Package 'ssel'

November 11, 2026

Type Package

Title Semi-Supervised Ensemble Learning

Version 0.5.2

Description Weighted-ensemble regression over base learners supported by 'caret' (Kuhn (2008) <doi:10.18637/jss.v028.i05>), with cross-validated hyperparameter selection, out-of-fold diagnostics, and signed residual-offset estimates. Multi-response problems use iterative input-space expansion related to Spyromitros-Xioufis et al. (2016) <doi:10.1007/s10994-016-5546-z>, with Jacobi or 'Gauss-Seidel' sweeps, package-defined companion gates and per-response iteration stitching. A package-defined pseudo-label stage promotes prediction rows by a cross-model and cross-dataset range ratio and accepts rounds with an out-of-fold squared-correlation gauge.

License AGPL (>= 3)

Encoding UTF-8

Language en-US

URL https://kanameishi.github.io/ssel/

Depends R (>= 4.1.0)

Imports caret, data.table, dbscan, digest, doParallel, earth, foreach, gbm, jsonlite, ranger, stats, utils

Suggests devtools, htmlwidgets, knitr, pkgdown, rcmdcheck, rmarkdown, roxygen2, rhub, testthat (>= 3.0.0), webshot2

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Alejandro Verri Kozlowski [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-8535-1170>)

Maintainer Alejandro Verri Kozlowski <averri@fi.uba.ar>

Repository CRAN

Date/Publication 2026-11-11 07:30:02 UTC

Contents

activeByImportance . . . 2
activeByShadow . . . 4
aggregateResponses . . . 5
auditOverfit . . . 9
auditQuantiles . . . 12
buildDataset . . . 14
chainPipeline . . . 17
computeActiveByImportance . . . 23
describeModule . . . 25
detectOutliers . . . 26
extractChainImportance . . . 26
modelPipeline . . . 28
oofEnsemble . . . 33
postProcess . . . 36
predictModel . . . 37
removeOutliersIQR . . . 41
semiSupervisedPipeline . . . 42
toNumeric . . . 48
trainModel . . . 49
trainRegressionModel . . . 52
which.nonnum . . . 57

Index **59**

`activeByImportance` *Select chain responses by mean normalized importance*

Description

Select chain-feature responses whose mean supplied importance reaches a package-defined floor. Use this table-only gate when normalized importance scores have already been assembled and a fixed percentage threshold is the desired policy.

Usage

```
activeByImportance(importanceDT, suffix = ".o", thresh = 0)
```

Arguments

`importanceDT`	A `data.table` with one row per importance observation and required columns `feature` and `normImportance`. `feature` must contain non-missing character names ending in `suffix`; `normImportance` must be numeric and is interpreted in normalized-importance percentage points. Extra columns are ignored. Values are not normalized or range-checked.

`suffix` A single non-missing, non-empty character suffix to remove from every `feature`, default `".o"`. Removal is positional: the function does not verify that names end in this suffix.

`thresh` A single numeric threshold in normalized-importance percentage points, default `0`. The comparison is inclusive. The function does not validate its length, whether it is finite, or its range; `NA` selects no response.

Value

A character vector of unique stripped response names in first-appearance order. It is empty when no group reaches `thresh`.

Gate definition

For each response name obtained by removing the final `nchar(suffix)` characters from `feature`, the function computes the arithmetic mean of all non-missing `normImportance` values. Every input row contributes separately, including duplicate rows. A response is returned when its mean is greater than or equal to `thresh`; an exact threshold tie therefore passes. A group with only missing scores has a `NaN` mean and is excluded.

The threshold and averaging rule are ssel policy. Averaging is mechanical: the function neither establishes nor checks that scores from different learners or other sources are comparable. Output order follows the first appearance of each stripped response name, and grouping removes duplicate names from the result.

Validation and side effects

`NULL` and a zero-row table return `character(0)`. A non-empty input must be a `data.table`; missing required columns or incompatible types produce underlying base R or data.table errors or warnings rather than a deliberate validation message. The input is copied before the temporary response column is added, so it is not modified by reference. The function reads and writes no files, uses no random numbers, and emits no message or warning for valid inputs.

See Also

`activeByShadow()` for the strict-majority shadow alternative. `extractChainImportance()` is the package table producer, and `computeActiveByImportance()` composes extraction with this gate; their model-file and stability contracts are separate from this table operation.

Examples

```
importance <- data.table::data.table(
  feature = c("beta.o", "alpha.o", "beta.o", "alpha.o", "gamma.o"),
  normImportance = c(1, 2, 3, NA, NA)
)

# beta ties the 2% floor; alpha ignores one NA; gamma has no usable score.
activeByImportance(importance, suffix = ".o", thresh = 2)
```

`activeByShadow` *Select chain responses by a shadow comparison*

Description

Select chain-feature responses whose supplied importance is strictly greater than a same-row shadow importance in more than half of their rows. Use this table-only gate when each row contains the feature and shadow scores to be compared directly.

Usage

```
activeByShadow(IMP, suffix = ".o")
```

Arguments

`IMP`	A `data.table` with one row per comparison cell and required columns `feature`, `normImportance`, and `shadowImportance`. `feature` must contain non-missing character names ending in `suffix`; both importance columns must be numeric on the same normalized-percentage scale. Extra columns are ignored. Values and row uniqueness are not validated.
`suffix`	A single non-missing, non-empty character suffix to remove from every `feature`, default `".o"`. Removal is positional: the function does not verify that names end in this suffix.

Value

A character vector of unique stripped response names in first-appearance order. It is empty when no response has a strict majority.

Gate definition

Each input row is one comparison cell; method, response, and dataset columns are not inspected. A row is a success only when `shadowImportance` is not missing and `normImportance` > `shadowImportance`. Equality is not a success. For each stripped response name, the denominator is the full row count, including duplicate rows and rows with a missing shadow score. The response passes only when successes are strictly greater than half of that count, so a tied half does not pass.

A missing `shadowImportance` counts as an unsuccessful row and remains in the denominator. A missing `normImportance` paired with a present shadow makes the group's success count `NA`, so that group is excluded. The helper assumes that the two score columns are comparable within each row; it does not establish comparability across learners or sources.

This strict-majority rule is ssel package policy. It uses a supplied shadow score as a reference but does not implement the iterative random-probe tests of the Boruta algorithm.

Validation and side effects

`NULL`, a zero-row table, or an input without `shadowImportance` returns `character(0)`. Otherwise a non-empty input must be a `data.table`; missing `feature` or `normImportance` columns and incompatible types produce underlying base R or data.table errors or warnings. The function does not modify `IMP` by reference. It reads and writes no files, uses no random numbers, and emits no message or warning for valid inputs.

References

Kursa, M. B. and Rudnicki, W. R. (2010). Feature Selection with the Boruta Package. *Journal of Statistical Software*, 36(11), 1–13. doi:10.18637/jss.v036.i11.

See Also

`activeByImportance()` for the inclusive fixed-threshold gate. `extractChainImportance()` is the package table producer; its model-file contract is separate from this table operation.

Examples

```
importance <- data.table::data.table(
  feature = c("alpha.o", "alpha.o", "alpha.o",
              "beta.o", "beta.o", "gamma.o", "gamma.o"),
  normImportance = c(2, 3, 0, 2, 0, 2, 2),
  shadowImportance = c(1, 1, 1, 1, 1, 1, NA)
)

# alpha wins 2/3 cells; beta ties 1/2; gamma's missing shadow counts.
activeByShadow(importance, suffix = ".o")
```

---

aggregateResponses *Select and assemble final response deliveries*

Description

Combine unseen ensemble point predictions with reconstructed training-row OOF predictions, add signed residual offsets, select one candidate per sample and response by cell R2, and write delivery/feature products. This final selector is distinct from `predictModel()`'s response-level RMSE-or-R2 selector.

Usage

```
aggregateResponses(
  .path.summary,
  .path.export,
  .path.datasets,
  key,
  responses,
  datasets,
```

```
  boundsMin = -Inf,
  boundsMax = Inf,
  .path.iter = NULL
)
```

Arguments

`.path.summary`	Existing directory containing the four summary inputs described in **Input tables**.
`.path.export`	Existing writable output directory. It is not created by this helper. `Yo.csv`, `X.csv`, and `Xo.csv` are assigned here.
`.path.datasets`	Directory containing assembled `dataset.csv` inputs used only for feature exports.
`key`	A non-empty character key name used when extracting feature tables. Response aggregation itself is hard-coded to `SampleID`; setting another key does not replace that requirement.
`responses`	Non-empty character vector excluded, with `set`, from feature columns in `X.csv` and `Xo.csv`.
`datasets`	Non-empty character vector of assembled dataset basenames. Feature extraction uses the last declared file that actually exists.
`boundsMin`, `boundsMax`	Numeric response bounds used to project every point-plus-offset value. Each must be a single non-`NA` numeric value; `Inf` and `-Inf` are allowed. Their order and finiteness are not validated.
`.path.iter`	`NULL`, or an iteration directory. When non-null, its parent path for `Y.csv` is created recursively and the training-row median-offset table is written or merged.

Value

Invisibly, `NULL`. Results are `Yo.csv`, optional `Y.csv`, and the feature files written under the rules above.

Input tables

The helper stops unless `response_long.csv`, `metrics.csv`, and `prediction_quantiles.csv` exist. It then separately requires `residuals_oof.csv`.

- `response_long.csv` must provide `SampleID`, `value`, `method`, `dataset`, and `response`. Only rows with `method == "ensemble.RMSE"` are retained and marked as unseen rows.
- `residuals_oof.csv` must provide `SampleID`, `y_hat_oof`, `dataset`, and `response`. Every row is converted to a training candidate with `value = y_hat_oof` and method `ensemble.RMSE`.
- `metrics.csv` must provide `method`, `response`, `dataset`, and `r2`. Only `ensemble.RMSE` metric rows enter.
- `prediction_quantiles.csv` must provide `response`, `dataset`, `q_50`, `q_90`, and `q_95`.

Extra columns can propagate through joins. Missing, duplicate, or incompatible keys retain data.table join behavior; no uniqueness assertion is made.

Offset construction and final selector

Metric and offset rows are joined to every unseen or training candidate by response and dataset. Let i index `SampleID`, r a response, d a joined candidate dataset, and $p \in \{0.50, 0.90, 0.95\}$ an offset probability. Let $\hat{y}_{ird}$ be a candidate point value and $q_{p,rd}$ its signed OOF-residual offset, all in response units. Let a and b be the bounds supplied by `boundsMin` and `boundsMax`, respectively. When $a \leq b$, define interval projection by $\Pi_{[a,b]}(z) = \min\{b, \max\{a, z\}\}$ for real z. The helper computes

$$Q_{p,ird} = \Pi_{[a,b]}\{\hat{y}_{ird} + q_{p,rd}\}.$$

These are projected point-plus-offset estimates, not calibrated predictive quantiles or intervals. Bound order and finiteness are not checked; with invalid bounds the nested minimum/maximum is still executed but is not an interval projection.

For each `(SampleID, response)`, all RMSE-ensemble candidates are ordered by descending joined cell `r2`; the first row is selected and supplies one dataset for all three probabilities:

$$d^*_{ir} = \text{first} \operatorname*{argmax}_d R^2_{rd}.$$

The score is cell-level even though selection is grouped by sample. Ties follow joined-row order. Finite scores precede missing scores; when all joined scores in a group are missing, the first joined row is still selected with missing `r2`. Missing offsets propagate missing delivery values because this helper does not remove them. This selector never considers `ensemble.R2` rows and is not the response-level metric selector used by `predictModel()`. Therefore `modelPipeline(ensemble = )` does not change the final candidate family assembled here.

Delivery and iteration products

The selected table is expanded to long probabilities `p = 0.50`, `0.90`, and `0.95`. It retains available identifier columns from `SampleID`, `HoleID`, `From`, and `To`, followed by `response`, `p`, `value`, `r2`, and `dataset`. Response values and offsets share response units; `r2` and `p` are dimensionless.

Unseen rows are written to `Yo.csv`. If that file exists, every old row whose response occurs in the current unseen result is removed, current rows are appended, and other responses are preserved. Replacement is by response, not dataset.

When `.path.iter` is non-null, training rows at `p = 0.50` are cast to `Y.csv` with one response column per `SampleID`. On merge, new response columns replace same-named old columns and old non-overlapping response columns remain. Thus `Y.csv` contains median signed-offset estimates; `p = 0.50` is not necessarily the unadjusted point value.

Feature products, conditions, and return

Among declared assembled files that exist, the last one is read. Every column except `key`, the declared responses, and `set` is treated as a feature. Rows with `set == "train"` write `X.csv`; rows with `set == "predict"` write `Xo.csv`. If no assembled file exists, a warning is emitted and neither feature file is written. If no feature column remains, a warning is emitted but key-only files are still written.

Assigned CSVs are overwritten and writes are not transactional. Required file, column, join, cast, and directory errors propagate after any earlier writes. Valid progress is emitted as a message of class `moduleDetail`. The helper uses no random numbers.

References

Parikh, N., and Boyd, S. (2014). Proximal Algorithms. *Foundations and Trends in Optimization*, 1(3), 127–239. doi:10.1561/2400000003.

See Also

`predictModel()` for cell evaluation and the first selector, `auditQuantiles()` for signed residual offsets, and `chainPipeline()` for the consumer of optional `Y.csv`.

Examples

```
root <- tempfile("ssel-aggregate-")
summary_dir <- file.path(root, "summary")
export_dir <- file.path(root, "export")
datasets_dir <- file.path(root, "datasets")
iter_dir <- file.path(root, "iter")
for (path in c(summary_dir, export_dir, datasets_dir)) {
  dir.create(path, recursive = TRUE)
}

data.table::fwrite(
  data.table::data.table(
    SampleID = rep(c("P1", "P2"), each = 2),
    value = c(10, 20, 11, 21),
    method = "ensemble.RMSE",
    dataset = rep(c("d1", "d2"), 2),
    response = "target"
  ),
  file.path(summary_dir, "response_long.csv")
)
data.table::fwrite(
  data.table::data.table(
    method = "ensemble.RMSE", response = "target",
    dataset = c("d1", "d2"), r2 = c(0.5, 0.9)
  ),
  file.path(summary_dir, "metrics.csv")
)
data.table::fwrite(
  data.table::data.table(
    response = "target", dataset = c("d1", "d2"),
    q_50 = c(0, 1), q_90 = c(1, 2), q_95 = c(2, 3)
  ),
  file.path(summary_dir, "prediction_quantiles.csv")
)
data.table::fwrite(
  data.table::data.table(
    response = "target", dataset = c("d1", "d2"),
    rowIndex = 1L, SampleID = "S1",
    y_obs = c(8, 18), y_hat_oof = c(9, 19), residual = -1
  ),
  file.path(summary_dir, "residuals_oof.csv")
)
```

```
for (dataset in c("d1", "d2")) {
  data.table::fwrite(
    data.table::data.table(
      SampleID = c("S1", "P1", "P2"),
      set = c("train", "predict", "predict"),
      feature = 1:3
    ),
    file.path(datasets_dir, paste0(dataset, ".csv"))
  )
}

aggregateResponses(
  .path.summary = summary_dir,
  .path.export = export_dir,
  .path.datasets = datasets_dir,
  key = "SampleID",
  responses = "target",
  datasets = c("d1", "d2"),
  .path.iter = iter_dir
)
data.table::fread(file.path(export_dir, "Yo.csv"))
data.table::fread(file.path(iter_dir, "Y.csv"))
# Dataset d2 wins because its RMSE-ensemble cell has the larger R2.

unlink(root, recursive = TRUE)
```

`auditOverfit` *Audit package-defined in-sample optimism*

Description

Compare projected in-sample model error with the resampling summaries stored in fitted caret models, then classify a package-defined relative-optimism ratio. Use this table as a diagnostic for unusually large training-versus- resampling gaps. The ratio and default thresholds are ssel policy; they are not a hypothesis test, a universal overfitting measure, or published cutoffs.

