Package 'featR'

September 14, 2026

Title A Unified Toolkit for Feature Selection

Version 0.1.0

Description Filter, wrapper, and embedded feature-selection methods behind a consistent set of functions that share one calling convention and one return type: correlation and chi-squared filters, information gain, LASSO and elastic net, Bayesian model comparison, Boruta, recursive feature elimination, random forest importance, multivariate adaptive regression splines, support vector machine recursive feature elimination, stepwise selection, and principal component / singular value decomposition helpers. The implemented methods follow Tibshirani (1996) <doi:10.1111/j.2517-6161.1996.tb02080.x>, Zou and Hastie (2005) <doi:10.1111/j.1467-9868.2005.00503.x>, Friedman (1991) <doi:10.1214/aos/1176347963>, Breiman (2001) <doi:10.1023/A:1010933404324>, Guyon, Weston, Barnhill and Vapnik (2002) <doi:10.1023/A:1012487302797>, Kursa and Rudnicki (2010) <doi:10.18637/jss.v036.i11>, and Vehtari, Gelman and Gabry (2017) <doi:10.1007/s11222-016-9696-4>. Heavy modeling engines are optional and only required by the functions that use them.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Config/roxygen2/version 8.1.0

Depends R (>= 4.1.0)

Imports data.table, parallel, stats, utils, withr

Suggests bigstatsr, Boruta, brms, caret, doParallel, e1071, earth, foreach, furrr, future, ggplot2, glmnet, kernlab, knitr, loo, MASS, Matrix, MLmetrics, pbapply, polycor, pROC, PRROC, randomForest, rmarkdown, RSpectra, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

URL https://github.com/elkronos/featR

BugReports https://github.com/elkronos/featR/issues

NeedsCompilation no

Author Justin Chase [aut, cre, cph]

Maintainer Justin Chase <jchase.msu@gmail.com>

Repository CRAN

Date/Publication 2026-09-14 15:40:02 UTC

Contents

`fs_bayes` *Bayesian feature selection for model optimization*

Description

Fits a brms model for every candidate predictor combination and compares them with `loo::loo_compare()`, returning the selected combination as an `fs_result`.

Usage

```
fs_bayes(
  data,
  target,
  predictors,
  date_col = NULL,
  brm_family = stats::gaussian(),
  prior = NULL,
  brm_args = list(),
  rule = c("1se", "best"),
  max_comb_size = NULL,
  sample_combinations = NULL,
  parallel_combinations = FALSE,
  seed = NULL,
  verbose = FALSE,
  n_cores = 1L
)
```

Arguments

data	A data.frame or data.table. It is copied, never modified. Only `target`, `predictors` and `date_col` are carried forward, and rows with a missing value in any of them are dropped before any model is fitted.
target	Character. Name of the target (response) column, which must exist in `data` and suit `brm_family` (numeric for `stats::gaussian()`, 0/1 or logical for `brms::bernoulli()`).
predictors	Character vector. Names of the candidate predictor columns; at least one, all present in `data`.
date_col	Character or NULL. Name of a date column. When provided, an `iso_week_id` feature (ISO year * 100 + ISO week) is added to the predictors. Note: `iso_week_id` enters the models as a continuous covariate, which is a rough encoding of seasonality; a categorical or cyclic encoding is deferred to a future version. Default NULL (no date feature; the date column itself is never used as a predictor).
brm_family	A model family accepted by brms::brm(), for example stats::gaussian() (default) or brms::bernoulli().
prior	A brms prior specification (default NULL).
brm_args	List. Extra arguments for brms::brm() (for example iter, warmup, seed), overriding featR's defaults of iter = 2000, adapt_delta = 0.99, max_treedepth = 15 and refresh = 0. `chains` and `cores` are respected when evaluating combinations sequentially (`cores` is capped at the detected core count; the defaults are 4 chains on 1 core), but both are forced to 1 when `parallel_combinations = TRUE`. Default `list()`.
rule	Selection rule: `"1se"` (default) keeps the most parsimonious model within one standard error of the best elpd; `"best"` keeps the raw elpd maximum.
max_comb_size	Whole number >= 1, or NULL. Largest number of predictors allowed in a combination; values above the number of candidate predictors are capped rather than rejected. Default NULL (all sizes).

`sample_combinations`
: Whole number >= 1, or NULL. Randomly sample this many combinations instead of evaluating all of them; ignored when fewer combinations exist. Pass `seed` to make the draw reproducible. Default NULL (evaluate every combination).

`parallel_combinations`
: Logical. Evaluate predictor combinations in parallel via parallel::mclapply(). Not available on Windows (falls back to sequential evaluation with a message), and falls back to sequential evaluation when `n_cores` resolves to 1. Because each model is then held to a single MCMC chain, cross-chain convergence diagnostics such as R-hat become unavailable; a warning says so. Default FALSE.

`seed`
: Optional integer. When supplied, seeds the random sampling of combinations (see `sample_combinations`) locally; the previous RNG state is restored on exit. Default NULL: fs_bayes() never seeds the RNG unless asked. Note this does not seed the samplers; pass `seed` inside `brm_args` to control brms itself.

`verbose`
: Logical. Print progress information, and show a progress bar for sequential evaluation when the suggested pbapply package is installed. Default FALSE.

`n_cores`
: Whole number >= 1. Worker count used when `parallel_combinations = TRUE`, and ignored otherwise. Default 1 (sequential); requests are capped at the detected core count.

Details

Use this when you want the predictor subset itself chosen by out-of-sample predictive fit under a fully Bayesian model, and you can afford to fit one model per subset. The search is exhaustive by default: with `p` candidate predictors it fits every non-empty subset, `2^p - 1` models, and each one compiles and samples its own Stan program. Use `max_comb_size` or `sample_combinations` to bound the search.

Model selection uses `loo::loo_compare()` rather than the raw elpd maximum. With `rule = "1se"` (the default) the chosen model is the most parsimonious one – fewest predictors, ties broken by the higher elpd – among those whose elpd difference from the best model is no larger in absolute value than one standard error of that difference (the `se_diff` column of the comparison table). With `rule = "best"` the raw elpd maximum wins, which is the older behavior and is more prone to over-fitting the comparison. When no usable comparison table is available, both rules fall back to the raw elpd maximum, ties broken by fewer predictors.

Combinations whose model fails to fit are excluded from selection and counted in `details$n_failed_fits`, with a warning. If no fit yields a finite `elpd_loo` at all, the first successfully fitted model is returned with a warning; that is an arbitrary fallback, not a selection.

Two caveats are worth stating plainly. `details$mae` and `details$rmse` are in-sample errors computed on the same rows used to fit and to select, so they are optimistic. And anything read off the returned `brmsfit` (posterior intervals, effect sizes) is post-selection inference: the model was chosen by looking at the same data it is reported on.

Value

An object of class `fs_result` with:

selected Character vector of the predictors in the chosen model.

scores NULL: no per-feature score is comparable across combination models. details$n_features records how many candidate predictors were offered.

method "bayes".

task "regression", or "classification" when brm_family is one of the categorical brms families.

model The selected brmsfit.

details A list with data (the complete-case modeling data.table: target, predictors, date_col and iso_week_id where applicable, plus appended fitted_values, residuals, abs_residuals and squared_residuals columns), mae and rmse (in-sample, post-selection errors computed on the same rows used to fit and select, so they are optimistic), formula (the selected model's formula string), best_elpd (the selected model's elpd_loo, possibly NA), loo_comparison (the loo::loo_compare() table – a matrix or data.frame depending on the loo version – or NULL when fewer than two models could be compared or the comparison failed), n_failed_fits (how many combinations failed to fit) and n_features (the number of candidate predictors, including iso_week_id when date_col is supplied).

call The matched call.

Examples

```
# Fitting needs more than the brms package: brms compiles and samples a
# Stan program for each candidate model, which requires a working C++
# toolchain. The call is therefore guarded on brms being installed, limited
# to two single-predictor fits via max_comb_size = 1, and wrapped in try()
# so that a machine without a usable Stan toolchain skips the example
# instead of failing it.

if (requireNamespace("brms", quietly = TRUE)) {
  x1 <- seq(-2, 2, length.out = 40)
  x2 <- rep(c(-1, 1), 20)
  d <- data.frame(
    y  = 1 + 2 * x1 + sin(seq_len(40)),
    x1 = x1,
    x2 = x2
  )
  res <- try(
    fs_bayes(
      d, target = "y", predictors = c("x1", "x2"),
      max_comb_size = 1,
      brm_args = list(chains = 1, iter = 500, refresh = 0),
      rule = "1se", verbose = FALSE
    ),
    silent = TRUE
  )
  if (!inherits(res, "try-error")) {
    print(res$selected)
    print(res$details$loo_comparison)
  }
}
```

`fs_boruta` *Feature selection using Boruta*

Description

Runs the Boruta all-relevant feature selection algorithm on a dataset. Preprocesses predictors, optionally seeds the RNG locally, optionally resolves tentative features, and optionally prunes correlated features from the confirmed set. Pruning is importance-aware: within a group of correlated features the one with the highest median Boruta importance is kept.

Usage

```
fs_boruta(
  data,
  target,
  maxRuns = 250,
  cutoff_features = NULL,
  cutoff_cor = 0.7,
  resolve_tentative = TRUE,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

`data`	A data frame (or data-frame-like object, or matrix). Predictor columns may be numeric, factor, character, logical, Date or POSIXt; character and logical columns are converted to factors and Date/POSIXt columns to numeric, and any other type is an error. Neither the target nor the predictors may contain missing values, since Boruta's underlying random forest cannot fit them.
`target`	Name of the target column in `data`. A factor target is treated as classification, a numeric target as regression; any other type is an error, as is a target containing NAs.
`maxRuns`	Whole number of at least 11 (the minimum Boruta itself accepts). Maximum number of Boruta iterations. Default 250.
`cutoff_features`	Optional whole number capping the number of returned features. When supplied, the top features by median Boruta importance are retained, applied after any correlation pruning. Default NULL (no cap).
`cutoff_cor`	Numeric correlation cutoff between 0 and 1 used to drop redundant features from the selected set. Within each group of features correlated above the cutoff, the feature with the highest median Boruta importance is kept and the rest are dropped. Only numeric predictors are compared (absolute Pearson correlation); factor predictors are never pruned. Set NULL to skip this step. Default 0.7.

`resolve_tentative`
: Logical; if TRUE, apply `Boruta::TentativeRoughFix()` and return only confirmed attributes. If FALSE, tentative attributes are included in the selected set. Default TRUE.

`seed`
: Optional integer for reproducibility. Applied locally: the previous RNG state is restored when the function exits. Default NULL (the RNG is never seeded unless requested).

`verbose`
: Logical; if TRUE, report progress and name any features dropped by correlation pruning. This maps to Boruta's `doTrace = 1` (decisions are reported as they are made); FALSE maps to `doTrace = 0`. Default FALSE.

Details

Boruta answers "which features carry any information about the target?", not "which minimal subset predicts best". It is an all-relevant selector: across up to `maxRuns` random-forest fits it compares each feature against "shadow" features built by permuting the predictors, and confirms every feature that beats the best shadow often enough to be unlikely by chance. Redundant-but-informative features are therefore all confirmed. That makes it a good fit for understanding a dataset, and a poor fit when you need a compact model. It also costs many forest fits, and the outcome varies from run to run unless `seed` is supplied.

The `cutoff_cor` pruning and the `cutoff_features` cap are featR additions, applied to Boruta's output in that order; both rank features by median Boruta importance. Features still undecided when `maxRuns` is reached stay Tentative; `resolve_tentative = TRUE` settles them with the `Boruta::TentativeRoughFix()` heuristic rather than with further evidence.

Value

An object of class `fs_result` with:

selected Character vector of selected feature names, after optional correlation pruning and the optional `cutoff_features` cap.

scores Named numeric vector of median Boruta importance for every candidate feature (`NA` for attributes with no importance history).

method "boruta".

task "classification" for a factor target, "regression" for a numeric one.

model The Boruta object, after `Boruta::TentativeRoughFix()` when `resolve_tentative = TRUE` and something was still tentative.

details A list with `boruta_obj` (the same Boruta object as `model`), `decisions` (the per-feature Confirmed/Tentative/Rejected factor, as it stands after any tentative fix), `dropped_correlated` (features removed by correlation pruning, empty when `cutoff_cor` is NULL), and `n_features` (the number of candidate features).

call The matched call.

Examples

