Package 'featR'

November 9, 2026

Title A Unified Toolkit for Feature Selection

Version 0.2.0

Description Filter, wrapper, and embedded feature-selection methods behind a consistent set of functions that share one calling convention and one return type: univariate threshold filters, correlation and chi-squared filters, information gain, the least absolute shrinkage and selection operator (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, methods, parallel, stats, utils, withr (>= 2.4.0)

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,
https://elkronos.github.io/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 2026-11-09 00:30:02 UTC

Contents

fs_bayes 2
fs_boruta 7
fs_chi 9
fs_correlation 12
fs_elastic 15
fs_infogain 19
fs_lasso 22
fs_mars 26
fs_pca 30
fs_randomforest 32
fs_recursivefeature 37
fs_stepwise 42
fs_supervised 44
fs_svd 47
fs_svm 50
fs_unsupervised 55
print.fs_result 58
selected 59
summary.fs_result 60

Index **62**

`fs_bayes` *Bayesian predictor-subset selection with brms and LOO*

Description

Fits one brms model for each candidate combination of predictors, estimates each model's out-of-sample fit with PSIS-LOO, and picks a combination from the `loo::loo_compare()` table. The selected `brmsfit` is returned as the `model` of 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 kept, and rows with a missing value in any of them are dropped before any model is fitted. Infinite values in those columns are an error. None of those columns may be named `fitted_values`, `residuals`, `abs_residuals`, or `squared_residuals`, because those columns are appended to `details$data`.
target	Name of the target (response) column. It must exist in `data` and suit `brm_family`: numeric for `stats::gaussian()`, 0/1 or logical for `brms::bernoulli()`.
predictors	Character vector naming the candidate predictor columns: at least one, without duplicates, all present in `data`.
date_col	Name of a date column, or `NULL`. When given, an `iso_week_id` feature (ISO year * 100 + ISO week) is added to the predictors, so no target, predictor, or date column may itself be named `iso_week_id`. `iso_week_id` enters the models as a continuous covariate, a rough encoding of seasonality; a categorical or cyclic encoding may come in a future version. The date column itself is never used as a predictor. Default `NULL` (no date feature).
brm_family	A model family accepted by `brms::brm()`, for example `stats::gaussian()` (the default) or `brms::bernoulli()`.
prior	A brms prior specification. Default `NULL` (the brms default priors).
brm_args	List of extra arguments for `brms::brm()`, such as `iter`, `warmup`, or `seed`. They override featR's defaults of `iter = 2000`, `adapt_delta = 0.99`, `max_treedepth = 15`, and `refresh = 0`. `chains` and `cores` are respected when combinations are evaluated sequentially (the defaults are 4 chains on 1 core, and `cores` is capped at the detected core count), but both are forced to 1 when `parallel_combinations = TRUE`. The PSIS-LOO step uses the same core count, never `getOption("mc.cores")`. 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. See Details.

`max_comb_size` Largest number of predictors in a combination, a whole number >= 1, or `NULL`. Values above the number of candidate predictors are capped. Default `NULL` (all sizes).

`sample_combinations` Number of combinations to draw at random instead of evaluating all of them, a whole number >= 1, or `NULL`. Ignored when fewer combinations exist. Pass `seed` to make the draw reproducible. Default `NULL` (evaluate every combination).

`parallel_combinations` Logical. If `TRUE`, evaluate combinations in parallel with `parallel::mclapply()`. Each model is then limited to a single MCMC chain, so cross-chain convergence diagnostics such as R-hat are unavailable; a warning says so. On Windows, or when `n_cores` resolves to 1, evaluation falls back to sequential with a message. Default `FALSE`.

`seed` Optional whole number within the R integer range; fractional values are an error. When supplied, the whole call runs on a local RNG stream and your RNG state is restored on exit. The seed drives the random draw of combinations (see `sample_combinations`) and, unless `brm_args$seed` is given, one Stan seed per combination, so the MCMC fits are reproducible too, including with `parallel_combinations = TRUE`. An explicit `brm_args$seed` is used for every model instead. Default `NULL`: `fs_bayes()` never seeds the RNG unless you ask, and brms and rstan then draw their seeds from the global stream.

`verbose` Logical. If `TRUE`, print progress and every per-model warning, and show a progress bar for sequential evaluation when the suggested package 'pbapply' is installed. Default `FALSE`.

`n_cores` Number of workers when `parallel_combinations = TRUE`, a whole number >= 1; ignored otherwise. Requests are capped at the detected core count. Default 1 (sequential).

Details

PSIS-LOO (Pareto-smoothed importance sampling leave-one-out) approximates leave-one-out cross-validation from a single posterior fit. It yields `elpd_loo`, an estimate of the expected log predictive density for new data; higher is better. Use `fs_bayes()` when you want the predictor subset chosen by that criterion under a fully Bayesian model and can afford one model per subset.

Cost of the search:

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: 4 predictors mean 15 models, 10 predictors mean 1023. Use `max_comb_size` or `sample_combinations` to bound the search, and `parallel_combinations` to spread it over cores.

How a model is chosen:

With `rule = "1se"` (the default), the chosen model is the one with the fewest predictors, ties broken by the higher elpd, among those whose elpd difference from the best model is at most one

standard error of that difference (the `se_diff` column of the comparison table) in absolute value. With `rule = "best"`, the raw elpd maximum wins; that was 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.

Failures and diagnostics:

Combinations whose model fails to fit are left out of the selection, counted in `details$n_failed_fits`, and reported in a warning that quotes the first brms error. Fitted models for which `loo::loo()` fails or gives no finite elpd are left out in the same way, counted in `details$n_loo_failed`, and reported in a warning. If `loo::loo_compare()` itself fails, a warning says that the raw elpd maximum is used and `rule = "1se"` is not applied. If no fit yields a finite `elpd_loo` at all, the first successfully fitted model is returned with a warning. That model is an arbitrary fallback that no criterion chose.

Sampler warnings (divergent transitions, R-hat, effective sample size) and loo's Pareto-k warnings are muffled while the models are fitted, so a search over many subsets does not flood the console. They are counted instead: after the search, one warning reports how many fits raised sampler warnings and another how many have observations with Pareto k > 0.7 (unreliable PSIS-LOO estimates), and each says whether the selected model is among them. `verbose = TRUE` shows every individual warning.

What is optimistic:

`details$mae` and `details$rmse` are in-sample errors, computed on the same rows used to fit and to select, so they are optimistic.

`details$best_elpd` and the `loo_comparison` table are optimistic too. PSIS-LOO is roughly unbiased for one fixed model, but the selected model's `elpd_loo` is the maximum (or, under `rule = "1se"`, a near-maximum) over `details$n_compared` candidates. It therefore overstates the chosen model's out-of-sample performance, and more so the larger the search (selection-induced bias; Piironen and Vehtari, 2017). An honest estimate needs the whole search repeated inside an outer cross-validation, or held-out data.

Anything read off the returned `brmsfit`, such as posterior intervals or effect sizes, is post-selection inference: the model was chosen using the same data it is reported on. The Validation and leakage article covers both points.

Value

An object of class `fs_result` with:

selected Character vector of the predictors in the chosen model.

scores `NULL`, because no per-feature score is comparable across models with different predictors. `details$n_features` records how many candidate predictors were offered.

method `"bayes"`.

task `"classification"` when `brm_family` is a binary, categorical, or ordinal family (`bernoulli`, `binomial`, `beta_binomial`, `categorical`, `multinomial`, `cumulative`, `sratio`, `cratio`, `acat`), otherwise `"regression"`.

model The selected `brmsfit`.

details A list with:

- `data`: the complete-case modeling data.table (`target`, `predictors`, and `date_col` and `iso_week_id` where applicable), with `fitted_values`, `residuals`, `abs_residuals`, and `squared_residuals` columns appended. Families without a single fitted value per row (categorical, multinomial, and multivariate responses) get no appended columns.
- `mae`, `rmse`: in-sample errors on the rows used to fit and select, so optimistic; `NA` for the families without a single fitted value per row.
- `formula`: the selected model's formula, as a string.
- `best_elpd`: the selected model's `elpd_loo`, possibly `NA`; optimistic because it was selected (see Details).
- `loo_comparison`: the `loo::loo_compare()` table (a matrix or a 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.
- `n_loo_failed`: how many fitted models were left out because `loo()` failed.
- `n_compared`: how many models with a finite `elpd_loo` competed in the selection.
- `n_features`: the number of candidate predictors, including `iso_week_id` when `date_col` is given.

call The matched call.

References

Piironen, J. and Vehtari, A. (2017). Comparison of Bayesian predictive methods for model selection. *Statistics and Computing*, 27(3), 711-735.

See Also

The Wrapper methods guide, which runs this function on a worked example.

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` *All-relevant feature selection with Boruta*

Description

Runs the Boruta algorithm to find every feature that carries information about the target, then optionally prunes correlated features and caps how many are returned. Boruta compares each feature's random-forest importance with that of "shadow" copies of the predictors whose values have been shuffled, and confirms the features that beat the best shadow more often than chance allows.

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 become factors, Date and POSIXt columns become numeric, and any other type is an error. Neither the target nor the predictors may contain missing values, because Boruta's random forest cannot fit them. Column names must be unique and non-empty, and `data` must hold at least one predictor besides the target.
`target`	Name of the target column in `data`. A factor target means classification and a numeric target means regression; any other type is an error, as is a target with `NA` or infinite values. Unused factor levels are dropped, and a factor target with fewer than two observed classes is an error.

`maxRuns` Maximum number of Boruta iterations: a whole number of at least 11, the minimum Boruta itself accepts. Default 250.

`cutoff_features` Optional whole number capping how many features are returned. The top features by median Boruta importance are kept, after any correlation pruning. Default `NULL` (no cap).

`cutoff_cor` Correlation cutoff between 0 and 1 for pruning redundant features from the selected set (see Details). Within a correlated group the feature with the highest median Boruta importance is kept. Only numeric predictors are compared; factor predictors are never pruned. Set `NULL` to skip pruning. Default 0.7.

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

`seed` Optional single whole number within the range of an R integer; fractional or out-of-range values are an error. The seed is applied locally and the previous RNG state is restored on exit. Default `NULL`: the RNG is never seeded unless you ask.

`verbose` Logical. If `TRUE`, report progress, which sets Boruta's `doTrace = 1` (decisions are reported as they are made), and name any features dropped by correlation pruning. `FALSE` (default) sets `doTrace = 0`.

Details

Boruta is an all-relevant selector: it looks for every feature that carries information about the target, rather than the smallest subset that predicts best. Redundant but informative features are therefore all confirmed, which helps you understand a dataset more than it helps you build a compact model. The search costs up to `maxRuns` random-forest fits, and its outcome varies from run to run unless `seed` is supplied. The forests are grown on a single thread (`num.threads = 1` is passed to Boruta's importance source), in line with featR's sequential default.

Features still undecided when `maxRuns` is reached are Tentative. `resolve_tentative = TRUE` settles them with the `Boruta::TentativeRoughFix()` heuristic, not with further evidence. The heuristic confirms a tentative feature whenever its median importance beats the median importance of the best shadow, a weaker test than the one that confirms features during the run, so it trades false negatives for false positives. In a simulation of 20 data sets with one real predictor and three pure-noise predictors (150 rows, 100 runs), Boruta alone confirmed 5 percent of the noise predictors and 8.3 percent after the rough fix. When false positives are costly, use `resolve_tentative = FALSE` and inspect `details$decisions`, or raise `maxRuns` so fewer features stay tentative.

Two featR steps then run on Boruta's selection, in this order, and both rank features by median Boruta importance:

1. Correlation pruning (`cutoff_cor`, on by default at 0.7). Starting from the most important selected feature, every remaining numeric feature whose absolute Pearson correlation with it exceeds `cutoff_cor` is dropped; the walk then moves to the next most important survivor. Factor predictors are never pruned. A constant column has an undefined correlation, which is treated as 0 with a warning.
2. The cap (`cutoff_features`), which keeps the top features by median importance.

Value

An object of class `fs_result` with:

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

scores Named numeric vector of median Boruta importance for every candidate feature (`NA` for a feature 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 some feature was still tentative.

details A list with `boruta_obj` (the same Boruta object as `model`), `decisions` (the per-feature Confirmed, Tentative, or Rejected factor, 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.