Usage

```
auditOverfit(.path.model, .path.train, .path.summary,
  .min = -Inf, .max = Inf, REL_OPTIMISM_CATASTROPHIC = 0.95,
  REL_OPTIMISM_HIGH = 0.60,
  .response_pattern = NULL, verbose = TRUE)
```

Arguments

`.path.model`	Path to an existing directory containing model files named `method_response_dataset.Rds`. The method identifier must be a non-empty string without an underscore. Matching objects must support `predict(model, newdata = TRAIN)` and contain a non-empty `resample` table with `RMSE` and `Rsquared` columns.

`.path.train` Path to an existing directory containing `response_dataset_TRAIN.csv`. Each eligible file must contain the fitted predictors and a column named `y` coercible to numeric.

`.path.summary` Path to the output directory. It is created recursively when absent; `overfit.csv` is overwritten.

`.min`, `.max` Numeric lower and upper response bounds, in response units. Only in-sample predictions are projected with `pmax(.min, pmin(.max, z))`. Bound order and finiteness are not validated.

`REL_OPTIMISM_CATASTROPHIC`
Numeric package-policy threshold for the `"catastrophic"` label. It is tested first and defaults to 0.95.

`REL_OPTIMISM_HIGH`
Numeric package-policy threshold for the `"high_optimism"` label. It is tested second and defaults to 0.60. Threshold length, finiteness, range, and ordering are not validated.

`.response_pattern`
A non-empty character regular expression inserted into the model-filename parser. Use noncapturing groups when grouping is needed because additional capturing groups change parser field positions. The default `NULL` deliberately fails.

`verbose` A non-missing logical value. If true, the output path and, when a model was audited, the flag counts are emitted as messages of class `moduleDetail`.

Value

Invisibly, a one-element list named `overfit`. Its value is the table with the schema above. When no model survives, this is an empty zero-column table, as described in **Files, skips, and side effects**.

Diagnostic definition

For an eligible model, let y_i be numeric `TRAIN$y` and let $\hat{y}_i^{in}$ be the generic in-sample prediction projected onto $[a, b]$. The helper reports

$$\mathrm{RMSE}_{in} = \left[n^{-1} \sum_i (y_i - \hat{y}_i^{in})^2 \right]^{1/2}$$

and, when the sample variance of y is nonzero,

$$R^2_{in} = \mathrm{cor}(y, \hat{y}^{in})^2.$$

It separately reads each row of `model$resample`, reports the mean, sample standard deviation, minimum, and maximum of its `RMSE` column, and the mean and sample standard deviation of `Rsquared`, using `na.rm = TRUE`. The output field `k_folds` is exactly the number of rows in `model$resample`; its name does not establish that those rows are unique folds rather than another caret resampling design.

With $\overline{\mathrm{RMSE}}_{CV}$ equal to that resample mean, ssel defines

$$\mathrm{rel_optimism} = \frac{\overline{\mathrm{RMSE}}_{CV} - \mathrm{RMSE}_{in}}{\max(\overline{\mathrm{RMSE}}_{CV}, 10^{-9})}.$$

The unscaled numerator is `optimism_rmse`. With finite response bounds, this compares projected in-sample error with caret's stored, unprojected resample RMSE summary; it is not a like-for-like reprojection of both terms.

Non-finite relative optimism is labeled `"unclassified"`. Otherwise, the catastrophic threshold is tested first, followed by the high threshold; remaining rows are `"reliable"`. Negative values are possible. These labels express only the configured package policy.

Files, skips, and side effects

For every retained model, `overfit.csv` contains `method`, `response`, `dataset`, `n_train`, `k_folds`, `rmse_in`, `rmse_cv_mean`, `rmse_cv_sd`, `rmse_cv_min`, `rmse_cv_max`, `optimism_rmse`, `rel_optimism`, `r2_in`, `r2_cv_mean`, `r2_cv_sd`, and `flag`. RMSE and optimism fields have response units; relative optimism and R-squared fields are dimensionless.

A filename that does not match `.response_pattern` is filtered out silently: excluding responses is what the pattern is for. A file that does match and still cannot be audited – absent TRAIN file, unreadable model, failed prediction, prediction-length mismatch, or a missing or empty resample table – is skipped and reported by a warning naming the file and the reason. A second warning reports an audit that produced nothing at all, distinguishing a pattern that matched no filename from matches that were all skipped. Numeric coercion and summary edge cases retain base R behavior, including missing or infinite fields and warnings.

When no model survives, the written file depends on what was already there. `data.table::fwrite()` refuses a column-less table, so an existing `overfit.csv` is left untouched and still holds the previous run's verdict with nothing in the file itself to mark it stale; the warnings above are that signal. With no previous file, `fwrite()` warns and creates a zero-byte `overfit.csv`. In both cases the function returns a zero-row, zero-column table. Malformed-column, data.table, and file errors not caught by the helper propagate.

The function reads models and training data, writes only `overfit.csv`, does not fit models, and neither sets a seed nor draws random values directly. Model-specific prediction methods govern any of their own random-number effects.

References

Kuhn, M. (2008). Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5), 1–26. doi:10.18637/jss.v028.i05.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*, second edition, Chapter 7. Springer. doi:10.1007/9780387848587.

See Also

`trainRegressionModel()` for model and resampling production, `oofEnsemble()` for row-level OOF reconstruction, and `auditQuantiles()` for the separate residual-offset summary.

Examples

```
root <- tempfile("ssel-overfit-")
model_dir <- file.path(root, "model")
train_dir <- file.path(root, "train")
summary_dir <- file.path(root, "summary")
```

```
dir.create(model_dir, recursive = TRUE)
dir.create(train_dir)

train <- data.frame(
  x = seq_len(8),
  y = c(1, 2, 2, 4, 5, 5, 7, 8)
)
fit <- stats::lm(y ~ x, data = train)
fit$resample <- data.frame(
  RMSE = c(0.2, 0.3),
  Rsquared = c(0.8, 0.9)
)
saveRDS(fit, file.path(model_dir, "lm_target_demo.Rds"))
data.table::fwrite(train, file.path(train_dir, "target_demo_TRAIN.csv"))

diagnostic <- auditOverfit(
  .path.model = model_dir,
  .path.train = train_dir,
  .path.summary = summary_dir,
  .response_pattern = "target",
  verbose = FALSE
)
diagnostic$overfit[, .(
  method, rmse_in, rmse_cv_mean, rel_optimism, flag
)]
# The label follows ssel's configured ratio thresholds.

unlink(root, recursive = TRUE)
```

`auditQuantiles` *Estimate signed OOF-residual offsets*

Description

Reconstruct projected OOF ensemble residuals with `oofEnsemble()` and write their cell-pooled sample quantiles. The resulting offsets can be added to a point prediction by `predictModel()`. They are descriptive residual-offset estimates: this helper does not construct a predictive distribution or establish conditional, marginal, conformal, or two-sided interval coverage.

Usage

```
auditQuantiles(.path.model, .path.train, .path.summary,
  .path.datasets = file.path("data", "datasets"), .min = -Inf, .max = Inf,
  .response_pattern = NULL, .sort_keys = character(0), verbose = TRUE)
```

Arguments

`.path.model`, `.path.train`, `.path.datasets`, `.min`, `.max`, `.response_pattern`, `.sort_keys`

Passed unchanged to `oofEnsemble()`. See that reference for the model, split, identifier, projection, regular-expression, and ordering contracts.

`.path.summary` Path to the output directory. It is created recursively when absent. Assigned CSV files are overwritten.

`verbose` A non-missing logical value. If true, the paths written and the cell count with the median offsets are emitted as messages of class `moduleDetail`.

Value

Invisibly, a named list with `residuals`, the merged row-level OOF table, and `quantiles`, the six-column cell summary.

Estimator

For cell (r, d) and each retained reconstructed cast row $h = (\texttt{rowIndex}, \texttt{obs})$, let $e_{hrd} = y_{hrd} - \hat{y}_{hrd}^{OOF}$ be the signed residual returned by `oofEnsemble()`. A row index can therefore contribute more than once when methods supplied different saved observations; each returned row is a separate quantile observation. For $p \in \{0.50, 0.90, 0.95\}$, the helper computes R's type-7 sample quantile. With the n retained residual rows sorted as $e_{(1)} \leq \cdots \leq e_{(n)}$, define $h = (n-1)p + 1$, $j = \lfloor h \rfloor$, and $\gamma = h - j$; then

$$q_{p,rd} = (1 - \gamma)e_{(j)} + \gamma e_{(j+1)},$$

with R's endpoint convention. The values `q_50`, `q_90`, and `q_95` are signed and have the response's units. They are computed separately for every response–dataset cell; no centering, symmetrization, covariate conditioning, or finite-sample calibration is performed.

Files, partial runs, and return

The helper first writes `residuals_oof.csv` with columns `response`, `dataset`, `rowIndex`, `SampleID`, `y_obs`, `y_hat_oof`, and `residual`. If a compatible file already exists, every prior row whose `response` occurs in the current result is removed before the current rows are appended. Replacement is by response, not by response–dataset cell: an omitted dataset for a current response is not retained. Rows for other responses remain.

Quantiles are recomputed from that complete merged residual table and `prediction_quantiles.csv` is overwritten with columns `response`, `dataset`, `n`, `q_50`, `q_90`, and `q_95`. Thus a partial response run carries prior other-response residuals and their recomputed summaries forward. Existing quantile rows are never merged independently. Row order is not a public sorting guarantee.

The return is an invisible list named `residuals` and `quantiles`, containing exactly the two tables written in the current call. If OOF reconstruction returns no row, the function stops after creating `.path.summary` and before writing either assigned CSV. Its message names the model directory, how many `.Rds` files it holds and the response pattern in force, and refers to the `oofEnsemble()` warnings for which files matched and why each was skipped. A failure after the residual write can leave that file updated while the quantile file remains from an earlier call or absent. Existing-file schema, file, quantile, data.table, and upstream reconstruction errors propagate. The helper does not fit models or alter the random-number state.

References

Hyndman, R. J., and Fan, Y. (1996). Sample quantiles in statistical packages. *The American Statistician*, 50(4), 361–365. doi:10.1080/00031305.1996.10473566.

Barber, R. F., Candes, E. J., Ramdas, A., and Tibshirani, R. J. (2021). Predictive inference with the jackknife+. *The Annals of Statistics*, 49(1), 486–507. doi:10.1214/20AOS1965. This reference distinguishes coverage procedures using fold- or leave-one-out-specific test predictions from the descriptive pooled offset computed here.

See Also

`oofEnsemble()` for the exact OOF estimator and identifier mapping, and `predictModel()` for the separate point-prediction-plus-offset delivery step.

Examples

```
root <- tempfile("ssel-offsets-")
model_dir <- file.path(root, "model")
train_dir <- file.path(root, "train")
summary_dir <- file.path(root, "summary")
dir.create(model_dir, recursive = TRUE)
dir.create(train_dir)

data.table::fwrite(
  data.table::data.table(y = c(1, 3)),
  file.path(train_dir, "target_demo_TRAIN.csv")
)
data.table::fwrite(
  data.table::data.table(SampleID = c("S1", "S2")),
  file.path(train_dir, "target_demo_TRAIN_ids.csv")
)
fit <- list(
  pred = data.frame(
    tune = 1L, pred = c(0, 2), obs = c(1, 3), rowIndex = 1:2
  ),
  bestTune = data.frame(tune = 1L)
)
saveRDS(fit, file.path(model_dir, "lm_target_demo.Rds"))

offsets <- auditQuantiles(
  .path.model = model_dir,
  .path.train = train_dir,
  .path.summary = summary_dir,
  .response_pattern = "target",
  verbose = FALSE
)
offsets$quantiles
# Both signed residuals equal 1, so all three offsets equal 1.

unlink(root, recursive = TRUE)
```

`buildDataset` *Build per-response training and prediction files*

Description

Read assembled dataset CSV files, choose response and predictor columns, and write the per-response files consumed by the model-fitting helpers. Use this low-level helper when the assembled CSVs already exist and their row roles and columns are known.

Usage

```
buildDataset(.path.input, .path.train, .file.index = NULL,
  Y.PATTERN = NULL, KEY = NULL, COLS.id = character(0),
  features = NULL, OUTLIER_REMOVAL = FALSE, SET_COL = NULL,
  ITER_SUFFIX = NULL, .shadow = FALSE)
```

Arguments

`.path.input`	Path to an existing directory. Each non-recursive file whose name ends in `.csv` is treated as one assembled dataset.
`.path.train`	Path to an existing writable output directory. The function does not create it.
`.file.index`	Reserved for compatibility; currently ignored.
`Y.PATTERN`	A non-empty regular expression used case-insensitively to select response column names. Required despite the `NULL` default.
`KEY`	A single non-empty name of the row-identifier column. Required despite the `NULL` default.
`COLS.id`	A character vector of additional identifier columns to exclude from inferred predictors. `SET_COL` is always excluded as well. Names absent from a given CSV are silently ignored.
`features`	`NULL`, or a character vector giving the exact base predictors in output order. `NULL` infers predictors as every column that is not a response, `KEY`, or an identifier. Exact names must exist and cannot name responses or identifiers.
`OUTLIER_REMOVAL`	A non-missing logical scalar. If `TRUE`, `detectOutliers()` is applied to each complete training split and rows it flags are removed.
`SET_COL`	A single non-empty name of the row-role column. The exact value `"train"` marks training rows; other non-missing values mark prediction rows. The column must contain no missing values. This precondition is not proactively validated. Required despite the `NULL` default.
`ITER_SUFFIX`	`NULL`, or a suffix identifying prior-iteration response columns. Available matching columns are added to an exact `features` set; each response's own suffixed column is excluded from its split.
`.shadow`	A non-missing logical scalar. If `TRUE`, add a standard-normal column named `shadow` using the fixed seed `12345`. It is inferred as a predictor when `features = NULL`; with exact features it must be named.

Value

Invisibly returns `NULL`. The result is the set of CSV files described in **Files written**.

Input CSV schema

The input filename without `.csv` becomes the dataset identifier used in output filenames. Each CSV must contain `KEY`, `SET_COL`, the columns matched by `Y.PATTERN`, and any exact `features`. Response columns are eligible only when numeric after cleaning. Predictor and response units are not converted or scaled; CSV unit metadata is neither read nor written, so callers must supply consistent units.

Before splitting, the helper reads `""`, `"N/I"`, `"N/A"`, and `"-"` as missing, removes a column named `*`, removes duplicate rows, and deletes ASCII spaces from every character value. It passes character columns other than `KEY` and `COLS.id` (including `SET_COL`) through `toNumeric()` and replaces a column only when every non-missing value converts successfully. A partially convertible column remains character.