```
if (requireNamespace("Boruta", quietly = TRUE)) {
  d <- data.frame(
```

```
  y  = factor(rep(c("a", "b"), each = 20)),
  x1 = rep(c(0, 1), each = 20) + seq(0, 1, length.out = 40),
  x2 = seq_len(40) %% 3
)
res <- fs_boruta(d, "y", maxRuns = 25, cutoff_cor = NULL, seed = 42)
res$selected
res$scores
}
```

`fs_chi` *Chi-square feature selection for categorical features*

Description

Tests association between each categorical feature and a (categorical) target via the chi-square test of independence. Character columns are automatically coerced to factors, and only factor features (excluding the target) are tested. Handles missing values per-feature, switches to simulation-based p-values when any expected cell count is < 5, and supports multiple-testing correction.

Usage

```
fs_chi(
  data,
  target,
  sig_level = 0.05,
  continuity_correction = NULL,
  p_adjust_method = "bonferroni",
  simulation_B = 2000,
  seed = NULL,
  verbose = FALSE,
  parallel = FALSE,
  n_cores = 2L
)
```

Arguments

`data`	A data.frame or data.table with features and target. Character columns are coerced to factor. The input object is never modified.
`target`	Character scalar: name of the target column. It is coerced to a factor if necessary and must have at least 2 non-NA levels.
`sig_level`	Numeric threshold for significance, strictly between 0 and 1 (default 0.05).
`continuity_correction`	NULL/TRUE/FALSE: apply Yates correction to 2x2 tables tested asymptotically. If NULL (default), auto-apply to every such table; TRUE is equivalent, FALSE disables it. It has no effect on tables larger than 2x2 or on simulation-based p-values, neither of which is ever corrected.

p_adjust_method
: Character: one of `stats::p.adjust.methods` (default "bonferroni"). Set to "none" to disable multiple-testing correction. Matching is case-insensitive. The correction counts only the features that were actually tested: a skipped feature has an `NA` p-value, and `stats::p.adjust()` drops NAs before its default `n = length(p)` is evaluated, so skipped features keep an `NA` adjusted p-value and do not inflate the multiplier for the others. With `k` tested and `s` skipped features, "bonferroni" therefore multiplies by `k`, not by `k + s`.

simulation_B
: Whole number >= 100: replicates for the simulation-based p-value used when any expected cell count is < 5 (default 2000).

seed
: Optional integer. Seeds the RNG locally (the previous RNG state is restored on exit), which makes simulation-based p-values reproducible in the sequential path. The parallel path draws from furrr's own L'Ecuyer-CMRG parallel streams (`furrr_options(seed = TRUE)`), so for the same `seed` parallel results are internally reproducible but differ from sequential results. Default NULL (never seeds).

verbose
: Logical; if TRUE, emits informative messages (target coercion, skipped features, worker count). Default FALSE.

parallel
: Logical; if TRUE, run features in parallel using the suggested furrr and future packages. Default FALSE (sequential).

n_cores
: Whole number >= 1. Number of workers used when `parallel = TRUE`; requests are capped at the detected core count. A `future::multisession` plan is set for the duration of the call and the previous plan is restored on exit. Default 2.

Details

The question this answers is, per feature: does the joint distribution of that feature and the target differ from what independence would predict? Features are examined one at a time, so the result describes marginal association only. It says nothing about interactions, and two features carrying the same information are both reported as significant. Only factor features are tested: numeric, logical and date columns are ignored entirely, so convert or discretize them first if you want them included.

Each feature is tested on its own complete cases (rows where both the feature and the target are observed), so `n` can differ between features. Levels left empty after that filtering are dropped; if either the feature or the target then has fewer than two levels, the feature is skipped with an `NA` p-value rather than tested.

A feature is tested with the asymptotic chi-square statistic when every expected cell count is at least 5. Otherwise the p-value comes from a Monte-Carlo simulation with `simulation_B` replicates, which makes it stochastic unless `seed` is set and leaves `df` as `NA`. Yates' continuity correction applies only on the asymptotic path and only to 2x2 tables; it is never applied to a simulated p-value or to a larger table, whatever `continuity_correction` says.

Finally, a p-value is evidence against independence, not an effect size: with enough rows a negligible association still clears any `sig_level`.

Value

An object of class `fs_result` with:

selected Character vector of features with adj_p_value < sig_level.

scores Named numeric vector of adjusted p-values, one per candidate categorical feature and NA for any feature that had to be skipped (smaller is stronger evidence of association).

method "chi".

task "classification".

model NULL; the chi-square filter fits no model.

details A list with `results` (the full results data.frame, one row per candidate categorical feature, ordered by adj_p_value then p_value so that skipped features sort last, with columns: feature; n (for tested features, the number of complete feature-target pairs; for skipped features, the feature's non-NA row count); df (NA for simulation-based tests, where the asymptotic degrees of freedom do not apply, and for skipped features); p_value; adj_p_value; significant (TRUE only when adj_p_value < sig_level, so FALSE for skipped features); method ("asymptotic" or "simulation", NA when skipped); correction_applied (TRUE/FALSE, NA when skipped); min_expected (minimum expected cell count, NA when skipped)), plus `sig_level`, `p_adjust_method` (the method string as supplied), and `n_features` (the number of candidate categorical features, including any that were skipped).

call The matched call.

Examples

```
d <- data.frame(
  f1 = factor(rep(c("A", "B", "A"), times = c(40, 40, 20))),
  f2 = factor(rep(c("X", "Y"), times = 50)),
  target = factor(rep(c("Yes", "No"), each = 50))
)
out <- fs_chi(d, "target")
out$selected
out$details$results
```

`fs_correlation` *Correlation-based feature selection*

Description

Flags variable pairs whose absolute correlation exceeds `threshold` and, by default, reduces each correlated group to a single representative.

Usage

```
fs_correlation(
  data,
  threshold,
  method = "pearson",
  prune = TRUE,
  na.rm = FALSE,
  sample_frac = 1,
```

```
  output_format = "matrix",
  diag_value = 0,
  seed = NULL,
  verbose = FALSE,
  parallel = FALSE,
  n_cores = 2L
)
```

Arguments

data	A data frame or matrix with at least 2 columns and unique column names (a correlation matrix with duplicated dimnames is ambiguous, so duplicates are rejected rather than silently resolved to the first match). For "pearson", "spearman", "kendall": all columns must be numeric. For "polychoric": all columns must be ordered factors. For "pointbiserial": columns may be numeric (continuous) or dichotomous (exactly 2 unique non-NA values).
threshold	Numeric between 0 and 1. Pairs with |correlation| > threshold are flagged as redundant. Required; there is no default.
method	One of "pearson" (default), "spearman", "kendall", "polychoric", "pointbiserial". Point-biserial correlations are computed as the Pearson correlation between the continuous variable and a 0/1 indicator of the dichotomous variable (1 for the second sorted unique value, e.g. the second factor level), which is the definition of the point-biserial coefficient. Only continuous-dichotomous pairs are defined under that method: cells for continuous-continuous and dichotomous-dichotomous pairs stay NA, so those pairs can never be flagged.
prune	Logical. If TRUE (default), selected is the reduced non-redundant set: variables are dropped one at a time until no two retained variables correlate above threshold, each step dropping the member of the strongest remaining pair with the HIGHER mean absolute correlation to the other retained variables (the caret::findCorrelation() heuristic, implemented here with base stats). Variables that were never flagged are always retained. If FALSE, selected is every variable appearing in at least one flagged pair, i.e. the redundant set, and nothing is dropped.
na.rm	Logical. If TRUE, missing values are removed pairwise for "pearson"/"spearman"/"kendall", per pair (complete observations within each variable pair) for "pointbiserial", and casewise (complete cases across all columns) for "polychoric". If FALSE (default), missing values propagate NA into the affected correlations, except for "polychoric", which stops with an error when missing values are present (silently deleting cases would contradict the behavior of the other methods). Default FALSE.
sample_frac	Numeric in (0, 1]. Fraction of rows sampled without replacement (rounded up, never below one row) before computing correlations. Default 1 (no sampling).
output_format	"matrix" (default) or "data.frame" for the correlation matrix stored in details$corr_matrix.
diag_value	Single numeric value, or NA, assigned to the diagonal of the reported correlation matrix. It is cosmetic: the diagonal never takes part in flagging, scoring, or pruning. Default 0.

seed	Optional integer seed for reproducible sampling, applied locally; the previous RNG state is restored afterwards. Default NULL (never seeds).
verbose	Logical. Emit progress messages (row sampling, the correlation method used, and any variables dropped by pruning). Default FALSE.
parallel	Logical. Use parallel processing (via the suggested foreach and doParallel packages) for point-biserial computations. Ignored by every other method, all of which are single-call. Default FALSE (sequential).
n_cores	Whole number >= 1. Number of workers if parallel = TRUE and method = "pointbiserial"; requests are capped at the detected core count. Default 2.

Details

This is an unsupervised redundancy filter: it looks only at how the variables relate to one another and never at an outcome. When two variables are near-interchangeable it therefore cannot prefer the one that predicts better; it keeps whichever is least entangled with the rest of the data. Use fs_boruta() when the choice inside a correlated group should be driven by importance for a target.

Pruning is greedy rather than group-wise: while any retained pair still exceeds threshold, the strongest such pair is taken and the member with the larger mean absolute correlation to the other retained variables is dropped. No two retained variables end up with a computable correlation above threshold, but the surviving set is not guaranteed to be the smallest one with that property, and it depends on the order in which pairs are resolved.

Correlations that cannot be computed come back as NA. Those pairs are never flagged and never pruned – an unknown correlation is treated as no evidence of redundancy – and a warning reports how many there are. A message (not a warning) is emitted when no pair at all exceeds threshold, whatever verbose is set to.

Value

An object of class fs_result with:

selected Character vector. With prune = TRUE, every variable that survives pruning (all columns except details$dropped), no two of which have a computable absolute correlation above threshold. With prune = FALSE, every variable appearing in at least one flagged pair (both members of each pair), which is the redundant set rather than the set to keep.

scores Named numeric vector giving each variable's maximum absolute correlation with any other variable (NA when no correlation with that variable could be computed).

method paste0("correlation_", method), e.g. "correlation_pearson".

task NA_character_; correlation filtering is unsupervised.

model NULL; no model is fitted.

details A list with corr_matrix (the correlation matrix, with the diagonal set to diag_value, reshaped to long form with columns Var1, Var2, Correlation when output_format = "data.frame"), pairs (a data.frame of the flagged pairs with columns Var1, Var2, Correlation, ordered by decreasing absolute correlation and unaffected by output_format), dropped (variables removed by pruning, in the order they were dropped; empty when prune = FALSE), redundant (every variable in at least one flagged pair, i.e. selected as it would be under prune = FALSE), and n_features (the number of variables considered).

call The matched call.

Examples

```
d <- data.frame(
  a = c(1, 2, 3, 4, 5, 6),
  b = c(2, 4, 6, 8, 10, 12),
  c = c(1.5, 0.9, 2.1, 0.4, 1.1, 0.8)
)
res <- fs_correlation(d, threshold = 0.9)
res$selected
res$details$pairs

# keep the legacy view: both members of every flagged pair
fs_correlation(d, threshold = 0.9, prune = FALSE)$selected
```

---

fs_elastic *Elastic Net Feature Selection and Model Training*

Description

Performs feature selection and model training using elastic net regularization via `caret::train(method = "glmnet")`. Supports regression (numeric outcomes) and classification (factor/character outcomes).

Usage