See Also

The Wrapper methods guide, which runs this function on a worked example.

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-squared tests for categorical features*

Description

Tests each categorical feature for association with a categorical target using the chi-squared test of independence, adjusts the p-values for multiple testing, and keeps the features whose adjusted p-value is below `sig_level`.

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 holding the features and the target. Column names must be unique, non-empty, and not NA. The input object is never modified.
target	Name of the target column. It is converted to a factor if necessary and must have at least 2 levels among its non-missing values.
sig_level	Significance threshold for the adjusted p-values, strictly between 0 and 1. Default 0.05.
continuity_correction	NULL, TRUE, or FALSE. Controls Yates' correction for 2x2 tables tested asymptotically. NULL (the default) and TRUE both apply it; FALSE turns it off. Tables larger than 2x2 and simulated p-values are never corrected.
p_adjust_method	Multiple-testing correction, one of stats::p.adjust.methods, matched case-insensitively. Default "bonferroni"; "none" turns correction off. An invalid name is an error before any feature is tested. Only tested features count toward the correction: a skipped feature keeps an NA adjusted p-value and does not raise the multiplier for the others. With k tested and s skipped features, "bonferroni" multiplies by k, not k + s.
simulation_B	Number of Monte Carlo replicates for the simulated p-value used when any expected cell count is below 5. A whole number of at least 100; default 2000. A simulated p-value is never below 1 / (simulation_B + 1), so after adjustment it may be unable to reach sig_level at all. With Bonferroni this happens once the number of tested features reaches sig_level * (simulation_B + 1), which is 101 at the defaults. A warning then names the affected features and the simulation_B you would need.
seed	Optional integer. When supplied, the RNG is seeded for this call and the previous RNG state is restored on exit, which makes simulated p-values reproducible. With parallel = TRUE the simulations draw from furrr's parallel-safe streams instead (furrr_options(seed = TRUE)), so a given seed gives reproducible parallel results that differ from the sequential ones. Default NULL (featR does not touch the RNG).

verbose — If TRUE, report target conversion, skipped features, and the worker count. Default FALSE.

parallel — If TRUE, test features in parallel using the suggested furrr and future packages. Default FALSE.

n_cores — Number of workers when parallel = TRUE, a whole number of at least 1, capped at the detected core count. Default 2. If the current future plan is sequential (the future default), featR sets a future::multisession plan with this many workers for the duration of the call and restores the previous plan on exit, even if worker start-up fails. If you have already set a non-sequential plan, for example future::plan(future::multisession, workers = 4), featR uses it as is, leaves its workers running, and ignores n_cores.

Details

For each feature, the test asks whether the feature and the target are distributed jointly the way independence would predict. Features are tested one at a time, so the result describes marginal association only. A feature that matters only in combination with another is missed, and two features carrying the same information are both reported as significant.

A p-value measures evidence against independence and says nothing about the size of the effect. With enough rows a negligible association still clears any sig_level.

Value

An fs_result object with:

selected Features with an adjusted p-value below sig_level.

scores Named numeric vector of adjusted p-values, one per candidate factor feature, NA for skipped features. Smaller means stronger evidence of association, the opposite of most featR scores.

method "chi".

task "classification".

model NULL; no model is fitted.

details A list with results, sig_level, p_adjust_method (as supplied), and n_features (the number of candidate factor features, skipped ones included). results is a data frame with one row per candidate feature, ordered by adjusted then raw p-value with skipped features last. Its columns are:

- feature: the column name.
- n: for tested features, the number of complete feature-target pairs; for skipped features, the feature's count of non-missing values.
- df: degrees of freedom; NA for simulated and skipped tests.
- p_value, adj_p_value: raw and adjusted p-values.
- significant: TRUE when adj_p_value < sig_level, so FALSE for skipped features.
- method: "asymptotic" or "simulation"; NA when skipped.
- correction_applied: whether Yates' correction was actually applied. A 2x2 table whose counts equal their expected values gets no correction even when one was requested. NA when skipped.
- min_expected: the smallest expected cell count; NA when skipped.

call The matched call.

Which columns are tested

Character columns are converted to factors, and then every factor column other than the target is tested. Numeric, logical, and date columns are ignored, so convert or discretize them first if you want them tested. If no factor features remain, the result has an empty selection and no error is raised. A numeric target is converted with `as.factor()`, so each distinct value becomes its own class.

Missing values and sparse tables

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 the feature or the target then has fewer than two levels, the feature is skipped: it gets an `NA` p-value and is not selected.

A feature is tested with the asymptotic chi-squared statistic when every expected cell count is at least 5. Otherwise the p-value comes from a Monte Carlo simulation with `simulation_B` replicates. A simulated p-value changes from run to run unless `seed` is set, and its `df` is `NA`. Yates' continuity correction applies only to 2x2 tables on the asymptotic path, whatever `continuity_correction` says.

See Also

The Filter methods guide, which runs this function on a worked example.

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` *Find and prune highly correlated variables*

Description

Finds pairs of variables whose absolute correlation exceeds `threshold` and, by default, drops variables until no such pair remains, keeping one representative of each correlated group.

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. Column names must be unique, non-empty, and not NA; duplicated names are an error because a correlation matrix with duplicated names is ambiguous. The allowed column types depend on method: • "pearson", "spearman", "kendall": all numeric. • "polychoric": all ordered factors. • "pointbiserial": numeric, or any type with exactly 2 distinct non-missing values. A numeric column with more than 2 distinct values counts as continuous, and a column with exactly 2 counts as dichotomous. At least one of each is needed, or every correlation is NA (with a warning).
threshold	Absolute-correlation cutoff between 0 and 1. Pairs with an absolute correlation strictly greater than threshold are flagged. Required; there is no default.
method	One of "pearson" (default), "spearman", "kendall", "polychoric" (needs the suggested polycor package), or "pointbiserial". The point-biserial correlation is the Pearson correlation between a continuous variable and a 0/1 indicator of a dichotomous one, where 1 marks the second sorted value (for a factor, the second level). Only continuous-dichotomous pairs have one, so under this method every continuous-continuous and dichotomous-dichotomous cell stays NA and those pairs are never flagged.
prune	If TRUE (default), selected is the reduced set left after pruning (see "How pruning works"). If FALSE, nothing is dropped and selected is every variable in at least one flagged pair: the redundant set, not the set to keep.
na.rm	How to handle missing values. If TRUE, they are removed pairwise for "pearson", "spearman", and "kendall"; within each pair for "pointbiserial"; and casewise (rows complete across all columns) for "polychoric". If FALSE (default), a missing value makes the affected correlations NA, except under "polychoric", which stops with an error instead of silently dropping rows.

sample_frac	Fraction of rows, in (0, 1], to sample without replacement before computing correlations. The sample size is rounded up and is never below one row. Default 1 (no sampling).
output_format	Shape of details$corr_matrix: "matrix" (default) or "data.frame" (long form).
diag_value	Single number, or NA, written on the diagonal of details$corr_matrix. The diagonal never takes part in flagging, scoring, or pruning. Default 0.
seed	Optional integer seed for the row sampling. It is applied for this call only and the previous RNG state is restored afterward. It has no effect when sample_frac = 1, and parallel point-biserial computation uses no random numbers. Default NULL (featR does not touch the RNG).
verbose	If TRUE, report row sampling, the correlation method, and any variables dropped by pruning. Default FALSE.
parallel	If TRUE, compute point-biserial correlations in parallel with the suggested foreach and doParallel packages. Every other method computes the matrix in one call and ignores this. Default FALSE.
n_cores	Number of workers when parallel = TRUE and method = "pointbiserial", a whole number of at least 1, 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, never at an outcome, so when two variables are nearly interchangeable it cannot keep the one that predicts better. It keeps whichever is less correlated with the rest of the data. Use fs_boruta() when the choice within a correlated group should depend on importance for a target.

Value

An fs_result object with:

selected With prune = TRUE, every variable not in details$dropped; no two of them have a computable absolute correlation above threshold. With prune = FALSE, both members of every flagged pair.

scores Named numeric vector giving each variable's largest absolute correlation with any other variable. Larger means more redundant. NA when no correlation with that variable could be computed.

method paste0("correlation_", method), for example "correlation_pearson".

task NA; the filter uses no outcome.

model NULL; no model is fitted.

details A list with:

- corr_matrix: the correlation matrix with diag_value on the diagonal, or, when output_format = "data.frame", the same values in long form with columns Var1, Var2, Correlation.
- pairs: a data frame of the flagged pairs (Var1, Var2, Correlation), strongest absolute correlation first, whatever output_format is.
- dropped: variables removed by pruning, in the order they were dropped; empty when prune = FALSE.

- `redundant`: every variable in at least one flagged pair, which is `selected` as it would be under `prune = FALSE`.
- `n_features`: the number of variables considered.

`call` The matched call.

How pruning works

Pruning is greedy. While any pair of retained variables still has an absolute correlation above `threshold`, the strongest such pair is taken, and the member with the larger mean absolute correlation to the other retained variables is dropped (on an exact tie, the later column). Afterward no two retained variables have a computable correlation above `threshold`. The surviving set is not guaranteed to be the smallest one with that property, and it can depend on the order in which pairs are resolved.

This borrows the tie-break of `caret::findCorrelation()` but not its scan order: caret visits columns in order of mean correlation rather than pairs in order of strength, so the two can drop different variables. Variables that were never flagged are always kept.

Correlations that cannot be computed

A correlation that cannot be computed is `NA`. Such pairs are never flagged and never pruned, because an unknown correlation is no evidence of redundancy, and a warning reports how many there are. When no pair exceeds `threshold`, a message (not a warning) says so, whatever `verbose` is set to.

See Also

The Filter methods guide, which runs this function on a worked example.

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

# Both members of every flagged pair, with nothing dropped
fs_correlation(d, threshold = 0.9, prune = FALSE)$selected
```

`fs_elastic` *Elastic-net feature selection with tuned alpha and lambda*

Description

Tunes an elastic-net model over a grid of `alpha` and `lambda` values with `caret::train(method = "glmnet")` and selects the predictors whose coefficient is non-zero at the winning pair. Handles regression (numeric outcome) and classification (factor, character or logical outcome).

Usage

```
fs_elastic(
  data,
  target,
  alpha_seq = seq(0.1, 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, with unique, non-empty column names.
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, 1, by = 0.1), which leaves out 0 (ridge) because ridge cannot zero a coefficient.
lambda_seq	Numeric vector of non-negative lambda values to tune over, or NULL (default) to use glmnet's own path at each alpha (see "Tuning grid and metrics").
trControl	Optional caret::trainControl() object; a partial list or a string is an error. If NULL (default), 5-fold CV with an NA-safe summary function is used. When use_pca = TRUE, pcaComp = nPCs is added to its preProcOptions. Its allowParallel is always overwritten with n_cores > 1, so a registered foreach backend is used only when n_cores > 1.
metric	Optional character. Performance metric to optimize. If NULL (default), "RMSE" for regression and "Accuracy" for classification. If caret does not find the name in the summary output, it warns and falls back to its default, and details$metric_name reports the metric actually used.
use_pca	Logical; default FALSE. Whether to project the predictors onto principal components inside each resample.
nPCs	Integer >= 1. Number of principal components to retain when use_pca = TRUE; must be strictly less than min(nrow, ncol) of the predictor matrix. Default NULL, which is an error when use_pca = TRUE and ignored otherwise.
seed	Optional whole number; default NULL, which leaves the RNG alone. When given, the seed is set for this call only and the previous RNG state is restored on exit. The resampling folds are drawn in the main R process and the glmnet fits are deterministic, so a seeded n_cores > 1 run gives the same result as a seeded sequential one. When trControl$seeds is unset (NA, the trainControl() default), caret draws one seed per resample from this seed in the main process and sets it on each worker, so seeded runs that use worker-side randomness (for example sampling = "up") are reproducible with any n_cores. A user-supplied trControl$seeds takes precedence.

verbose — Logical; default FALSE. Print progress messages.

n_cores — Integer >= 1; default 1 (sequential, and no cluster is created). Number of workers for parallel training; values above the detected core count are capped. Values above 1 need the 'foreach' and 'doParallel' packages.

