Package 'SVEMnet'

October 30, 2026

Type Package

Title Self-Validated Ensemble Models with Lasso and Relaxed Elastic Net Regression

Version 3.6.3

Date 2026-10-29

Description Implements the self-validated elastic-net and relaxed elastic-net ensemble modeling and multi-response optimization workflow described in Karl (2026) <doi:10.1016/j.chemolab.2026.105660>. Self-validated ensemble models (SVEM; Lemkus et al. (2021) <doi:10.1016/j.chemolab.2021.104439>) are fitted for small-sample design-of-experiments and related workflows using 'glmnet' (Friedman et al. (2010) <doi:10.18637/jss.v033.i01>). Fractional random-weight bootstraps with anti-correlated validation copies are used to tune penalty paths by validation-weighted AIC/BIC. Supports Gaussian and binomial responses, deterministic expansion helpers for shared factor spaces, prediction with bootstrap uncertainty, and a random-search optimizer that respects mixture constraints and combines multiple responses via desirability functions. Also includes a permutation-based whole-model test for Gaussian SVEM fits (Karl (2024) <doi:10.1016/j.chemolab.2024.105122>). Package code was drafted with assistance from generative AI tools.

URL https://doi.org/10.1016/j.chemolab.2026.105660, https://arxiv.org/abs/2511.20968

Depends R (>= 4.0.0)

Imports glmnet (>= 4.1-6), stats, cluster, ggplot2, lhs, parallel, utils

Suggests foreach, testthat (>= 3.0.0), withr, RhpcBLASctl

License GPL-2 | GPL-3

Encoding UTF-8

Config/testthat/edition 3

LazyData true

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Andrew T. Karl [cre, aut] (ORCID:
<https://orcid.org/0000-0002-5933-8706>)

Maintainer Andrew T. Karl <akarl@asu.edu>

Repository CRAN

Date/Publication 2026-10-30 17:50:02 UTC

Contents

SVEMnet-package . . . 3
bigexp_formula . . . 7
bigexp_prepare . . . 8
bigexp_terms . . . 9
bigexp_train . . . 15
coef.svem_model . . . 17
forward_aicc . . . 18
glmnet_with_cv . . . 20
lipid_screen . . . 26
plot.svem_binomial . . . 32
plot.svem_model . . . 34
plot.svem_significance_test . . . 35
predict.svem_model . . . 36
predict_cv . . . 40
print.bigexp_formula . . . 42
print.bigexp_spec . . . 42
print.svem_significance_test . . . 43
print.svem_thompson_batch . . . 44
SVEMnet . . . 44
svem_export_candidates_csv . . . 55
svem_forward . . . 56
svem_ij_variance . . . 59
svem_nonzero . . . 60
svem_random_table_multi . . . 62
svem_score_random . . . 64
svem_select_from_score_table . . . 74
svem_significance_test_parallel . . . 76
svem_thompson_batch . . . 82
svem_wmt_multi . . . 88
with_bigexp_contrasts . . . 92

Index **93**

SVEMnet-package *SVEMnet: Self-Validated Ensemble Models with Relaxed Lasso and Elastic-Net Regression*

Description

The SVEMnet package implements Self-Validated Ensemble Models (SVEM) using Elastic Net (including lasso and ridge) regression via glmnet. SVEM averages predictions from multiple models fitted to fractionally weighted bootstraps of the data, tuned with anti-correlated validation weights. The package supports multi-response optimization with uncertainty-aware candidate generation for iterative formulation and process development.

Details

The primary package reference is Karl (2026), doi:10.1016/j.chemolab.2026.105660, published in *Chemometrics and Intelligent Laboratory Systems*. It describes the elastic-net and relaxed elastic-net ensemble workflow, shared factor expansions, and multi-response optimization that form the package's core. The analyses in that paper used SVEMnet version 3.2.0; see the package NEWS for subsequent changes. Use citation("SVEMnet") for the package paper and related references.

