested`, effective `m` and effective `epsilon=2*m/(k+1)`, alongside the fold sizes. Implicit classification options are absent from effective settings; class levels, event class and loss are NULL. The calibrator is the unchanged IVAR or CVAR core. IVAR uses weighted points, including the midpoint when both weights vanish. CVAR averages complete fold points, lower bounds and upper bounds separately with equal weights, regardless of fold size; it never merges averaged bounds. Matched mode retains each source workflow/recipe. Full mode retains one fresh full workflow and applies its numeric prediction to every held-out calibrator. No rounding or probability constraint is applied to regression predictions. For OvO/OvR, the global `event_class` and effective `event_level` are NULL; events belong to subproblems in details. The calibrator stores the original classes and chosen multiclass method. For Cross, `calibrator` is

a `cross_venn_abers` core with `folds`. Binary has a correspondingly named `calibrators` list; multiclass has one binary Cross core per stored pair or class, each retaining its fold IDs and corresponding ordered Inductive calibrators. Inspect learned recipes with the public workflow extractors; predict with full workflows. RDS restoration for pure R engines works in a fresh process after loading vennabers, workflows, and required engine packages. External-pointer engines may require their own serialization support.

Explicit split validation

For an explicit plan, selected raw columns (in canonical predictor order, then outcome), values, column types, factor levels and row order must match local fit data. Container class and row names are ignored; equal observations still occupy distinct positions. Training and calibration indices must be finite, whole, unique, in range, disjoint, nonempty and cover all local rows. The single internal fold ID is always `Inductive1`; rsample's creation/outer IDs are metadata, not configurable inner fold IDs. Cross requires unique nonempty character IDs, at least two folds, exactly one assessment assignment for every local row, and the exact complementary analysis set in each fold. All folds are checked before any learning starts. Repeated inner folds and bootstrap duplicates are rejected. Ordinary and classification-stratified partitions may be mixed or non-shuffled. For an ordered Inductive holdout use `make_splits()` with the first `n - ceiling(n * q)` local rows for training and the remaining `ceiling(n * q)` for calibration, requiring both parts to be nonempty. This example convention rounds the calibration size upwards: 11 rows and `q = .25` give training 1–8 and calibration 9–11. Explicit plans are used as supplied, without sorting, mixing or stratification. Order is local to each outer analysis set. Non-shuffled Cross folds retain exact training complements, including later rows, and are not forward validation. Group, time and other specialized split classes are outside this partition contract. An unbound global positions plan cannot be reused on different outer analysis data. See installed `examples/reference-splits.R` for all eight training paths, known ordinary/stratified memberships, data roles and assessment scores. Equal R/Python seeds alone do not establish equal splits or training; numerical comparisons require identical model outputs, assignments and options.

Cross aggregation

Matched Cross retains exactly one full base workflow per fold and all corresponding binary subproblem calibrators, with no additional full-data model. New data traverse every matching learned recipe/model path. A failed fold fails the entire fit. Log loss uses the core's stable logarithmic aggregation of endpoints; Brier uses the equal-weight mean of fold Brier points. Neither merger weights by fold size or pools calibration data. Precision rounds only calibration probabilities, never new model probabilities. Fold details reconstruct the final probability using these mergers. For multiclass, merge each binary subproblem across folds first, then couple OvO or normalize OvR. Averaging already coupled/normalized multiclass fold matrices is a different procedure. This shared R fold plan does not establish Python pair-model training parity.

Outer resampling and assessment

Ordinary `tune::fit_resamples()` fits the complete predictor independently on every outer analysis set. The stored loss controls native predictions even when tune calls parsnip directly; there is no loss argument to fit_resamples and no implicit control modification. For binary, match `event_level` in the constructor and `tune::control_resamples()`. Yardstick's binary Brier is the binary vector score. For OvO/OvR, yardstick Brier is exactly half the manual matrix score,

including the two-class matrix contract. Its log-loss clipping affects scoring only, never stored predictions. Regression uses normal yardstick RMSE/MAE (`.estimator = "standard"`) in outcome units, scored on the complete procedure's held-out predictions.

Use `save_pred = TRUE` and all public collectors, including `collect_notes()`, to check complete unsummarized resample, repeat, configuration and assessment-row identities and finite metrics. Inner failures, metric errors, NA removal or missing rows mean incomplete evaluation. All fits failing may make a collector itself error; a mean of surviving fits is insufficient. The installed `examples/calibrated-resampling.R` shows binary Inductive/Cross and multiclass Inductive/Cross OvO/OvR, plus both regression methods with an executable completeness check. Five outer by five inner Cross folds require exactly 25 base fits; no final fit runs automatically. Keeping extracted Cross states retains all 25 workflows and their recipes/calibrators, so memory scales with the number of retained base fits. A separate final development fit costs another five fits.

Use `control_resamples(extract = workflows::extract_fit_engine)` for all fitted engine states, `extract = identity` for full fitted workflows, or a function selecting compact diagnostics. `save_workflow = TRUE` alone does not preserve each fitted composite. Inner training is serial; use `parallel_over = "resamples"` for outer parallelism. Fresh workers must load vennabers and required base-engine namespaces through `pkgs`. Reproducibility is checked for the same versions, seeds, threads, strategy and deterministic engine; changing strategies or external engine RNGs is not covered.