Row and predictor filtering

Predictors with at most one distinct non-missing training value are removed. Training rows incomplete in any retained predictor or the current response are omitted. Prediction rows are omitted when a retained predictor is missing or a character predictor has a value not observed in training; one warning reports the number omitted. No imputation or unit conversion is performed.

Files written

For every numeric response `response` in dataset `dataset`, three CSV files are written:

- `response_dataset_TRAIN.csv`: retained predictors followed by the response renamed `y`;
- `response_dataset_TEST.csv`: retained predictors followed by the original `KEY` column;
- `response_dataset_TRAIN_ids.csv`: one column named `SampleID`, containing the training identifiers aligned with the filtered TRAIN rows.

Existing files with those names are overwritten. Other files in `.path.train` are not removed. The function writes no files for a CSV with no numeric matched response. An input directory with no matching CSV files produces an error before any split file is written.

Validation and side effects

Missing required selectors, required columns, or output directories produce errors. Exact `features` also error when they contain missing, empty, duplicated, response, or identifier names. `SET_COL` values must be non-missing, but this is not checked before splitting; a missing row role can therefore produce an indirect downstream error. `COLS.id` names are not validated, and absent names are silently ignored. `KEY` uniqueness and missingness are not checked. Shadow generation restores an existing global random-number state; if no state existed, the call initializes one.

See Also

`toNumeric()` for the supported numeric-string grammar and `detectOutliers()` for optional training-row filtering.

Examples

```
root <- tempfile("ssel-buildDataset-")
input_dir <- file.path(root, "assembled")
output_dir <- file.path(root, "splits")
dir.create(input_dir, recursive = TRUE)
dir.create(output_dir)

assembled <- data.frame(
  SampleID = c("S1", "S2", "P1", "P2"),
  set = c("train", "train", "predict", "predict"),
  feature_a = c(1, 2, 3, 4),
  feature_b = c("1,5", "2,5", "3,5", "4,5"),
  target = c(10, 20, NA, NA)
)
utils::write.csv(assembled, file.path(input_dir, "demo.csv"),
                 row.names = FALSE, na = "")

buildDataset(
  .path.input = input_dir,
  .path.train = output_dir,
  Y.PATTERN = "^target$",
  KEY = "SampleID",
  SET_COL = "set"
)

sort(basename(list.files(output_dir)))
utils::read.csv(file.path(output_dir, "target_demo_TRAIN.csv"))
utils::read.csv(file.path(output_dir, "target_demo_TEST.csv"))

unlink(root, recursive = TRUE)
```

`chainPipeline` *Fit an iterative multi-response companion chain*

Description

Refit the supervised ssel workflow while using other responses' preceding delivery values as predictors. The helper supports simultaneous Jacobi updates and an asymmetric Gauss–Seidel update in which training companions can refresh within a sweep but prediction-row companions remain frozen. It is filesystem orchestration around `modelPipeline()`, not a standard regressor-chain estimator or a general fixed-point solver.

Usage

```
chainPipeline(
  .path.datasets,
  .path.train,
  .path.model,
  .path.metrics,
```

```
  .path.response,
  .path.summary,
  .path.export,
  .path.iter,
  key,
  responses,
  datasets,
  colsId = character(0),
  features = NULL,
  setCol = "set",
  methods = c("ranger", "earth", "gbm"),
  seed = NULL,
  ensemble = "ensemble.RMSE",
  boundsMin = -Inf,
  boundsMax = Inf,
  maxIter,
  tol,
  patience,
  sweep = "jacobi",
  suffix = ".o",
  chainGate = "fixed",
  chainThresh = 0,
  chainStable = 1L,
  resume = FALSE
)
```

Arguments

`.path.datasets`	Existing directory containing the original assembled CSV files. Every non-recursive CSV is copied into augmented iteration directories; `modelPipeline()` applies its all-CSV split contract.
`.path.train`, `.path.model`, `.path.metrics`, `.path.response`, `.path.summary`, `.path.export`	Mutable supervised-workflow directories. They are repeatedly overwritten or removed under the rules in **Files and final stitch**.
`.path.iter`	Mutable chain-state directory. A fresh run recursively removes it. It then contains augmented datasets, iteration response and metric copies, five-file summary snapshots, and `convergence.csv`.
`key`	Non-empty character key passed to joins and `modelPipeline()`. Final response products still inherit `aggregateResponses()`'s independent `SampleID` requirement.
`responses`	Non-empty character vector of response column names. The names also determine sorted Gauss–Seidel order, metric columns, and suffixed companion names. Supervised split-file discovery interprets the first underscore field as the response, so compatible response identifiers cannot contain underscores.
`datasets`	Non-empty character vector passed to `modelPipeline()` for final feature-file precedence. It does not restrict all-CSV split/model discovery. Upstream split parsers likewise require dataset identifiers without underscores.

`colsId` Character vector of identifier columns excluded from inferred predictors.

`features` `NULL`, or exact base predictor names. With exact names, available response companions are appended automatically by `buildDataset()`. The shadow predictor is not appended automatically and must be named explicitly when `chainGate = "shadow"`; `NULL` inference includes it.

`setCol` Non-empty character name of the assembled row-role column.

`methods` Character vector of caret method identifiers passed to every supervised fit. The default is `c("ranger", "earth", "gbm")`.

`seed` `NULL`, or a value passed to iteration-zero `modelPipeline()`. Iteration t receives `seed + t`. The function does not validate arithmetic compatibility.

`ensemble` Forwarded to `modelPipeline()` for its first delivery selector. Chain metric comparison and final iteration selection instead use `ensemble.RMSE` rows under their separate contracts. Active-set extraction does not use ensemble rows: it reads the individual fitted model files in `.path.model`. Final aggregation consumes the selected snapshots rather than reapplying this argument.

`boundsMin, boundsMax` Response bounds forwarded to every supervised call. They apply to response and companion deliveries as documented by the called helpers; order and finiteness are not checked here.

`maxIter` Numeric number of additional chain iterations to attempt. On resume it is additional to the last completed iteration, not an absolute total cap. Positivity, integrality, finiteness, and length are not proactively checked.

`tol` Numeric strict change-score threshold. The chain stops when the current δ_t is strictly less than this value. See **Change and stopping policies**.

`patience` Numeric strike threshold used independently for consecutive non-improvement and increasing change. It is not validated as a positive integer.

`sweep` One of `"jacobi"` or `"gauss-seidel"`.

`suffix` Character suffix appended to companion columns and passed to importance extraction. It is not validated as non-empty or collision-free.

`chainGate` One of `"fixed"` or `"shadow"`. The fixed policy uses model-specific importance and an inclusive mean threshold. The shadow policy uses ssel's strict-majority single-shadow rule; it is not Boruta.

`chainThresh` Numeric normalized-importance percentage threshold used only by the fixed gate. It is forwarded without validation.

`chainStable` Value coerced with `as.integer()` to the raw active-set intersection window. Missing, nonpositive, or otherwise unsuitable values are not proactively rejected.

`resume` Logical flag. Resume occurs only when both `convergence.csv` and `Y.csv` exist; otherwise a fresh run starts. The value is tested by ordinary logical conditions rather than validated as a scalar.

Value

Invisibly, `NULL`. Results are the mutable supervised products, iteration state, convergence history, and stitched response/summary files described above.

Companion values and row classes

Let $Y_k^{(t)}(i)$ denote response k's `p = 0.5` value in iteration t's wide `Y.csv` for training key i. Let $Z_k^{(t)}(u)$ denote its `p = 0.5` value in `Yo<t>.csv` for prediction key u. These are median signed-residual-offset delivery values after interval projection, in response k's units; they are not unadjusted point predictions.

Before a fit, the helper casts the selected `Yo` responses wide and joins them to `Y` by `key`. For a common key and response, `Y` takes precedence and `Yo` fills only a missing `Y` value. Companion columns are renamed with `suffix` and joined to every original assembled CSV. `buildDataset()` removes target j's own suffixed column from that target's predictors. Its complete-case rules can remove rows with missing companions.

Jacobi and Gauss–Seidel recurrences

Let $A^{(t-1)}$ be the active companion-response set entering sweep t. In a Jacobi sweep, one augmented dataset snapshot is built before any target is refitted. Every target j therefore uses

$$\{Y_k^{(t-1)}(i) : k \in A^{(t-1)}, k \neq j\}$$

for a training row and

$$\{Z_k^{(t-1)}(u) : k \in A^{(t-1)}, k \neq j\}$$

for a prediction row.

Gauss–Seidel targets are processed in sorted response-name order. That order does not depend on the session locale: ASCII case is ignored, and `_` and `.` sort before any other character. The augmented CSVs are rebuilt before every target. Earlier targets' new training-row values can therefore enter later targets through the evolving `Y.csv`; later targets retain their preceding-sweep values. In contrast, the `PrevYo` path is fixed for the entire sweep, so every target uses only $Z_k^{(t-1)}(u)$ on prediction rows. Thus the immediate-update analogy applies only to training companions. No within-sweep prediction-row companion is consumed.

These recurrences are ssel package definitions related to response-as-input multi-target methods. They do not inherit published regressor-chain ordering, out-of-fold, convergence, or optimality claims.

Active companion policy

This policy is independent of the `ensemble.RMSE` metric rows used for change history and final stitching. After each completed fit, `"fixed"` calls `extractChainImportance()` with `useModel = TRUE`, then applies `activeByImportance()` at `chainThresh`. The `"shadow"` policy calls the extractor with `useModel = FALSE` and applies `activeByShadow()`. Both routes scan individual `.Rds` model files in `.path.model`; neither extracts importance from an ensemble row. The shadow route is a strict-majority comparison to one injected Gaussian predictor and does not implement Boruta's iterative random probes or statistical tests.

An empty raw gate result is first replaced by all declared responses. If enough history exists, the current raw set is then intersected with the preceding `chainStable - 1` returned active sets. These stored sets can already reflect earlier intersections. A new intersection can still be empty and is recorded with active-set size zero. If another iteration is attempted from that empty set, augmentation reaches data.table column renaming with no companion columns and errors before the supervised fit. The pipeline invokes `computeActiveByImportance()` with its stability disabled and applies this common history rule itself for both gate types.

Change and stopping policies

For metrics, only `ensemble.RMSE` rows are considered and the first minimum-RMSE dataset row is retained for each response. Let $R^2_{r,t}$ be that row's squared sample correlation. An iteration is improving only when its unweighted mean across responses is strictly greater than the mean at the historical accepted metric baseline. That baseline is replaced only on strict improvement.

With response columns of `Y.csv` compared by current row and column position, ssel defines

$$\delta_t = \max_{i,r} \frac{|Y_r^{(t)}(i) - Y_r^{(t-1)}(i)|}{\max\{1, |Y_r^{(t-1)}(i)|\}},$$

removing missing cells in the final maximum. Compatible table order is therefore a precondition; no key join aligns the two matrices. Although numerically dimensionless, δ_t is not invariant to response-unit rescaling or translation because of the fixed floor 1. It is a package stopping heuristic, not proof that an equation solver converged.

Non-improvement strikes reset after an improving iteration. Divergence strikes increment only when δ_t strictly exceeds the preceding value and reset otherwise. After all current artifacts are written, stop tests run in this order: current change strictly below `tol`, divergence strikes at `patience`, then non-improvement strikes at `patience`. If every cellwise change is missing, the final `max(..., na.rm = TRUE)` produces `-Inf`; that value is written and satisfies `Delta < tol` for a finite `tol`, so the tolerance stop can fire. If the current or accepted mean R2 is missing, the strict comparison yields `NA` and the subsequent `if (!OK)` condition errors before the current convergence row and snapshots are written. Other invalid or non-finite inputs retain their underlying base R and data.table comparison behavior. Stored change and metric values are rounded as described in **Files and final stitch**.

Resume behavior

A fresh run recursively removes `.path.iter`, runs iteration zero on the original assembled CSVs, computes the first active set, and writes its snapshots. A recognized resume additionally requires current summary `metrics.csv`; absence stops with a diagnostic. Older convergence files lacking `nActive` are rewritten with that missing column.

Resume takes the largest recorded iteration, last recorded change, current summary metrics, and the matching `Yo` snapshot when present. It resets both strike counters, does not reconstruct active-set history, and sets the first resumed active set to `NULL`; that first augmentation therefore retains every available companion. It attempts `maxIter` further iterations. Resume state and subsequent writes are not transactional.

Files and final stitch

Iteration zero copies `Y0.csv`, `Yo0.csv`, `metrics_0.csv`, and a `summary_0` directory. Every later iteration writes `datasets_t`, `Yt.csv`, `Yot.csv`, `metrics_t.csv`, `summary_t`, and a convergence row. Summary snapshots cover `metrics.csv`, `response_long.csv`, `residuals_oof.csv`, `prediction_quantiles.csv`, and `overfit.csv`. Missing source summary files are simply absent from a snapshot. Convergence rows contain iteration, change rounded to eight decimals, mean RMSE to six, mean R2 and per-response R2 to four, improvement, strike count, and active-set size.

At completion, response r selects the first maximum R2 among baseline iteration zero and recorded iterations. Baseline R2 is read without rounding; later candidates use the four-decimal convergence values, so rounding can affect selection. `Yo.csv` and all five current summary files are then

overwritten response by response from their selected snapshots. Missing snapshots or a missing `response` column stop the stitch.

Models, TRAIN/TEST splits, cell metric/response files, `Y.csv`, `X.csv`, and `Xo.csv` remain from the last fitted iteration; they are not stitched. Filesystem deletion, fitting, and writes have no rollback. Progress uses ssel's condition channels. RNG and bounded PSOCK effects are those of the composed supervised helpers; each fitted iteration receives the seed described above.

References

Spyromitros-Xioufis, E., Tsoumakas, G., Groves, W., and Vlahavas, I. (2016). Multi-target regression via input space expansion: treating targets as inputs. *Machine Learning*, 104, 55–98. doi:10.1007/s109940165546z.

See Also

`modelPipeline()` for each supervised refit, `aggregateResponses()` for `Y`/`Yo` construction, `extractChainImportance()` for model-file importance, and `activeByImportance()` and `activeByShadow()` for the two table gates.