```
fs_elastic(
  data,
  target,
  alpha_seq = seq(0, 1, by = 0.1),
  lambda_seq = NULL,
  trControl = NULL,
  metric = NULL,
  use_pca = FALSE,
  nPCs = NULL,
  seed = NULL,
  verbose = FALSE,
  n_cores = 1L
)
```

Arguments

`data`	A data frame (or data.table) containing the target and the candidate predictors.
`target`	Single string naming the outcome column in `data`.
`alpha_seq`	Numeric vector of alpha values to tune over, each in `[0, 1]`. Default `seq(0, 1, by = 0.1)`.
`lambda_seq`	Numeric vector of non-negative lambda values to tune over, or `NULL` (default) to use glmnet's own path per alpha.

trControl	Optional caret::trainControl() object. If NULL (default), 5-fold CV with an NA-safe summary function is used. When use_pca = TRUE, pcaComp = nPCs is injected into its preProcOptions.
metric	Optional character. Performance metric to optimize. If NULL (default), "RMSE" is used for regression and "Accuracy" for classification.
use_pca	Logical. Whether to project predictors onto principal components inside each resample. Default FALSE.
nPCs	Integer >= 1. Number of principal components to retain when use_pca = TRUE. Must be strictly less than min(nrow, ncol) of the predictor matrix. Default NULL, which is an error when use_pca = TRUE and ignored otherwise.
seed	Optional integer seed applied locally (and restored on exit) before resampling and tuning. Default NULL (never seeds by default).
verbose	Logical. Print progress messages. Default FALSE.
n_cores	Integer >= 1. Number of workers for parallel training. Default 1 (sequential; no cluster is created). Values above the detected core count are capped.

Details

Use this when the question is "which predictors survive a jointly tuned L1/L2 penalty?". Both alpha and lambda are chosen by resampling, and every predictor with a non-zero coefficient at the winning pair is reported. When the winning alpha is below 1 the ridge component spreads weight across correlated predictors, so a group of collinear columns tends to survive together instead of being reduced to a single representative.

The model formula is built internally from data and target: every other column of data is a candidate predictor, and non-syntactic names are backticked. Predictors then go through stats::model.matrix() and the intercept column is removed, so a k-level factor or character column contributes k - 1 dummy columns and selected, scores and details$coef name design-matrix columns rather than the original columns.

Rows with a missing response, or a missing value in any predictor, are dropped before fitting (an error if that leaves nothing), and a constant predictor column is a hard error rather than a silently degenerate fit. A logical response is converted to a two-level factor with a message; a numeric response with only two distinct values is still treated as regression, with a warning telling you to convert it to a factor if you meant classification.

scores are absolute coefficients on the scale of the columns the model saw, not standardized ones, so they rank predictors fairly only when those columns are on comparable scales – unlike fs_lasso(), this function does not rescale them for you.

When use_pca = TRUE the PCA is **not** fitted up front. caret is asked for preProcess = c("center", "scale", "pca") with pcaComp = nPCs in trControl$preProcOptions, so centering, scaling and the component loadings are refit on the training part of every resample and the held-out fold never contributes to them. The model is then fitted on components, so selected, scores and details$coef are named PC1, PC2, ... rather than after the original columns. nPCs must be smaller than the number of rows each resample trains on as well as smaller than the number of predictors.

lambda_seq = NULL (the default) tunes over the lambda path glmnet::glmnet() itself proposes at each alpha (up to 50 values per alpha), which is scaled to the data, instead of a fixed sequence that spends most of its fits on irrelevant lambdas. Only the candidate values come from the full data,

exactly as in caret's own default glmnet grid; which pair wins is still decided by resampling. With `use_pca = TRUE` that path is computed on the original predictors, so it is only an approximation of the scale the components live on; pass `lambda_seq` explicitly if you need to control it.

Value

An `fs_result` object with:

`selected`	Predictors (or components, when `use_pca = TRUE`) whose coefficient at the chosen alpha/lambda is non-zero; for multinomial fits, the union across classes.
`scores`	Named numeric vector of absolute coefficients at the chosen alpha/lambda, one entry per column the model saw (predictors shrunk to zero are kept, with a score of 0); `NULL` for multinomial fits, where a predictor has one coefficient per class and no single score exists.
`method`	`"elastic_net"`.
`task`	`"regression"` or `"classification"`.
`model`	The `caret::train` object.
`details`	List of `coef` (coefficients at the best lambda, intercept included: a sparse matrix, or a list of them for multinomial fits), `best_alpha` and `best_lambda` (the winning tuning pair), `metric_name` (the metric optimized), `metric_value` (its resampled value for that pair, `NA` when the metric is absent from caret's results table), `use_pca`, and `n_features` (number of columns the model saw, i.e. `nPCs` when `use_pca = TRUE`).
`call`	The matched call.

Examples

```
if (requireNamespace("caret", quietly = TRUE) &&
    requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("Matrix", quietly = TRUE)) {
  # x1 and x2 drive y; x3 is noise
  df <- data.frame(
    x1 = seq(-2, 2, length.out = 60),
    x2 = rep(c(-1, 0, 1), 20),
    x3 = cos(seq_len(60))
  )
  df$y <- 2 * df$x1 - df$x2 + 0.1 * cos(seq_len(60) * 3)
  res <- fs_elastic(df, "y", alpha_seq = c(0.5, 1), seed = 1)
  selected(res)
  res$scores
  res$details$best_alpha
}
```

`fs_infogain` *Feature Selection via Information Gain*

Description

Accepts either a single data.frame or a list of data.frames and scores every predictor by its information gain (in bits) with respect to the target, optionally normalized to a gain ratio.

Usage

```
fs_infogain(
  data,
  target,
  numeric_bins = NULL,
  normalize = c("none", "gain_ratio"),
  top_n = NULL,
  remove_na = TRUE,
  verbose = FALSE
)
```

Arguments

`data`	A data.frame (a data.table is accepted and is copied, never modified in place), or a list of data.frames each containing `target`.
`target`	Character. Name of the target column.
`numeric_bins`	Optional whole number >= 1 (values below 2 are clamped to 2) overriding the automatic bin count for numeric predictors and for a numeric target. Default `NULL` (bins chosen per column).
`normalize`	One of `"none"` (default, raw information gain in bits) or `"gain_ratio"` (gain divided by the predictor's split entropy). See the section on cardinality bias.
`top_n`	Optional whole number >= 1. When supplied, `selected` holds the `top_n` highest-scoring features (fewer if fewer were scored). This is a rank cut, not a score floor: a zero-scoring feature is selected if the ranking reaches it. When `NULL` (default), `selected` instead holds every feature whose score is strictly greater than 0.
`remove_na`	Logical. If `TRUE` (default), rows with NA in the target are removed up front. See Details for its narrow practical effect.
`verbose`	Logical. If `TRUE`, report how many features were scored, how many names collided across data.frames, and how many were selected. Default `FALSE`.

Details

Use this to rank candidate predictors cheaply, before any model is fitted: it answers "how many bits of uncertainty about the target does knowing this one predictor remove?". It needs no distributional assumptions and handles mixed column types, but it is a univariate filter – each predictor is scored on its own, so two redundant copies of the same information both score highly, and a predictor that

only matters in combination with another scores low. Treat the ranking as a shortlist, not a final feature set.

- Numeric predictors are discretized into `max(Freedman-Diaconis, Sturges)` equal-width bins (never fewer than 2), unless `numeric_bins` overrides the count. Under the automatic rule, a column with a zero range or a zero interquartile range falls back to 2 bins; a column with a single distinct value becomes a single-level factor either way and scores 0.
- The target is always treated categorically, so `task` is always `"classification"`. Numeric targets are discretized the same way, ONCE on all rows with a non-NA target, so bin breaks are shared and scores are comparable across predictors.
- Date-like predictors are expanded into `*_year`, `*_month`, `*_day` columns (base R; the originals are dropped). Date-like targets are treated as categorical days.
- NAs are handled per predictor/target pair; rows are never dropped globally for other predictors. A predictor whose score is undefined (`NA`) is never selected.
- `remove_na` has a deliberately narrow effect: rows with NA in the target are excluded per pair anyway, so the observable difference is only when the target is entirely NA – `remove_na = TRUE` stops with an error, while `remove_na = FALSE` returns `NA` information gain for every predictor.

Value

An object of class `fs_result` with elements:

- `selected`: character vector of selected feature names, ordered by decreasing score. Features with an undefined (`NA`) score are never selected.
- `scores`: named numeric vector of the (possibly normalized) score for every candidate feature, ranked in decreasing order with `NA` scores last. For list input this is the union of features across data.frames; a name occurring in several data.frames keeps its highest score.
- `method`: `"infogain"`.
- `task`: `"classification"` (the target is always discretized).
- `model`: `NULL` (this is a filter; nothing is fitted).
- `details`: a list holding `table` (the full scored table, one row per scored column of each input: `Variable`, `InfoGain`, plus `SplitEntropy` and `GainRatio` when `normalize = "gain_ratio"`, plus `Origin` for list input), `normalize` (as resolved), `numeric_bins` (the requested bin count as an integer, `NULL` when automatic), `n_features` (the number of distinct scored features, i.e. `length(scores)`), and, for list input, `collisions` (a table of the feature names found in more than one data.frame, with `Kept` marking the row whose score won; zero rows when there are none).
- `call`: the matched call.

Cardinality bias and the gain ratio

Raw information gain systematically favors predictors with many distinct levels. In the limit, a near-unique identifier column splits the data into near-singleton groups, drives the conditional entropy $H(Y \mid X)$ to zero and therefore attains the largest gain any predictor can attain, $H(Y)$ – while carrying no generalizable signal whatsoever. Comparing raw gains across predictors of different cardinality is therefore comparing unlike things.

normalize = "gain_ratio" applies Quinlan's correction: each predictor's gain is divided by that predictor's own split entropy H(X), the entropy of its (discretized) level distribution. H(X) grows with cardinality – it is log2(k) for a predictor with k equally frequent levels – so dividing by it charges a predictor for the fineness of the split it makes. A constant predictor has H(X) = 0 and, rather than dividing by zero, is assigned a gain ratio of 0 (its gain is zero too). Because gain ratios are scaled gains, do not compare them against thresholds calibrated for raw gains in bits.

Examples

```
# Single data.frame:
df <- data.frame(
  A = rep(1:10, each = 10),
  B = rep(c("yes", "no"), 50),
  when = as.Date("2020-01-01") + rep(0:24, 4),
  target = rep(1:2, 50)
)
res <- fs_infogain(df, target = "target")
res$selected
res$scores
res$details$table

# Normalize by split entropy to offset the bias toward many-leveled
# predictors, and keep the two best-ranked features:
fs_infogain(df, target = "target", normalize = "gain_ratio", top_n = 2)

# List of data.frames:
df1 <- data.frame(A = rep(1:5, 20), target = rep(1:2, 50))
df2 <- data.frame(B = rep(c("yes", "no"), 50), target = rep(letters[1:2], 50))
fs_infogain(list(df1 = df1, df2 = df2), target = "target")
```

---

fs_lasso *Lasso Feature Selection with Cross-Validation*

Description

Fits a lasso (or elastic-net) model with glmnet::cv.glmnet() and reports which predictors survive at lambda.min. Numeric outcomes only (gaussian family).

Usage

```
fs_lasso(
  data,
  target,
  alpha = 1,
  nfolds = 5,
  standardize = TRUE,
  custom_folds = NULL,
  impute = c("none", "mean"),
  return_model = FALSE,
```