Details

The elastic net penalizes coefficients with a mix of an L1 (lasso) and an L2 (ridge) penalty: alpha sets the mix and lambda the overall strength. Both are chosen by resampling, and every column with a non-zero coefficient at the winning pair is reported. When the winning alpha is below 1, the ridge part spreads weight across correlated predictors, so a group of collinear columns tends to survive together instead of being reduced to one representative as fs_lasso() tends to do.

alpha = 0 is pure ridge, which never sets a coefficient to exactly zero and so cannot select. The default alpha_seq starts at 0.1 for that reason. If you include 0 and it wins, every column is selected and a warning says so.

The winning pair is the one with the best resampled metric (caret's default selectionFunction = "best"). With hundreds of alpha/lambda candidates that choice is noisy and leans towards too little penalty: on pure-noise data (100 rows, 50 predictors) it selected 8.5 predictors on average and at least one in every run. A trControl with selectionFunction = "oneSE" picks the most penalized pair within one standard error of the best and was much sparser in the same simulation (1.8 predictors on average).

For the same reason details$metric_value, the best value over the whole grid, is an optimistic estimate of performance. fs_elastic() holds out no test rows and reports no honest performance estimate; that needs an outer resampling loop around the whole call.

Value

An fs_result object with:

selected — Columns (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; columns 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 winning 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 caret optimized, which differs from metric when caret fell back to its default), metric_value (its resampled value for the winning pair, NA when the metric is absent from caret's results table; optimistic, see Details), use_pca, and n_features (number of columns the model saw, which is nPCs when use_pca = TRUE).

call — The matched call.

How the design matrix is built

Every column of `data` except `target` is a candidate predictor. The formula is built for you (non-syntactic names are backticked), and the predictors go through `stats::model.matrix()` with the intercept column removed, so a k-level factor or character column contributes k - 1 dummy columns. `selected`, `scores` and `details$coef` therefore name design-matrix columns rather than the original columns. Non-syntactic names are reported as you wrote them, without the backticks `model.matrix()` adds. Above 500,000 cells the matrix is converted to a sparse matrix unless `use_pca = TRUE`.

These inputs are errors:

- fewer than two design-matrix columns (`glmnet` cannot fit a single-column `x`), so a single numeric predictor fails;
- a factor or character predictor with fewer than two distinct values;
- a dummy name that collides with another column (factor `g` with level `b` next to a column called `gb`);
- Inf or -Inf in the target or any predictor;
- a predictor column that is constant, or constant up to floating-point round-off (range at most `64 * .Machine$double.eps` times its largest absolute value), checked the same way for dense and sparse input.

Missing values and the outcome type

Rows with a missing response or a missing value in any predictor are dropped before fitting; if no rows remain, that is an error. The task is then inferred from the remaining rows. A class that occurred only in dropped rows is removed with a message, and a class with fewer than two complete rows is an error. When rows are dropped, a user-supplied `trControl` carrying `index`, `indexOut` or `indexFinal` is an error, because those row numbers no longer match.

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

`scores` are absolute coefficients on the scale of the columns the model saw, not standardized ones. They rank predictors fairly only when those columns are on comparable scales; unlike `fs_lasso()`, this function does not rescale them for you. Coefficients are taken at exactly the winning lambda (the path is refitted if that lambda is not one of the final model's knots), not interpolated between knots.

Principal components inside resampling

With `use_pca = TRUE`, caret is asked for `preProcess = c("center", "scale", "pca")` with `pcaComp = nPCs` in `trControl$preProcOptions`. Centering, scaling and the component loadings are therefore refitted on the training part of every resample, and the held-out fold never contributes to them. The model is fitted on components, so `selected`, `scores` and `details$coef` are named `PC1`, `PC2`, and so on, rather than after the original columns. `nPCs` must be smaller than the number of predictors and than the number of rows each resample trains on.

Tuning grid and metrics

`lambda_seq = NULL` (the default) tunes over the lambda path that `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, as in caret's own default glmnet grid; which pair wins is still decided by resampling. With `use_pca = TRUE` the path is computed on the original predictors, so it only approximates the scale of the components; pass `lambda_seq` if you need control over it.

The resampled metrics in `model$results` and `details$metric_value` are caret's. For regression they are `RMSE`, `Rsquared` and `MAE`. `Rsquared` is the squared Pearson correlation between held-out predictions and observations (`caret::postResample()`), not `1 - SSE/SST`, so it is not comparable with an `R2` defined as `1 - SSE/SST` (for example `fs_randomforest()`'s). It lies in [0, 1], except that with the default `trControl = NULL` featR's summary function records a fold where it is undefined (constant predictions, typical at the large-lambda end of the path) as `-Inf`, so such grid rows show `-Inf` in `model$results`. For classification the metrics are `Accuracy` and Cohen's `Kappa`, which need the suggested package 'e1071'. featR checks for it before fitting when the default summary function is used; a custom `summaryFunction` in `trControl` is responsible for its own dependencies.

See Also

The Regularization and embedded importance guide, which runs this function on a worked example.

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$x2 + 0.1 * cos(seq_len(60) * 3)
  res <- fs_elastic(df, "y", alpha_seq = c(0.5, 1), seed = 1)
  print(selected(res))
  print(res$scores)
  print(res$details$best_alpha)
}
```

`fs_infogain` *Rank features by information gain*

Description

Scores every predictor by its information gain about the target, in bits, optionally divided by the predictor's own entropy to give a gain ratio. Accepts a single data frame or a list of data frames.

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, or a list of data frames that each contain target. A data.table is copied, never modified in place. Column names must be unique, non-empty, and not NA within each data frame, and list columns are not supported.
target	Name of the target column.
numeric_bins	Optional whole number of bins for numeric predictors and a numeric target, replacing the automatic count. Values below 2 are raised to 2. Default NULL (bins chosen per column).
normalize	"none" (default) for raw information gain in bits, or "gain_ratio" for gain divided by the predictor's split entropy. See "Cardinality bias and the gain ratio".
top_n	Optional whole number of at least 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, so a feature scoring 0 is selected if the ranking reaches it. When NULL (default), selected holds every feature with a score strictly greater than 0. A gain within floating-point rounding of zero, as an exactly independent predictor produces, is reported as exactly 0 and is not selected.
remove_na	If TRUE (default), rows with a missing target are removed first. See "Missing values" for why this rarely changes the result.
verbose	If TRUE, report how many features were scored, how many names occurred in more than one data frame, and how many were selected. Default FALSE.

Details

Information gain is the drop in the entropy of the target once you know the value of one predictor: how many bits of uncertainty about the target that predictor removes. It makes no distributional assumptions and handles mixed column types, which makes it a cheap way to rank candidates before fitting any model.

It is a univariate filter. Each predictor is scored on its own, so two copies of the same information both score highly, and a predictor that matters only in combination with another scores low. Treat the ranking as a shortlist rather than a final feature set.

The default selection rule keeps every feature with a score above 0, which screens out almost nothing. On a finite sample, a predictor unrelated to the target still has a positive gain in almost every data set, and more so with more bins or levels. In a 200-row example with 50 pure-noise columns, all 50 are selected. Use top_n, or compare against gains computed on a permuted target, to make a real cut.

Value

An `fs_result` object with:

`selected` Selected feature names in decreasing order of score. A feature with an `NA` score is never selected.

`scores` Named numeric vector of the score (gain or gain ratio) for every candidate feature, in decreasing order with `NA` last. For list input it covers the union of features across data frames, and a name found in several data frames keeps its highest score.

`method` `"infogain"`.

`task` `"classification"`, because the target is always treated as categorical.

`model` `NULL`; no model is fitted.

`details` A list with:

- `table`: the full scored table, one row per scored column of each input, with columns `Variable` and `InfoGain`, plus `SplitEntropy` and `GainRatio` when `normalize = "gain_ratio"`, plus `Origin` for list input.
- `normalize`: the value used.
- `numeric_bins`: the requested bin count as an integer, or `NULL` when automatic.
- `n_features`: the number of distinct scored features, equal to `length(scores)`.
- `collisions` (list input only): the feature names found in more than one data frame, with `Kept` marking the row whose score was used. It has zero rows when no name repeats.

`call` The matched call.

How columns are prepared

- Numeric predictors are cut into equal-width bins. The number of bins is the larger of the Freedman-Diaconis and Sturges counts, at least 2, unless `numeric_bins` sets it. Under the automatic rule, a column with zero range or zero interquartile range gets 2 bins. A column with a single distinct value is one level either way and scores 0.
- Character and logical predictors become factors.
- Date and date-time predictors are replaced by three numeric columns, `<name>_year`, `<name>_month`, and `<name>_day`, which are then binned like any numeric column. If one of those names already exists in `data`, the call stops.
- The target is always treated as categorical, so `task` is always `"classification"`. A numeric target is binned by the same rule, once, on all rows where it is observed, so every predictor is scored against the same bins. A date target is treated as one class per day.

Missing values

Missing values are handled for each predictor-target pair separately: a missing value in one predictor does not remove that row for the others. A predictor with no complete pairs scores `NA` and is never selected.

`remove_na` has a narrow effect. Rows with a missing target are excluded pair by pair anyway, so the only difference shows when the target is entirely missing: `remove_na = TRUE` stops with an error, and `remove_na = FALSE` returns an `NA` score for every predictor.

Cardinality bias and the gain ratio

Raw information gain favors predictors with many distinct levels. A near-unique identifier column splits the data into groups of about one row, which drives the conditional entropy H(Y | X) to zero and gives it the largest gain any predictor can reach, H(Y), while carrying no signal that generalizes. Raw gains of predictors with different numbers of levels are therefore not comparable.

`normalize = "gain_ratio"` applies Quinlan's correction: each predictor's gain is divided by its split entropy H(X), the entropy of its (binned) level distribution. H(X) grows with the number of levels (it is `log2(k)` for `k` equally frequent levels), so the division penalizes a predictor for how finely it splits the data. A constant predictor has H(X) = 0 and gets a gain ratio of 0 instead of a division by zero. Gain ratios are on a different scale from raw gains, so do not reuse a threshold chosen for gains in bits.

See Also

The Filter methods guide, which runs this function on a worked example.

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) regression with `glmnet::cv.glmnet()` and selects the predictors whose coefficient is non-zero at the cross-validated penalty. The outcome must be numeric; only the gaussian family is fitted.

Usage