Examples

```
root <- tempfile("ssel-chain-")
datasets_dir <- file.path(root, "datasets")
dir.create(datasets_dir, recursive = TRUE)

assembled <- data.frame(
  SampleID = c(paste0("S", 1:18), "P1", "P2"),
  set = c(rep("train", 18), "predict", "predict"),
  x = 1:20,
  z = rep(c(0, 1), 10),
  targetA = c(1.0, 2.4, 1.8, 3.7, 2.9, 5.1, 4.0, 6.2, 5.4,
              7.0, 6.1, 8.3, 7.2, 9.1, 8.0, 10.5, 9.4, 11.2,
              NA, NA),
  targetB = c(2.2, 1.3, 3.6, 2.7, 4.1, 3.8, 5.7, 4.6, 6.5,
              5.9, 7.8, 6.4, 8.6, 7.5, 9.9, 8.7, 10.4, 9.6,
              NA, NA)
)
utils::write.csv(assembled, file.path(datasets_dir, "demo.csv"),
                 row.names = FALSE, na = "")

suppressMessages(chainPipeline(
  .path.datasets = datasets_dir,
  .path.train = file.path(root, "train"),
  .path.model = file.path(root, "model"),
  .path.metrics = file.path(root, "metrics"),
  .path.response = file.path(root, "response"),
  .path.summary = file.path(root, "summary"),
  .path.export = file.path(root, "export"),
  .path.iter = file.path(root, "iter"),
  key = "SampleID",
  responses = c("targetA", "targetB"),
```

```
  datasets = "demo",
  methods = "lm",
  seed = 2026,
  maxIter = 1,
  tol = 0,
  patience = 2,
  sweep = "jacobi"
))

data.table::fread(file.path(root, "iter", "convergence.csv"))
data.table::fread(file.path(root, "export", "Yo.csv"))
# Final Yo rows can come from different best iterations by response.

unlink(root, recursive = TRUE)
```

`computeActiveByImportance` *Extract, threshold, and intersect fixed-gate chain responses*

Description

Compose model-specific companion-importance extraction with `activeByImportance()` and an optional intersection over supplied preceding active sets. This convenience wrapper implements only the fixed threshold policy. `chainPipeline()` separately applies its fallback and common history policy around this result.

Usage

```
computeActiveByImportance(pathModel, suffix = ".o", thresh, stableK,
  history)
```

Arguments

`pathModel`	Directory passed to `extractChainImportance()`.
`suffix`	Companion suffix passed to extraction and positional response name removal in `activeByImportance()`.
`thresh`	Numeric inclusive mean normalized-importance threshold in percentage points. It has no default and is forwarded without validation.
`stableK`	Value compared with 1 and used as the number of current plus supplied preceding sets to intersect. It is not coerced or validated here.
`history`	List-like object whose final `stableK - 1` elements are supplied to `intersect()` after the current raw set. The function does not establish whether these are raw or previously stabilized sets.

Value

A character vector of stripped response names, in current fixed-gate order, that survive the optional intersections. It can be empty.

Composition rule

Let G_0 be the character vector returned by `activeByImportance(extractChainImportance(...), thresh = thresh)`. Extraction always uses its default `useModel = TRUE`. If `stableK <= 1`, or fewer than `stableK - 1` history elements are available, the helper returns G_0 unchanged. Otherwise, for the retained history tail $G_1, \ldots, G_{K-1}$, it returns

$$G_0 \cap G_1 \cap \cdots \cap G_{K-1}.$$

Base `intersect()` preserves the current vector's order while removing duplicate names. No fallback replaces an empty extraction or intersection.

In `chainPipeline()`, this wrapper is called with `stableK = 1` and an empty history. The pipeline first replaces an empty raw result with all declared responses, then applies its own window by intersecting the current set with preceding returned active sets. Consequently, standalone wrapper output and pipeline fallback/stability must not be described as one rule.

Conditions and side effects

Validation and failures are inherited from extraction, the fixed table gate, `tail()`, `Reduce()`, and `intersect()`. In particular, a missing `stableK` comparison can error, and unusual history element types retain base R behavior. The helper reads models through the extractor but writes no file, sets no seed, draws no random value directly, and emits no deliberate condition.

See Also

`extractChainImportance()` for model parsing and normalization, `activeByImportance()` for threshold semantics, and `chainPipeline()` for pipeline fallback and the common fixed/shadow history policy.

Examples

```
root <- tempfile("ssel-chain-stability-")
dir.create(root)
train <- data.frame(
  alpha.o = 1:12,
  beta.o = rep(c(0, 1), 6),
  x = rep(1:4, 3),
  y = c(2.1, 1.7, 3.5, 2.8, 5.2, 4.1,
        6.4, 5.3, 7.1, 6.8, 8.2, 7.5)
)
fit <- caret::train(
  y ~ ., data = train, method = "lm",
  trControl = caret::trainControl(method = "none")
)
saveRDS(fit, file.path(root, "lm_target_demo.Rds"))

computeActiveByImportance(
  pathModel = root,
  suffix = ".o",
  thresh = 0,
  stableK = 2,
```

```
  history = list("alpha")
)
# Only a currently active name also present in the supplied set survives.

unlink(root, recursive = TRUE)
```

`describeModule` *Describe the installed module*

Description

Diagnoses the installation of the ssel command without repairs; the installed command prints this report for `ssel doctor`. Every check runs even when another one fails, so the report is always complete.

Usage

```
describeModule(resources)
```

Arguments

`resources`	The resource directory of the installed command, `PREFIX/libexec/ssel`, or `NULL` outside an installed product, where the checks that read it fail.

Value

A list with `schema` (1), `package`, `version` (the version that loads), `summary` and `checks`. Each check has `name`, `required`, `status` (`ok` or `fail`) and `detail`, its result or the reason it failed. The required checks are `library` (the version that loads is the one of `lib/DESCRIPTION`), `exports` (the exports that `install/requirements.R` declares exist), `packages` (the packages it declares load), `tools` (the tools it declares are on `PATH`) and `receipt` (the files that the receipt `install.json` records keep their MD5). `summary` holds `status`, `fail` when a required check fails and `ok` otherwise. Reads `resources` and the files of its receipt; writes nothing.

Examples

```
# Outside an installed command the checks of the installation fail with
# their reason, and the report stays complete.
Report <- describeModule(NULL)
Report$summary$status
vapply(Report$checks, function(x) paste(x$name, x$status), "")
```

detectOutliers *Flag regression training outliers*

Description

Adds robust response, local-outlier-factor, and optional Cook's-distance scores to a regression training table.

Usage

```
detectOutliers(DT, threshold.mad = 3.5, threshold.lof = 2.0,
  threshold.cooks = Inf, minPts = 5, use.cooks = FALSE)
```

Arguments

DT	Training data.table containing numeric response column y.
threshold.mad	Modified Z-score threshold.
threshold.lof	Local outlier factor threshold.
threshold.cooks	Cook's distance threshold, or Inf for default rule.
minPts	Minimum points for local outlier factor.
use.cooks	Logical; include Cook's distance.

Value

A copy of DT with .score.mad, .score.lof, and .outlier columns; .score.cooks is added when requested.

extractChainImportance *Extract normalized importance for response companions*

Description

Read fitted model files, ask caret::varImp() for each model's importance table, normalize scores within that table, and retain features ending in a configured companion suffix. This helper produces the table consumed by the fixed and shadow chain gates; it does not itself select active responses.

Usage

```
extractChainImportance(pathModel, suffix = ".o", useModel = TRUE)
```

Arguments

pathModel	Directory scanned non-recursively for case-sensitive filenames ending in .Rds.
suffix	Character suffix matched with endsWith() against every importance feature name. Length, missingness, emptiness, and collisions are not proactively validated.
useModel	Value passed unchanged to caret::varImp(model, useModel = useModel). TRUE requests caret's model-specific path and is used by the fixed gate. FALSE requests caret's model-independent regression path and is used by the shadow gate. The helper does not validate a logical scalar.

Value

A data.table with one row per retained model–companion pair and columns feature, response, dataset, method, normImportance, and shadowImportance. Importance columns are in normalized percentage points. If no row is retained, returns an empty zero-column data.table.

File parser and skips

A model basename is split on underscores after removing its extension. The first field becomes method, the last becomes dataset, and all interior fields joined by underscores become response. Method and dataset identifiers therefore cannot be reconstructed when they contain underscores; a response may contain them. A valid basename must have at least three fields: method, one or more response fields, and dataset. This precondition is not validated. With two fields, for example lm_demo.Rds, the index sequence reverses and the response becomes demo_lm; with one field, lm.Rds, it becomes NA_lm while both method and dataset are lm. Such malformed stems can therefore contribute spurious response labels rather than being rejected. Files are processed in list.files() order.

An unreadable model, a caught caret::varImp() error, an empty importance table, or a table with no suffix-matching feature contributes no row. The returned caret table is expected to become columns rn and Overall; incompatible schemas fail at column renaming rather than being skipped. Apart from the unvalidated short-stem behavior above, other type, filesystem, and data.table errors propagate.

Normalization and shadow field

For model file c, let I_{fc} be caret's Overall score for feature f, and let

$$T_c = \sum_f I_{fc},$$

removing missing scores. When $T_c > 0$, ssel writes

$$J_{fc} = 100 I_{fc} / T_c$$

rounded to four decimals; otherwise it writes zero for every row. The sum is over all importance rows before companion filtering, so retained companion percentages need not sum to 100. Negative, infinite, duplicated, or otherwise unusual caret scores are not range-checked.

The exact feature name `shadow`, when it occurs once, supplies that model's normalized `shadowImportance` on the same row-local scale. Zero or multiple exact matches yield a missing shadow score. Only features whose names end in `suffix` are returned. Their names are not stripped here.

Model-specific importance definitions can differ across learners. This normalization is a package aggregation convention and does not establish cross-learner comparability. Likewise, the model-independent/shadow path is one package null-reference input; it is not the Boruta algorithm.

Side effects and return

The helper reads model files and calls their caret importance methods. It writes no file, sets no seed, draws no random value directly, and emits no deliberate progress condition. Caught model/importance failures are silent.

See Also

`activeByImportance()` for the fixed table gate, `activeByShadow()` for the strict shadow-majority table gate, and `computeActiveByImportance()` for fixed-gate composition.

Examples

```
root <- tempfile("ssel-chain-importance-")
dir.create(root)
train <- data.frame(
  alpha.o = 1:12,
  beta.o = rep(c(0, 1), 6),
  x = rep(1:4, 3),
  y = c(2.1, 1.7, 3.5, 2.8, 5.2, 4.1,
        6.4, 5.3, 7.1, 6.8, 8.2, 7.5)
)
fit <- caret::train(
  y ~ ., data = train, method = "lm",
  trControl = caret::trainControl(method = "none")
)
saveRDS(fit, file.path(root, "lm_target_demo.Rds"))

importance <- extractChainImportance(root, suffix = ".o")
importance[, .(feature, response, dataset, normImportance)]
# Percentages use all model features in their denominator.

unlink(root, recursive = TRUE)
```

---

modelPipeline *Orchestrate the supervised ssel workflow*

Description

Compose split construction, fitting, evaluation, residual-offset and optimism audits, response-level selection, and final aggregation. This function is filesystem orchestration around public helpers; it is not a separate statistical estimator and it does not read a project manifest.

Usage

```
modelPipeline(
  .path.datasets,
  .path.train,
  .path.model,
  .path.metrics,
  .path.response,
  .path.summary,
  .path.export,
  .path.iter = NULL,
  key,
  responses,
  datasets,
  colsId = character(0),
  features = NULL,
  setCol = "set",
  methods = c("ranger", "earth", "gbm"),
  seed = NULL,
  ensemble = "ensemble.RMSE",
  boundsMin = -Inf,
  boundsMax = Inf,
  iterSuffix = NULL,
  mode = "full",
  responsesActive = NULL,
  wipe = NULL,
  shadow = FALSE
)
```

Arguments

`.path.datasets`	Existing directory of assembled CSV files. Every non-recursive filename ending in `.csv` is processed by `buildDataset()`, whether or not its basename occurs in `datasets`.
`.path.train`, `.path.model`, `.path.metrics`, `.path.response`, `.path.summary`, `.path.export`	Output directories for splits, fitted models, cell metrics, cell response predictions, summary products, and final exports. Missing directories are created recursively after any configured wipe. See **Wipe and cache policy** for destructive behavior.
`.path.iter`	`NULL`, or an iteration directory. When supplied, `aggregateResponses()` writes or updates `Y.csv` there for a later chain iteration.
`key`	A non-empty character key name passed to `buildDataset()` and `aggregateResponses()`. Final response aggregation currently requires a column named `SampleID` independently of this argument.
`responses`	Non-empty character vector of assembled response-column names. Names are inserted unescaped into regular expressions used for split and model discovery.

`datasets` Non-empty character vector of assembled dataset basenames used only by final `aggregateResponses()` feature extraction. Among these declared files, the last one that exists supplies `X.csv` and `Xo.csv`. This vector does not restrict upstream split construction, fitting, OOF reconstruction, or prediction.

`colsId` Character vector of additional identifier columns excluded from inferred predictors.

`features` `NULL`, or exact base predictor names passed to `buildDataset()`. `NULL` delegates predictor inference to that helper.

`setCol` Non-empty character name of the assembled row-role column, expected to distinguish `"train"` and `"predict"`.

`methods` Character vector of fitted caret method identifiers. The default is `c("ranger", "earth", "gbm")`.

`seed` `NULL`, or a value passed to `set.seed()` before split construction in every mode. It controls fitting/resampling and other downstream random operations only to the extent documented by each helper.

`ensemble` Response-level selector for `predictModel()`: `"ensemble.RMSE"` or `"ensemble.R2"`, checked before filesystem changes in `"full"` and `"predict"` modes. This argument is unused in `"train"` mode. Final `aggregateResponses()` output uses `ensemble.RMSE` candidates.

`boundsMin, boundsMax` Numeric response bounds forwarded through OOF, point-prediction, audit, and aggregation paths. Each must be a single non-`NA` numeric value; `Inf` and `-Inf` are allowed and are the defaults. Their order and finiteness are not validated.

`iterSuffix` `NULL`, or the suffix identifying prior-iteration response features in `buildDataset()`, commonly `".o"`.

`mode` A value in `c("full", "train", "predict")`.

`responsesActive` `NULL`, or a value used without subset or type validation to filter fitting, OOF/audit reconstruction, and prediction. `NULL` uses `responses`. All `responses` are still split and passed to final aggregation. See **Partial-response state**.

`wipe` `NULL`, or any value. `NULL` and the literal logical scalar `TRUE` enable wiping; every other value disables it through `isTRUE()`. See **Wipe and cache policy**.

`shadow` Logical value passed to `buildDataset()` to add its Gaussian shadow predictor.

Value

Invisibly, `NULL`. Results are the files produced by the selected mode and its composed helper calls.

Mode composition

Initial validation checks the type/non-emptiness conditions shown for `key`, `responses`, `datasets`, `setCol`, and non-null `features`, plus membership of `mode` and the mode-dependent `ensemble` conditions above. Each response bound must be a single non-`NA` numeric value; their order and finiteness are not validated. Initial validation leaves `responsesActive`, `methods`, `wipe`, and `shadow` unchecked. A non-null seed is passed directly to `set.seed()`. Other conditions are delegated to the called helpers and can therefore fail after destructive or expensive work.

Every mode prepares directories and calls buildDataset(), which scans every CSV in .path.datasets. The response-column expression is the unescaped anchored alternation ^(r1|r2|...)$, applied case-insensitively. Fitting and prediction then discover dataset identifiers independently from the generated split filenames. Active response filtering of those names is exact and case-sensitive; the OOF and overfit model-file parsers receive an unescaped, unanchored active-response alternation inside their case-sensitive regular expressions.

- "train" then calls trainModel() for active responses and stops. No OOF audit, prediction, overfit audit, or aggregation runs.
- "predict" skips fitting, then calls auditQuantiles(), predictModel(), auditOverfit(), and aggregateResponses() in that order. Existing compatible models are therefore required.
- "full" performs the training call followed by the complete prediction sequence above.

The offset audit must precede prediction because prediction consumes prediction_quantiles.csv. Aggregation then consumes unadjusted long point predictions, OOF residual rows, metric rows, and offsets. The overfit audit is diagnostic; aggregation does not use overfit.csv.