```
  seed = NULL,
  verbose = FALSE,
  parallel = FALSE,
  n_cores = 2L
)
```

Arguments

data	A data.frame or data.table holding the target and the candidate predictors. A numeric matrix with column names is also accepted; non-numeric matrices are not (supply a data.frame so factors and characters can be expanded by `model.matrix()`).
target	Single string naming the outcome column in `data`. It must be numeric and free of NA/NaN/Inf.
alpha	Numeric in (0, 1]; default 1 (lasso). Use values in (0, 1) for elastic-net. `alpha = 0` (pure ridge) is rejected: it never sets a coefficient to exactly zero, so it cannot select.
nfolds	Integer >= 3; default 5. Number of cross-validation folds passed to `glmnet::cv.glmnet()`, which rejects anything smaller than 3. When `custom_folds` is supplied those fold IDs define the folds instead, and `nfolds` only bounds which IDs are valid (so `custom_folds` must also define at least 3 folds).
standardize	Logical; default TRUE. Passed to `glmnet::cv.glmnet()` and also controls the scale `scores` is ranked on (see Details).
custom_folds	Optional integer vector of fold IDs (one per row of `data`, covering 1..`nfolds` with no empty folds); default NULL.
impute	How to handle missing predictor values: `"none"` (default) errors and names the offending columns, `"mean"` imputes column means computed on the full data and warns about the leakage.
return_model	Logical; keep the fitted cv.glmnet object in the result (and pass `keep = TRUE` to `cv.glmnet()`); default FALSE.
seed	Optional whole number for reproducibility, applied locally and restored on exit; default NULL (never seeds by default).
verbose	Logical; default FALSE. When TRUE, reports the parallel backend status and how many design-matrix columns were selected.
parallel	Logical; default FALSE. When TRUE, cross-validation runs on `n_cores` workers (requires the 'doParallel' and 'foreach' packages); if `n_cores` resolves to one worker it stays sequential and says so under `verbose = TRUE`.
n_cores	Integer >= 1; number of workers used only when `parallel = TRUE`. Default 2. Values above the detected core count are capped.

Details

Use this when the question is "which predictors keep a non-zero coefficient under a cross-validated L1 penalty (or, for `alpha < 1`, an elastic-net penalty)?". Selection happens at `lambda.min`, the penalty that minimizes cross-validated error; the more conservative `lambda.1se` is reported in `details` but is not used to select. The main caveat is that lasso tends to keep one member of a

group of strongly correlated predictors and zero out the rest, so an absent feature is not evidence that it is unrelated to the outcome.

The design matrix is built internally from every column of `data` except `target`, via `stats::model.frame(na.action = stats::na.pass)` followed by `stats::model.matrix()`: factors and characters are expanded to dummies and rows carrying NAs are preserved rather than silently dropped. The matrix carries no intercept column of its own (`glmnet` fits its own intercept, which is dropped from the reported coefficients), so `selected`, `scores` and `details$coefficients` name design-matrix columns – for a factor predictor, its expanded dummy columns rather than the original column.

`scores` ranks features on the STANDARDIZED coefficient scale when `standardize = TRUE`: each coefficient is multiplied by the standard deviation of its design-matrix column, which makes the ranking independent of the units the predictors happen to be measured in. glmnet standardizes internally for fitting but reports coefficients back on the input scale, so those raw coefficients are kept in `details$coefficients`. With `standardize = FALSE`, `scores` is the raw table and the two are identical.

Raw coefficient magnitude is a scale-dependent notion of importance: a predictor measured in small units earns a large coefficient for the same effect. That applies to `details$coefficients` always, and to `scores` when `standardize = FALSE`, so compare those numbers across predictors only when the predictors share a scale.

Missing predictor values are an error under the default `impute = "none"`, because imputing before cross-validation lets the folds see each other. `impute = "mean"` fills them with column means computed on the whole data set – not on the training part of each fold – and warns that this leaks. Columns that are entirely NA are an error either way.

Value

An `fs_result` object with:

`selected`	Design-matrix columns whose raw coefficient at `lambda.min` is non-zero, ordered by decreasing `scores` magnitude. (Selection reads the raw coefficients, so a constant column that glmnet nonetheless gave a non-zero coefficient is reported even though its standardized score is 0.)
`scores`	data.frame with columns Variable, Coefficient and AbsCoefficient, one row per design-matrix column (including those shrunk to zero), ordered by decreasing AbsCoefficient, on the standardized scale when `standardize = TRUE` (see Details).
`method`	`"lasso"`, regardless of `alpha`.
`task`	`"regression"` (gaussian family only).
`model`	The fitted cv.glmnet object when `return_model = TRUE`, else NULL.
`details`	List of `lambda_min` (lambda minimizing CV error, the one selection uses), `lambda_1se` (largest lambda within 1 SE of the minimum; reported only), `coefficients` (the same three-column table as `scores` but always on the raw coefficient scale) and `n_features` (number of design-matrix columns considered).
`call`	The matched call.

Examples

```
if (requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("Matrix", quietly = TRUE)) {
  n <- 100
  df <- data.frame(
    x1 = rnorm(n),
    x2 = rnorm(n),
    cat = sample(letters[1:3], n, TRUE)
  )
  df$y <- 2 * df$x1 - 3 * df$x2 + rnorm(n)
  res <- fs_lasso(df, "y", seed = 123)
  selected(res)
  head(res$scores)
}
```

`fs_mars` *MARS (earth) feature selection*

Description

Trains a Multivariate Adaptive Regression Splines model with `caret::train(method = "earth")` and repeated cross-validation, evaluates it on a held-out test set, and reports the predictors earth retained together with their variable importance.

Usage

```
fs_mars(
  data,
  target,
  train_ratio = 0.8,
  degree = 1:3,
  nprune = c(5, 10, 15),
  tuneLength = 10L,
  search = "grid",
  number = 5,
  repeats = 3,
  sample_size = 10000,
  corr_cut = 0.95,
  remove_nzv = TRUE,
  seed = NULL,
  verbose = FALSE,
  n_cores = 1L
)
```

Arguments

data	data.frame or data.table with predictors and the target. A data.table is copied, never modified in place.
target	Character. Name of the target column in data. A factor or character target is treated as classification, a numeric one as regression.
train_ratio	Numeric in (0, 1). Training proportion (default 0.8).
degree	Integer vector, each element >= 1. Interaction degrees to tune (default 1:3). Used when search = "grid".
nprune	Integer vector, each element >= 2. Numbers of retained terms to tune (default c(5, 10, 15)). Used when search = "grid".
tuneLength	Integer >= 1. Number of random hyperparameter combinations evaluated when search = "random" (default 10; ignored for grid search).
search	Character. "grid" (default) or "random"; see Details.
number	Integer >= 2. Cross-validation folds (default 5).
repeats	Integer >= 1. Cross-validation repeats (default 3).
sample_size	Integer >= 1. Maximum number of rows used; larger data sets are randomly down-sampled first, after missing rows are dropped and before the train/test split (default 10000).
corr_cut	Numeric between 0 and 1. Correlation cutoff for dropping highly correlated numeric predictors (default 0.95; 0 disables).
remove_nzv	Logical. Remove near-zero-variance predictors (default TRUE).
seed	Optional whole number for reproducibility, applied locally and restored on exit. Default NULL (never seeds by default). When NULL, no deterministic resampling seed lists are constructed either.
verbose	Logical. Print progress messages, caret's per-fold iteration log, and the small-class notice (default FALSE).
n_cores	Integer >= 1. Number of parallel workers for model training (default 1 = sequential; no cluster is created and caret's allowParallel stays FALSE). Values above the detected core count are capped; requires 'doParallel' and 'foreach' when greater than 1.

Details

Use this when you want a model that discovers non-linear effects and interactions on its own and then tells you which predictors it used: earth fits piecewise-linear hinge terms, prunes them back, and the surviving terms define the selected set. Selection is model-based rather than a filter, so it reflects one particular fitted model and its tuning.

The pipeline, in order: rows with any missing value are dropped; the data are randomly down-sampled to at most sample_size rows; for classification each class must have at least two rows; a stratified caret::createDataPartition() split keeps train_ratio of the rows for training and holds the rest out; near-zero-variance and strongly correlated predictors (see remove_nzv and corr_cut) are identified on the training rows only and dropped from both halves; then caret::train() fits earth under repeated cross-validation with preProcess = c("center", "scale").

Selection is read off the fitted model: `caret::varImp(model, scale = FALSE)` is computed on the caret `train` object and every predictor with a strictly positive importance is reported in `selected`. Predictors earth pruned away score 0 and are not selected. caret expands factors into dummy columns before fitting, so each importance row is attributed back to the source predictor by name (the longest candidate name it starts with) and a predictor keeps the largest importance of its encoded columns; that name-based attribution is ambiguous if one predictor's name happens to be a prefix of another predictor's encoded column name.

For classification, class imbalance is handled by `sampling = "up"` inside `caret::trainControl()`, i.e. upsampling happens within each resample; the data are never upsampled before cross-validation (which would leak duplicated rows across folds). The FIRST factor level is treated as the positive class for ROC/PR AUC, and the test-set ROC AUC is computed with a fixed direction, so a worse-than-chance model scores below 0.5 rather than being silently flipped. Factor levels are sanitized with `make.names(unique = TRUE)`, so distinct labels can never be merged.

For regression, the reported `R2` comes from `caret::R2()`, which is the squared correlation between predictions and observations – not `1 - SSE/SST` – and can be high even for a biased model.

Everything in `details$metrics` comes from the single held-out split, so on small data sets these numbers are noisy; the per-resample tuning results are in `model$resample`.

`search = "grid"` tunes over `expand.grid(nprune, degree)`; `search = "random"` ignores that grid and evaluates `tuneLength` random hyperparameter combinations instead.

Value

An object of class `fs_result` with:

selected Character vector of the predictors earth retained (strictly positive `caret::varImp()` importance), ordered by decreasing importance. Empty when no importance is available.

scores Named numeric vector of unscaled variable importance, one entry per candidate predictor that entered training, floored at 0 (predictors earth pruned away score 0). `NULL` when caret cannot compute variable importance for the fitted model, in which case `selected` is empty.

method "mars".

task "classification" for a factor or character target (characters are coerced to factor), "regression" for a numeric one.

model The `caret::train` object.

details A list with `predictions` (test-set predictions; a factor carrying the training levels for classification), `metrics` (RMSE/MAE/R2 for regression; Accuracy/Kappa plus ROC_AUC/PR_AUC when the optional 'pROC'/'PRROC' packages are installed for binary classification), `confusion_matrix` (classification only, else NULL), `varimp` (the `caret::varImp()` object, or NULL when unsupported), `removed_predictors` (a list with `nzv` and `corr` naming the dropped predictors), `train_index` (integer row indices of the training rows, into the cleaned and optionally downsampled data), `test_data` (the held-out rows after preprocessing) and `n_features` (number of candidate predictors that entered training).

call The matched call.

Examples

```
if (requireNamespace("caret", quietly = TRUE) &&
```

```
  requireNamespace("earth", quietly = TRUE)) {
# x1 and x2 drive y; x3 is noise
df <- data.frame(
  x1 = rnorm(150),
  x2 = rnorm(150),
  x3 = rnorm(150)
)
df$y <- 2 * df$x1 - df$x2 + rnorm(150, sd = 0.5)
res <- fs_mars(df, "y", degree = 1, nprune = c(5, 10),
               number = 3, repeats = 1, seed = 42)
res$selected
res$scores
res$details$metrics
}
```

`fs_pca` *Principal component analysis with tidy results and optional plotting*

Description

Answers "how much of the spread in these numeric columns lives in a handful of directions, and which columns drive them?" Runs a PCA on the numeric columns of `data`. Character and factor columns are excluded from the feature set and kept as label candidates; an explicitly supplied `label_col` (numeric or not) is likewise excluded from the features and only used for labeling. Rows with missing values in the numeric columns and zero-variance columns are dropped before the decomposition.

Usage