A typical workflow is:

1. Build a wide, deterministic factor expansion (optionally via bigexp_terms) and reuse it across responses with bigexp_formula.
2. Fit one or more SVEM models with SVEMnet.
3. Optionally run whole-model testing via svem_significance_test_parallel (and svem_wmt_multi) to assess factor relationships or reweight response goals.
4. Call svem_score_random to draw random points in the factor space, compute multi-response Derringer–Suich scores, optional WMT-reweighted scores, and an uncertainty measure; then use svem_select_from_score_table to pick a single "best" row and diverse medoid candidates, and svem_export_candidates_csv to export candidate tables for the next experimental round.
5. Run new experiments at the suggested candidates, append the data, refit the models, and repeat as needed (closed-loop optimization).

Core modeling and summaries

SVEMnet Fit an SVEMnet model using Elastic Net regression (including relaxed elastic net) on fractionally weighted bootstraps.

predict.svem_model Predict method for SVEM models (ensemble-mean aggregation by default, optional debiasing, and percentile intervals across bootstrap member predictions).

coef.svem_model Averaged (optionally debiased) coefficients from an SVEM model.

svem_nonzero Bootstrap nonzero percentages for each coefficient, with an optional quick plot.

plot.svem_model Quick actual-versus-predicted plot for a fitted model (with ROC, precision-recall, and calibration displays for binomial fits).

Deterministic wide expansions (bigexp helpers)

The `bigexp_*` helpers build and reuse a locked polynomial/interaction expansion across multiple responses and datasets:

`bigexp_terms` Build a deterministic expanded RHS (polynomials, interactions, optional partial-cubic terms) with locked factor levels and numeric ranges.

`bigexp_prepare` Coerce new data to match a stored `bigexp_spec`, including factor levels and numeric types.

`bigexp_formula` Reuse a locked expansion for another response to ensure an identical factor space across models.

`with_bigexp_contrasts` Temporarily restore the contrast options used when a `bigexp_spec` was built.

`bigexp_train` Convenience wrapper that builds a `bigexp_spec` and prepares training data in one call.

Random tables, optimization, and candidate generation

`svem_random_table_multi` Generate one shared random predictor table (with optional mixture constraints) from cached factor-space information and obtain predictions from multiple SVEM models at those points. Supports both Gaussian and binomial models; binomial predictions are returned on the probability scale. This is the lower-level sampler used by `svem_score_random`.

`svem_score_random` Random-search scoring for multiple responses with Derringer–Suich desirabilities, user weights, optional whole-model-test (WMT) reweighting, percentile CI-based uncertainty, and (optionally) scoring of existing experimental data. Returns a scored random-search table and, when `data` is supplied, an augmented copy of the original data with `<resp>_pred`, desirabilities, scores, and an `uncertainty_measure`.

`svem_select_from_score_table` Given a scored table (typically `svem_score_random()$score_table`), select one "best" row under a chosen objective and a small, diverse set of medoid candidates via PAM clustering on predictors.

`svem_export_candidates_csv` Concatenate one or more selection objects from `svem_select_from_score_table` and export candidate tables (with metadata, predictions, and response columns) to CSV or return them in-memory for inspection.

Whole-model testing and plotting

`svem_significance_test_parallel` Whole-model significance test with serial execution by default and an optional private PSOCK cluster for explicit multi-core runs. Supports mixture-constrained sampling and reuse of a locked `bigexp_spec`; designed for continuous (Gaussian) responses.

`svem_wmt_multi` Helper to run `svem_significance_test_parallel` across multiple responses and construct whole-model p-values and reweighting multipliers for use in `svem_score_random`.

`plot.svem_significance_test` Plot helper for visualizing one or more significance-test outputs (boxplots of observed vs permutation distances).

Auxiliary utilities and data

`glmnet_with_cv` Convenience wrapper around repeated `cv.glmnet()` selection for robust lambda (and optional alpha) choice.

`lipid_screen` Example dataset for multi-response modeling, whole-model testing, and mixture-constrained optimization demonstrations.