Wipe and cache policy

With wipe = NULL or true, the function recursively removes and recreates .path.train, .path.metrics, .path.response, and .path.summary in every mode. In "full" mode only, it also recursively removes .path.model and .path.export; this same condition sets OVERRIDE = TRUE for fitting. These operations are destructive and have no rollback.

In "train" or "predict" mode, wiping preserves existing model and export directories. Training mode then uses OVERRIDE = FALSE, so surviving models can be reused under trainModel()'s predictor-name-only cache rule. With wipe = FALSE, every existing output is preserved, missing directories are created, and full-mode fitting also uses OVERRIDE = FALSE. Current metric, residual, export, and iteration helpers then apply their own merge or overwrite rules.

Products, selectors, and state

Every mode can overwrite split files. Train mode can additionally write fitted model and hash files, then returns without summary or delivery calls. Prediction mode never fits or writes model/hash files; it can write cell metrics and responses, metrics.csv, response_long.csv, residuals_oof.csv, prediction_quantiles.csv, overfit.csv, Yo.csv, X.csv, Xo.csv, optional Y.csv, and dataset-named intermediate exports. Full mode can write all train- and prediction-mode products.

The ensemble argument governs only predictModel()'s per-response dataset selector. aggregateResponses() independently forms its final rowwise selection from ensemble.RMSE candidates and cell R2. These selectors must not be described as one common "best model" rule.

Supplying seed changes the caller's RNG state. Both fitting and the evaluation engine invoked by predictModel() register a bounded PSOCK backend and leave foreach sequential under their accepted contracts. Shadow generation and prediction method randomization have their documented RNG effects. Errors and partial files from any helper propagate; orchestration is not transactional.

Partial-response state

With a non-null responsesActive, the helpers do not follow one common accumulation rule. auditQuantiles() replaces residual rows for active responses in an existing residuals_oof.csv

and regenerates offsets from the merged residual table. predictModel() overwrites response_long.csv with current active-response predictions; its metrics.csv is the union of every cell-metric CSV currently present. auditOverfit() overwrites overfit.csv from models matching the active-response expression. aggregateResponses() preserves old Yo.csv rows only for responses absent from the current unseen result, and optional Y.csv preserves old non-overlapping response columns; both use their own replacement rules. X.csv and Xo.csv are overwritten from the last existing declared dataset. Thus wipe = FALSE preserves inputs for mixed accumulation but does not make every summary cumulative.

See Also

buildDataset() for assembled input, trainModel() for fitting, predictModel() for the first selector, and aggregateResponses() for final delivery.

Examples

```
root <- tempfile("ssel-pipeline-")
datasets_dir <- file.path(root, "datasets")
dir.create(datasets_dir, recursive = TRUE)

assembled <- data.frame(
  SampleID = c(paste0("S", 1:16), "P1", "P2"),
  set = c(rep("train", 16), "predict", "predict"),
  x = 1:18,
  z = rep(c(0, 1), 9),
  target = c(1:16 / 3 + rep(c(-0.5, 0.5), 8), NA, NA)
)
utils::write.csv(
  assembled,
  file.path(datasets_dir, "demo.csv"),
  row.names = FALSE,
  na = ""
)

suppressMessages(modelPipeline(
  .path.datasets = datasets_dir,
  .path.train = file.path(root, "train"),
  .path.model = file.path(root, "model"),
  .path.metrics = file.path(root, "metrics"),
  .path.response = file.path(root, "response"),
  .path.summary = file.path(root, "summary"),
  .path.export = file.path(root, "export"),
  key = "SampleID",
  responses = "target",
  datasets = "demo",
  methods = "lm",
  seed = 2026,
  ensemble = "ensemble.RMSE",
  mode = "full"
))

sort(basename(list.files(file.path(root, "summary"))))
```

```
data.table::fread(file.path(root, "export", "Yo.csv"))

unlink(root, recursive = TRUE)
```

`oofEnsemble` *Reconstruct a projected OOF ensemble*

Description

Recover the selected-tuning out-of-fold (OOF) predictions stored in fitted caret models and reconstruct ssel's inverse-RMSE ensemble for every response–dataset cell. Use this helper to inspect training-row ensemble residuals. It recomputes cell-local weights from the saved OOF predictions; it does not reuse the unseen-row weight state described by `trainRegressionModel()`.

Usage

```
oofEnsemble(.path.model, .path.train,
  .path.datasets = file.path("data", "datasets"), .min = -Inf, .max = Inf,
  .response_pattern = NULL, .sort_keys = character(0))
```

Arguments

`.path.model`	Path to an existing directory containing model files named `method_response_dataset.Rds`. The method identifier must be a non-empty string without an underscore. Matching files must contain an object with caret-compatible `pred` and `bestTune` components.
`.path.train`	Path to an existing directory containing `response_dataset_TRAIN.csv` and, preferably, `response_dataset_TRAIN_ids.csv` files. A TRAIN file must exist for a model to be eligible. Its values are read but are not used to replace the observed responses stored in the model.
`.path.datasets`	Path to assembled `dataset.csv` files used only by the compatibility mapping when a TRAIN-ID sidecar is absent.
`.min`, `.max`	Numeric lower and upper response bounds, in response units. Predictions are projected with `pmax(.min, pmin(.max, z))`. The function does not validate finiteness or `.min <= .max`.
`.response_pattern`	A non-empty character regular expression inserted into the model-filename parser to identify responses. Use noncapturing groups when grouping is needed: additional capturing groups change the parser's field positions. The default `NULL` deliberately fails.
`.sort_keys`	Character vector of columns that must already define the assembled-dataset row order when the compatibility ID mapping is used. An empty vector disables the order check.

Value

A `data.table` with one retained row per reconstructed cell and distinct `(rowIndex, obs)` cast key. Columns are `response`, `dataset`, integer `rowIndex`, character `SampleID`, `y_obs`, `y_hat_oof`, and `residual`. The final three columns have the response's units. Row indices and sample IDs can repeat when saved observations disagree across methods. If no row is reconstructed, returns an empty zero-column table.

OOF reconstruction

For a matching model, the function inner-joins `model$pred` to `model$bestTune` by every best-tuning column. It converts `pred`, `obs`, and `rowIndex` to numeric or integer values and retains only rows where all three are finite. Predictions are then projected onto the configured bounds. Repeated rows for one method, response, dataset, and row index are reduced by taking separate arithmetic means of `obs` and the projected prediction.

Let $\Pi_{[a,b]}(z) = \min\{b, \max\{a, z\}\}$ be interval projection, let y_{mi} be the saved observed value, and let $\tilde{y}_{mi}$ be method m's averaged projected OOF prediction at row i. The observed value is method-specific at this stage; equality across methods is not checked. In cell (r, d), the helper computes

$$\widetilde{\mathrm{RMSE}}_{mrd} = \left[n_m^{-1} \sum_i (y_{mi} - \tilde{y}_{mi})^2 \right]^{1/2}$$

and defines the package weights

$$\tilde{w}_{mrd} = \frac{1/\widetilde{\mathrm{RMSE}}_{mrd}}{\sum_j (1/\widetilde{\mathrm{RMSE}}_{jrd})}.$$

These weights are a package reconstruction policy, not an optimality claim. A cell is skipped when any method-level RMSE is non-finite or zero. A one-method cell is retained with weight one.

The wide reconstruction is keyed jointly by `rowIndex` and the averaged saved `obs`. Let $h = (i, o)$ denote one distinct joint key and let A_h be the methods present at that key. The reconstructed value is

$$\hat{y}_h^{OOF} = \Pi_{[a,b]} \left(\frac{\sum_{m \in A_h} \tilde{w}_m \tilde{y}_{mh}}{\sum_{m \in A_h} \tilde{w}_m} \right).$$

For a complete key, A_h contains every cell method and the denominator is one. A key with no method prediction is excluded. If methods supply different averaged observed values for the same row index, they create separate keys, each is aggregated with its available methods, and the result can contain repeated `rowIndex` and `SampleID` values. The function does not validate a common observed response across methods. The returned signed residual is $e_h = o - \hat{y}_h^{OOF}$, in response units.

Sample identifiers

The preferred mapping reads the `SampleID` column from `response_dataset_TRAIN_ids.csv`; row i supplies the identifier for caret `rowIndex = i`. No sidecar length or uniqueness check is made before indexing, but an OOF index beyond its length stops the call.

If the sidecar is absent, the compatibility path reads `dataset.csv`. That file must contain `SampleID`, the response, and every `.sort_keys` column. When sort keys are supplied, the file must already

equal its ordering by those keys. Identifiers are then taken only from rows whose response is non-missing. This older mapping is safe only when TRAIN rows are exactly those assembled rows, in unchanged order, with no additional removal or reordering. Feature complete-case filtering, outlier removal, or any other extra filter can silently shift identifiers even when the maximum-index check passes; the sidecar is required in those cases. A missing dataset, missing required column, broken ordering invariant, or out-of-range OOF index stops the call.

Skips, ordering, and side effects

A filename that does not match `.response_pattern` is filtered out silently: excluding responses is what the pattern is for. A file that does match and still cannot be used – absent TRAIN file, unreadable model, failed best-tune merge, empty saved predictions, or no finite saved row – is skipped and reported by a warning naming the file and the reason. A cell can also disappear under the RMSE and row-availability rules above; a warning then reports how many of the cells were reconstructed. A third warning reports a reconstruction that produced nothing at all, distinguishing a pattern that matched no filename from matches that were all skipped. If nothing remains, the function returns an empty zero-column `data.table`. Result order is an implementation consequence and is not a sorting guarantee.

The helper reads files but writes none, does not fit models, and does not alter the random-number state. Base R, data.table, and file-reading errors not listed above propagate.

References

Kuhn, M. (2008). Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5), 1–26. doi:10.18637/jss.v028.i05.

Parikh, N., and Boyd, S. (2014). Proximal Algorithms. *Foundations and Trends in Optimization*, 1(3), 127–239. doi:10.1561/2400000003.

See Also

`trainRegressionModel()` for producing the fitted caret objects and saved OOF predictions, `buildDataset()` for TRAIN-ID sidecars, and `auditQuantiles()` for writing residual-offset summaries.

Examples

```
root <- tempfile("ssel-oof-")
model_dir <- file.path(root, "model")
train_dir <- file.path(root, "train")
dir.create(model_dir, recursive = TRUE)
dir.create(train_dir)

data.table::fwrite(
  data.table::data.table(y = c(1, 3)),
  file.path(train_dir, "target_demo_TRAIN.csv")
)
data.table::fwrite(
  data.table::data.table(SampleID = c("S1", "S2")),
  file.path(train_dir, "target_demo_TRAIN_ids.csv")
)
save_oof <- function(method, prediction) {
```

```
  fit <- list(
    pred = data.frame(
      tune = 1L, pred = prediction, obs = c(1, 3), rowIndex = 1:2
    ),
    bestTune = data.frame(tune = 1L)
  )
  saveRDS(fit, file.path(model_dir, paste0(method, "_target_demo.Rds")))
}
save_oof("lower", c(0, 2))
save_oof("mixedCase", c(0.5, 2.5))

oof <- oofEnsemble(
  .path.model = model_dir,
  .path.train = train_dir,
  .response_pattern = "target"
)
oof[, .(SampleID, y_obs, y_hat_oof, residual)]
# Both learners have RMSE 1 and 0.5, so their weights are 1/3 and 2/3.

unlink(root, recursive = TRUE)
```

`postProcess` *Assemble user-facing prediction deliverables.*

Description

Builds two CSVs in `.path.export`: `predictions.csv` (wide best per response, optional Transformed twin) and `audit.csv` (long disaggregated per method, raw values).

Usage

```
postProcess(
  .path.summary,
  .path.export,
  key,
  responses,
  location,
 locationSources = c("data/domains/index.csv", file.path(.path.export, "X.csv"),
    file.path(.path.export, "Xo.csv")),
  transform = NULL,
  normalize = NULL,
  boundsMin = -Inf,
  boundsMax = Inf
)
```

Arguments

`.path.summary`	Directory holding ssel computed aggregates (`response_long.csv`, `residuals_oof.csv`, `metrics.csv`, `prediction_quantiles.csv`).

.path.export	Directory holding ssel intermediate Yo.csv and where outputs (predictions.csv, audit.csv) are written. Created if missing.
key	Identity column name (typically "SampleID").
responses	Character vector of response column names.
location	Character vector of location columns to carry from the project's identity source into predictions.csv / audit.csv.
locationSources	Character vector of candidate file paths from which to read (key, location). All existing files carrying the required columns are combined in source order, retaining the first row for each key.
transform	Element-wise transform spec, a list with op and op-specific params. Supported: list(op="identity") (default when NULL), list(op="log", base=10), list(op="exp", base=exp(1)).
normalize	Row-wise normalize spec. Supported: list(op="identity") (default), list(op="closure", target=100).
boundsMin	Lower clamp for predicted values. Default -Inf.
boundsMax	Upper clamp for predicted values. Default Inf.

Value

Invisible NULL. Side effect: writes predictions.csv and audit.csv to .path.export.

---

predictModel *Evaluate fitted cells and select response-level deliveries*

Description

Run trainRegressionModel() in evaluation mode, collect its point predictions and OOF metrics, select one response–dataset cell per response, and add matching signed OOF-residual offsets. This is the first delivery selector. The later aggregateResponses() selector has a different candidate set and grouping rule.

Usage

```
predictModel(.path.train, .path.model, .path.metrics,
  .path.response, .path.summary, .path.export, .file.index = NULL,
  METHODS, PPROC = NULL, probs = c(0.50, 0.90, 0.95),
  .min = -Inf, .max = Inf, .ensemble = "ensemble.R2",
  .deliverable_datasets = NULL, .responses = NULL)
```

Arguments

.path.train	Path to an existing split directory. Training files are discovered non-recursively by the suffix _TRAIN.csv. Their first two underscore-separated fields are interpreted as response and dataset, so those identifiers cannot contain underscores.
.path.model	Existing directory containing the fitted model files required by trainRegressionModel() evaluation mode.
.path.metrics	Existing writable directory for cell metric CSVs. Evaluation can overwrite assigned files; this helper then reads every CSV in the directory and overwrites metrics.csv under .path.summary with their unique union.
.path.response	Existing writable directory for cell response CSVs. Evaluation can overwrite assigned files; matching files are then read to assemble the long prediction table.
.path.summary	Summary-output directory. It is created non-recursively when absent, so its parent must exist. A compatible prediction_quantiles.csv from auditQuantiles() must already exist.
.path.export	Export directory. It is created non-recursively when absent, so its parent must exist. Assigned dataset CSVs are overwritten.
.file.index	NULL, a nonexistent path, or a CSV index path. When an existing file is supplied, its first column is the join key used to rewrite every CSV under .path.summary and .path.export that contains that name.
METHODS	Non-empty character vector of fitted caret method identifiers. Duplicates are removed and order is randomized before evaluation.
PPROC	Forwarded as .preProcess to evaluation mode. The current trainRegressionModel() evaluation path does not use preprocessing.
probs	Numeric vector of offset probabilities. Each value is converted to an integer percentage with as.integer(100 * p) and requires a matching q_XX column in prediction_quantiles.csv. The standard producer writes only 0.50, 0.90, and 0.95.
.min, .max	Numeric response bounds. Base point predictions are projected by the evaluation engine; each point-plus-offset value is projected again. Each must be a single non-NA numeric value; Inf and -Inf are allowed. Bound order and finiteness are not validated.
.ensemble	A value in c("ensemble.R2", "ensemble.RMSE"). It selects both the metric and method label used by the response-level selector. The default is "ensemble.R2".
.deliverable_datasets	NULL, or a character vector restricting dataset eligibility only for response-level selection. It does not change split discovery, evaluation, long outputs, or the list of dataset-named files considered for export.
.responses	NULL, or a character vector restricting discovered responses before evaluation. NULL uses all parsed responses.