```
fs_pca(
  data,
  num_pc = NULL,
  scale_data = TRUE,
  center_data = TRUE,
  label_col = NULL,
  plot = FALSE,
  verbose = FALSE
)
```

Arguments

`data`	A data.frame or data.table with at least two rows and at least one numeric column.
`num_pc`	Number of principal components to retain: a whole number `>= 1`, or `NULL` (the default) to retain `min(2, max_possible)`. `max_possible` is `min(nrow - 1, ncol)` of the *usable* numeric data, that is after incomplete rows and zero-variance columns have been dropped. Explicit values larger than `max_possible` raise an error.

scale_data	Logical; scale numeric columns to unit variance. Cannot be TRUE while center_data is FALSE: stats::prcomp() and bigstatsr::big_scale() treat uncentered scaling differently (the latter ignores it), so allowing it would make the decomposition depend on which engine the data size selected. That combination is an error. Default TRUE.
center_data	Logical; center numeric columns. Default TRUE.
label_col	Optional single string naming a column of data used to label and color points. The column is excluded from the PCA features but carried into pca_df. Default NULL, which means no labels and therefore no plot.
plot	Logical; if TRUE, the PC1 vs PC2 scatterplot is printed. Default FALSE, unconditionally: it does not depend on whether label_col was supplied. The plot object is returned in $plot either way whenever one can be built (see Details).
verbose	Logical; emit progress messages (the non-numeric columns held back as labels, which engine was chosen, and any fallback from 'bigstatsr'). Default FALSE.

Details

The caveat is what PCA is. This is unsupervised dimensionality reduction, not feature selection: every component is a linear combination of *all* the retained numeric columns, chosen without reference to any outcome. A PCA therefore does not shorten the list of variables you have to measure, and the leading components are the highest-variance directions, which need not be the ones related to a response.

Data with fewer than 1e7 cells is decomposed with stats::prcomp(). From 1e7 cells up, bigstatsr::big_SVD() runs on a temporary file-backed matrix (the backing file is deleted when the call returns) if the suggested package 'bigstatsr' is installed; otherwise the call falls back to prcomp(), which is reported only when verbose = TRUE.

var_explained reports the proportion of *total* variance explained by each retained component under both engines, so it sums to 1 only when every available component is retained; with the default two components it sums to the fraction those two capture. The engines obtain that denominator differently. prcomp() computes every component, so the total is the sum of all the eigenvalues. big_SVD() computes only the top num_pc singular values, which are *not* the whole spectrum, so the total variance is recovered separately from the column statistics of the (implicitly centered and scaled) matrix. Under the 'bigstatsr' engine in particular, read the entries as shares of the whole and never as shares of the components that happened to be computed.

plot controls printing only, never construction. Whenever a plot can be built at all (a label_col was supplied, at least two components were retained, and the suggested package 'ggplot2' is installed) the ggplot object is returned in $plot, whether or not plot is TRUE. Setting plot = TRUE additionally prints it, and in that case 'ggplot2' is required: its absence becomes an error rather than a silently missing $plot. plot = TRUE without a label_col, or with fewer than two retained components, warns and skips the plot.

Value

A plain list. fs_pca() is dimensionality reduction rather than feature selection, so it returns its own PCA structure and not the fs_result object produced by the package's selection functions. The components are:

- pc_loadings: numeric matrix of variable loadings, one row per surviving numeric feature and one column per retained PC, with the feature names as row names and PC1, PC2, ... as column names.
- pc_scores: numeric matrix of observation scores, one row per kept observation and one column per retained PC.
- var_explained: numeric vector of length num_pc, the proportion of total variance carried by each retained PC (see Details, especially for the large-data engine).
- pca_df: data.table of the scores with the label columns (every character/factor column, plus label_col) bound alongside, restricted to the rows that were kept.
- meta: list with numeric_cols (the features actually decomposed), rows_kept (logical vector over the rows of data), n_rows_used, and n_cols_used.
- plot: the ggplot object. Present only when a plot could be built, that is when label_col was supplied, at least two PCs were retained, and 'ggplot2' is available. Absent otherwise, so test for it with "plot" %in% names(res) rather than assuming it is there.

Examples

```
res <- fs_pca(mtcars, num_pc = 2, label_col = "cyl")
res$var_explained
head(res$pca_df)


if (requireNamespace("ggplot2", quietly = TRUE)) {
  # plot = FALSE (the default) draws nothing, but $plot is built anyway
  res <- fs_pca(iris, label_col = "Species", verbose = TRUE)
  print(inherits(res$plot, "ggplot"))

  # plot = TRUE draws it as well
  res <- fs_pca(iris, label_col = "Species", plot = TRUE)
}
```

---

fs_randomforest *Random forest importance and held-out evaluation*

Description

Answers "how much does each predictor contribute to a random forest, and how well does that forest do on rows it has never seen?" The pipeline runs in this order: optional preprocessing of the full table, target cleaning, optional downsampling, a stratified train/test split, the optional control$feature_select hook on the training rows, character-to-factor coercion and level alignment, near-zero-variance removal, imputation, training (optionally across several workers), and evaluation on the held-out rows. Permutation importance is reported as the per-feature score.

Usage

```
fs_randomforest(
  data,
  target,
  task = c("classification", "regression"),
  control = list(),
  seed = NULL,
  verbose = FALSE,
  n_cores = 1L
)
```

Arguments

data	A data.frame or data.table with at least one row and one column, holding the target and the candidate predictors (every other column). Date columns are converted to numeric before modeling.
target	Single string naming the target column of data; a column index is not accepted.
task	One of "classification" (the default) or "regression", matched with match.arg(). A classification target is coerced with as.factor() and must retain at least two levels; a regression target is coerced with as.numeric(). Either way, rows whose target is NA after coercion are dropped with a warning.
control	List of method-specific options; every accepted entry and its default is listed under Details. Unknown entries are an error, not a silent no-op. Default list(), i.e. all defaults.
seed	Optional single finite number for reproducibility, truncated to an integer with as.integer(). Applied for the duration of the call only; the previous RNG state is restored on exit. With more than one worker it is also handed to parallel::clusterSetRNGStream(), and is then the only way to make the run reproducible. Default NULL (the RNG is never seeded unless requested).
verbose	Logical; emit progress messages (split sizes, how many predictors the hook kept, the tree/worker counts, and a note when AUC is skipped because 'pROC' is missing). Default FALSE.
n_cores	Whole number >= 1. Number of workers used to grow the forest. Default 1L (sequential; no cluster is created). Capped at the detected core count and at control$ntree; an effective value above 1 requires the suggested packages 'foreach' and 'doParallel', and costs the OOB metrics as described below.

Details

Random forests take any mix of numeric, factor, character, and Date predictors and need no scaling, which makes this a good default when the feature types are messy. The main caveat is that a forest *ranks* rather than *selects*: unless you supply control$feature_select, selected is every predictor that reached the forest, ordered by importance, and choosing the cutoff is left to you. Permutation importance also tends to favor predictors with many distinct values and to split credit between correlated predictors, so treat close scores as ties.

control list (every accepted entry, with its default):

- train_ratio = 0.75 – training proportion of the stratified split; must be strictly between 0 and 1.
- sample_size = NULL – optional whole number. When supplied, the data is downsampled to approximately this many rows *before* the split (proportionally within each target class for classification, uniformly at random for regression). Values above the available row count are capped. NULL keeps every row.
- ntree = 500 – number of trees to grow; whole number >= 1.
- importance = TRUE – compute permutation importance. When FALSE, scores and details$importance are NULL and selected keeps plain column order.
- scale_importance = TRUE – passed as scale to randomForest::importance(), which divides each mean decrease in accuracy by its permutation standard error. Set to FALSE for the raw, unscaled permutation importance.
- mtry = NULL – predictors sampled at each split. NULL means floor(sqrt(p)) for classification and floor(p / 3) for regression; any value, supplied or derived, is clamped to [1, p], where p counts the predictors that survive selection and near-zero-variance removal.
- nodesize = NULL – minimum size of terminal nodes; whole number >= 1. NULL leaves randomForest's own default (1 for classification, 5 for regression).
- maxnodes = NULL – maximum number of terminal nodes per tree; whole number >= 2, or NULL for no limit.
- sampsize = NULL – rows drawn to grow each tree, passed to randomForest. A single number for regression (a vector is an error); one number per class is allowed for classification. Values are clamped to the training row count, and NA entries are replaced by the full training size when replace = TRUE and by ceiling(0.632 * n) otherwise.
- classwt = NULL – class priors for classification, forwarded to randomForest unchanged and unvalidated.
- strata = NULL – stratification variable for the per-tree sampling, forwarded to randomForest unchanged and unvalidated.
- replace = TRUE – sample rows with replacement when growing each tree.
- preprocess = NULL – function dt -> dt, applied to the full data before the split. It must return a data.frame/data.table that still contains the target. See the leakage note below.
- feature_select = NULL – function dt -> dt; runs on the training split only, and must select existing columns while retaining the target – see the note below.
- impute = TRUE – median (numeric) or modal (factor/character) values learned on the training data for every predictor and applied to NAs in train and test.
- drop_zerovar = TRUE – near-zero variance removal with caret::nearZeroVar(), using training data only.
- oob = TRUE – include out-of-bag metrics in details$oob (accuracy for classification, RMSE for regression); see the note below about parallel training.
- return_test_data = FALSE – when TRUE, the held-out rows are returned in details$test_data.
- positive_class = NULL – optional level name treated as the positive class for binary AUC. Falls back to the second factor level both by default and, silently, when the string does not name a level of the target.

Unknown control entries are rejected, so typos and arguments that moved out of control (seed, n_cores, the former split_ratio) fail loudly.

Selection is train-only: control$feature_select runs *after* the train/test split and sees the training rows only, so a selection rule that looks at the target no longer leaks the held-out rows into the reported test metrics. Because the same columns must be applied to the test rows, the hook may only subset and reorder existing columns; put any feature engineering in control$preprocess, which runs on the full data before the split – subject to the caveat in the next paragraph. Near-zero-variance removal happens after the hook, so details$feature_names (the predictors the forest actually used) can be a subset of selected.

What still sees the full data: control$preprocess runs before the split by design, so any quantity it learns – an imputation value, a scaling constant, anything that consults the target – is computed from the held-out rows too and will flatter details$metrics. Keep it to row-wise transformations that do not depend on other rows. Optional downsampling (control$sample_size) also happens before the split, which is why details$train_index indexes the cleaned and down-sampled table rather than the rows of the original data.

OOB metrics and parallel training: when more than one worker is actually used (n_cores after the caps described above) the forest is assembled with randomForest::combine(), which drops the out-of-bag error structures (err.rate, mse, confusion). In that case details$oob is NULL and a warning is emitted if control$oob = TRUE. The held-out metrics in details$metrics are unaffected; use n_cores = 1 when you want the OOB numbers.

Character predictors are converted to factors, with the levels learned on the training split. Test values unseen in training become NA and are imputed when control$impute = TRUE; with control$impute = FALSE those NAs survive into predict(), which randomForest cannot handle. details$metrics$auc is filled in only when the classification target has exactly two levels *and* the suggested package 'pROC' is installed; it is NA otherwise, which includes every multi-class fit. Regression fits report RMSE, MAE, and R2 instead, and R2 is NA when the held-out target has no variance.

Value

An object of class fs_result with:

selected Character vector. The predictors kept by control$feature_select when a hook is supplied; otherwise every predictor that reached the forest, since a plain random forest ranks rather than selects. Ordered by decreasing importance when importance was computed.

scores Named numeric vector of permutation importance from randomForest::importance(type = 1) – mean decrease in accuracy for classification, percent increase in MSE for regression – scaled or not according to control$scale_importance. NULL when control$importance = FALSE.

method "randomforest".

task "classification" or "regression", echoing the task argument.

model The fitted randomForest object; the result of randomForest::combine() when more than one worker was used.

details A list with metrics (a named list: accuracy, kappa, auc for classification, or RMSE, MAE, R2 for regression), predictions (test-set predictions), probabilities (test-set class probability matrix, classification only, NULL if predict() cannot produce one), importance (data.frame with columns feature and importance, sorted descending, NULL when control$importance

= FALSE), confusion (Observed x Predicted table, classification only), oob (accuracy for classification or RMSE for regression; NULL when trained in parallel or switched off), feature_names (predictors the forest actually used), train_index (integer training rows of the cleaned and optionally down-sampled data), test_data (the held-out rows, only when control$return_test_data = TRUE, NULL otherwise), control (the merged control list) and n_features (candidate predictors after cleaning and before selection).

call The matched call.

Examples

```
if (requireNamespace("randomForest", quietly = TRUE) &&
    requireNamespace("caret", quietly = TRUE)) {
  res <- fs_randomforest(
    iris,
    target = "Species",
    task = "classification",
    control = list(ntree = 100),
    seed = 42
  )
  # every predictor, ranked: a forest ranks rather than selects
  print(res$selected)
  print(res$scores)
  print(res$details$metrics)
}
```

---

fs_recursivefeature *Recursive feature elimination with held-out evaluation*

Description

Answers "how few predictors can I keep before resampled performance starts to fall off?" Splits the data into stratified train/test partitions, optionally one-hot encodes the predictors (encoder fitted on the training rows only), runs caret::rfe() on the training set, evaluates the fitted RFE model on the held-out test rows, and optionally trains a final caret model on the training rows.

Usage

```
fs_recursivefeature(
  data,
  target,
  sizes = NULL,
  train_ratio = 0.8,
  rfe_control = list(method = "cv", number = 5),
  train_control = list(method = "cv", number = 5),
  model_method = "rf",
  handle_categorical = FALSE,
  return_final_model = FALSE,
```