```
fs_lasso(
  data,
  target,
  alpha = 1,
  nfolds = 5,
  standardize = TRUE,
  custom_folds = NULL,
  impute = c("none", "mean"),
  return_model = FALSE,
  lambda = c("min", "1se"),
  seed = NULL,
  verbose = FALSE,
  parallel = FALSE,
  n_cores = 2L
)
```

Arguments

`data`	A data.frame or data.table holding the target and the candidate predictors, with unique, non-empty column names. A numeric matrix with column names is also accepted. A non-numeric matrix is 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 and Inf.
`alpha`	Numeric in (0, 1]; default 1 (lasso). Values in (0, 1) give an elastic net. `alpha = 0` (pure ridge) is rejected because ridge 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 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 sets the scale `scores` is ranked on (see "How scores are scaled").
`custom_folds`	Optional integer vector of fold IDs, one per row of `data`, covering 1..`nfolds` with no empty fold; default `NULL`.
`impute`	How to handle missing predictor values: `"none"` (default) errors and names the affected columns; `"mean"` imputes column means computed on the full data and warns about the leak.
`return_model`	Logical; default `FALSE`. When `TRUE`, the fitted `cv.glmnet` object is kept in `model` and `keep = TRUE` is passed to `cv.glmnet()` so the prevalidated fits are retained.
`lambda`	Penalty to select at: `"min"` (default, `lambda.min`) or `"1se"` (`lambda.1se`, sparser). See Details.
`seed`	Optional whole number; default `NULL`, which leaves the RNG alone. When given, the seed is set for this call only and the previous RNG state is restored on exit.

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, which needs the 'doParallel' and 'foreach' packages. If n_cores resolves to one worker, the run stays sequential and says so when verbose = TRUE.
n_cores	Integer >= 1; default 2. Number of workers, used only when parallel = TRUE. Values above the detected core count are capped.

Details

The lasso adds an L1 penalty to least squares, which shrinks coefficients and sets some of them to exactly zero. fs_lasso() chooses the penalty strength (lambda) by nfolds-fold cross-validation and reports the predictors that keep a non-zero coefficient. With alpha < 1 the penalty becomes an elastic net, a mix of L1 and L2 penalties; fs_elastic() tunes alpha as well and also handles classification.

By default selection happens at lambda.min, the penalty with the lowest cross-validated error. lambda.min is tuned for prediction and tends to over-select, keeping noise variables alongside the true ones. lambda = "1se" selects at lambda.1se, the largest penalty whose CV error is within one standard error of the minimum, which gives a sparser, more conservative set. Both values are reported in details whichever one is used.

The 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.

fs_lasso() reports no performance estimate. The lambdas are tuned on all rows, so judging the selected set needs an outer resampling loop around the whole call.

Value

An fs_result object with:

selected	Design-matrix columns whose coefficient at the chosen lambda is non-zero, ordered by decreasing scores magnitude. Constant and numerically constant columns always have coefficient 0, so they are never selected.
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.
method	"lasso", whatever the value 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), lambda_1se (largest lambda within one standard error of the minimum), lambda_used ("lambda.min" or "lambda.1se", the one that selection and all coefficients are read at), coefficients (the same three-column table as scores, always on the raw coefficient scale) and n_features (number of design-matrix columns considered).
call	The matched call.

How the design matrix is built

Every column of `data` except `target` is a candidate predictor. The design matrix comes from `stats::model.frame(na.action = stats::na.pass)` followed by `stats::model.matrix()` without an intercept column, so rows with NAs reach the missing-value policy instead of being dropped silently. Factors and characters are expanded to dummy columns: the first factor (in column order) gets one dummy per level and each later factor gets one per level except the first. `glmnet` fits its own intercept, which is left out of the reported coefficients.

`selected`, `scores` and `details$coefficients` therefore name design-matrix columns, so a factor predictor appears as its dummy columns rather than under its own name. Non-syntactic column names are reported as you wrote them, without the backticks `model.matrix()` adds.

These inputs are errors:

- a design matrix with fewer than two columns (`glmnet` cannot fit a single-column `x`);
- a factor or character predictor with fewer than two observed levels;
- a dummy name that collides with another column or the target (factor `g` with level `b` next to a column `gb`), because `selected` and `scores` identify columns by name;
- Inf or -Inf in any predictor (NaN counts as a missing value);
- a column whose variance overflows double precision (values around 1e154 or larger); rescale it first.

How scores are scaled

With `standardize = TRUE` (the default), `scores` ranks features on the standardized coefficient scale: each coefficient is multiplied by the standard deviation of its design-matrix column, so the ranking does not depend on the units the predictors are measured in. `glmnet` standardizes internally for fitting but reports coefficients on the input scale, and those raw coefficients are kept in `details$coefficients`. With `standardize = FALSE`, `scores` holds the raw coefficients and the two tables are identical.

A raw coefficient depends on scale: a predictor measured in small units earns a large coefficient for the same effect. Compare `details$coefficients` (and `scores` when `standardize = FALSE`) across predictors only when the predictors share a scale.

Constant columns get a coefficient of exactly 0 and are never selected. The same holds for columns that are constant up to floating-point round-off: standardization would otherwise inflate that round-off into a full-scale predictor with a raw coefficient around 1e15, so such columns are held at 0 during the fit. A coefficient whose standardized value is at most `1e-10` times the largest standardized coefficient is floating-point residue and is set to exactly 0. Without that step, the redundant dummy of the first factor, which is collinear with the intercept, would keep a coefficient around 1e-13 and be selected.

Missing values

Under the default `impute = "none"`, a missing predictor value is an error that names the affected columns, because imputing before cross-validation lets the folds see each other. `impute = "mean"` fills missing values with column means computed on all rows, not on the training part of each fold, and warns about that leak. A column that is entirely NA is an error under either setting. The target must be free of NA, NaN and Inf.

See Also

The Regularization and embedded importance guide, which runs this function on a worked example.

Examples

```
if (requireNamespace("glmnet", 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$x2 + rnorm(n)
  res <- fs_lasso(df, "y", seed = 123)
  print(selected(res))
  print(head(res$scores))
}
```

---

fs_mars *MARS (earth) feature selection*

Description

Trains a multivariate adaptive regression splines (MARS) model with `caret::train(method = "earth")` under repeated cross-validation, evaluates it on a held-out test set, and selects the predictors that the final model uses, 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 the predictors and the target, with unique, non-empty column names and no infinite values. 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. Unused levels of a factor target are dropped.
train_ratio	Numeric in (0, 1). Proportion of rows used for training (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 (default 10000). Larger data sets are randomly down-sampled after missing rows are dropped and before the train/test split.
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; default NULL, which leaves the RNG alone and builds no resampling seed lists. When given, the seed is set for this call only and the previous RNG state is restored on exit. It drives the down-sampling and the train/test split, and one seed per resample and tuning candidate is drawn from it and passed to caret::trainControl(seeds =). That seed list is what keeps a seeded n_cores > 1 run reproducible; the parallel workers themselves are not given seeded RNG streams.
verbose	Logical. Print progress messages, caret's per-fold iteration log, and a warning when a class has fewer than 10 rows (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. Values above 1 need 'doParallel' and 'foreach'; the workers need 'caret' and 'earth' installed, not attached.

Details

MARS builds a model from piecewise-linear hinge terms, optionally multiplied together into interactions, and then prunes the terms back. It finds non-linear effects and interactions without you specifying them, and the predictors that appear in the surviving terms form the selected set. That set comes from one fitted and tuned model, not from a per-predictor test.

The steps run in this order:

1. Rows with any missing value are dropped.
2. The data are randomly down-sampled to at most `sample_size` rows.
3. For classification, unused target levels are dropped and each class must have at least two rows.
4. A stratified `caret::createDataPartition()` split keeps `train_ratio` of the rows for training and holds the rest out.
5. Near-zero-variance predictors (see `remove_nzv`) and highly correlated numeric predictors (see `corr_cut`) are identified on the training rows only and dropped from both halves.
6. `caret::train()` tunes earth under repeated cross-validation with `preProcess = c("center", "scale")` applied inside each resample.
7. The final model is evaluated on the held-out rows.

`search = "grid"` tunes over `expand.grid(nprune, degree)`. `search = "random"` ignores that grid and evaluates `tuneLength` random hyperparameter combinations instead. Tuning optimizes `RMSE` for regression, `ROC` for binary classification when 'pROC' is installed, and `Accuracy` otherwise.

Everything in `details$metrics` comes from the single held-out split. The selection and preprocessing never saw those rows, so the numbers are honest, but on small data sets they are noisy. The cross-validated figures in `model$results` were used to pick the tuning values, so the winning row is optimistic. The per-resample results for every candidate are in `model$resample`.

Value

An object of class `fs_result` with:

selected Character vector of the predictors that appear in a term of the final earth model, ordered by decreasing importance. If `caret::varImp()` fails (with a warning), those predictors are still reported, in model order.

scores Named numeric vector of unscaled variable importance, one entry per candidate predictor that entered training. Predictors earth pruned away score 0; a predictor in the final model keeps its importance even when that is zero or negative. `NULL` when caret cannot compute variable importance for the fitted model, in which case a warning carries caret's error message.

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; for classification, a factor with the training levels), `metrics` (RMSE, MAE and R2 for regression; Accuracy and Kappa for classification; for binary classification also ROC_AUC when the optional 'pROC' package is installed and PR_AUC when 'PRROC' is installed as well; `ROC_AUC` is `NA` and `PR_AUC` absent when the held-out rows contain only one class), `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 within the cleaned and possibly down-sampled data), `test_data` (the held-out rows after preprocessing) and `n_features` (number of candidate predictors that entered training).

call The matched call.

How predictors are selected and scored

`caret::varImp(model, scale = FALSE)` is computed on the caret `train` object, and every predictor that appears in a term of the final (pruned) earth model is reported in `selected`. Predictors earth pruned away score 0 and are not selected. earth's GCV importance can be zero or negative for a predictor the final model still uses; such a predictor is selected and keeps its signed importance.

caret expands factors into dummy columns before fitting, so each importance row is attributed back to its source predictor by name (the longest candidate name it starts with), and a predictor keeps the largest importance of its dummy columns. This attribution is ambiguous when one predictor's name is a prefix of another predictor's dummy column name. The backticks caret adds around non-syntactic names are stripped first, so such columns are attributed like any other.

Classification

Class imbalance is handled by `sampling = "up"` in `caret::trainControl()`, so upsampling happens inside each resample. The data are never upsampled before cross-validation, which would leak duplicated rows across folds.

The first factor level, after unused levels are dropped and labels sanitized, is the positive class for ROC and PR AUC and for the sensitivity and specificity (`Sens`, `Spec`) that `caret::twoClassSummary()` reports in `model$results`. There is no override; reorder the target's levels with `stats::relevel()` to change it. `fs_randomforest()` differs: its default positive class is the second level. ROC AUC does not depend on which class is positive, but PR AUC, sensitivity and specificity do. The test-set ROC AUC is computed with a fixed direction, so a worse-than-chance model scores below 0.5 instead of being flipped.

Factor levels are sanitized with `make.names(unique = TRUE)`, so distinct labels such as "class 1" and "class.1" are never merged. Classification needs the 'e1071' package (used by `caret::confusionMatrix()` and caret's resampling summaries), which is checked before any training.

Regression

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

See Also

The Regularization and embedded importance guide, which runs this function on a worked example.

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)
```

```
  print(res$selected)
  print(res$scores)
  print(res$details$metrics)
}
```

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

Description

Runs a principal component analysis (PCA) on the numeric columns of `data` and returns loadings, scores, and the share of variance each retained component explains. PCA finds the directions of greatest spread in the data; use it to see how much of that spread a few components carry and which columns drive them.

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. Column names must be unique and non-empty. A non-numeric (label) column named like a retained score column (`PC1`, `PC2`, ...) would be hidden by the scores in `pca_df`, so that is an error. Columns in extreme units (beyond about `1e154` or below `1e-154`) are kept, because the zero-variance check is relative to each column's magnitude; with `scale_data = FALSE` their variance cannot be represented and they are an error.
`num_pc`	Number of principal components to retain: a whole number `>= 1`, or `NULL` (the default) for `min(2, max_possible)`. `max_possible` is `min(nrow - 1, ncol)` of the usable numeric data, after incomplete rows and zero-variance columns are dropped. A larger explicit value is an error.
`scale_data`	Logical; scale numeric columns to unit variance. Default `TRUE`. `scale_data = TRUE` with `center_data = FALSE` is an error, because `stats::prcomp()` and `bigstatsr::big_scale()` handle uncentered scaling differently (the latter ignores it), and the result would depend on which engine the data size selected.
`center_data`	Logical; center numeric columns. Default `TRUE`.

label_col	Optional single string naming a column of data used to label and color points. It is left out of the PCA features and carried into pca_df. Default NULL: no labels, and therefore no plot.
plot	Logical; if TRUE, print the PC1 vs PC2 scatterplot. Default FALSE, whether or not label_col is supplied. The plot object is returned in $plot either way whenever one can be built (see Details).
verbose	Logical; print progress messages naming the non-numeric columns held back as labels, the engine chosen, and any fallback from 'bigstatsr'. Default FALSE.

Details

PCA 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.

Before the decomposition:

- Non-numeric columns (character, factor, logical, Date, ...) are left out of the features and carried along as labels. An explicit label_col, numeric or not, is also left out of the features.
- Rows with a missing value in any numeric feature column are dropped. An infinite value in a feature column is an error.
- Zero-variance columns are dropped with a warning. A column counts as zero-variance when its standard deviation is at most 1000 * .Machine$double.eps (about 2e-13) times its largest absolute value. A column that is constant up to floating-point round-off is therefore dropped, while small-unit or large-offset columns (timestamps, say) are kept.