Value

Invisibly, a list with DTL, the unadjusted long point table; DTW, the selected wide point-plus-offset table; and DT.metrics, the deduplicated union of metric CSVs before any index rewrite.

Evaluation and assembly

Discovered response and dataset names are deduplicated independently and passed to `trainRegressionModel()` with `TRAIN = FALSE`. The method vector is sampled with R's current random-number generator, so this call advances RNG state even though fitted models are reused. The engine reconstructs OOF metrics and predicts unseen split rows according to its accepted reference contract. Because this wrapper does not override the engine's `PARALLEL` argument, evaluation registers a PSOCK backend with at most two workers, stops it on exit, and leaves foreach sequential. A restricted environment that forbids local server sockets can therefore fail before evaluation even though no model is fitted.

Every existing metric CSV is row-bound and deduplicated into `DT.metrics`. Every existing `response_dataset.csv` under `.path.response` is read, deduplicated, and tagged with `set = "response_dataset"`. Their union, `DTL`, retains base-method and both ensemble point predictions. Missing response files are skipped. If none is read, cleanup of the never-created intermediate can error before a result is written.

Response-level selector and offsets

Let r index a response, d a dataset cell, x one unseen predictor row, and $\mathcal{D}_r$ the eligible metric rows for response r. Eligibility includes the requested ensemble label and, when supplied, membership in `.deliverable_datasets`. For the RMSE ensemble the selected row is

$$d_r^{RMSE} = \operatorname*{first\,argmin}_{d \in \mathcal{D}_r} \mathrm{RMSE}_{rd},$$

among rows labeled `ensemble.RMSE`. For the R2 ensemble it is

$$d_r^{R2} = \operatorname*{first\,argmax}_{d \in \mathcal{D}_r} R^2_{rd},$$

among rows labeled `ensemble.R2`. Missing scores are ignored when a non-missing score exists, and an extremum tie selects the first metric row. If every score in a response group is missing, or filtering leaves no row for that response, the selector contributes no selected row for it. Later offset assembly and reshaping can consequently omit that response or fail when no usable selection remains.

For the selected response and dataset, let $\hat{y}_r(x)$ be the matching ensemble point prediction and let $q_{p,rd}$ be the signed, cell-pooled OOF-residual offset at requested probability p, written by `auditQuantiles()`. Let a and b be bounds in response units, supplied by `.min` and `.max`, respectively. When $a \leq b$, interval projection is $\Pi_{[a,b]}(z) = \min\{b, \max\{a, z\}\}$ for real z. The delivered value is

$$Q_{p,r}(x) = \Pi_{[a,b]}\{\hat{y}_r(x) + q_{p,rd}\}.$$

This is a projected point-plus-residual-offset estimate. It is not a predictive-distribution quantile and carries no conditional, marginal, or conformal coverage guarantee. Exactly one matching offset row is required for every selected cell. Rows with any missing field are removed before wide reshaping. Bound order and finiteness are not checked; with invalid bounds the nested minimum/maximum remains the executed operation but is not an interval projection.

Files, index joins, and return

The helper overwrites `metrics.csv` with `DT.metrics` and `response_long.csv` with the unadjusted long point table `DTL`. For each discovered dataset, it finds sample IDs occurring in long-table

set names ending in `_dataset` and writes a dataset-named CSV when at least one selected wide row has such an ID. Dataset names enter an unescaped regular expression. Each wide file has one row per retained sample/probability, a `SampleID` column, one column per selected response, and `p`. Existing assigned files are overwritten.

If an index file is supplied, every CSV currently present under both output directories is read. Files containing the index's first column are replaced by the index-to-file data.table join, which can add columns, change order, or duplicate rows when the key is not unique. This includes pre-existing CSVs not created by the current call. In-memory return tables precede these joins.

Errors from evaluation, absent quantile files or columns, non-unique selected rows, malformed schemas, empty casts, joins, and file operations can occur after earlier assigned files were written; there is no transactional rollback. The files written are reported as messages of class `moduleDetail`.

References

Hyndman, R. J., and Fan, Y. (1996). Sample quantiles in statistical packages. *The American Statistician*, 50(4), 361–365. doi:10.1080/00031305.1996.10473566.

Parikh, N., and Boyd, S. (2014). Proximal Algorithms. *Foundations and Trends in Optimization*, 1(3), 127–239. doi:10.1561/2400000003.

See Also

`trainRegressionModel()` for evaluation and ensemble point predictions, `auditQuantiles()` for the offset estimator, and `aggregateResponses()` for the later RMSE-candidate rowwise selector.

Examples

```
root <- tempfile("ssel-predict-")
train_dir <- file.path(root, "train")
model_dir <- file.path(root, "model")
metrics_dir <- file.path(root, "metrics")
response_dir <- file.path(root, "response")
summary_dir <- file.path(root, "summary")
export_dir <- file.path(root, "export")
for (path in c(train_dir, model_dir, metrics_dir, response_dir,
               summary_dir, export_dir)) {
  dir.create(path, recursive = TRUE)
}

train <- data.frame(
  x = 1:6,
  y = c(1, 2, 2, 4, 5, 7)
)
test <- data.frame(x = c(7, 8), SampleID = c("P1", "P2"))
utils::write.csv(train, file.path(train_dir, "target_demo_TRAIN.csv"),
                 row.names = FALSE)
utils::write.csv(test, file.path(train_dir, "target_demo_TEST.csv"),
                 row.names = FALSE)

fit <- stats::lm(y ~ x, data = train)
fit$pred <- data.frame(
```

```
  intercept = TRUE,
  pred = c(1.5, 1.5, 3, 3, 6, 6),
  obs = train$y,
  rowIndex = 1:6
)
fit$bestTune <- data.frame(intercept = TRUE)
saveRDS(fit, file.path(model_dir, "lm_target_demo.Rds"))
data.table::fwrite(
  data.table::data.table(
    response = "target", dataset = "demo", n = 6L,
    q_50 = 0, q_90 = 0.5, q_95 = 1
  ),
  file.path(summary_dir, "prediction_quantiles.csv")
)

set.seed(2026)
result <- suppressMessages(predictModel(
  .path.train = train_dir,
  .path.model = model_dir,
  .path.metrics = metrics_dir,
  .path.response = response_dir,
  .path.summary = summary_dir,
  .path.export = export_dir,
  METHODS = "lm",
  .ensemble = "ensemble.RMSE"
))
result$DTW
# p identifies three signed-residual offsets around one selected cell.

unlink(root, recursive = TRUE)
```

`removeOutliersIQR` *Remove rows outside feature IQR envelopes*

Description

Fit a ranger model to rank the predictors by impurity importance, then drop every row that falls outside the interquartile-range envelope of any of the most important numeric ones. The envelopes apply together: a row survives only when it is inside all of them.

Usage

```
removeOutliersIQR(.data, n = 3, q = 3)
```

Arguments

`.data`	Training data containing a numeric response column named `y`. The response name is the package's split contract: `buildDataset()` renames each response to `y` when it writes the training splits.

n — Number of top-ranked predictors to inspect. Non-numeric predictors among them are ignored; when none of the n is numeric the data is returned unchanged.

q — Multiplier of the interquartile range. The envelope of a column runs from its first quartile minus q times its interquartile range to its third quartile plus the same amount, so a larger q keeps more rows. At the default of 3 an approximately normal column loses no row.

Value

A data.table holding the surviving rows and every original column.

Missing values

The ranger fit runs through caret's formula interface, whose default na.action is na.fail, so a missing value anywhere in .data stops the call before any envelope is computed. Resolve missing values before calling. A missing value in a column being inspected would not by itself remove its row: this function removes outliers, not incomplete rows.

See Also

detectOutliers(), which scores rows instead of removing them and is the routine buildDataset() applies when outlier removal is enabled.

---

semiSupervisedPipeline

Fit a range-ratio pseudo-label promotion loop

Description

Refit the supervised ssel workflow while promoting selected prediction rows as ordinary ensemble.RMSE pseudo-labels. Selection uses a package-defined cross-method/dataset prediction range; acceptance uses a separate out-of-fold squared-correlation gauge. This is a pseudo-label policy, not tri-training, co-training regression, held-out validation, or calibrated predictive uncertainty.

Usage

```
semiSupervisedPipeline(
  .path.datasets,
  .path.work,
  key,
  responses,
  datasets,
  methods,
  seed,
  tau,
  maxRounds = 10L,
  epsRevert = 0.02,
```

```
  promotionCap = 0.8,
  boundsMin = -Inf,
  boundsMax = Inf,
  .path.iter = NULL,
  allResponses = responses
)
```

Arguments

`.path.datasets`	Directory scanned non-recursively for assembled CSV files whose basenames occur in `datasets`. It is ignored after a non-`NULL` `.path.iter` selects an iteration-specific dataset directory.
`.path.work`	Working directory. Every fit recursively removes and recreates its `round` child without rollback.
`key`	Character key column used for joins and pool membership. The final OOF gauge normally requires `"SampleID"` because that is the key column in the accepted residual schema. Missing key values are not rejected; their downstream behavior is described under input discovery.
`responses`	Character vector of target response columns. Exactly one is required when `.path.iter` is supplied. A standalone call can receive multiple responses; each final TEST split uses its fitted TRAIN schema.
`datasets`	Character dataset identifiers matched to CSV basenames. Matching preserves `list.files()` order, not this vector's order.
`methods`	Character vector of caret method identifiers. The initial check requires at least two entries; promotion additionally requires at least two distinct method names among available predictions.
`seed`	Value passed unchanged to every full `modelPipeline()` fit. It is not passed to the final serial re-emission.
`tau`	Numeric strict upper threshold for the dimensionless range ratio D. Equality does not promote. Length, missingness, and finiteness are not proactively validated.
`maxRounds`	Value passed to `seq_len()` as the maximum number of candidate rounds after the baseline fit. No terminal history row is added merely because this sequence is exhausted.
`epsRevert`	Numeric absolute tolerance in squared-correlation units. A candidate reverts only below `bestR2 - epsRevert`; the value is not validated as finite or nonnegative.
`promotionCap`	Numeric multiplier used in the key budget `floor(promotionCap * length(U))`. It is not restricted to $[0, 1]$.
`boundsMin, boundsMax`	Numeric response bounds forwarded to supervised fitting and final evaluation. Let a be `boundsMin` and b be `boundsMax`. When $a \leq b$, both ordinary and reconstructed OOF predictions use interval projection $\Pi_{[a,b]}(z) = \min\{b, \max\{a, z\}\}$. Order and finiteness are not checked. If $a > b$, the two executed nestings differ: ordinary evaluation computes $\min\{b, \max\{a, z\}\} = b$, whereas OOF reconstruction computes $\max\{a, \min\{b, z\}\} = a$. Neither is then an interval projection.

`.path.iter` NULL, or a `chainPipeline()` iteration directory containing `metrics_<integer>.csv` and the corresponding `datasets_<integer>` directories. Selection is described in **Input discovery and optional chain warm start**.

`allResponses` Character vector intended to name every response column. Before each fit, named non-target responses are removed. Missing names are ignored; an omitted non-target response remains and is numerically coerced as a predictor. The default is `responses`.

Value

Visibly, a list with:

`history` A `data.table` with integer `round`, `promoted`, and `cumulative`; numeric `oofR2`; and character `action`, under the counting and action rules above.

`promoted` NULL, or a `data.table` with the configured key, `response`, and numeric `value`, one row per accepted matching ensemble label row.

`baselineR2` The numeric baseline gauge g_0.

`finalR2` The numeric historical maximum gauge, not necessarily the final fitted-state gauge.

`iterStar` `NA_integer_` without `.path.iter`; otherwise the selected iteration integer.

Input discovery and optional chain warm start

With `.path.iter` = NULL, non-recursive `.csv` files in `.path.datasets` are filtered by basename membership in `datasets`; `list.files()` order is retained. With `.path.iter` supplied, the function reads every lexically ordered `metrics_<integer>.csv`, retains rows for `method == "ensemble.RMSE"` and the single target response, and selects the first row attaining the global maximum `r2`. Its integer becomes `iterStar`, and the input directory becomes `datasets_<iterStar>`.

This selects one augmented dataset snapshot. It does not consume a stitched model, stitch response-specific features, or rerun a chain sweep. The selected directory must exist; in particular, no fallback maps iteration zero to the original dataset directory. With no retained metric row or only missing `r2`, `which.max()` returns no position, no CSV is discovered, and the later non-empty-file assertion fails.

The first retained CSV alone defines the labelled-key vector $L = \{u : \mathrm{set}(u) = \text{"train"}\}$ and original prediction-key vector $U = \{u : \mathrm{set}(u) = \text{"predict"}\}$. Row order and duplicate keys are retained. Missing key values are also retained without a missingness check. They enter membership filters, update joins, `length(U)`, unique-key accounting, and the final completeness count under ordinary R/data.table semantics. Round CSV writing and subsequent type inference can materialize a missing character key as an empty string while preserving a missing numeric key, so no cross-file missing-key normalization is guaranteed. Other files are not checked for the same key pools, and missing `set` or `key` columns fail through data.table evaluation. The base budget size $|U|$ is `length(U)`, including duplicates and missing values.

Before every fit, each retained CSV is copied after removing only names in `setdiff(allResponses, responses)`. Every remaining column except `key` and `set`, including targets and predictors, is replaced by `as.numeric()`. Text and factor-like input can therefore warn and become missing. Missing features, targets, or role columns are left to the composed supervised helpers and data.table operations to reject or omit under their contracts.

Destructive round fit

For each baseline, candidate, or restoration fit, the function recursively deletes `.path.work/round`, recreates its `datasets` directory, and writes all selected CSV copies. Accepted pseudo-label rows update targets by `(key, response)`, and every matching key is changed to `set = "train"`. Duplicate label keys and labels from multiple datasets are not collapsed; update-join order can make the last matching value material.

The rewritten tables are passed to `modelPipeline()` in its default `"full"` mode with `ensemble = "ensemble.RMSE"` and the same `seed` on every fit. Round products therefore follow that helper's split, fit, OOF, prediction, summary, aggregation, bounded-PSOCK, and deletion contracts. Ordinary prediction `ensemble.RMSE` rows used below are distinct from the reconstructed OOF inverse-RMSE estimator underlying the acceptance gauge.

Range-ratio promotion score

For original prediction key u, response r, method m, dataset d, and round s, let $f^{(s)}_{mdr}(u)$ be every available projected ordinary base-method prediction. Pool all method and dataset rows into

$$\mathcal{A}^{(s)}_{ur} = \{f^{(s)}_{mdr}(u)\}, \qquad S^{(s)}_{ur} = \max \mathcal{A}^{(s)}_{ur} - \min \mathcal{A}^{(s)}_{ur}.$$

A pair is retained only when at least two distinct method names occur; two datasets alone do not suffice.