Extensions beyond the package paper

The package also includes newer methods for further study: `svem_forward` and its single-model benchmark `forward_aicc`, the bootstrap-member batch proposer `svem_thompson_batch`, and the FRW variance diagnostic `svem_ij_variance`. The opt-in `complexity = "edf"` selector in `SVEMnet` is also experimental; the published package workflow uses `complexity = "support"`. These extensions are not established by the results in the package paper. Opt-in polynomial centering in `bigexp_terms` is a later design-matrix construction option and was not evaluated in that paper.

Bootstrap member-percentile intervals describe variation among member predictions. They do not include new-observation noise and do not by themselves provide calibrated confidence or prediction intervals.

Families

SVEMnet currently supports:

- Gaussian responses (`family = "gaussian"`) with identity link and optional debiasing / bootstrap member-percentile intervals.
- Binomial responses (`family = "binomial"`) with logit link. The response must be 0/1 numeric or a two-level factor (first level treated as 0). Use `predict(..., type = "response")` for event probabilities or `type = "class"` for 0/1 labels (threshold = 0.5 by default).

Some higher-level utilities place additional constraints:

- `svem_significance_test_parallel` is designed and interpreted for continuous (Gaussian) responses.
- `svem_score_random` supports mixed Gaussian + binomial response sets, treating binomial predictions and CIs on the probability scale, but WMT-based goal reweighting (via `svem_wmt_multi` and the `wmt` argument) is only allowed when all responses are Gaussian.

Acknowledgments

Generative AI tools were used to assist with coding and roxygen documentation; all content was reviewed and finalized by the author.

Author(s)

Maintainer: Andrew T. Karl <akarl@asu.edu> (ORCID)

Authors:

- Andrew T. Karl <akarl@asu.edu> (ORCID)

References

Karl, A. T. (2026). SVEMnet: An R package for self-validated elastic-net ensembles and multi-response optimization in small-sample mixture-process experiments. *Chemometrics and Intelligent Laboratory Systems*, *271*, 105660. doi:10.1016/j.chemolab.2026.105660

Gotwalt, C., & Ramsey, P. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-Using/ev-p/849873/redirect_from_archived_page/true

Karl, A. T. (2024). A randomized permutation whole-model test heuristic for Self-Validated Ensemble Models (SVEM). *Chemometrics and Intelligent Laboratory Systems*, *249*, 105122. doi:10.1016/j.chemolab.2024.105122

Karl, A., Wisnowski, J., & Rushing, H. (2022). JMP Pro 17 Remedies for Practical Struggles with Mixture Experiments. JMP Discovery Conference. doi:10.13140/RG.2.2.34598.40003/1

Lemkus, T., Gotwalt, C., Ramsey, P., & Weese, M. L. (2021). Self-Validated Ensemble Models for Design of Experiments. *Chemometrics and Intelligent Laboratory Systems*, 219, 104439. doi:10.1016/j.chemolab.2021.104439

Xu, L., Gotwalt, C., Hong, Y., King, C. B., & Meeker, W. Q. (2020). Applications of the Fractional-Random-Weight Bootstrap. *The American Statistician*, 74(4), 345–358. doi:10.1080/00031305.2020.1731599

Ramsey, P., Gaudard, M., & Levin, W. (2021). Accelerating Innovation with Space Filling Mixture Designs, Neural Networks and SVEM. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Accelerating-Innovation-with-Space-Filling-Mixture-Designs/ev-p/756841

Ramsey, P., & Gotwalt, C. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-ev-p/849647/redirect_from_archived_page/true

Ramsey, P., Levin, W., Lemkus, T., & Gotwalt, C. (2021). SVEM: A Paradigm Shift in Design and Analysis of Experiments. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/SVEM-A-Paradigm-Shift-in-Design-and-Analysis-of-Experiments-2021/ev-p/756634

Ramsey, P., & McNeill, P. (2023). CMC, SVEM, Neural Networks, DOE, and Complexity: It's All About Prediction. *JMP Discovery Conference*.

Friedman, J. H., Hastie, T., and Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1), 1-22.

Meinshausen, N. (2007). Relaxed Lasso. Computational Statistics & Data Analysis, 52(1), 374-393.

Kish, L. (1965). *Survey Sampling*. Wiley.