Data with fewer than 1e7 cells is decomposed with stats::prcomp(). From 1e7 cells up, if the suggested package 'bigstatsr' is installed, bigstatsr::big_SVD() runs on a temporary file-backed matrix (the backing file is deleted when the call returns) and can compute at most one fewer component than there are usable columns. Without 'bigstatsr' the call falls back to prcomp(), and says so only when verbose = TRUE.

var_explained is the proportion of total variance explained by each retained component under both engines. It sums to 1 only when every available component is retained; with the default two components it sums to the fraction those two capture. prcomp() computes every component, so the total is the sum of all eigenvalues. big_SVD() computes only the top num_pc singular values, so the total is recovered separately from the column statistics of the (implicitly centered and scaled) matrix. Under either engine, read the entries as shares of the whole, never as shares of the components that happened to be computed.

plot controls printing only, never construction. Whenever a plot can be built (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, whatever the value of plot. plot = TRUE also prints it and makes 'ggplot2' required: its absence is then an error rather than a missing $plot. plot = TRUE without a label_col, or with fewer than two retained components, warns and skips the plot.

Value

A plain list, not an fs_result: fs_pca() reduces dimensions rather than selecting features, so it has no selected set to report. The components are:

- `pc_loadings`: numeric matrix of variable loadings, one row per retained numeric feature (named) and one column per retained component (`PC1`, `PC2`, ...).
- `pc_scores`: numeric matrix of observation scores, one row per kept observation and one column per retained component.
- `var_explained`: numeric vector of length `num_pc`, the proportion of total variance carried by each retained component (see Details).
- `pca_df`: data.table of the scores with the label columns (every non-numeric column, plus `label_col`) bound alongside, restricted to the kept rows.
- `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 (a `label_col` was supplied, at least two components were retained, and 'ggplot2' is installed). Test for it with `"plot" %in% names(res)`.

See Also

The Dimensionality reduction guide, which runs this function on a worked example.

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 with a held-out evaluation*

Description

Fits a random forest on a stratified training split, scores each predictor by permutation importance, and evaluates the forest on the held-out rows. Unless you supply `control$feature_select`, `selected` holds every predictor the forest used, ordered by importance, and you choose where to cut the list.

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). Column names must be unique and non-empty, and predictors may not contain infinite values. Date columns are converted to numeric before modeling.
target	Name of the target column of data, as a single string (a column index is not accepted). For classification the target is coerced with as.factor(), an ordered factor is treated as unordered, and at least two levels must remain. For regression it is coerced with as.numeric(), a factor through its labels (as.numeric(as.character(y))) and never its internal level codes, and an infinite value is an error. Either way, rows whose target is NA after coercion are dropped with a warning.
task	"classification" (the default) or "regression", matched with match.arg(). The task is not inferred from the target. If you leave task at its default and the target is numeric, a warning says that every distinct value will become a class.
control	List of method-specific options. Every accepted entry and its default is listed under "The control list" in Details. Default list(), meaning all defaults.
seed	Optional single whole number within the range of an R integer; fractional or out-of-range values are an error. The seed applies for the duration of the call and the previous RNG state is restored on exit. With more than one worker it is also passed to parallel::clusterSetRNGStream(), and it is then the only way to make the run reproducible. Default NULL: the RNG is never seeded unless you ask.
verbose	Logical. If TRUE, print the split sizes, how many predictors the hook kept, the tree and worker counts, and a note when AUC is skipped because 'pROC' is missing. Default FALSE.
n_cores	Number of workers used to grow the forest, a whole number >= 1. Default 1L (sequential; no cluster is created). Capped at the detected core count and at control$ntree. An effective value above 1 needs the suggested packages 'foreach' and 'doParallel', and leaves details$oob empty (see Details).

Details

A random forest averages many decision trees, each grown on a bootstrap sample of the training rows with a random subset of predictors tried at each split. Forests take any mix of numeric, factor, character, and Date predictors and need no scaling, so fs_randomforest() is a reasonable default

when the feature types are messy. Permutation importance is the drop in out-of-bag accuracy (or rise in out-of-bag MSE) when one predictor's values are shuffled. It tends to favor predictors with many distinct values and to split credit between correlated predictors, so treat close scores as ties.

The honest performance estimate is `details$metrics`, computed on rows the forest and the selection hook never saw (`control$preprocess` does see them; see "What sees the held-out rows"). The Validation and leakage article lists which numbers in each featR result are held out and which are optimistic.

Order of operations:

1. `control$preprocess` runs on the full table.
2. `Date` columns become numeric, and the target is coerced and cleaned (see `target`).
3. If `control$sample_size` is set, the table is downsampled.
4. The rows are split into training and test sets, stratified on the target, with `control$train_ratio` of them used for training.
5. `control$feature_select` runs on the training rows.
6. Character predictors become factors, unused factor levels are dropped, and the test factors are re-leveled to the training levels.
7. Near-zero-variance predictors are removed, judged on the training rows.
8. Missing predictor values are imputed with values learned on the training rows.
9. The forest is grown, on `n_cores` workers.
10. The forest predicts the test rows, and permutation importance is read from it.

Because near-zero-variance removal comes after the hook, `details$feature_names` (the predictors the forest used) can be a subset of `selected`.

The control list:

Every accepted entry, with its default:

- `train_ratio` = `0.75`: training proportion of the stratified split, strictly between 0 and 1.
- `sample_size` = `NULL`: optional whole number. When set, the data is downsampled to about this many rows *before* the split, proportionally within each class for classification and uniformly at random for regression. Values above the available row count are capped. `NULL` keeps every row.
- `ntree` = `500`: number of trees, a whole number >= 1.
- `importance` = `TRUE`: compute permutation importance. When `FALSE`, `scores` and `details$importance` are `NULL` and `selected` keeps column order.
- `scale_importance` = `TRUE`: passed as `scale` to `randomForest::importance()`, which divides each permutation importance by its standard error over trees. Set `FALSE` for the raw importance. With more than one worker the standard error is pooled across the sub-forests, so scaled scores do not depend on `n_cores`.
- `mtry` = `NULL`: predictors tried 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 left after the hook and near-zero-variance removal.
- `nodesize` = `NULL`: minimum size of terminal nodes, a whole number >= 1. `NULL` keeps the `randomForest` default (1 for classification, 5 for regression).
- `maxnodes` = `NULL`: maximum number of terminal nodes per tree, a whole number >= 2, or `NULL` for no limit.

- sampsize = NULL: rows drawn to grow each tree, passed to randomForest. Regression takes a single number (a vector is an error); classification also accepts one number per class. Entries must be whole numbers >= 1. A single value is clamped to the training row count. A per-class (or, with strata, per-stratum) vector has each entry clamped to that class's (stratum's) count in the training rows, with a warning, because randomForest refuses larger entries and you cannot know the training split in advance. An NA entry becomes the full training size when replace = TRUE and ceiling(0.632 * n) otherwise, and is then clamped the same way.
- classwt = NULL: class priors for classification, passed to randomForest unchanged and unvalidated.
- strata = NULL: name of a column of data to stratify the per-tree sampling on (used with sampsize). Its values are read from the training rows so they line up with the rows the forest is grown on. Passing the values as a vector is an error, because the training rows are chosen inside the function. The column must not contain NA, and it stays in the model as a predictor unless you remove it.
- replace = TRUE: sample rows with replacement when growing each tree.
- preprocess = NULL: a function dt -> dt applied to the full data before the split. It must return a data.frame or data.table that still contains the target. See "What sees the held-out rows" below.
- feature_select = NULL: a function dt -> dt that runs on the training rows only. It must keep the target and may only subset and reorder existing columns, because the same columns are then taken from the test rows. Put feature engineering in preprocess.
- impute = TRUE: replace missing predictor values with the median (numeric) or the most frequent value (factor, character, logical), learned on the training rows and applied to both splits.
- drop_zerovar = TRUE: remove near-zero-variance predictors with caret::nearZeroVar(), using the training rows only. With FALSE, training rows in which every predictor is constant are an error, since the trees have nothing to split on.
- oob = TRUE: report out-of-bag metrics in details$oob (accuracy for classification, RMSE for regression). They are optimistic after a supervised hook and unavailable after parallel training; see below.
- return_test_data = FALSE: when TRUE, the held-out rows are returned in details$test_data.
- positive_class = NULL: level treated as the positive class for binary AUC. NULL means the second factor level. A string that does not name a level of the cleaned target is an error.

Unknown entries are an error, so typos and arguments that moved out of control (seed, n_cores, and split_ratio, now train_ratio) fail loudly.

What sees the held-out rows:

control$feature_select runs after the split on the training rows only, so a hook that looks at the target does not leak the test rows into details$metrics.

The out-of-bag metrics in details$oob do not get that protection. The forest is grown on the same training rows the hook selected on, so an out-of-bag row was left out of some trees but was still used for the selection. With a hook that consults the target, details$oob is optimistic, and on pure noise it can read well above chance.

control$preprocess runs on the full data by design. Any quantity it learns from other rows, such as an imputation value, a scaling constant, or anything computed from the target, includes the test rows and will flatter details$metrics. Keep it to row-wise transformations. Downsampling with control$sample_size also happens before the split, which is why details$train_index indexes the cleaned and downsampled table rather than the rows of data.

When more than one worker is used (n_cores after the caps), the sub-forests are merged with randomForest::combine(), which drops the out-of-bag error structures (err.rate, mse, confusion). details$oob is then NULL, with a warning if control$oob = TRUE. details$metrics is unaffected; use n_cores = 1 to get the out-of-bag numbers.

Factor levels and missing values:

Factor predictors carry only the levels seen in the training rows. A test value unseen in training becomes NA and is imputed when control$impute = TRUE. With control$impute = FALSE, any NA left in a predictor reaches randomForest, which fails on it.

Metrics:

Classification reports accuracy, kappa, and auc on the held-out rows. kappa is Cohen's kappa from the held-out confusion table, NA when chance agreement is 1. auc is filled in only when the target has exactly two levels and the suggested package 'pROC' is installed. It is NA for every multi-class fit, and also (with a warning) when the held-out rows contain only one of the two classes.

For a binary target, the positive class (whose predicted probability is scored) is control$positive_class or, by default, the *second* factor level after unused levels are dropped. fs_mars() uses the *first* level. ROC AUC is the same whichever class is called positive, so auc does not depend on this choice, but class-specific quantities such as fs_mars()'s PR AUC or sensitivity are not directly comparable with anything computed here.

Regression reports RMSE, MAE, and R2 on the held-out rows. R2 is 1 - SSE/SST, so it can be negative; it is not the squared correlation that fs_mars() reports. It is NA when the held-out target has no variance, including variance that is only floating-point round-off relative to the target's magnitude.

Value

An object of class fs_result with:

selected Character vector. With control$feature_select, the predictors the hook kept; otherwise every predictor that reached the forest. 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, mean increase in MSE for regression (randomForest names that column with a percent sign, but it is not a percentage). 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, or the result of randomForest::combine() when more than one worker was used.

details A list with:

- metrics: held-out metrics, a named list with accuracy, kappa, and auc for classification, or RMSE, MAE, and 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$importance = FALSE`.
- `confusion`: Observed x Predicted table (classification only).
- `oob`: out-of-bag `accuracy` or `RMSE`; `NULL` when trained in parallel or switched off.
- `feature_names`: the predictors the forest used.
- `train_index`: integer training rows of the cleaned and optionally downsampled data.
- `test_data`: the held-out rows when `control$return_test_data = TRUE`, `NULL` otherwise.
- `control`: the merged control list.
- `n_features`: candidate predictors after cleaning and before selection.

call The matched call.

See Also

The Regularization and embedded importance guide, which runs this function on a worked example.