Optional package registration

Integration packages are optional Suggests and version-checked on use. Neither core loading nor manual calibration requires or loads parsnip. Registration uses public parsnip APIs, is idempotent in either load order, and rejects foreign model registrations under the name `vennabers`. Packages are never auto-installed.

Pairwise model training

Global factor levels fix the class order before decomposition. Each stable `pair_id` names its two classes through `pair_models[[pair_id]]$class_levels`; the second is always the event. Each complete binary workflow learns exclusively from its pair's training rows, including all recipe estimates. Only its held-out event probabilities fit the existing binary calibrator. For Cross, the existing binary core first aggregates within each pair using stable log-loss aggregation or equal-weight Brier points. Only then are pair points combined with the same reciprocal OvO merger as shared training; coupled multiclass fold predictions are never averaged. The global schema remains unchanged; only the unfitted recipe's outcome type is adapted to the two-level local domain, retaining steps, roles and blueprint options.

`pair_models[[pair_id]]` retains `pair_id`, `class_levels`, `event_class`, `rows` (positions in the current global fit domain), `folds` (positions in the local pair, plus sizes and, with requested independent sizes, the pair's `unassigned` positions; the resolved counts in `effective` are then named by pair ID), `models`, `calibrator`, and `calibration`. Folds, models and calibration records are named by local fold ID: `Inductive1` or the supplied Cross IDs. The Cross calibrator retains its ordered, named binary fold calibrators. Prediction re-derives the pair order from the global class levels and rejects components whose `pair_id`, `class_levels` or fold names disagree, as well as models whose fitted two-class outcome mold is not the pair's. These replace the shared model/fold/calibrator fields. Calibration records retain local row positions, binary truths and unrounded probabilities. Details add `pair_id`, `model_training="pairwise"`, `prediction_model`, and `model_source` equal to the pair ID in matched mode. The pair ID together with `fold` identifies

the source workflow and calibrator in matched mode. Full mode instead names each new model source `pair_N/Full1`, separately from the unchanged pair/fold calibrator. Decoding changes class identities while preserving pair/fold/source IDs. Zero-row outputs retain all columns without calling any base model.

All pair plans and data bindings are validated before the first base fit. For Inductive, both classes must occur in each pair's training and calibration portions. Cross permits nonempty one-sided calibration folds when the training complement supports a trainable engine returning correct two-class probabilities. Within each pair, assessment folds partition every local row exactly once and training is the exact complement. Repeated/overlapping or incomplete plans, foreign data and missing/duplicate IDs fail before training. Automatic Cross uses the requested `v` without reduction; explicit valid partitions may have different fold counts across pairs. Equal fold IDs in different pairs have no shared meaning. Ordinary/stratified and non-shuffled reference partitions follow the same local IID split rules as shared training. Observations may recur across pairs. Explicit plans retain their local position order; an ordered pair holdout does not imply a global time cutoff. With a fixed `seed`, each pair resolver/automatic plan starts from that same seed in a restored local scope, so plans do not depend on preceding fits or call counters. Base fits then consume the enclosing fit's seeded stream in stable pair order. With NULL, splitting and fitting consume the caller's stream. Prepared assignments use `local_calibration_splits()` over every local pair, which rejects conflicting plans for identical data. Resampling callbacks see only the current outer analysis pair; development-wide plans cannot be replayed on different data.

For K classes, P=K*(K-1)/2 pairs cost P base fits for Inductive and the sum of local fold counts for Cross (three classes with five folds cost 15 fits). Each nonempty prediction call uses each retained model exactly once. Recipes, calibrators and models remain stored; prediction never learns. With `prediction_model="full"`, all source workflows and calibrators are built first, then one extra full workflow learns its recipe/model on each complete local two-class domain. This order preserves source RNG draws and held-out values. After every full workflow succeeds, `models` retains only `Full1` per pair; all calibrators, held-out records and identities remain. No in-sample recalibration or OOF pooling occurs. Full Inductive costs 2P fits; full Cross costs the sum of local fold counts plus P (18 for three classes and five folds). Each nonempty prediction uses each pair full model once and shares its named event probability across that pair's calibrators before binary aggregation and global coupling. Outer assessment/test rows never enter these fits. Any failed component fails the whole fit. See installed `examples/pairwise-inductive.R` for separate binary references, explicit pair plans, independent coupling, decoding and complete outer evaluation. The installed `examples/pairwise-cross.R` adds explicit unequal Cross partitions, reconstruction, native/decoded diagnostics and outer fit costs. Both examples compare matched/full predictions, costs and held-out scores; matching-model guarantees do not automatically transfer and better quality is not promised.