Lumley, T. (2004). Analysis of complex survey samples. *Journal of Statistical Software*, 9(1), 1–19.

Lumley, T. and Scott, A. (2015). AIC and BIC for modelling with complex survey data. *Journal of Survey Statistics and Methodology*, 3(1), 1–18.

Tibshirani, R. J. and Taylor, J. (2012). Degrees of freedom in lasso problems. *The Annals of Statistics*, 40(2), 1198–1232. doi:10.1214/12AOS1003

See Also

Useful links:

- doi:10.1016/j.chemolab.2026.105660
- https://arxiv.org/abs/2511.20968

`bigexp_formula` *Construct a formula for a new response using a bigexp_spec*

Description

bigexp_formula() lets you reuse an existing expansion spec for multiple responses. It keeps the right hand side locked but changes the response variable on the left hand side.

Usage

```
bigexp_formula(spec, response)
```

Arguments

`spec`	A "bigexp_spec" object created by bigexp_terms().
`response`	Character scalar giving the name of the new response column in your data. If omitted, the original formula is returned unchanged.

Details

This is useful when you want to fit separate models for several responses on the same factor space while guaranteeing that they all use exactly the same design columns and coding.

Value

A formula of the form `response ~ rhs`, where the right-hand side is taken from the locked expansion stored in `spec`.

Examples

```
set.seed(1)
df2 <- data.frame(
  y1 = rnorm(10),
  y2 = rnorm(10),
  X1 = rnorm(10),
  X2 = rnorm(10)
)

spec2 <- bigexp_terms(
  y1 ~ X1 + X2,
  data             = df2,
  factorial_order  = 2,
```

```
  polynomial_order = 2
)

f2 <- bigexp_formula(spec2, "y2")
f2
```

`bigexp_prepare` *Prepare data to match a* `bigexp_spec`

Description

`bigexp_prepare()` coerces a new data frame so that it matches a previously built `bigexp_terms` spec. It:

- applies the locked factor levels for categorical predictors,
- enforces that continuous variables remain numeric (and errors if they are not), and
- optionally warns about or errors on unseen factor levels.

Usage

```
bigexp_prepare(spec, data, unseen = c("warn_na", "error"))
```

Arguments

`spec`	Object returned by `bigexp_terms`.
`data`	New data frame (for example, training, test, or future batches).
`unseen`	How to handle unseen factor levels in `data`: `"warn_na"` (default) maps unseen levels to `NA` and issues a warning, or `"error"` stops with an error if any unseen levels are encountered.

Details

Columns that are not listed in `spec$vars` (for example, the response or extra metadata columns) are left unchanged.

The goal is that `model.matrix(spec$formula, data)` will produce the same set of columns in the same order across all datasets prepared with the same spec, even if some levels are missing in a particular batch.

Value

A list with two elements:

- `formula`: the expanded formula stored in the spec (same as `spec$formula`).
- `data`: a copy of the input data with predictor columns coerced to match the spec (types and levels), suitable for `model.frame()` / `model.matrix()`.

See Also

bigexp_terms

Examples

```
set.seed(1)
train <- data.frame(
  y  = rnorm(10),
  X1 = rnorm(10),
  X2 = rnorm(10),
  G  = factor(sample(c("A", "B"), 10, replace = TRUE))
)

spec <- bigexp_terms(
  y ~ X1 + X2 + G,
  data             = train,
  factorial_order  = 2,
  polynomial_order = 2
)

newdata <- data.frame(
  y  = rnorm(5),
  X1 = rnorm(5),
  X2 = rnorm(5),
  G  = factor(sample(c("A", "B"), 5, replace = TRUE))
)

prep <- bigexp_prepare(spec, newdata)
str(prep$data)
```

---

bigexp_terms *Create a deterministic expansion spec for wide polynomial and interaction models*

Description

`bigexp_terms()` builds a specification object that:

- decides which predictors are treated as continuous or categorical,
- optionally treats selected variables as blocking factors that enter the model only additively and never in interactions or polynomials,
- locks factor levels from the supplied data,
- records the contrast settings used when the model matrix is first built, and
- constructs a reusable right-hand side (RHS) expression string for a large expansion that can be shared across multiple responses and datasets.