Let $\bar{f}^{(s)}_{ur,\mathrm{last}}(u)$ be the last matching ordinary `ensemble.RMSE` value in response-file concatenation and update-join order. ssel defines

$$D^{(s)}_{ur} = \frac{S^{(s)}_{ur}}{\max\{1, |\bar{f}^{(s)}_{ur,\mathrm{last}}(u)|\}}.$$

This dimensionless one-dimensional range ratio is package policy, not a variance, probability, interval, or calibrated uncertainty. The fixed denominator floor makes it non-invariant to response rescaling and translation. Pooling across datasets and choosing the last ensemble match also make row composition and order material.

Neither maximum nor minimum removes missing values. A missing base value makes the score missing; it cannot satisfy `D < tau`. Identical finite base predictions give $D = 0$, which promotes only when `tau > 0`. A score equal to `tau` does not promote. Equal scores retain their preceding group order. Previously accepted `(key, response)` pairs are removed before this strict gate.

Key budget and pseudo-label rows

Let c be `promotionCap`, let P be the accepted pseudo-label table, and let $K(P)$ be its unique keys. The remaining budget is

$$B_s = \lfloor c|U| \rfloor - |K(P)|.$$

New keys are encountered in ascending-score, first-row order. When they exceed B_s, only the first B_s new keys are retained, together with all their eligible response pairs and new response pairs on already promoted keys. A nonpositive budget or absence of a new key produces `saturation-stop`. Because $|U|$ counts rows while spending counts unique keys, duplicate prediction keys can allow `promotionCap < 1` to admit every unique key. Invalid cap values retain base arithmetic and condition behavior.

Pseudo-labels are all response-file rows matching `method == "ensemble.RMSE"` and the candidate pairs; no median or cross-dataset reduction is applied. Multiple datasets can therefore contribute multiple label rows for one pair. `history$promoted` counts candidate pairs, whereas accepted `history$cumulative` and the returned `promoted` table count label rows.

OOF gauge and accept/revert rule

Let $H_r^{(s)}$ contain all rows of `residuals_oof.csv` for response r whose configured key occurs in the first-file labelled vector L. Rows from all datasets remain pooled. For each response, compute squared sample correlation and then define

$$g_s = \mathrm{mean}_r\{\mathrm{cor}(y, \hat{y}_{\mathrm{OOF}})^2\},$$

removing response-level missing values and weighting each surviving response equally rather than by row count. The gauge is dimensionless. It is not held-out risk, a fully nested OOF estimate, or a no-degradation guarantee. Duplicate pooled rows retain their multiplicity. Missing values or constant vectors can make a response correlation missing; if every response is missing, g_s is `NaN` and a later logical comparison can error. A key other than `"SampleID"` generally fails when the accepted residual product has no such column.

Baseline round zero sets historical `bestR2` to g_0. Candidate round s reverts only when

$$g_s < \mathrm{bestR2} - \varepsilon,$$

where ε is `epsRevert`. Equality is accepted. An accepted round may be worse than the historical maximum by up to this absolute tolerance; `bestR2` becomes `max(bestR2, g_s)`. Thus the returned `finalR2` is a historical maximum and need not equal the final fitted state's gauge. A rejected candidate writes a `REVERT-stop` history row, then the accepted pool is materially refitted, deleting the rejected round state. That restoration gauge is not recomputed.

Stops and history

`history` begins with action `"baseline"`, zero counts, and g_0. Accepted rounds use `"accept"`, the candidate-pair count, accepted label-row count, and candidate gauge. If no pair passes, `"empty-stop"` records zero promoted rows, the accepted label-row count, and historical `bestR2`. A nonpositive/exhausted budget or no new key similarly records `"saturation-stop"`. A rejection uses uppercase `"REVERT-stop"`, records the attempted pair count, the preceding accepted label-row count, and the rejected gauge. Exhausting `maxRounds` adds no separate stop row.

No-promotion runs return `promoted = NULL`. Depending on cap and duplicate keys, every unique prediction key can be promoted. Subsequent scoring uses only rows still emitted as predictions by the latest accepted full fit.

Final serial re-emission

After the loop and any restoration, the function rebuilds one TEST file for every declared dataset/response cell. It uses all rows from each original selected CSV whose key belongs to the first-file U, taking feature names from that cell's final TRAIN split, excluding its `y` column. Other target responses and discarded features do not enter TEST. Missing declared datasets or required feature columns fail through file/data.table operations.

`trainRegressionModel()` then runs with `TRAIN = FALSE` and `PARALLEL = FALSE`, overwriting cell metric and response CSVs. For each cell, it asserts that the number of unique `ensemble.RMSE` keys

equals `length(U)` and that the key sets agree. Duplicate keys in U make the assertion fail even when all unique keys were emitted.

The shared prediction-summary calculation and `aggregateResponses()` then refresh summaries and exports for the original prediction pool, including promoted keys. Aggregation uses the original selected datasets for observed and prediction row membership. The accepted fitted models, OOF residuals, `auditQuantiles()` offsets and `auditOverfit()` results are retained; refreshing the delivery does not fit or evaluate models again.

Conditions, RNG, and side effects

Apart from the method-count and optional single-response/file assertions, most validation is delegated to base R, data.table, caret, and the composed ssel helpers. Paths, scalar lengths, response/dataset agreement, bounds, thresholds, caps, and tolerances are not proactively normalized. Failures can leave partial round files; there is no transaction or rollback.

Every full round receives the same `seed`, so `modelPipeline()` resets RNG before its sampled method/dataset order and downstream fitting. Reproducible results remain conditional on the R, RNG, caret, learner, and parallel environment. Rounds are sequential, each full round uses the supervised bounded-PSOCK contracts, and final re-emission is explicitly serial. The helper emits the conditions of its composed functions and returns no fitted model or prediction object.

See Also

`modelPipeline()` for each destructive full fit, `trainRegressionModel()` for ordinary prediction and final serial re-emission, `oofEnsemble()` for the reconstructed OOF estimator, and `chainPipeline()` for the optional augmented-dataset source.

Examples

```
root <- tempfile("ssel-semi-")
datasets_dir <- file.path(root, "datasets")
dir.create(datasets_dir, recursive = TRUE)

n_train <- 30L
assembled <- data.frame(
  SampleID = c(paste0("S", seq_len(n_train)), "P1", "P2"),
  set = c(rep("train", n_train), "predict", "predict"),
  x = seq_len(n_train + 2L),
  z = rep(c(0, 1), length.out = n_train + 2L),
  target = c(
    1.5 * seq_len(n_train) + rep(c(-0.4, 0.2, 0.6), 10),
    NA, NA
  )
)
utils::write.csv(
  assembled, file.path(datasets_dir, "demo.csv"),
  row.names = FALSE, na = ""
)

result <- suppressMessages(semiSupervisedPipeline(
  .path.datasets = datasets_dir,
```

```
  .path.work = file.path(root, "work"),
  key = "SampleID",
  responses = "target",
  allResponses = "target",
  datasets = "demo",
  methods = c("lm", "glm"),
  seed = 2026,
  tau = 0.1,
  maxRounds = 1,
  epsRevert = Inf,
  promotionCap = 0.5
))

result$history
result$promoted
c(baseline = result$baselineR2, historical_best = result$finalR2)
# The one-round policy promotes at most one of the two original keys.
# iterStar is NA because no chain iteration directory was supplied.

unlink(root, recursive = TRUE)
```

`toNumeric` *Convert supported numeric strings to numeric values*

Description

Normalize the package's supported decimal and grouping separators, then coerce values to numeric. Use this helper for simple character columns whose separator convention is known; it is not a general locale or units parser.

Usage

```
toNumeric(x)
```

Arguments

`x`	An atomic vector or factor. Values are processed as character text and returned as a plain numeric vector of the same length.

Value

A numeric vector with one output per input value. Failed conversions are `NA_real_`; numeric `NaN` remains a missing numeric value.

Supported grammar

ASCII spaces are removed first. A value containing two or more period-plus- digits groups has every period removed; the analogous rule removes every comma from a value containing two or more comma-plus-digits groups. After those rules, one comma in a complete string of the form `-?[0-9]+,[0-9]+` is changed to a decimal point. Base R numeric conversion handles the remaining text, including ordinary point decimals, scientific notation, and `Inf`.

Group widths are not validated. Thus `"1.234.567"` and `"1,234,567"` become `1234567`, while `"1,2,3"` becomes `123`. A single separator is interpreted as decimal: both `"1.234"` and `"1,234"` become `1.234`.

Missing, ambiguous, and unit-bearing values

Missing inputs remain missing. Conversion warnings are suppressed and text outside the supported grammar becomes `NA_real_`; this includes mixed separators such as `"1,234.56"`, currency symbols, and unit suffixes. Numbers are not scaled and units are not converted or preserved as metadata. Resolve ambiguous separators and harmonize units before calling this helper.

See Also

`which.nonnum()` for positions that fail direct base R numeric conversion. `buildDataset()` applies `toNumeric()` to character columns and replaces a column only when all of its non-missing values convert.

Examples

```
x <- c("1 234", "1.234.567", "1,25", "1,234", "1,234.56", NA)
data.frame(input = x, numeric = toNumeric(x))
```

---

trainModel *Fit every discovered response–dataset training cell*

Description

Training-only wrapper around `trainRegressionModel()`. It discovers split files created by `buildDataset()`, constructs the response–dataset grid, and requests every learner under ssel's standard fitting defaults. It does not evaluate fitted models, write metrics, or predict unseen rows.

Usage

```
trainModel(.path.train, .path.model, .path.metrics, .path.response,
  METHODS, PPROC = NULL, OVERRIDE = FALSE, .min = -Inf, .max = Inf,
  .responses = NULL)
```

Arguments

.path.train	Path to an existing directory containing split files. Files are discovered non-recursively by the suffix _TRAIN.csv.
.path.model	Path to the model-output directory. It is created non-recursively when absent, so its parent must exist and be writable.
.path.metrics	Path to a metrics directory. It is created when absent but receives no metric file from this training-only wrapper.
.path.response	Path to a prediction directory. It is created when absent but receives no prediction file from this training-only wrapper.
METHODS	A character vector of method identifiers accepted by caret::train(). Duplicate identifiers are removed. Their execution order is randomized for each dataset. An empty vector is not rejected, but no learner is then fitted.
PPROC	NULL, a character vector, or another preprocessing specification accepted by the preProcess argument of caret::train().
OVERRIDE	A non-missing logical scalar. If FALSE, an existing model is reused only when its companion feature-name hash exists and matches. If TRUE, every discovered cell is fitted again.
.min, .max	Numeric response bounds, in the response's units. They are forwarded to the cell engine for API consistency; this wrapper invokes training mode only, so the bounds do not alter fitting or training data. Each must be a single non-NA numeric value; Inf and -Inf are allowed. Their order and finiteness are not validated.
.responses	NULL, or a character vector restricting the discovered response names. NULL fits every discovered response. At least one TRAIN split must remain after filtering; otherwise the function produces an error.

Value

Invisibly returns NULL. The fitted .Rds and .Rds.hash files are the result; .path.metrics and .path.response are only prepared for later evaluation calls.

Choosing the API layer

Use modelPipeline() when starting from assembled dataset CSVs and the complete supervised workflow is required. Use trainModel() when TRAIN split files already exist and the standard five-fold/RMSE fitting policy is appropriate. Call trainRegressionModel() with TRAIN = TRUE only when the requested response–dataset identifier grid, fold count, tuning granularity, preprocessing, metric, or parallel mode must be controlled directly. Fitted-model evaluation and delivery belong to predictModel(); this wrapper performs no evaluation despite receiving metrics and response paths.

Split-file contract

A training filename must have the form response_dataset_TRAIN.csv; suffix matching is case-sensitive. Discovery interprets the first two underscore-separated fields as response and dataset; those identifiers therefore cannot contain underscores. This restriction is not proactively validated.

The parsed responses and datasets are deduplicated separately, then their Cartesian product is attempted. In a sparse set of source files, reconstructed pairs that were not present therefore warn and skip. Each CSV that is found must satisfy the `trainRegressionModel()` training schema: predictor columns followed by, or otherwise including, a numeric response column named `y` with at least two distinct non-missing values. A missing split directory, an empty set of TRAIN files, or a response filter with no matching split produces an error before the output directories are created.

Dataset order and the unique learner order are randomized with R's current random-number generator. Call `set.seed()` immediately before this function when repeatable ordering and caret resampling are required. Separate learner fits do not receive a shared caret resampling index, so the function does not promise identical folds across learners.

Fitting and cache policy

The wrapper fixes five-fold caret cross-validation, `tuneLength = 10`, `metric = "RMSE"`, saved final-tuning holdout predictions, and the `trainRegressionModel()` default parallel policy. `tuneLength` requests a method-specific grid granularity; it is not ten values for every tuning parameter.

A successful learner fit writes a caret `train` object to `method_response_dataset.Rds` and a companion `.Rds.hash` text file under `.path.model`. The eight-character hash is derived only from the sorted predictor names. With `OVERRIDE = FALSE`, equal predictor names reuse the model even if response values, predictor values or classes, preprocessing, resampling, tuning, or other fitting arguments changed. Set `OVERRIDE = TRUE` whenever any of those inputs changed.

Conditions and side effects

A malformed discovered basename can be parsed into a reconstructed cell filename that does not exist; that cell is skipped with a warning. A split with no usable `y`, or with a constant `y`, is also skipped with a warning. A learner-specific caret failure warns and skips that learner; successfully fitted cells continue. The wrapper muffles caret's generic missing-resample- performance warning because the cell engine emits method/cell counts for missing tuning-grid or selected-fold RMSE values after a successful fit. Unused-connection warnings are muffled without a replacement diagnostic.

The fitting engine registers a PSOCK backend with at most two workers, then stops it and leaves foreach registered sequentially. Progress is emitted as messages of class `moduleProgress`. Existing model/hash files may be reused or overwritten as described above; other files are preserved.

References

Kuhn, M. (2008). Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5), 1–26. doi:10.18637/jss.v028.i05. See also the caret documentation for `caret::train()` and `caret::trainControl()`.

See Also

`buildDataset()` for creating the split files and `trainRegressionModel()` for the complete cell-engine contract.

Examples

```
root <- tempfile("ssel-trainModel-")
train_dir <- file.path(root, "train")
```

```
model_dir <- file.path(root, "model")
metrics_dir <- file.path(root, "metrics")
response_dir <- file.path(root, "response")
dir.create(train_dir, recursive = TRUE)

split <- data.frame(
  x = seq_len(20),
  z = rep(c(0, 1), 10),
  y = seq_len(20) / 3 + rep(c(-1, 1), 10)
)
utils::write.csv(
  split,
  file.path(train_dir, "target_demo_TRAIN.csv"),
  row.names = FALSE
)

set.seed(2026)
suppressMessages(trainModel(
  .path.train = train_dir,
  .path.model = model_dir,
  .path.metrics = metrics_dir,
  .path.response = response_dir,
  METHODS = "lm"
))
sort(basename(list.files(model_dir)))
# The result is one caret model and its predictor-name hash.

unlink(root, recursive = TRUE)
```

`trainRegressionModel` *Fit or evaluate a requested response–dataset grid*

Description

Low-level engine for a requested response–dataset–learner grid. Training mode fits and caches caret learners. Evaluation mode first reconstructs projected out-of-fold (OOF) metrics and weighted ensembles, then predicts matching unseen rows. The higher-level `trainModel()` and `predictModel()` wrappers call these two modes; direct use is intended for grid-level control.