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, in decreasing order of importance
  print(res$selected)
  print(res$scores)
  print(res$details$metrics)
}
```

`fs_recursivefeature` *Recursive feature elimination with held-out evaluation*

Description

Finds how few predictors a model needs before its resampled performance starts to drop. `fs_recursivefeature()` splits the data into stratified training and test rows, runs `caret::rfe()` on the training rows, and scores the chosen subset on the test rows. It can also one-hot encode the predictors first and fit a final caret model on the selected ones.

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 holding the target and the candidate predictors (every other column). It needs at least one row, unique non-empty column names, and at least two candidate predictors after any one-hot encoding. It is converted to a plain data.frame on entry.
target	Single string naming the target column; a column index is not accepted. A factor, character, or logical target means classification (unused factor levels are dropped), anything else regression. The target may not contain missing or infinite values.
sizes	Whole-number subset sizes to evaluate; fractional sizes are an error. Default NULL uses 1:p, where p is the predictor count after any one-hot encoding. Values outside [1, p] are dropped with a warning, and the call stops if none are left.
train_ratio	Training proportion of the stratified split, strictly between 0 and 1. Default 0.8.
rfe_control	List of arguments for caret::rfeControl(). Default list(method = "cv", number = 5). • method and number are required. method is one of "cv", "repeatedcv", "LOOCV", "boot", "boot632", "LGOCV" or "test" (matched ignoring case), and number must be at least 2 for "cv" and "repeatedcv". With "repeatedcv" and no repeats, caret's default of 1 is used and a warning says so. • functions is the caret RFE function set, default caret::rfFuncs. A supplied set must contain the functions summary, fit, pred, rank, selectSize and selectVar. • index, indexOut and indexFinal are an error: resampling runs on the internal training split, so row numbers built for data would point at the wrong rows. • allowParallel is dropped with a warning; use parallel instead. verbose, if present, overrides the verbose argument for caret::rfe().
train_control	List of arguments for caret::trainControl(), used only when return_final_model = TRUE and then checked, together with model_method, before the search starts.

It must contain `method`, and also `number` unless `method = "none"` (at least 2 for `"cv"` and `"repeatedcv"`). `index`, `indexOut` and `indexFinal` are an error, for the same reason as in `rfe_control`. Unless set here, `allowParallel` follows `parallel`, so a sequential call does not use a foreach backend you registered yourself. Default `list(method = "cv", number = 5)`.

`model_method` Single string: the caret model key for the final model, such as `"rf"` or `"lm"`. It must be in caret's built-in model library. Used only when `return_final_model = TRUE`. Default `"rf"`.

`handle_categorical`
Logical; one-hot encode the predictors with full-rank dummies, fitted on the training rows and applied to the test rows. A dummy name (`<variable><level>`) that matches another column is an error. Function sets that work on a model matrix, such as `caret::lmFuncs`, rank dummy columns rather than factors, so they need `handle_categorical = TRUE` when factor predictors are present; without it the call stops with an error that says so. Default `FALSE`.

`return_final_model`
Logical; train a final caret model on the training rows using the selected predictors and return it as `model` (the `rfe` object stays in `details$rfe`). Near-zero-variance and linearly dependent predictors are dropped from the selected set first, each with a warning, and the survivors are recorded in `details$final_model_variables`. Default `FALSE`.

`seed` Optional whole number within the range of an R integer. Fractional or out-of-range values are an error, not truncated. Default `NULL`, which never seeds. See "Seeds and parallel runs" in Details.

`verbose` Logical; print progress messages and let `caret::rfe()` report its own progress, unless `rfe_control$verbose` overrides the latter. Default `FALSE`.

`parallel` Logical. `TRUE` registers a two-worker PSOCK cluster (capped at the detected core count) for the duration of the call, then stops it and restores whichever foreach backend was registered before. Requires the suggested packages 'foreach' and 'doParallel'. The RFE resampling and the final model's resampling both follow this setting. A custom function set such as `caret::caretFuncs` builds its own inner `caret::train()` control and is not covered. Default `FALSE`.

Details

Recursive feature elimination (RFE) fits a model on all candidate predictors and ranks them by importance, then refits on the top `k` for each subset size `k` in `sizes`, inside every resample of `rfe_control`. The size with the best resampled performance wins, so you do not have to pick a cut-off yourself. The price is one model fit per candidate size per resample, which makes RFE one of the slowest methods in featR. The answer is also tied to the model family in `rfe_control$functions` (random forests by default): it tells you which predictors that model needs, not which predictors are informative in general.

The honest performance estimate is `details$test_metrics`. The Validation and leakage article lists which numbers in each featR result are held out and which are optimistic.

Required packages:

'caret' is always required. The default function set, `caret::rfFuncs`, fits random forests and needs 'randomForest'; so does the default final model (`model_method = "rf"`) when `return_final_model`

= TRUE. To avoid random forests, pass another set, for example rfe_control = list(method = "cv", number = 5, functions = caret::lmFuncs). Classification also needs 'e1071', because the metrics come from caret::postResample(). parallel = TRUE needs 'foreach' and 'doParallel'. All of these are checked before any model is fitted.

What the held-out rows see:

Everything learned from data is learned on the training rows: the one-hot encoder, the factor levels the test columns are aligned to, the elimination itself, and the final model. The test rows play no part in choosing either the predictors or the subset size.

details$test_metrics applies caret::postResample() to the test-row predictions of the fitted rfe object, which is caret's refit on all training rows restricted to the optimal subset. It is the only held-out estimate on the result. Two other summaries reuse the training rows and are optimistic:

- details$resampling_results is the resampled performance at each candidate size, computed inside the training rows. The size RFE picked is the best of these, so its entry is biased upward.
- With return_final_model = TRUE, model$results reports train_control resampling inside the same training rows, on predictors that were already chosen using those rows.

Reading the metrics:

The metrics are caret's. For regression they are RMSE, Rsquared and MAE, where Rsquared is the squared Pearson correlation between predictions and observations, not 1 - SSE/SST. It is never negative and ignores bias and scale errors, so it cannot be compared with an R2 defined as 1 - SSE/SST, such as details$metrics$R2 from fs_randomforest(). For classification they are Accuracy (the proportion of test rows classified correctly) and Kappa (Cohen's kappa, accuracy corrected for chance agreement).

Data preparation and requirements:

- Infinite values (Inf, -Inf) anywhere in data are an error.
- Missing values in the training predictors are an error; impute or drop incomplete rows before calling. Missing values in the test rows are not imputed. Depending on the function set, predict() then either stops (random forests) or returns NA for those rows, which caret::postResample() leaves out of details$test_metrics. A warning reports how many rows were left out.
- Character and logical predictors become factors. Factor predictors keep only the levels observed in the training rows; a test value with a level the training rows never saw becomes missing, with a warning.
- Predictors that are constant on the training rows (a single value, or a factor with one observed level) are removed with a warning before RFE. They carry no information, and a random forest fitted on a single constant predictor never returns.
- Predictor names that are not syntactic R names, such as "a b", break caret's function sets, so the rfe object and the final model use their make.names() versions. selected, scores, details$variable_importance and details$final_model_variables report the original names.
- With handle_categorical = TRUE, the encoded test rows are checked against the input row count, so an encoding that loses rows is an error instead of a misaligned metric.

Seeds and parallel runs:

`seed` covers the split and the RFE resampling. It is applied for the duration of the call only, and the previous RNG state is restored on exit. With `seed = NULL` the call draws from your current RNG stream without reseeding it. Under `parallel = TRUE`, a seed also sets `rfe_control$seeds = NULL` unless you supplied `seeds`, so caret draws one seed per resample from the locally seeded RNG. That is what makes two seeded parallel runs agree. The workers also get L'Ecuyer-CMRG streams, but those alone would not be enough, because foreach hands each resample to whichever worker is free. A seeded parallel run need not match a seeded sequential one, because the sequential run keeps caret's default `seeds = NA`. See Reproducibility and parallelism.

Value

An object of class `fs_result` with:

selected Character vector of the predictors RFE kept: caret's `optVariables` at the optimal subset size.

scores Named numeric vector with one entry per candidate predictor: its importance in the full-size fit (all candidates present), averaged over every resample. `caret::varImp()` on the `rfe` object instead averages a predictor only over the resamples where it survived to the optimal size, which inflates marginal predictors and omits predictors that never survived. When the function set records no per-resample importance, the `caret::varImp()` values are used instead. `NULL` when neither is available.

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 with:

- `rfe`: the caret `rfe` object, present even when `model` holds the final model.
- `optimal_size`: the subset size RFE chose.
- `test_metrics`: `caret::postResample()` on the test rows, the only held-out estimate on the result.
- `resampling_results`: the RFE resampling summary over candidate sizes, computed inside the training rows.
- `variable_importance`: the `caret::varImp()` data.frame. It averages each predictor only over the resamples where it survived to the optimal size, and `selected` follows caret's `optVariables` rule, which is weighted the same way.
- `preprocessor`: the `dummyVars` encoder, or `NULL` when `handle_categorical = FALSE`.
- `train_index`, `test_index`: row indices into `data` for the two partitions.
- `final_model_variables`: the predictors the final model used after near-zero-variance and linear-combination filtering, or `NULL` when no final model was requested.
- `n_features`: the number of candidate predictors offered to RFE, counted after one-hot encoding and after constant predictors were removed.

call The matched call.

See Also

The Wrapper methods guide, which runs this function on a worked example.

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 by AIC*

Description

Finds the subset of columns that AIC keeps in a linear model. Uses `MASS::stepAIC()` to add or drop one term at a time from a linear regression of `target` on all other columns of `data`, stopping when no single move lowers AIC. Requires the suggested package 'MASS'.

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, with unique, non-empty column names and no infinite values. Every column other than `target` is offered to the search. For backward and both-direction searches, the complete rows must outnumber the coefficients of the full model.
`target`	Single string naming the numeric target column of `data`. Unquoted symbols and column indices are not accepted. Non-syntactic names work, because they are backticked in the formula.
`direction`	Direction of the search: `"both"` (the default), `"backward"`, or `"forward"`.

verbose — Logical. If TRUE, print progress messages and the stepAIC() trace. Default FALSE.

... — Further arguments passed to MASS::stepAIC(), for example k (k = log(nrow(data)) gives BIC), steps, or scope (which then replaces the default scope). trace is controlled by verbose, and a trace supplied here is dropped with a warning. See Details for what else ... absorbs.

Details

This is the cheapest wrapper method in featR and the easiest to read: the result is an ordinary lm that the usual generics work on. 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.

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 are not valid for formal inference; use them only as a rough ordering of the retained terms.

Only linear regression is supported: the target must be numeric, and every other column of data is a candidate predictor. direction = "backward" starts from the full model; "forward" starts from the intercept-only model with the full model as the upper scope; "both" starts from the full model with the same scope, so terms can be added back. There is no seed argument: stepAIC() is deterministic, so the same call on the same data returns the same model, and the RNG is never touched.

Rows with a missing value in any column are dropped, with a warning, before the search, because stepAIC() cannot compare models fitted on different rows. Infinite values are an error.

Backward and both-direction searches start from the full model, which must leave residual degrees of freedom. With as many coefficients (intercept and factor dummies included) as complete rows, the full model fits perfectly, its AIC is -Inf, and the call stops with an error giving the counts. A forward search may have more candidate predictors than rows; it warns if the model it ends on has no residual degrees of freedom, since that perfect fit makes the selection meaningless. A factor or character predictor with fewer than two observed levels (after incomplete rows are dropped) is an error, as is a factor dummy name that collides with another column or the target (factor g with level b next to a column gb).

The models reach their data through a private environment attached to the model formula, so predict(), summary(), anova(), add1(), drop1() and similar generics work on the returned model. Nothing is assigned to the global environment or written to disk. That environment holds a copy of the complete-case data and is kept alive by the returned model, so the returned object is at least as large as that copy. Plain update(model) fails with an object-not-found error, because update() re-evaluates the stored call in the caller's environment, where the private data is not visible. Pass the data explicitly instead: update(model, . ~ . - x, data = my_data).

Non-syntactic column names are backticked in the formula and reported without the backticks, so selected matches names(data).

... is forwarded to MASS::stepAIC() unchanged, so it also swallows arguments this function does not have. The removed arguments seed and return_models are passed through and ignored rather than rejected: a call written for the older interface still runs, without the effect it asked for. Check your argument names against the list below.

Value

An object of class `fs_result` with:

selected Character vector of the selected predictor terms, without the intercept.

scores Named numeric vector with one entry per selected term, keyed like `selected`: the absolute t statistic of the term's coefficient or, for a term with several coefficients (a factor, an interaction), the largest `|t|` among them (`NA` when all are aliased). The per-coefficient values (for example `grpb` and `grpc` for the term `grp`) are in `details$coefficient_scores`. These statistics are computed after selection on the same data, so they are optimistically biased and not valid for inference.

method `"stepwise"`.

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

model The `lm` chosen 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, biased like `scores`), `coefficient_scores` (named `|t|` per non-intercept coefficient), `selected_terms` (the same labels as `selected`), `direction` (the search direction used), `n_features` (the number of candidate predictors, `ncol(data) - 1`), and `dropped_na_rows` (the number of rows removed for missing values, `0L` when none were).

call The matched call.

See Also

The Wrapper methods guide, which runs this function on a worked example.

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 that scores each feature against a target*

Description

Scores every column of `data` except `target` against the target, one feature at a time, and keeps or drops each feature by comparing its score with `threshold`. A numeric target is scored by absolute Pearson correlation, a categorical target by the one-way ANOVA F statistic.

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 holding target and the candidate features. Every column other than target must be numeric. The input is copied, never modified in place.
target	Single string naming the target column of data. The target may be numeric, factor, character, or logical; any other type is an error. It is removed from the candidates and used only for scoring.
method	One of "auto" (default), "correlation", or "anova". See Details.
threshold	Non-negative, finite number compared with each feature's score. Default 0. The scale depends on the method: [0, 1] for "correlation", [0, Inf] for "anova". With the defaults (threshold = 0, direction = "above", action = "keep") nothing is filtered: any non-constant feature, pure noise included, has a sample $\|r\| > 0$ or $F > 0$, so every feature with a defined score is kept. Set a threshold to make a real selection.
direction	"above" (default) or "below": which side of threshold a score must fall on to meet the condition.
action	"keep" (default) retains the features that meet the condition; "remove" drops them.
include_equal	Logical. If TRUE, a score equal to threshold meets the condition. Default FALSE.
na_rm	Logical. If TRUE (the default), rows with NA in a feature or in the target are left out when that feature is scored. If FALSE, an NA anywhere in a feature or in the target makes that feature's score NA.
output	Shape of the return value. "Candidate features" means the columns of data other than target, in their original order. • "result" (default): an fs_result object (see Value). • "matrix", "dt", "data.frame": the selected features as a numeric matrix, a data.table, or a data.frame. • "mask": logical vector with one element per candidate feature, named after the candidates; TRUE marks a selected feature.

- "indices": integer positions of the selected features among the candidates (that is, in data without target), named after the selected features.