Usage

```
bigexp_terms(
  formula,
  data,
  factorial_order = 3L,
  polynomial_order = 3L,
  include_pc_2way = TRUE,
  include_pc_3way = FALSE,
  intercept = TRUE,
  blocking = NULL,
  discrete_numeric = NULL,
  audit = c("warn", "error", "none"),
  audit_numeric_rate = 0.9,
  audit_unique_ratio = 0.8,
  audit_min_n = 12L,
  report = TRUE,
  center_polynomials = FALSE,
  mixture_vars = NULL
)
```

Arguments

formula	Main-effects formula of the form `y ~ X1 + X2 + G` or `y ~ .`. The right-hand side should contain main effects only; do not include `:` (interactions), `^` (factorial shortcuts), `I()` powers, transformed terms such as `log(X1)`, or inline polynomial expansions. The helper will generate interactions and polynomial terms automatically. Dot exclusions such as `y ~ . - X1` are honored. Note that with `y ~ .` every non-response column of `data` becomes a predictor, including any other response columns present in the data frame; list the main effects explicitly (or use exclusions) when the data holds several responses.
data	Data frame used to decide types and lock factor levels.
factorial_order	Integer >= 1. Maximum order of factorial interactions among the non-blocking main effects. For example, 1 gives main effects only, 2 gives up to two-way interactions, 3 gives up to three-way interactions, and so on.
polynomial_order	Integer >= 1. Maximum polynomial degree for continuous non-blocking predictors. A value of 1 means only linear terms; 2 adds squares `I(X^2)`; 3 adds cubes `I(X^3)`; in general, all powers `I(X^k)` for `k` from 2 up to `polynomial_order` are added.
include_pc_2way	Logical. If `TRUE` (default), `polynomial_order` >= 2, and `factorial_order` >= 2, include partial-cubic two-way terms of the form `Z:I(X^2)` where `X` is continuous and `Z` is another non-blocking predictor. (These are cross terms, so `factorial_order = 1` — "main effects only" — suppresses them.)
include_pc_3way	Logical. If `TRUE`, `polynomial_order` >= 2, and `factorial_order` >= 3, in-

clude partial-cubic three-way terms `I(X^2):Z:W` among non-blocking predictors.

`intercept` Logical. If `TRUE` (default), include an intercept in the expansion; if `FALSE`, the generated RHS drops the intercept. The no-intercept form is available for standalone design-matrix workflows; `SVEMnet()` and `glmnet_with_cv()` require the intercept convention used by the published SVEM methods and reject such a specification.

`blocking` Optional character vector of column names in `data` to treat as blocking factors. These variables are included in the spec and typed like other predictors (categorical vs continuous), but they enter the model only as additive main effects and never appear in interactions, polynomials, or partial-cubic terms. **Important:** when using `y ~ .`, blocking variables are automatically excluded from the "non-blocking" predictor set so they do not trigger a conflict error. When using an explicit RHS (for example `y ~ X1 + X2`), blocking variables must not also be explicitly listed on the right-hand side.

`discrete_numeric`

Optional specification of "discrete numeric" predictors for downstream sampling (for example in `svem_random_table_multi()`). These predictors are still treated as numeric for modeling and expansion (that is, they remain continuous in the design matrix and may participate in polynomial and interaction terms). This option only records a finite set of preferred numeric levels to be used when randomly generating recipes. Supply either:

- a character vector of predictor names, in which case the allowed levels are inferred as the sorted unique finite values observed in `data`; or
- a named list mapping predictor names to numeric vectors of allowed levels. If an entry is `NULL` or length zero, levels are inferred from `data` for that predictor.

`audit` How to handle suspicious typing / high-cardinality issues when building the spec. One of `"warn"` (default), `"error"`, or `"none"`. Audits cover numeric-like character/factor columns (including percent strings like `"25%"`), and very high-cardinality categorical predictors that are likely IDs or mis-typed numerics.