```
  seed = NULL,
  verbose = FALSE,
  parallel = FALSE
)
```

Arguments

data	A data.frame or data.table with at least one row and one column, holding the target and the candidate predictors (every other column). It is converted to a plain data.frame on entry.
target	Single string naming the target column of data; a column index is not accepted. A factor, character, or logical target means classification, anything else regression.
sizes	Numeric vector of feature-subset sizes to evaluate. Default NULL, which uses 1:p, where p is the predictor count after any one-hot encoding. Values outside [1, p] are dropped with a warning; if that leaves nothing, the call is an error rather than a silent empty run.
train_ratio	Numeric, strictly between 0 and 1: the training proportion of the stratified split. Default 0.8.
rfe_control	List of arguments for caret::rfeControl(); must contain at least method and number. Default list(method = "cv", number = 5). functions selects the caret RFE function set and defaults to caret::rfFuncs. An allowParallel entry is dropped with a warning (use the parallel argument instead), while a verbose entry, if present, overrides the verbose argument for caret::rfe(). With method = "repeatedcv" and no repeats, caret's default of 1 is used and a warning says so.
train_control	List of arguments for caret::trainControl(), used only when return_final_model = TRUE. Must contain method, and also number unless method = "none". Default list(method = "cv", number = 5).
model_method	Single string; the caret model key used for the final model, for example "rf" or "lm". Used only when return_final_model = TRUE. Default "rf".
handle_categorical	Logical; one-hot encode predictors with full-rank dummies (fitted on the training rows, applied to the test rows). Default FALSE.
return_final_model	Logical; train a final caret model on the training rows using the selected features and return it as model, with the rfe object still available in details$rfe. Near-zero-variance and linearly dependent predictors are dropped from the selected set first, each with a warning, and what survives is recorded in details$final_model_variables. Default FALSE.
seed	Optional single finite number, truncated to an integer with as.integer(). It covers the split and the RFE resampling, is applied for the duration of the call only (the previous RNG state is restored on exit), and defaults to NULL, which never seeds. Under parallel = TRUE the seed is also used to set reproducible L'Ecuyer-CMRG streams on the workers, so two seeded parallel runs agree with each other; because the workers draw from their own streams, a seeded parallel run need not equal a seeded sequential one.

verbose	Logical; print progress messages and let caret::rfe() report its own progress, unless rfe_control$verbose overrides the latter. Default FALSE.
parallel	Logical. If TRUE, registers a two-worker PSOCK cluster (capped at the detected core count) for the duration of the call and stops it again when the call returns; requires the suggested packages 'foreach' and 'doParallel'. Default FALSE.

Details

RFE is a wrapper method: it refits the underlying model once per candidate subset size per resample, so it is by far the most expensive method here, and its answer is specific to the model family in rfe_control$functions rather than being a general statement about the features. In exchange, the subset it reports is tuned to the model you actually intend to use, and the subset size is chosen by resampling instead of by a threshold you invent.

Requires the suggested package 'caret'. The RFE function set comes from rfe_control$functions and defaults to caret::rfFuncs, which fits random forests, so the suggested package 'randomForest' must also be installed unless you supply a different set (for example rfe_control = list(method = "cv", number = 5, functions = caret::lmFuncs)). Parallel execution additionally requires 'foreach' and 'doParallel'; classification metrics use caret::postResample(), which needs 'e1071'.

Everything that could leak is fitted on the training rows: the one-hot encoder, the factor levels the test columns are aligned to, the elimination itself, and the final model. details$test_metrics is therefore a genuine held-out estimate. It applies caret::postResample() to the predictions of the fitted rfe object (caret's own refit on all the training rows, restricted to the optimal subset) on test rows that took no part in choosing either the features or the subset size.

Two other summaries on the result are *not* held-out estimates, by construction. details$resampling_results summarizes resampling performed inside the training rows across candidate subset sizes, and, when return_final_model = TRUE, model$results reports train_control resampling inside those same training rows on features that were already selected. Only details$test_metrics is computed on data the search never saw.

Missing values in the *training* predictors are rejected with an error; impute or drop incomplete rows before calling. NAs in the held-out rows are not checked here and will propagate through predict() into details$test_metrics. With handle_categorical = TRUE the encoded test rows are row-count checked against their input, so an encoding that quietly loses rows becomes an error rather than a silently misaligned metric.

Value

An object of class fs_result with:

selected Character vector of the variables RFE kept (optVariables at the optimal subset size).

scores Named numeric vector of resample-averaged importance from caret::varImp() on the rfe object: its "Overall" column when caret supplies one, otherwise the row means of whatever numeric columns it did supply. NULL when there is nothing usable.

method "rfe".

task "classification" or "regression", inferred from the target.

model The final caret::train model when return_final_model = TRUE, otherwise the rfe object.

details A list, in snake_case, with `rfe` (the caret `rfe` object, always present even when `model` holds the final model), `optimal_size` (the subset size RFE chose), `test_metrics` (`caret::postResample()` on the held-out rows – the only held-out estimate on the object), `resampling_results` (the RFE resampling summary over candidate sizes, computed inside the training rows), `variable_importance` (the `caret::varImp()` data.frame), `preprocessor` (the `dummyVars` encoder, or `NULL` when `handle_categorical = FALSE`), `train_index` and `test_index` (row indices into `data` for the two partitions), `final_model_variables` (predictors the final model actually used after NZV/linear-combination filtering, or `NULL` when no final model was requested) and `n_features` (candidate predictors offered to RFE, counted after one-hot encoding when `handle_categorical = TRUE`).

call The matched call.

Examples

```
if (requireNamespace("caret", quietly = TRUE) &&
    requireNamespace("randomForest", quietly = TRUE) &&
    requireNamespace("e1071", quietly = TRUE)) {
  res <- fs_recursivefeature(
    iris,
    target = "Species",
    sizes = 1:4,
    rfe_control = list(method = "cv", number = 3),
    seed = 42
  )
  print(res$selected)
  print(res$details$optimal_size)
  # the only held-out estimate on the object
  print(res$details$test_metrics)
}
```

`fs_stepwise` *Stepwise linear-regression feature selection via AIC*

Description

Answers "which subset of these columns does AIC keep in a linear model?" Uses `MASS::stepAIC()` to perform forward, backward, or both-direction stepwise selection on a linear regression of `target` against all other columns of `data`. For `direction = "forward"` and `"both"` a proper null model and scope are set up so that forward moves are possible.

Usage

```
fs_stepwise(
  data,
  target,
  direction = c("both", "backward", "forward"),
  verbose = FALSE,
  ...
)
```

Arguments

data	A data.frame (or data.table) with at least two columns: the numeric target and at least one candidate predictor. Every column other than target is offered to the search.
target	Single string naming the numeric target column of data. Unquoted symbols and column indices are not accepted. Non-syntactic names are backticked into the formula, so they work.
direction	Direction of the search: "both" (the default), "backward", or "forward", matched with match.arg().
verbose	Logical. If TRUE, emits progress messages and enables the stepAIC() trace output on the console. Default FALSE.
...	Additional arguments passed to MASS::stepAIC(), for example k or steps. trace is the one exception: it is controlled by verbose, and a user-supplied trace is dropped with a warning. See Details for what else ... quietly absorbs.

Details

This is the cheapest wrapper method in featR and the easiest to read: what comes back is an ordinary lm that the usual generics work on. The price is that the search is greedy, so it can walk past the AIC-best subset, and unstable, so small changes in the data can change the retained set. On top of that, the statistics it reports about its own answer are not valid; see the post-selection inference caveat below.

Requires the suggested package 'MASS'. The function is linear-regression-only: the target must be numeric, and every other column of data is offered to the search as a candidate predictor. There is no seed argument, because stepAIC() is deterministic: repeating a call on the same data returns the same model, and the RNG is never touched.

Post-selection inference caveat. The reported scores (absolute t statistics) and the p-values in details$coefficients are computed on the same data that drove the search. They are optimistically biased – the selective-inference problem – and must not be used for formal inference, only as a rough ordering of the retained terms.

The fitted models reach their data through a private environment attached to the model formula, so predict(), summary(), anova(), add1()/drop1() and similar generics keep working on the returned model; nothing is assigned to the global environment and nothing is written to disk. That environment holds a copy of the complete-case data and is kept alive by the returned model, so the result carries the data with it and is correspondingly large. To refit the returned model with update() from another environment, pass the data explicitly, e.g. update(model, . ~ . - x, data = my_data). Plain update(model) does *not* work: update() re-evaluates the stored call in the *caller's* environment, where the private data object is not visible, so it fails with an object-not-found error.

... is forwarded to MASS::stepAIC() verbatim, which means it also absorbs arguments this function does not have. The removed seed and return_models are passed through and ignored rather than rejected, so a call written against the older API still runs and quietly does nothing with them; check your argument names against the list below.

Rows containing missing values in any column are dropped (with a warning) before the search, because stepAIC() cannot compare models fitted on differing row sets.

Value

An object of class `fs_result` with:

selected Character vector of the selected predictor terms (excluding the intercept).

scores Named numeric vector of absolute t statistics from the final model's coefficient table, excluding the intercept. **Caveat**: these statistics (and the p-values in `details$coefficients`) are computed after selection on the same data, so they are optimistically biased and are not valid for inference.

method "stepwise".

task "regression"; `fs_stepwise()` fits linear models only.

model The `lm` selected by `stepAIC()`. It carries a copy of the complete-case data in a private environment (see Details).

details A list with `final_model` (the same `lm`), `coefficients` (the `summary()` coefficient matrix, same caveat as `scores`), `selected_terms` (the same term labels as `selected`), `direction` (the search direction actually used), `n_features` (the number of candidate predictors offered to the search, i.e. `ncol(data) - 1`) and `dropped_na_rows` (how many rows were removed for missing values, `0L` when none were).

call The matched call.

Examples

```
if (requireNamespace("MASS", quietly = TRUE)) {
  res <- fs_stepwise(mtcars, target = "mpg", direction = "both")
  print(res$selected)
  # |t| of the retained terms: a rough ordering, not valid inference
  print(res$scores)

  # the returned model is an ordinary lm, but update() needs 'data'
  refit <- stats::update(res$model, . ~ . - wt, data = mtcars)
  print(stats::formula(refit))
}
```

`fs_supervised` *Supervised Filter-Based Feature Selection*

Description

Performs supervised, univariate, filter-based feature selection: every column of `data` except `target` is scored against the target, and features are selected or dropped by comparing that score to `threshold`.

Usage

```
fs_supervised(
  data,
  target,
  method = c("auto", "correlation", "anova"),
  threshold = 0,
  direction = c("above", "below"),
  action = c("keep", "remove"),
  include_equal = FALSE,
  na_rm = TRUE,
 output = c("result", "matrix", "dt", "data.frame", "mask", "indices", "names", "list"),
  verbose = FALSE
)
```

Arguments

data	A data.frame, data.table, or matrix containing target and the candidate feature columns. Every column other than target must be numeric. The input is copied, never modified in place.
target	Character. Name of the target column in data. It is removed from the candidate features and used as the scoring target. The target column itself may be numeric, factor, character, or logical; any other type is an error.
method	One of "auto" (default), "correlation", "anova".
threshold	Non-negative, finite numeric scalar threshold applied to the feature scores (not to the target directly). Default 0. Note that the two methods put scores on different scales: [0, 1] for "correlation" and [0, Inf) for "anova".
direction	One of "above" (default), "below"; compares scores to threshold.
action	One of "keep" (default), "remove"; determines whether features meeting the condition are retained or dropped.
include_equal	Logical; if TRUE, comparisons are inclusive (greater/less than or equal) instead of strict. Default FALSE.
na_rm	Logical; if TRUE (the default), rows with NA in a feature or in the target are dropped when computing that feature's score. If FALSE, an NA anywhere in a feature or the target makes that feature's score undefined.
output	One of "result" (default), "matrix", "dt", "data.frame", "mask", "indices", "names", "list". Throughout, "candidate features" means the columns of data other than target, in their original order. • "result" (default): an fs_result object (see Value). • "matrix": numeric matrix of the selected features. • "dt": data.table of the selected features. • "data.frame": data.frame of the selected features. • "mask": logical vector, one element per candidate feature, named after the candidate features; TRUE marks a selected column. • "indices": integer vector of the selected column indices, indexing the candidate features (that is, data without target), named after the selected columns.

- "names": character vector of the selected column names.
- "list": list with components filtered (matrix), mask, indices, names, scores (all as above), and meta, a list recording method_arg (the method as requested), method_used (the method after "auto" was resolved), threshold, direction, action, include_equal, na_rm, n_input_cols, and n_kept_cols.

verbose Logical; emit progress messages. Default FALSE.

Details

This is one of the cheapest supervised screens in featR: no predictive model is fitted, each feature is scored on its own, and the cost is linear in the number of columns, so it scales to wide data. The flip side is that scoring is strictly univariate: it cannot see interactions between features, and it will happily keep a whole group of near-duplicate columns that all correlate with the target. Use fs_correlation() to prune that redundancy afterwards, or a wrapper such as fs_recursivefeature() or fs_svm() when the joint contribution is what matters.

Supported methods:

- "correlation": Absolute Pearson correlation (numeric target), so scores lie in [0, 1].
- "anova": One-way ANOVA F-statistic (categorical target), so scores lie in [0, Inf).
- "auto": Chooses "correlation" for a numeric target and "anova" for a categorical one. Because the two score scales differ, a message reports the resolved method when verbose = TRUE.

Features whose score is undefined (NA) are never selected, under both action = "keep" and action = "remove"; a warning reports how many such features were excluded.

Columns are subset by integer index, never by name, so duplicated column names cannot select the wrong columns.

Value

With the default output = "result", an object of class fs_result with elements:

- selected: character vector of selected feature names.
- scores: named numeric vector of per-feature scores, in column order, NA where the score is undefined.
- method: "supervised_" followed by the resolved scoring method, for example "supervised_correlation".
- task: "regression" for a numeric target, "classification" for a factor target, NA otherwise.
- model: NULL.
- details: a list holding, in this order, mask (the logical keep-mask over the candidate features), indices (the selected column indices, named after the selected columns), filtered (the selected columns as a data.table), threshold, direction, action, and n_features (the number of candidate features, that is ncol(data) - 1).
- call: the matched call.

Any other `output` returns that shape instead, exactly as documented above. When no feature meets the selection criteria, a warning is issued and the tabular shapes come back empty: `"matrix"` and `"data.frame"` have zero columns and keep the input row count, while the `"dt"` shape (and `details$filtered`) is only guaranteed to have zero columns – data.table represents a zero-column table as having zero rows, so its row count is not preserved.

Examples