- "names": character vector of the selected feature names.
- "list": a list with filtered (matrix), mask, indices, names, scores, and meta. meta records method_arg (the method as requested), method_used (after "auto" is resolved), threshold, direction, action, include_equal, na_rm, n_input_cols, and n_kept_cols.

verbose Logical; print progress messages. Default FALSE.

Details

No model is fitted, so the cost grows linearly with the number of columns and the function copes with wide data. Each feature is scored on its own, which has two consequences: the filter cannot see interactions, and it keeps every member of a group of near-duplicate columns that all track the target. Prune that redundancy afterwards with fs_correlation(), or use a wrapper such as fs_recursivefeature() or fs_svm() when the joint contribution of features is what you care about.

The three values of method:

- "correlation" requires a numeric target. The score is the absolute Pearson correlation, in [0, 1]. A constant feature or a constant target has no defined correlation and scores NA. The constancy check is relative to each column's magnitude, so a feature measured on a very small or very large scale is scored normally.
- "anova" requires a factor, character, or logical target. The score is the one-way ANOVA F statistic, in [0, Inf]. A feature that separates the classes perfectly (no spread within any class) scores Inf, and a constant feature scores NA.
- "auto" (the default) uses "correlation" for a numeric target and "anova" otherwise. The two scales differ, so a threshold that suits one is meaningless for the other; with verbose = TRUE a message reports which method was chosen.

A feature whose score is NA is never selected, whether action is "keep" or "remove", and a warning reports how many features were excluded this way.

Columns are subset by position, never by name, so duplicated feature names cannot pull in the wrong column. The target's name must be unique, and every column needs a non-empty, non-NA name. A matrix keeps its own column names; V1, V2, ... are used only when it has none.

Value

With output = "result" (the default), an fs_result object:

- selected: character vector of selected feature names.
- scores: named numeric vector with one score per candidate feature, in column order, NA where the score is undefined.
- method: "supervised_" followed by the method actually used, for example "supervised_correlation".
- task: "regression" for a numeric target, "classification" for a factor target, and NA for a character or logical target, whose intended task is ambiguous.
- model: NULL.

- `details`: a list holding, in this order, `mask` (the logical keep-mask over the candidates), `indices` (selected positions, named), `filtered` (the selected columns as a data.table), `threshold`, `direction`, `action`, and `n_features` (the number of candidates, `ncol(data) - 1`).
- `call`: the matched call.

Any other `output` returns the shape described under `output`.

When no feature meets the criteria, a warning is issued and the tabular shapes come back empty. `"matrix"` and `"data.frame"` then have zero columns and keep the input row count. The `"dt"` shape and `details$filtered` have zero columns and also zero rows, because data.table represents a zero-column table that way.

See Also

The Filter methods guide, which runs this function on a worked example.

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 singular value decomposition (SVD) of a numeric matrix, optionally centering and scaling it first and keeping only the leading singular values and vectors. The SVD factors a matrix into u %*% diag(d) %*% t(v); the leading terms give the best low-rank approximation of the matrix in the least-squares sense.

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). It must contain no NA, NaN, or 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)). A value above min(dim(x)) is an error.
scale_input	How to transform the columns before decomposing: TRUE (default) centers and scales, FALSE leaves the matrix alone, "center" subtracts column means only, and "scale" divides only. Any other value is an error. The two dividing forms (TRUE and "scale") need every column to have non-zero variance, meaning a standard deviation above 1000 * .Machine$double.eps (about 2e-13) times the column's largest absolute value, so the check does not depend on units. Offending columns are named in the error, or numbered when the matrix has no column names. "scale" follows base::scale(): without centering it divides by the root mean square, not 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 it. Default 100.
approx_args	List of extra arguments passed to RSpectra::svds(), for example tol or opts. It must not include A or k, which are set from x and n_singular_values.
verbose	Logical; print progress messages. Default FALSE.

Details

Use it for a low-rank summary of a numeric matrix: compression, denoising, or a latent-factor representation. The components are linear combinations of every input column, so nothing is dropped

and the components cannot be read as individual features; use one of featR's selection functions when you need to name the features you keep. `fs_pca()` covers the same ground with labeled output and variance-explained figures; `fs_svd()` is the lower-level decomposition.

The exact solver is `base::svd()`. The approximate solver is `RSpectra::svds()` (from **RSpectra**, a Suggests dependency needed only when that solver runs), and it can only return fewer than `min(dim(x))` singular values. The default `n_singular_values = NULL` asks for all `min(dim(x))` of them and so always takes the exact path; request fewer to make the approximate solver available.

With `svd_method = "auto"`, the approximate solver runs only when `n_singular_values < min(dim(x))` and `min(dim(x)) > svd_threshold`. Otherwise the exact solver runs, and a message explains why when a large matrix ends up on the exact path because all singular values were requested. `svd_method = "approx"` with `n_singular_values` equal to `min(dim(x))` is not an error: it falls back to the exact solver with a message. If `RSpectra::svds()` converges on fewer singular values than requested, the call warns and recomputes them with the exact solver.

Centering and scaling happen before the decomposition, so the returned values and vectors factorize the transformed matrix. With the default `scale_input = TRUE` they reconstruct `scale(x)`, not `x`. The column means and standard deviations are not returned; keep them yourself if you need to map results back to the original units.

Value

A plain list, not an `fs_result`: `fs_svd()` reduces dimensions rather than selecting features. Writing `k` for the number 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 block of tied singular values), so column signs can differ between the exact and approximate solvers and between platforms.

See Also

The Dimensionality reduction guide, which runs this function on a worked example.

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 a support vector machine (SVM) classifier or regressor with caret's **kernlab** engines, tunes it by cross-validation, and reports its performance on held-out rows. With `feature_select = TRUE` it first selects features on the training rows, by SVM-RFE (the default) or by random-forest RFE. It can also up-sample minority classes and train on several workers.

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 (or data.table) containing the predictors and the target, with unique, non-empty column names.
`target`	A string naming the target column.

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 proportion of the stratified split, strictly between 0 and 1. Default 0.7.

nfolds — Number of cross-validation folds for tuning the final model, a whole number greater than 1. Default 5. SVM-RFE also uses it for its size search (capped there at 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 data frame of tuning values for caret::train(). If NULL, the default grid for the kernel is used: C = 2^(-5:10) for "linear"; sigma = 2^(-15:-5) crossed with C = 2^(-1:10) for "radial"; degree = 2:5, scale = c(0.1, 0.5, 1) and C = 2^(-3:8) crossed for "polynomial".

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; 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 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", the selection is truncated to its n_features highest-scoring entries, and n_features is also the minimum the random-forest fallback keeps (see Details). Ignored when feature_select = FALSE. Default NULL (the size is chosen automatically).

class_imbalance — Logical; if TRUE and the task is classification, up-sample classes within each cross-validation resample. Default FALSE.

seed — Optional whole-number seed within the R integer range. Fractional values are an error, not truncated. Default NULL, which never seeds. See "Packages, seeds and parallel runs" in Details.

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. Above 1, a cluster is created for the duration of the call and stopped on exit.

Details

For classification, an SVM fits the boundary between classes that stays furthest from the nearest training points while tolerating some points on the wrong side. For regression, it fits a function that keeps most points within a margin around it. The cost C sets how heavily points that break these rules are penalized. The kernel sets the shape of the boundary or function: a hyperplane for "linear", curved surfaces for "radial" and "polynomial".

Use `fs_svm()` when the selector and the final model should come from the same family. SVM-RFE ranks features by the weights of a linear SVM, and the result carries the tuned model and its held-out performance next to the chosen features. The cost is 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()`.

The honest performance estimate is `details$performance`. The Validation and leakage article lists which numbers in each featR result are held out and which are optimistic.

Order of operations:

1. The target is coerced for `task` (see "Data requirements").
2. The rows are split into training and test sets, stratified on the target, with `train_ratio` of them used for training. For classification, test rows whose class does not occur in the training rows are dropped with a warning.
3. Factor and character predictors in the test rows are re-leveled to the levels observed in the training rows. Test rows with a level the training rows never saw are dropped with a warning.
4. Predictors are dummy-encoded with full-rank dummies, fitted on the training rows.
5. With `feature_select = TRUE`, the selector runs on the encoded training predictors, and the training and test matrices are cut down to the selected columns.
6. The final SVM is fitted with `caret::train()`: predictors are centered and scaled, and `tune_grid` is searched by `nfolds`-fold cross-validation on the training rows.
7. The tuned model predicts the test rows, which gives `details$performance`.

SVM-RFE:

SVM-RFE (Guyon, Weston, Barnhill and Vapnik, 2002) fits a linear SVM on the centered and scaled encoded training matrix and ranks the features by their squared primal weight `w^2`. For a multi-class fit, `w^2` is summed over the pairwise problems. The lowest-ranked feature is dropped (the lowest 10 percent while more than 50 features remain), and the SVM is refitted on the rest until one feature is left. Reversing the elimination order gives the ranking, so rank 1 is the feature eliminated last. These fits use a fixed cost of `C = 1` and are separate from the final model, which is tuned over `tune_grid`.

SVM-RFE requires `kernel = "linear"`, because the primal weight vector only exists for a linear kernel. Combining it with another kernel is an error that suggests `select_method = "rf_rfe"`.

With `n_features` supplied, SVM-RFE keeps exactly that many features from the top of the ranking. Otherwise it scores 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. Each size is scored by `nfolds`-fold cross-validation with the same linear SVM, on folds shared by every size. Inside each fold, the standardization and the whole elimination are re-run on that fold's training rows, and the top features of that fold's ranking are scored on its held-out rows, so no size is scored by a ranking that has seen the rows it is scored on. The size with the highest mean accuracy (classification) or the lowest mean RMSE (regression) wins, ties going to the smaller size. The kept features are the top of the ranking computed on the whole training split. If no size could be scored, all features are kept with a warning.

Each entry of `details$selection$size_scores` is an honest estimate for its own size. The entry for the chosen size, though, is the best of up to six noisy estimates and so is optimistic; on pure noise it typically sits a few points above chance.

Random-forest screening:

`select_method = "rf_rfe"` screens features with random-forest RFE (`caret::rfFuncs`) and works with every kernel. It runs its own 10-fold cross-validation over subset sizes 1 to `p`, independent of `nfolds`. Encoded predictors that are numerically constant are removed first; their score is `NA` and they are never selected.

If `caret::rfe()` fails or selects nothing, a plain random forest is fitted instead, with a warning. Every predictor with positive permutation importance (mean decrease in accuracy, or `%IncMSE` for regression) is kept, or, if fewer than `n_features` qualify (1 when `n_features = NULL`), the top `n_features` by that importance. This fallback is a weak screen: roughly half of pure-noise predictors have positive permutation importance. With `n_features` supplied, the selection is then truncated to the `n_features` selected predictors with the highest `scores`, so the fallback keeps exactly that many (capped at the number available).

Which numbers are optimistic:

Selection runs on the training split only, so the test rows never inform which features survive. The final model's tuning cross-validation reuses the training rows the selector has already seen, though. With `feature_select = TRUE`, the resampled metrics in `model$results` and `model$resample` (what printing `model` shows) are therefore biased upward, and on pure noise they can approach perfect accuracy. Taking the best point of `tune_grid` adds further optimism. Only `details$performance`, computed on the untouched test split, is an honest estimate.

Class-imbalance handling (`class_imbalance = TRUE`, classification only) up-samples within each cross-validation resample via `caret::trainControl(sampling = "up")`, so duplicated rows do not leak across folds.

Reading the performance:

For regression, `details$performance` holds `RMSE`, `MAE` and `Rsquared` from `caret::postResample()`. `Rsquared` is the squared Pearson correlation between test predictions and observations, not `1 - SSE/SST`. It is never negative and ignores bias and scale errors (predictions of `2 * y` score 1), so it cannot be compared with an `R2` defined as `1 - SSE/SST`, such as the one from `fs_randomforest()`. For classification it is a `caret::confusionMatrix`, whose `overall` element holds `Accuracy` and Cohen's `Kappa`. Test rows with a missing prediction are dropped from the metrics with a warning.

Data requirements:

- Missing values in the predictors or the target are an error, and so are infinite values.
- For classification, no class label may be the empty string. A numeric target is coerced to a factor (with a message) only when it has at most 10 unique values; more than 10 is an error suggesting `task = "regression"`. For regression the target must be numeric.
- A factor or character predictor with fewer than two distinct non-missing values is an error naming the column. So is one left with a single level in the training rows because its other levels fell only into the test rows.
- Unused factor levels are dropped before encoding, so they produce no all-zero dummy columns.
- Dummy columns are named by caret's `dummyVars()`: `color.green` for a factor with three or more levels, `g1` for a two-level factor `g`. If a dummy name matches another column or the target (factor `g` with levels `"0"` and `"1"` next to a numeric `g1`), the call stops and asks for a rename.

Packages, seeds and parallel runs:

`fs_svm()` always needs the suggested packages **caret** and **kernlab**. It also needs **e1071** for classification, **randomForest** for `select_method = "rf_rfe"` with `feature_select = TRUE`, and **doParallel** and **foreach** when `n_cores > 1`.

seed is applied for the duration of the call and the previous RNG state is restored afterwards. When n_cores > 1, caret's resampling controls get seeds = NULL, so caret draws one seed per resample from the locally seeded RNG. That is what makes two seeded parallel runs agree. The workers also get L'Ecuyer-CMRG streams, but those alone would not be enough, because foreach hands each resample to whichever worker is free. A seeded parallel run need not match a seeded sequential one. See Reproducibility and parallelism.

Value

An object of class fs_result with:

selected Character vector of selected encoded feature names. With feature_select = FALSE this is every encoded predictor. With "svm_rfe" it is the surviving subset, from most to least important. With "rf_rfe" it is the subset in the order caret::rfe() reports it, or, when n_features truncated it, ordered by decreasing scores.

scores Named numeric vector covering every encoded predictor, not only the survivors. For "svm_rfe" it is the squared primal weight w^2 from the first, full-feature fit. For "rf_rfe" it is the mean random-forest importance recorded across resamples at the full subset size, NA for predictors caret::rfe() never scored; when the fallback ran, it is the permutation importance of the fallback forest. NULL when feature_select = FALSE, because no selector produced scores.

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

task "classification" or "regression".

model The fitted caret::train object. With feature_select = TRUE its resampled metrics are optimistic (see Details); use details$performance.

details A list with:

- test_set: the test split, with its coerced target and aligned factor levels.
- predictions: test-set predictions in the row order of test_set; a factor for classification, an unnamed numeric vector for regression.
- performance: a caret::confusionMatrix for classification, or a named RMSE/Rsquared/MAE vector for regression, where Rsquared is the squared correlation (see Details).
- selection: NULL when no selection ran, otherwise a list with method, ranking (most to least important), scores, and the size search's sizes, size_scores and size_metric. Those last three are 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.
- n_features: the number of encoded predictors considered, counted before any were dropped.

call The matched call.

See Also

The Wrapper methods guide, which runs this function on a worked example.

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
  )
  print(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
  )
  print(sel$selected)
  print(sel$scores)
}
```