`audit_numeric_rate`

Numeric in (0,1). If at least this fraction of non-missing values parse as numeric (after stripping commas and an optional trailing `%`), the column is flagged as numeric-like when stored as character/factor.

`audit_unique_ratio`

Numeric in (0, 1). For categorical predictors, warn/error if `unique(non-missing) / n_nonmissing >= audit_unique_ratio`.

`audit_min_n` Integer >= 1. Minimum number of non-missing values required before audits are applied.

`report` Logical. If `TRUE` (default), print a compact summary of the inferred predictor types and settings (via `print.bigexp_spec`) when `bigexp_terms()` returns.

`center_polynomials`

Logical; default `FALSE`. If `TRUE`, subtract each eligible predictor's fixed training-data mean before raising it to a power. Applies to numeric, non-blocking predictors except those in `mixture_vars`. See *Centered polynomial powers*.

`mixture_vars` Optional character vector naming numeric, non-blocking predictors whose polynomial powers must remain uncentered. This only excludes them from polynomial centering; it does not impose mixture sum constraints or replace `mixture_groups` in downstream sampling.

Details

The expansion for non-blocking predictors can include:

- full factorial interactions among the listed main effects, up to a chosen order;
- polynomial terms `I(X^k)` for continuous predictors up to a chosen degree; and
- optional partial-cubic interactions of the form `Z:I(X^2)` and `I(X^2):Z:W`.

Predictor types are inferred from `data`:

- factors, characters, and logicals are treated as categorical;
- all other numeric predictors are treated as continuous, and their observed ranges are stored.

Variables listed in `blocking` are included in the spec and are typed using the same rules as other predictors (for example, a numeric blocking variable with many distinct values is treated as continuous). However, blocking variables enter the model only as additive main effects, without interactions or polynomial terms, regardless of `factorial_order` or `polynomial_order`.

Once built, a `"bigexp_spec"` can be reused to create consistent expansions for new datasets via `bigexp_prepare` and `bigexp_formula`. The RHS and contrast settings are locked, so the same spec applied to different data produces design matrices with the same columns in the same order (up to missing levels for specific batches).

Value

An object of class `"bigexp_spec"` with components:

- `formula`: expanded formula of the form `y ~ <big expansion>`, using the response from the input formula.
- `rhs`: right-hand side expansion string (reusable for any response).
- `vars`: character vector of predictor names (including blocking variables) in the order discovered from the formula and data.
- `is_cat`: named logical vector indicating which predictors are treated as categorical (`TRUE`) versus continuous (`FALSE`).
- `levels`: list of locked factor levels for categorical predictors.
- `num_range`: 2 x p numeric matrix of ranges for continuous variables (rows `c("min","max")`).
- `settings`: list of expansion settings, including `factorial_order`, `polynomial_order`, `include_pc_2way`, `include_pc_3way`, `intercept`, `blocking`, and stored contrast information. When polynomial centering or mixture exclusions are requested, also records `center_polynomials`, the named numeric vector `polynomial_centers`, and `mixture_vars`.

Centered polynomial powers

With `center_polynomials = TRUE`, a power `I(X^k)` is replaced by `I((X - c)^k)`, where `c` is the arithmetic mean of the finite values of `X` in the data supplied to `bigexp_terms()`. The same replacement is used wherever that power occurs in a partial-cubic term. Main effects, ordinary factorial interactions, and the non-powered factors in partial-cubic terms remain on their original scales.

Each center is computed separately from its predictor, without excluding rows because another predictor or response is missing. This keeps the expansion independent of the response used to build the spec. The constants are recorded in `spec$settings$polynomial_centers` and embedded in the formula; `bigexp_prepare()`, `bigexp_formula()`, and prediction reuse them without recentering new data. Predictor ranges, discrete sampling levels, and supplied data values are unchanged. Only design-matrix construction changes; fitting algorithms are unchanged, although selecting or penalizing terms in the new basis can change the fitted model.

Predictors named in `mixture_vars` retain raw powers. Mixture membership is explicit and is not inferred from names, ranges, or observed sums.