```
df <- data.frame(
  strong = c(1, 2, 3, 4),
  mirror = c(4, 3, 2, 1),
  weak   = c(1, 0, 1, 0),
  y      = c(1, 2, 3, 4)
)

# Default: an fs_result
res <- fs_supervised(df, target = "y", method = "correlation",
                     threshold = 0.5)
res$selected
res$scores
res$details$filtered

# The classic shapes are still available
fs_supervised(df, target = "y", method = "correlation", threshold = 0.5,
              output = "names")

# ANOVA against a factor target
df_fac <- data.frame(
  wide = c(1, 2, 10, 11),
  mild = c(1, 3, 2, 4),
  grp  = factor(c("a", "a", "b", "b"))
)
fs_supervised(df_fac, target = "grp", method = "anova", threshold = 1,
              output = "matrix")
```

`fs_svd` *Singular Value Decomposition with Optional Scaling and Truncation*

Description

Computes the SVD of a matrix with options for centering/scaling, truncation to the leading singular triplets, and an approximate solver for large matrices.

Usage

```
fs_svd(
  x,
  n_singular_values = NULL,
  scale_input = TRUE,
```

```
  svd_method = c("auto", "exact", "approx"),
  svd_threshold = 100,
  approx_args = list(),
  verbose = FALSE
)
```

Arguments

x	Numeric matrix, or a data.frame whose columns are all numeric (it is coerced to a matrix). Must contain no NA/NaN/Inf values. The argument is called x rather than data because it is a matrix of values, not a table of observations and features.
n_singular_values	Positive whole number of singular values and vectors to keep, or NULL (default) for min(dim(x)). Values exceeding min(dim(x)) are an error.
scale_input	TRUE (center and scale, the default), FALSE (leave the matrix alone), "center" (subtract column means only), or "scale" (divide only). Any other value is an error. The two dividing forms (TRUE and "scale") require every column to have non-zero variance, and offending columns are reported by name (or by position when the matrix has no column names); "center" has no such requirement. Note that "scale" inherits base::scale()'s semantics: with no centering it divides by the root mean square, not by the standard deviation.
svd_method	"auto" (default), "exact", or "approx". See Details for how "auto" decides and when "approx" falls back to the exact solver.
svd_threshold	Positive number; with svd_method = "auto", the approximate solver is considered only when min(dim(x)) exceeds this value (default 100).
approx_args	List of extra arguments passed to RSpectra::svds() (for example tol or opts). Must not include A or k, which are set internally.
verbose	Logical; emit progress messages. Default FALSE.

Details

Reach for this when you want a low-rank summary of a numeric matrix – the leading components for compression, denoising, or a latent-factor representation – rather than a subset of the original columns. Because the components are linear combinations of every input column, nothing is dropped and individual features stay uninterpretable; use one of the package's selection functions when you need to name the features you keep. fs_pca covers the same ground with tidy, labeled output and variance-explained figures; fs_svd() is the lower-level decomposition.

The exact path uses base::svd(). The approximate path uses RSpectra::svds() (package **RSpectra**, a Suggests dependency, required only when that path is actually taken) and is only applicable when fewer than min(dim(x)) singular values are requested. By default n_singular_values = NULL resolves to min(dim(x)), which implies the exact path; to enable the approximate solver on a large matrix, request n_singular_values < min(dim(x)).

With svd_method = "auto", the approximate solver is chosen only when n_singular_values < min(dim(x)) and min(dim(x)) > svd_threshold; otherwise the exact solver is used, and a message explains why when a large matrix still ends up on the exact path because all singular values were requested. Asking for svd_method = "approx" outright when n_singular_values

equals `min(dim(x))` is not an error: it falls back to the exact solver with a message, because `RSpectra::svds()` cannot return the full set.

Centering and scaling, when requested, happen *before* the decomposition, so the returned triplets factorize the transformed matrix rather than the raw input: with the default `scale_input = TRUE` they reconstruct `scale(x)`, not `x`. The transformation itself is not returned, so keep the column means and standard deviations yourself if you need to map results back to the original units.

Value

A plain list. `fs_svd()` is dimensionality reduction rather than feature selection, so it returns its own decomposition structure and not the `fs_result` object produced by the package's selection functions. Writing `k` for the number of triplets kept (`n_singular_values`, or `min(dim(x))` by default), the components are:

- `singular_values`: numeric vector of length `k`, in non-increasing order.
- `left_singular_vectors`: `nrow(x)` by `k` matrix (the `u` of `base::svd()`).
- `right_singular_vectors`: `ncol(x)` by `k` matrix (the `v` of `base::svd()`).

Singular vectors are unique only up to sign (and up to rotation within a tied block), so column signs may differ between the exact and approximate solvers and between platforms.

Examples

```
m <- matrix(
  c(4, 0, 0, 3,
    0, 5, 1, 2,
    2, 1, 6, 0,
    1, 3, 2, 7,
    5, 2, 0, 1,
    0, 4, 3, 2),
  nrow = 6, ncol = 4, byrow = TRUE
)

# Two leading components of the centered and scaled matrix
res <- fs_svd(m, n_singular_values = 2, scale_input = TRUE)
res$singular_values
res$left_singular_vectors
res$right_singular_vectors

# Share of the total (scaled) variance captured by those two components
all_values <- fs_svd(m, scale_input = TRUE)$singular_values
sum(res$singular_values^2) / sum(all_values^2)

# Untransformed, the full decomposition reconstructs the input exactly
full <- fs_svd(m, scale_input = FALSE)
recon <- full$left_singular_vectors %*%
  (full$singular_values * t(full$right_singular_vectors))
all.equal(recon, m)
```

`fs_svm` *Train and evaluate an SVM, with optional SVM-RFE feature selection*

Description

Trains an SVM classifier or regressor using **caret** (with the **kernlab** engines), with options for dummy encoding of predictors, feature selection, class-imbalance handling, and hyperparameter tuning via cross-validation. Optional parallel training uses an explicit worker count.

Usage

```
fs_svm(
  data,
  target,
  task,
  train_ratio = 0.7,
  nfolds = 5,
  kernel = c("linear", "radial", "polynomial"),
  tune_grid = NULL,
  feature_select = FALSE,
  select_method = c("svm_rfe", "rf_rfe"),
  n_features = NULL,
  class_imbalance = FALSE,
  seed = NULL,
  verbose = FALSE,
  n_cores = 1L
)
```

Arguments

`data`	A data frame containing predictors and the target.
`target`	A string naming the target variable.
`task`	Either `"classification"` or `"regression"`. Required; there is no default, because guessing it from the target is exactly the mistake this argument exists to prevent.
`train_ratio`	Training set proportion, strictly between 0 and 1 (default `0.7`).
`nfolds`	Number of CV folds for hyperparameter tuning, a whole number greater than 1 (default `5`). Also the number of folds used by the SVM-RFE subset-size search (clamped there to at most the number of training rows); `select_method = "rf_rfe"` ignores it and uses 10 folds.
`kernel`	One of `"linear"` (default), `"radial"`, or `"polynomial"`.
`tune_grid`	Optional tuning grid data frame. If `NULL`, a default grid for the chosen kernel is used.
`feature_select`	Logical; if `TRUE`, run feature selection on the dummy-encoded training predictors (default `FALSE`).

`select_method` Which selector to run when `feature_select = TRUE`: `"svm_rfe"` (default, true SVM-RFE, linear kernel only) or `"rf_rfe"` (random-forest screening, any kernel). Ignored when `feature_select = FALSE`, and so is the linear-kernel requirement, which is only enforced when SVM-RFE will actually run. An unrecognized value is always an error.

`n_features` Optional whole number of features to keep, capped at the number of encoded predictors. For `"svm_rfe"` the top `n_features` ranked features are kept and the cross-validated size search is skipped; for `"rf_rfe"` it truncates the selection to its first `n_features` entries and sets how many features the random-forest fallback keeps. Ignored when `feature_select = FALSE`. Default `NULL` (the size is chosen automatically).

`class_imbalance` Logical; if `TRUE` and the task is classification, up-samples classes within CV resampling (default `FALSE`).

`seed` Optional seed, applied locally for the duration of the call and restored afterwards; also used to set reproducible RNG streams on parallel workers when `n_cores > 1`. Default `NULL` (never seeds by default).

`verbose` Logical; if `TRUE`, report progress (including each SVM-RFE elimination step). Default `FALSE`.

`n_cores` Number of parallel workers (default 1, sequential). Requests are capped at the detected core count; when greater than 1 a cluster is created for the duration of the call and stopped on exit.

Details

This is the wrapper to reach for when the selector and the final model should belong to the same family: SVM-RFE ranks features by the weights of a linear SVM rather than by an external proxy criterion, and the returned object carries the tuned model and its held-out performance alongside the chosen features. That comes at a price – a full SVM fit at every elimination step, plus a cross-validated size search – so on wide data screen first with a filter such as `fs_supervised`.

- Feature selection (`feature_select = TRUE`) defaults to **SVM-RFE** (Guyon, Weston, Barnhill and Vapnik, 2002). A linear SVM is fitted on the centered and scaled encoded training matrix, the features are ranked by the squared primal weight `w^2` (recovered from the fit's support vectors and coefficients, summed over the pairwise problems of a multi-class fit), the lowest-ranked feature is dropped (the lowest 10\ SVM is *refitted* on the reduced set until one feature is left. Reversing the elimination order gives the ranking, so rank 1 is the feature eliminated last. SVM-RFE requires `kernel = "linear"`, because the ranking criterion is the primal weight vector, which only exists for a linear kernel; combining it with another kernel is an error that points at `select_method = "rf_rfe"`. The elimination and size-search fits use a fixed cost of `C = 1` and are separate from the final model, which is tuned over `tune_grid`.
- How many features SVM-RFE keeps: with `n_features` supplied, exactly that many (the top of the ranking). Otherwise a short ladder of candidate sizes – the powers of two up to the number of features, plus the full size, trimmed to at most six entries but always including 1 and the full size – is scored by `nfolds`-fold cross-validation with the same linear SVM, on folds shared by every candidate size. The winner is the size with the highest mean accuracy (classification) or the lowest mean RMSE (regression), ties going to the smaller size; if no size could be scored, all features are kept.

- `select_method = "rf_rfe"` keeps the older random-forest screening (`caret::rfFuncs`) and works with every kernel. It runs its own 10-fold cross-validation over subset sizes 1 to p, independent of `nfolds`. If `rfe()` fails or selects nothing, a plain random forest is fitted and its most important features are used instead, with a warning – and because the fallback keeps exactly `n_features` of them, a failure with `n_features = NULL` leaves a single feature.
- Selection always runs on the training split only, so the test set never informs which features survive.
- Class-imbalance handling (`class_imbalance = TRUE`, classification only) up-samples *within* each cross-validation resample via `caret::trainControl(sampling = "up")`, so no resampled rows leak across folds.
- After the train/test split, factor (and character) predictor levels in the test set are aligned to the training levels; test rows containing levels unseen in training are dropped with a warning.
- A numeric target with `task = "classification"` is coerced to a factor (with a message) only when it has at most 10 unique values; more than 10 unique values is an error suggesting `task = "regression"`.

Suggested packages required at runtime: **caret** and **kernlab** always, **e1071** for classification metrics, **randomForest** when `feature_select = TRUE` and `select_method = "rf_rfe"`, and **doParallel/foreach** when `n_cores > 1`.

Value

An object of class `fs_result` with:

selected Character vector of selected encoded feature names. When `feature_select = FALSE` this is every encoded predictor. With `"svm_rfe"` it is the surviving subset, ordered from most to least important; with `"rf_rfe"` it is the subset in the order `caret::rfe()` reports it.

scores Named numeric vector covering every encoded predictor, not just the survivors: the SVM-RFE criterion (the squared primal weights `w^2` of the first, full-feature fit) or, for `select_method = "rf_rfe"`, the mean random-forest importance recorded across resamples (`NA` for predictors `rfe()` never scored, or the mean decrease in node impurity when the fallback ran). `NULL` when `feature_select = FALSE`, because no selector produced comparable scores.

method `"svm_"` followed by the kernel, for example `"svm_linear"`. The selector that ran, if any, is reported in `details$selection$method`.

task `"classification"` or `"regression"`.

model The fitted `caret::train` object.

details A list with `test_set` (the test split with its coerced target and aligned factor levels), `predictions` (test-set predictions), `performance` (a `caret::confusionMatrix` for classification, or a named RMSE/Rsquared/MAE vector for regression), `selection` (`NULL` when no selection ran, otherwise a list with `method`, `ranking` from most to least important, `scores`, and the size search's `sizes`, `size_scores` and `size_metric` – those last three being `NULL`, `NULL` and `NA` whenever no size search ran, which is always the case for `"rf_rfe"` and for `"svm_rfe"` with an explicit `n_features`), `encoder` (the fitted `caret::dummyVars` object) and `n_features` (the number of encoded predictors considered, counted before any were dropped).

call The matched call.

Examples