---

fs_unsupervised *Unsupervised filter that scores each feature without a target*

Description

Scores each column of `data` with a summary that ignores any outcome (variance, spread, missingness, or number of distinct values) and keeps or drops each column by comparing its score with `threshold`.

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 whose columns are all 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", or "n_unique". See Details.
threshold	Non-negative, finite number compared with each feature's score. Default 0. Each method has its own scale (a variance is in squared units, "missing_prop" lies in [0, 1], "n_unique" is a count), so a threshold chosen for one method does not carry over to another.
direction	"above" (default) or "below": which side of threshold a score must fall on to meet the condition.
action	"keep" (default) retains the features that meet the condition; "remove" drops them.
include_equal	Logical. If TRUE, a score equal to threshold meets the condition. Default FALSE.
na_rm	Logical. If TRUE (the default), NAs are removed before computing "variance", "mad", "iqr", and "range"; if FALSE, an NA in a column makes its score NA under those methods. "missing_prop" and "n_unique" ignore na_rm.
output	Shape of the return value. • "result" (default): an fs_result object (see Value). • "matrix", "dt", "data.frame": the selected features as a numeric matrix, a data.table, or a data.frame. • "mask": logical vector of length ncol(data), named after the columns of data; TRUE marks a selected column. • "indices": integer positions of the selected columns, named after them. • "names": character vector of the selected column names. • "list": a list with filtered (matrix), mask, indices, names, scores, and meta. meta records method, threshold, direction, action, include_equal, na_rm, n_input_cols, and n_kept_cols.
verbose	Logical; print progress messages. Default FALSE.

Details

Use it for the first cleaning pass: dropping constant or near-constant columns, columns that are mostly missing, or columns with too few distinct values. No outcome informs the scores, so it is safe to run before a train/test split. For the same reason it says nothing about whether a feature is useful. A high-variance column can be pure noise and a low-variance one can be your best predictor; fs_supervised() or a model-based method answers that question.

The six values of method:

- `"variance"` (default): sample variance (denominator n - 1), in squared units. For values below about `1e-162` in magnitude the variance underflows to 0, so to find constant columns in extreme units use `"range"` with threshold 0 or `"n_unique"` with threshold 1.
- `"mad"`: median absolute deviation from `stats::mad()`, which multiplies by the consistency constant 1.4826. Scores are on the scale of a standard-deviation estimate, not the raw median absolute deviation; choose thresholds on that scale.
- `"iqr"`: interquartile range.
- `"range"`: maximum minus minimum.
- `"missing_prop"`: proportion of missing values, in `[0, 1]`. With the default `threshold = 0`, `direction = "above"`, and `action = "keep"`, this keeps the columns that have any missing values and drops the complete ones, which is almost never what you want. A warning then suggests `action = "remove"` or `direction = "below"`. The warning fires only when all three of `threshold`, `direction`, and `action` are left at their defaults; passing any one of them explicitly silences it.
- `"n_unique"`: number of distinct non-`NA` values.

A feature whose score is `NA` is never selected, whether `action` is `"keep"` or `"remove"`, and a warning reports how many features were excluded this way.

Columns are subset by position, never by name, so duplicated column names cannot pull in the wrong column. Every column needs a non-empty, non-`NA` name. A matrix keeps its own column names; `V1`, `V2`, ... are used only when it has none.

Value

With `output = "result"` (the default), an `fs_result` object:

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

Any other `output` returns the shape described under `output`.

When no feature meets the criteria, a warning is issued and the tabular shapes come back empty. `"matrix"` and `"data.frame"` then have zero columns and keep the input row count. The `"dt"` shape and `details$filtered` have zero columns and also zero rows, because data.table represents a zero-column table that way.

See Also

The Filter methods guide, which runs this function on a worked example.

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 the method, how many features were kept, and their names.

Usage

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

Arguments

x	An fs_result object.
n	Maximum number of selected names to list. Default 10.
...	Unused, for consistency with the generic.

Details

The output has at most five lines:

1. A <fs_result> header naming the method, followed by the task in parentheses when one is known.
2. "Selected k of N features", or "Selected k features" when the number of candidates cannot be recovered from the result.

3. The selected names, truncated to the first `n` with a "... (m more)" tail. This line is left out when the selection is empty.
4. A "Model:" line giving the class of `x$model`, when the method fitted one.
5. A "Details:" line naming the elements of `x$details`.

Value

`x`, invisibly. Called for the 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

Returns the names of the features a selection function kept. It gives the same value as `x$selected` and works on the result of any `fs_*()` selection function. Code that calls `selected()` keeps working if the internal layout of the result changes.

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. When nothing met the selection criteria it is `character(0)`, never `NULL`.

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 table of scores ranked strongest first.

Usage

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

Arguments

object	An fs_result object.
n	Maximum number of rows in the score table. Default 20.
...	Unused, for consistency with the generic.

Details

The output starts with everything print() shows, then the recorded call. For methods that produce per-feature scores it ends with a table of every scored feature, with columns feature, score (rounded to four significant digits), and selected (* marks the kept features). The table is truncated to n rows with a "... (m more)" tail.

Value

object, invisibly. Called for the printed output.

How rows are ranked

For most methods a larger score means a stronger feature, so rows are in decreasing order of score. A negative importance (a feature that did worse than its permuted copy) therefore ranks below every positive one. The other rules are:

- `fs_lasso()` scores are signed standardized coefficients. The sign gives the direction of the effect, not its strength, so rows are in decreasing order of absolute value.
- `fs_chi()` scores are adjusted p-values, where smaller is stronger, so rows are in increasing order and the most significant feature comes first.
- `fs_unsupervised()` and `fs_supervised()` also use increasing order when they kept the low side of the threshold (`direction = "below"` with `action = "keep"`, or `"above"` with `"remove"`), so the kept features come first.

Two methods select by something other than a cut on the score. For them the selected features are listed first, each group ordered by score, so a kept feature is never ranked below, or truncated out behind, a dropped one:

- `fs_mars()` selects every predictor the final earth model uses. A used predictor can have a zero or negative GCV importance while pruned predictors score 0.
- `fs_correlation()` with `prune = TRUE` scores each variable by its largest absolute correlation with any other (larger means more redundant) and keeps one member of each redundant pair. Rows are in increasing order within each group, least redundant first. With `prune = FALSE` the selection is the redundant set itself, and rows are in decreasing order of score.

When the result records a per-feature keep-mask (`details$mask`, as the threshold filters do), the `selected` marks are matched by position, so two features sharing a name are marked separately. Otherwise they are matched by name.

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

fs_bayes, 2
fs_boruta, 7
fs_boruta(), *14*
fs_chi, 9
fs_chi(), *61*
fs_correlation, 12
fs_correlation(), *46*, *61*
fs_elastic, 15
fs_infogain, 19
fs_lasso, 22
fs_lasso(), *61*
fs_mars, 26
fs_mars(), *36*, *61*
fs_pca, 30
fs_pca(), *49*
fs_randomforest, 32
fs_randomforest(), *40*, *53*
fs_recursivefeature, 37
fs_recursivefeature(), *46*
fs_stepwise, 42
fs_supervised, 44
fs_supervised(), *52*, *56*, *61*
fs_svd, 47
fs_svm, 50
fs_svm(), *46*
fs_unsupervised, 55
fs_unsupervised(), *61*

print.fs_result, 58

selected, 59
summary.fs_result, 60