Usage

```
trainRegressionModel(.path.data, .path.model, .path.metrics,
  .path.response, OVERRIDE = FALSE, TRAIN = TRUE, PARALLEL = TRUE,
  .min = -Inf, .max = Inf, .numberCV = 10, .responses = NULL,
  .methods = NULL, .datasets = NULL, .tuneLength = 10,
  .preProcess = NULL, .metric = "RMSE")
```

Arguments

.path.data	Path to an existing directory containing the split CSV files described in **Split files**.
.path.model	Model directory. Training mode requires it to exist and be writable for .Rds files and their .Rds.hash companions. Evaluation mode requires it to exist and be readable; it reads .Rds files and does not use the hashes. This engine does not create the directory.
.path.metrics	Metrics directory. Training mode does not touch this path, so it need not exist for a training-only call. Evaluation mode requires an existing writable directory, overwrites assigned cell-named CSVs, and reads every .csv already present; use a dedicated clean directory for one run.
.path.response	Prediction directory. Training mode does not touch this path, so it need not exist for a training-only call. Evaluation mode requires an existing writable directory and overwrites one cell-named CSV per cell that reaches prediction. This engine does not create it.
OVERRIDE	A non-missing logical scalar used only in training mode. FALSE reuses a model when both model and hash files exist and the sorted predictor-name hash matches; TRUE fits it again.
TRAIN	A non-missing logical scalar. TRUE selects fitting and returns after model creation. FALSE runs both OOF metric reconstruction and unseen-row prediction; it is not a prediction-only switch.
PARALLEL	A non-missing logical scalar. TRUE registers a PSOCK cluster capped at two logical cores; FALSE registers foreach sequential execution. The backend is left sequential after the call.
.min, .max	Numeric lower and upper response bounds, in the response's units. In evaluation mode each base OOF and unseen prediction is projected onto this interval. The function does not validate finiteness or .min <= .max. The bounds are unused in training mode.
.numberCV	A positive integer passed to the number argument of caret::trainControl() in training mode. The default is 10.
.responses	A character vector of response identifiers. Despite the NULL default, a non-empty vector is required to process cells.
.methods	A character vector of caret method identifiers. Despite the NULL default, at least one successfully fitted method is required for evaluation and ensemble output.
.datasets	A character vector of dataset identifiers. Despite the NULL default, a non-empty vector is required to process cells.
.tuneLength	A positive integer passed to caret::train() in training mode. It requests method-specific grid granularity and is not a count per tuning parameter. The default is 10.
.preProcess	NULL, a character vector, or another preprocessing specification accepted by caret::train() in training mode. It is unused in evaluation mode.
.metric	A single caret performance metric used to select tuning settings in training mode. The default is "RMSE"; it does not change the fixed metrics written by evaluation mode.

Value

In training mode, invisibly NULL. In evaluation mode, invisibly a data.table with the metric schema described in **Files written and return value**.

Choosing the API layer

This function is not a second supervised workflow. trainModel() discovers split files and calls TRAIN = TRUE with ssel's standard defaults. predictModel() discovers the same grid, calls TRAIN = FALSE, and then performs response-level selection and delivery assembly. semiSupervisedPipeline() also calls TRAIN = FALSE, PARALLEL = FALSE for its final serial re-emission. Use this engine directly only when those wrappers do not expose the required identifier grid, resampling, tuning, projection, or execution control.

Split files

The function expands .responses, .datasets, and .methods as a Cartesian product, in response–dataset–method loop order; it does not accept a sparse list of selected tuples. For each requested response and dataset, the training file is response_dataset_TRAIN.csv. It must contain predictor columns and a numeric column named y. Integer columns are promoted to double. A cell is skipped with a warning when the file is absent, y is absent, y has no non-missing value, or fewer than two distinct non-missing response values remain.

Evaluation mode additionally requires response_dataset_TEST.csv. It must contain the fitted predictors and exactly one column name absent from the training file; that column is the row key. Zero or multiple such names produce an error. Predictor and response values are not scaled or assigned units, so every file, bound, and interpretation must use consistent response units.

Training and reproducibility

Each learner is fitted by caret::train() with ordinary cross-validation, savePredictions = "final", the requested fold count, preprocessing, tuning granularity, and optimization metric. The function does not supply a common index or indexOut; separate learner calls therefore do not promise identical resampling folds. There is no seed argument. Call set.seed() before the function for reproducibility in a fixed R, caret, learner, and parallel environment.

A successful fit writes the caret train object to method_response_dataset.Rds and writes its eight-character predictor-name hash to method_response_dataset.Rds.hash. The hash uses only the sorted predictor names. With OVERRIDE = FALSE, an equal hash reuses the model even when data values or classes, response values, preprocessing, resampling, tuning, or metric settings changed. Training failures warn and skip the learner; successful cells continue.

OOF metrics and ensemble definitions

When a = .min <= .max = b, define interval projection by $\Pi_{[a,b]}(z) = \min\{b, \max\{a, z\}\}$. For method m, evaluation mode merges the saved final-tuning caret holdout predictions, orders them by caret row index, and applies this projection.

The first requested method that reaches this scoring step for a cell stores its ordered obs vector as the cell's shared reference response. Every later method is scored against that stored vector rather than its own merged obs. The function does not verify cross-method equality of observations, row indices, or vector lengths. Consequently, the supplied method order is material when fitted objects

contain inconsistent saved OOF rows. In the equations below, y_i denotes this first-method reference vector and $\hat{y}_{mi}$ the projected OOF prediction for method m and row $i = 1, \ldots, n$. The reported scores are

$$\mathrm{RMSE}_m = \left[n^{-1} \sum_i (y_i - \hat{y}_{mi})^2 \right]^{1/2},$$

$$\mathrm{MAE}_m = n^{-1} \sum_i |y_i - \hat{y}_{mi}|,$$

and squared sample correlation

$$R^2_m = \mathrm{cor}(y, \hat{y}_m)^2.$$

RMSE and MAE have the response's units; `r2` is dimensionless; `n` is the training-row count; and `p` is the predictor count.

For the methods with non-missing scores in a cell, the package defines two OOF weighted means. The R2 ensemble uses

$$w_m^{(R2)} = R^2_m / \sum_j R^2_j,$$

and the RMSE ensemble uses

$$w_m^{(RMSE)} = (1/\mathrm{RMSE}_m) / \sum_j (1/\mathrm{RMSE}_j).$$

Here j ranges over the retained methods. Each ensemble prediction is the corresponding weighted mean $\sum_m w_m \hat{y}_m$. These are package weighting policies, not claims of optimal weighting. A zero, missing, or otherwise unusable weight sum can skip an ensemble or the cell. The ensemble rows are named `ensemble.R2` and `ensemble.RMSE`.

Unseen-row prediction weight state

The OOF pass stores the exact named R2 and RMSE score lists under each response–dataset cell. Before predicting unseen rows for a cell, the function restores that cell's pair of lists. It does not reconstruct these numerical weights from metric CSVs, and a cell with no stored OOF score state has no unseen-row ensemble.

Each unseen cell uses the intersection of methods predicted successfully for that cell and the restored score-list names. A missing model or a method returning any missing prediction removes that method only from the current cell's local lists. For the R2 ensemble, the unseen pass substitutes equal weights if a retained R2 is missing or their sum is zero; otherwise it normalizes the retained R2 values. For the RMSE ensemble it first takes reciprocal retained RMSE values, then substitutes equal weights if a reciprocal is missing or their sum is zero; otherwise it normalizes those reciprocals. The formulas and fallbacks match the preceding policy, while score state and pruning remain response–dataset-cell local.

Base unseen predictions are projected before weighting.

Evaluation conditions

During OOF reconstruction, a missing training file, missing model, failed merge of saved holdout predictions, unsupported response, no valid method, invalid R2 or RMSE weight vector, or unusable ensemble prediction emits a warning and skips the affected method, ensemble, or cell according to

where it occurs. An invalid R2 vector skips the remainder of that cell before its metrics file is written. A missing test file warns and skips unseen prediction; a missing companion training file is skipped silently. During unseen prediction, a missing model or missing predicted value warns and prunes that method from the current cell, while no common method names warns and skips the cell. A prediction error, `NULL` prediction, row-count mismatch, or invalid key schema stops the call.

Files written and return value

A cell-named metrics file is written only when the OOF cell reaches the post-ensemble write. The file `.path.metrics/response_dataset.csv` contains every current-call metric row accumulated to that point, not only rows for the cell in its name. Base rows accumulated for a cell that skipped before writing can therefore first appear in a later cell's file. Columns are `method`, `set`, `dataset`, `response`, `rmse`, `mae`, `r2`, `n`, and `p`. Existing assigned files are overwritten.

After the OOF pass, every CSV in `.path.metrics` is row-bound to determine which base methods are eligible for unseen-row prediction. Stale or unrelated compatible rows can therefore affect method discovery even though ensemble weights come only from the current call's cell-local score lists. A clean, dedicated metrics directory is required for an isolated evaluation.

Each predicted cell writes `.path.response/response_dataset.csv` with one row per unseen row and available base or ensemble method. Columns are the original key name, `value`, `method`, `dataset`, and `response`; `value` has the response's units. Training mode invisibly returns `NULL`. Evaluation mode invisibly returns the cumulative metrics `data.table` assembled during the current call. Progress is emitted as messages of class `moduleProgress`; skips and degraded cells emit warnings of class `moduleWarning`.

References

Kuhn, M. (2008). Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5), 1–26. doi:10.18637/jss.v028.i05. See also the caret documentation for `caret::train()` and `caret::trainControl()`.

See Also

`trainModel()` for split discovery and standard fitting defaults, `predictModel()` for summary and residual-offset products, and `auditQuantiles()` for the separate OOF residual-offset estimator.

Examples

```
root <- tempfile("ssel-trainRegression-")
data_dir <- file.path(root, "data")
model_dir <- file.path(root, "model")
metrics_dir <- file.path(root, "metrics")
response_dir <- file.path(root, "response")
for (path in c(data_dir, model_dir, metrics_dir, response_dir)) {
  dir.create(path, recursive = TRUE)
}

train <- data.frame(
  x = seq_len(18),
  z = rep(c(0, 1, 2), 6),
  y = seq_len(18) / 4 + rep(c(-1, 0.5, 1), 6)
```

```
)
test <- data.frame(
  x = c(19, 20),
  z = c(0, 1),
  SampleID = c("P1", "P2")
)
utils::write.csv(
  train,
  file.path(data_dir, "target_demo_TRAIN.csv"),
  row.names = FALSE
)
utils::write.csv(
  test,
  file.path(data_dir, "target_demo_TEST.csv"),
  row.names = FALSE
)

set.seed(2026)
suppressMessages(trainRegressionModel(
  .path.data = data_dir,
  .path.model = model_dir,
  .path.metrics = metrics_dir,
  .path.response = response_dir,
  TRAIN = TRUE,
  PARALLEL = FALSE,
  .numberCV = 3,
  .responses = "target",
  .methods = "lm",
  .datasets = "demo",
  .tuneLength = 1
))

metrics <- suppressMessages(trainRegressionModel(
  .path.data = data_dir,
  .path.model = model_dir,
  .path.metrics = metrics_dir,
  .path.response = response_dir,
  TRAIN = FALSE,
  PARALLEL = FALSE,
  .responses = "target",
  .methods = "lm",
  .datasets = "demo"
))
metrics[, .(method, response, dataset, rmse, r2)]
utils::read.csv(file.path(response_dir, "target_demo.csv"))
# Base and two package-defined ensemble rows are returned per SampleID.

unlink(root, recursive = TRUE)
```

---

which.nonnum *Locate values outside base R's numeric grammar*

Description

Return the positions of present values that become missing when converted by `as.numeric(as.character(x))`. Use this diagnostic to find labels, unit suffixes, localized separators, or other text that base R cannot parse as a number.

Usage

```
which.nonnum(x)
```

Arguments

`x`	An atomic vector or factor. Factor levels are tested through their labels. The result has the same positional indexing as `x`.

Details

Conversion warnings are suppressed. Original missing values, including a numeric `NaN`, are excluded from the result. A present string such as `"NA"` or `"NaN"` is reported because its conversion is missing while the original element is not. Finite numbers and `Inf` are accepted by base R.

This helper does not apply `toNumeric()` first. Consequently, `"1,25"` is reported even though `toNumeric("1,25")` returns `1.25`. Values containing units or currency symbols are also reported; neither units nor locale are inferred.

Value

An integer vector of ascending one-based positions. Returns `integer(0)` when every non-missing value is accepted.

See Also

`toNumeric()` to normalize the package's supported separator formats before numeric conversion.

Examples

```
x <- c("2.5", "1,25", "12 kg", NA, "Inf")
which.nonnum(x)
x[which.nonnum(x)]
```

Index

activeByImportance, 2
activeByImportance(), *5*, *20*, *22–24*, *28*
activeByShadow, 4
activeByShadow(), *3*, *20*, *22*, *28*
aggregateResponses, 5
aggregateResponses(), *18*, *22*, *29–32*, *37*, *40*, *47*
as.integer(), *19*
auditOverfit, 9
auditOverfit(), *31, 32*, *47*
auditQuantiles, 12
auditQuantiles(), *8*, *11*, *31*, *35*, *38–40*, *47*, *56*

buildDataset, 14
buildDataset(), *19, 20*, *29–32*, *35*, *41, 42*, *49*, *51*

caret::train(), *50, 51*, *53, 54*, *56*
caret::trainControl(), *51*, *53*, *56*
caret::varImp(), *26, 27*
chainPipeline, 17
chainPipeline(), *8*, *23, 24*, *44*, *47*
computeActiveByImportance, 23
computeActiveByImportance(), *3*, *20*, *28*

describeModule, 25
detectOutliers, 26
detectOutliers(), *15, 16*, *42*

endsWith(), *27*
extractChainImportance, 26
extractChainImportance(), *3*, *5*, *20*, *22–24*

intersect(), *23, 24*
isTRUE(), *30*

list.files(), *27*, *43, 44*

modelPipeline, 28
modelPipeline(), *17–19*, *22*, *43*, *45*, *47*, *50*

oofEnsemble, 33
oofEnsemble(), *11–14*, *47*

postProcess, 36
predictModel, 37
predictModel(), *5*, *7, 8*, *12*, *14*, *30–32*, *50*, *52*, *54*, *56*

Reduce(), *24*
removeOutliersIQR, 41

semiSupervisedPipeline, 42
semiSupervisedPipeline(), *54*
seq_len(), *43*
set.seed(), *30*, *51*, *54*

tail(), *24*
toNumeric, 48
toNumeric(), *16*, *58*
trainModel, 49
trainModel(), *31, 32*, *52*, *54*, *56*
trainRegressionModel, 52
trainRegressionModel(), *11*, *33*, *35*, *37–40*, *46, 47*, *49–51*

which.max(), *44*
which.nonnum, 57
which.nonnum(), *49*