```
if (requireNamespace("caret", quietly = TRUE) &&
    requireNamespace("kernlab", quietly = TRUE) &&
    requireNamespace("e1071", quietly = TRUE)) {
  res <- fs_svm(
    data = iris,
    target = "Species",
    task = "classification",
    nfolds = 3,
    kernel = "linear",
    tune_grid = data.frame(C = 1),
    seed = 42
  )
  res$details$performance

  # SVM-RFE keeps the two most useful measurements
  sel <- fs_svm(
    data = iris,
    target = "Species",
    task = "classification",
    nfolds = 3,
    kernel = "linear",
    tune_grid = data.frame(C = 1),
    feature_select = TRUE,
    select_method = "svm_rfe",
    n_features = 2,
    seed = 42
  )
  sel$selected
  sel$scores
}
```

`fs_unsupervised` *Unsupervised Filter-Based Feature Selection*

Description

Performs unsupervised, univariate, filter-based feature selection by scoring each feature using a chosen unsupervised criterion and selecting or dropping features based on a threshold on that score.

Usage

```
fs_unsupervised(
  data,
  method = c("variance", "mad", "iqr", "range", "missing_prop", "n_unique"),
  threshold = 0,
  direction = c("above", "below"),
  action = c("keep", "remove"),
```

```
  include_equal = FALSE,
  na_rm = TRUE,
 output = c("result", "matrix", "dt", "data.frame", "mask", "indices", "names", "list"),
  verbose = FALSE
)
```

Arguments

`data`	A data.frame, data.table, or matrix; all columns must be numeric. Every column is a candidate feature. The input is copied, never modified in place.
`method`	One of `"variance"` (default), `"mad"`, `"iqr"`, `"range"`, `"missing_prop"`, `"n_unique"`.
`threshold`	Non-negative, finite numeric scalar threshold applied to the feature scores. Default 0. The scales differ by method (a variance is in squared units, `"missing_prop"` is in `[0, 1]`, `"n_unique"` is a count), so a threshold is not portable between methods.
`direction`	One of `"above"` (default), `"below"`; compares scores to `threshold`.
`action`	One of `"keep"` (default), `"remove"`; determines whether features meeting the condition are retained or dropped.
`include_equal`	Logical; if TRUE, comparisons are inclusive (greater/less than or equal) instead of strict. Default FALSE.
`na_rm`	Logical; if TRUE (the default), remove NAs when computing scores. Has no effect on `"missing_prop"` and `"n_unique"`, which always account for NAs the same way.
`output`	One of `"result"` (default), `"matrix"`, `"dt"`, `"data.frame"`, `"mask"`, `"indices"`, `"names"`, `"list"`.

- `"result"` (default): an `fs_result` object (see Value).
- `"matrix"`: numeric matrix of the selected features.
- `"dt"`: data.table of the selected features.
- `"data.frame"`: data.frame of the selected features.
- `"mask"`: logical vector of length `ncol(data)`, named after the columns of `data`; TRUE marks a selected column.
- `"indices"`: integer vector of the selected column indices, named after the selected columns.
- `"names"`: character vector of the selected column names.
- `"list"`: list with components `filtered` (matrix), `mask`, `indices`, `names`, `scores` (all as above), and `meta`, a list recording `method`, `threshold`, `direction`, `action`, `include_equal`, `na_rm`, `n_input_cols`, and `n_kept_cols`.

`verbose`	Logical; emit progress messages. Default FALSE.

Details

No target is involved, so this is the right tool for the first cleaning pass – dropping constant or near-constant columns, columns that are mostly missing, or columns with too few distinct values – and it is safe to run before a train/test split, since nothing about the outcome informs it. It says nothing about whether a feature is *useful*: a high-variance column can be pure noise, and a low-variance

one can be the best predictor you have. Use `fs_supervised()` or a model-based method for that judgment.

Supported methods:

- `"variance"`: Sample variance (denominator n - 1).
- `"mad"`: Median absolute deviation, computed with `stats::mad()`'s default consistency constant 1.4826 (normal-consistent). Scores are therefore on the standard-deviation (sigma) estimate scale, not the raw median-absolute-deviation scale, and thresholds should be chosen accordingly.
- `"iqr"`: Interquartile range.
- `"range"`: Max - Min.
- `"missing_prop"`: Proportion of missing values, in `[0, 1]`. Note that with the default `threshold = 0`, `direction = "above"`, and `action = "keep"`, this KEEPS the features with the most missing values, which is almost never the intent; a warning suggests `action = "remove"` or `direction = "below"`. The warning fires only when all three of `threshold`, `direction`, and `action` are left at their defaults, so passing any one of them explicitly opts out of the advisory.
- `"n_unique"`: Number of unique non-NA values.

`na_rm` affects only the methods that summarize the observed values (`"variance"`, `"mad"`, `"iqr"`, `"range"`); `"missing_prop"` and `"n_unique"` always look at the whole column and treat NA as NA.

Features whose score is undefined (`NA`) are never selected, under both `action = "keep"` and `action = "remove"`; a warning reports how many such features were excluded.

Columns are subset by integer index, never by name, so duplicated column names cannot select the wrong columns.

Value

With the default `output = "result"`, an object of class `fs_result` with elements:

- `selected`: character vector of selected feature names.
- `scores`: named numeric vector of per-feature scores, in column order, `NA` where the score is undefined.
- `method`: `"unsupervised_"` followed by the scoring method, for example `"unsupervised_variance"`.
- `task`: `NA_character_` (unsupervised selection has no task).
- `model`: `NULL`.
- `details`: a list holding, in this order, `mask` (the logical keep-mask over the candidate features), `indices` (the selected column indices, named after the selected columns), `filtered` (the selected columns as a data.table), `threshold`, `direction`, `action`, and `n_features` (the number of candidate features, that is `ncol(data)`).
- `call`: the matched call.

Any other `output` returns that shape instead, exactly as documented above. When no feature meets the selection criteria, a warning is issued and the tabular shapes come back empty: `"matrix"` and `"data.frame"` have zero columns and keep the input row count, while the `"dt"` shape (and `details$filtered`) is only guaranteed to have zero columns – data.table represents a zero-column table as having zero rows, so its row count is not preserved.

Examples

```
df <- data.frame(
  spread = c(1, 2, 3, 4, 100),
  flat   = c(2, 2, 2, 2, 2),
  gappy  = c(1, NA, 3, NA, 5)
)

# Default: an fs_result
res <- fs_unsupervised(df, method = "variance", threshold = 0.5)
res$selected
res$scores
res$details$filtered

# The classic shapes are still available
fs_unsupervised(df, method = "variance", threshold = 0.5,
                output = "matrix")

# Remove features with missing proportion >= 0.2
fs_unsupervised(df, method = "missing_prop", threshold = 0.2,
                direction = "above", action = "remove",
                include_equal = TRUE, output = "names")
```

---

print.fs_result *Print a featR result*

Description

Prints a compact summary of what a selection run kept.

Usage

```
## S3 method for class 'fs_result'
print(x, n = 10L, ...)
```

Arguments

x	An fs_result object.
n	Maximum number of features to list (default 10).
...	Unused, for consistency with the generic.

Details

The output is at most five lines: a <fs_result> header naming the method, followed by the task in parentheses when one is known; a count, either "Selected k of N features" or, when the number of candidates cannot be recovered, "Selected k features"; the selected names, truncated to the first n with a "... (m more)" tail; a "Model:" line giving the class of x$model when the method fitted one; and a "Details:" line naming the elements of x$details. Nothing is printed for the names when the selection is empty.

Value

x, invisibly. Called for its printed output.

Examples

```
fs_unsupervised(
  data.frame(a = c(1, 5, 2, 8), b = c(1, 1, 1, 1)),
  method = "variance", threshold = 0.5
)
```

`selected` *Extract the selected features from a featR result*

Description

The generic accessor for the one thing every featR method produces: the names of the features it kept. Equivalent to `x$selected`, but stable against future changes in the internal layout, and safe to call on the result of any `fs_*()` selection function.

Usage

```
selected(x, ...)

## S3 method for class 'fs_result'
selected(x, ...)
```

Arguments

x	An `fs_result` object.
...	Unused, for future methods.

Value

A character vector of selected feature names, possibly empty when nothing met the selection criteria.

Examples

```
res <- fs_unsupervised(
  data.frame(a = c(1, 5, 2, 8), b = c(1, 1, 1, 1)),
  method = "variance", threshold = 0.5
)
selected(res)

# Empty selections are returned as character(0), not NULL
none <- suppressWarnings(
  fs_unsupervised(
    data.frame(a = c(1, 5, 2, 8), b = c(1, 1, 1, 1)),
```

```
    method = "variance", threshold = 1e6
  )
)
selected(none)
```

`summary.fs_result` *Summarize a featR result*

Description

Prints the result, the call that produced it, and a ranked score table.

Usage

```
## S3 method for class 'fs_result'
summary(object, n = 20L, ...)
```

Arguments

`object`	An `fs_result` object.
`n`	Maximum number of scored features to show (default 20).
`...`	Unused, for consistency with the generic.

Details

Everything `print()` shows, then the recorded call, then – for methods that produce per-feature scores – a table of every scored feature ranked by decreasing absolute score, with columns `feature`, `score` (four significant digits) and `selected` (`*` marks the features that were kept). The table is truncated to `n` rows with a "... (m more)" tail.

The ranking direction follows the method. For most methods a larger score means a stronger feature, and rows are ordered by decreasing absolute score so that signed scores such as regression coefficients sort sensibly. For methods that score by p-value, where smaller is better, rows are ordered ascending instead, so the most significant feature is listed first.

Value

`object`, invisibly. Called for its printed output.

Examples

```
res <- fs_unsupervised(
  data.frame(a = c(1, 5, 2, 8), b = c(1, 1, 1, 1)),
  method = "variance", threshold = 0.5
)
summary(res)
```

Index