Full-model transfer

For `prediction_model="full"`, the split isolates calibration observations from source learning only. The extra full model includes them. Existing matching-model calibration or coverage guarantees do not automatically transfer to this differently trained model; improved quality is not guaranteed. Binary, shared multiclass and regression grids give Full1 its own local model/recipe search; the training method and prediction-model mode remain fixed per run. `fit_resamples()` rebuilds the complete procedure inside each outer analysis domain. `last_fit()` trains solely on its analysis domain, never analysis plus test. `calibrate_workflow()` remains a frozen-model path and does not accept this training option. Shared multiclass full-model transfer is an R extension, not Python pair-

model training parity. It uses the same full class matrix for every subproblem and fold, with binary aggregation before multiclass combination. See the installed examples `full-model-workflow.R` and `full-multiclass-workflow.R` under `system.file("examples", package="vennabers")`; the latter shows native/decoded diagnostics, repeated outer assessment, `last_fit()` and the unchanged yardstick Brier factor of two. `full-regression-workflow.R` demonstrates IVAR/CVAR transfer, RMSE/MAE, decoded numeric predictions and reconstruction of descriptive bounds. The Python 1.5.4 full-model regression option belongs to the exported lower `VennAbersCV` class, not its regression facade.

Printed costs

The fitted engine prints source mappings, the number of base fits (including temporary source fits discarded in full mode), models used per nonempty prediction, and stored calibration rows containing held-out truth and model predictions. Pairwise calibration rows count membership in each pair.

See Also

`calibrated_workflow` for arguments and data-role limitations, `venn_abers_details` for prediction diagnostics, `local_calibration_splits` for prepared plans, workflow_postprocessing for B/P compositions, and `venn_abers_tuning` for local parameter searches.

Index

∗ **documentation**
vennabers_examples, 52
∗ **package**
vennabers-package, 3

brier_score, *3*, 4, *14, 15*, *17*, *20*, *34*, *36*

calibrate_workflow, *3*, 5, *9*, *12, 13*, *27–29*, *31*, *41, 42*, *47*, *53, 54*, *58*
calibrated_workflow, *3*, *5*, *7, 8*, 9, *18*, *22*, *26*, *28, 29*, *31*, *39*, *41, 42*, *44*, *48*, *51*, *53*, *58*, *64*
calibration_plot, *3*, 13, *16, 17*, *20*, *30, 31*, *38*
corp_decomposition_plot, *3*
corp_decomposition_plot *(*corp_plot*)*, 15
corp_plot, *3*, *14*, 15, *31*, *38*
cross_venn_abers, *3*, *13*, 18, *32–34*, *36*, *39, 40*
cross_venn_abers_regression, *3*, 22, *44*, *46*

isoreg, *16*

local_calibration_splits, *10*, 25, *48*, *64*
log_loss, *3*, *20*, *34*, *36*
log_loss *(*brier_score*)*, 4

model_adapter, *3*, *13*, 26, *54*, *58*

plot(), *4*
plot.cross_venn_abers, *19*
plot.cross_venn_abers *(*plot.venn_abers*)*, 29
plot.cross_venn_abers_regression, *24*
plot.cross_venn_abers_regression *(*plot.venn_abers*)*, 29
plot.venn_abers, *14, 15*, *17*, 29, *40*
plot.venn_abers_regression, *37, 38*, *46*
plot.venn_abers_regression *(*plot.venn_abers*)*, 29
plot.vennabers_fit, *8*, *12*
plot.vennabers_fit *(*plot.venn_abers*)*, 29
predict._vennabers_fit *(*calibrated_workflow*)*, 9
predict.cross_venn_abers, *19, 20*, 32, *34*
predict.cross_venn_abers_regression *(*cross_venn_abers_regression*)*, 22
predict.venn_abers, *30*, *32*, 34, *40*, *44*
predict.venn_abers_regression, *38*
predict.venn_abers_regression *(*venn_abers_regression*)*, 44
predict.vennabers_frozen *(*calibrate_workflow*)*, 5
predict.vennabers_workflow *(*calibrated_workflow*)*, 9
print.cross_venn_abers, *19, 20*, 36
print.venn_abers, 37, *40*
print.vennabers_frozen *(*calibrate_workflow*)*, 5

quantile, *16*

regression_plot, *4*, *14*, *17*, *31*, 37

venn_abers, *3*, *5, 6*, *8, 9*, *13–15*, *18*, *34–36*, 39, *53*
venn_abers_decode, *42*, *44*
venn_abers_decode *(*venn_abers_schema*)*, 46
venn_abers_details, *3*, *8*, *12*, *30*, 41, *58*, *64*
venn_abers_encode *(*venn_abers_schema*)*, 46
venn_abers_one_hot, *3*, *12*, *34–36*, 43
venn_abers_regression, *3*, *8, 9*, *22–24*, *38*, *40*, 44
venn_abers_schema, *6*, *8*, *13*, 46
venn_abers_tuning, *3*, *11, 12*, *42*, 48, *53*, *58*, *64*
vennabers *(*vennabers-package*)*, 3

vennabers-package, 3
vennabers_examples, *3*, *8*, *10*, *13*, *20*, *24*, *28*, *40*, *46*, *51*, 52, *58*

workflow_postprocessing, *3*, *12*, *42*, *51*, 53, *53*, *64*
workflow_training, *3*, *11*, *12*, *28*, *42*, *51*, 58, *58*