Typical workflow

In a typical multi-response workflow you:

1. Call `bigexp_terms()` once on your training data to build and lock the expansion (types, levels, contrasts, RHS).
2. Fit models using `spec$formula` and the original data (for example, `SVEMnet(spec$formula, data, ...)` or `lm(spec$formula, data)`).
3. For new batches, call `bigexp_prepare` with the same `spec` so that design matrices have exactly the same columns and coding.
4. For additional responses on the same factor space, use `bigexp_formula` to swap the left-hand side while reusing the locked expansion.

See Also

`bigexp_prepare`, `bigexp_formula`, `bigexp_train`

Examples

```
## Example 1: small design with one factor
set.seed(1)
df <- data.frame(
  y  = rnorm(20),
  X1 = rnorm(20),
  X2 = rnorm(20),
  G  = factor(sample(c("A", "B"), 20, replace = TRUE))
)

## Two-way interactions and up to cubic terms in X1 and X2
spec <- bigexp_terms(
  y ~ X1 + X2 + G,
  data             = df,
  factorial_order  = 2,
  polynomial_order = 3
```

```
)

print(spec)

## Optional centering of powers using fixed training-data means
spec_centered <- bigexp_terms(
  y ~ X1 + X2 + G, data = df,
  factorial_order = 2, polynomial_order = 2,
  center_polynomials = TRUE
)
spec_centered$settings$polynomial_centers
## For mixture-process data, also supply mixture_vars = c("A", "B", "C")
## to keep the named mixture components' powers on their raw scales.

## Example 2: pure main effects (no interactions, no polynomial terms)
spec_main <- bigexp_terms(
  y ~ X1 + X2 + G,
  data             = df,
  factorial_order  = 1,  # main effects only
  polynomial_order = 1   # no I(X^2) or higher
)

## Example 3: blocking factors (categorical and continuous)
set.seed(2)
df_block <- data.frame(
  y           = rnorm(30),
  X1          = rnorm(30),
  X2          = rnorm(30),
  G           = factor(sample(c("A", "B"), 30, replace = TRUE)),
  Operator    = factor(sample(paste0("Op", 1:3), 30, replace = TRUE)),
  AmbientTemp = rnorm(30, mean = 22, sd = 2)  # continuous blocking covariate
)

## Here Operator (categorical) and AmbientTemp (continuous) are blocking factors:
## they enter additively, but do not appear in interactions or polynomials.
spec_block <- bigexp_terms(
  y ~ X1 + X2 + G,
  data             = df_block,
  factorial_order  = 2,
  polynomial_order = 3,
  blocking         = c("Operator", "AmbientTemp")
)

print(spec_block)
spec_block$rhs

## Example 4: discrete numeric predictors (finite numeric support)
## A common case is a numeric process setting that only takes a small set
## of allowed values (e.g., 0.5, 1, 2, 4). Use `discrete_numeric` in
## bigexp_terms() so downstream sampling respects those levels automatically.
set.seed(3)
D_allowed <- c(0.5, 1, 2, 4)
df_disc <- data.frame(
```

```
  y  = rnorm(40),
  D  = sample(D_allowed, 40, replace = TRUE),   # numeric with discrete support
  X1 = rnorm(40),
  G  = factor(sample(c("A", "B"), 40, replace = TRUE))
)

# Record that D should be treated as "discrete numeric" for downstream sampling.
# Levels are inferred automatically from the training data.
spec_disc <- bigexp_terms(
  y ~ D + X1 + G,
  data             = df_disc,
  factorial_order  = 2,
  polynomial_order = 2,
  discrete_numeric = "D"
)


# Fit. The discrete support is retained in fit$sampling_schema.
fit_disc <- SVEMnet(spec_disc, df_disc, nBoot = 8,
                    glmnet_alpha = 1, relaxed = FALSE)

# Score random candidates; sampled D values stay in D_allowed
scored <- svem_score_random(
  objects         = list(y = fit_disc),
  goals           = list(y = list(goal = "max", weight = 1)),
  n               = 200,
  numeric_sampler = "random",
  verbose         = FALSE
)

table(scored$score_table$D)
stopifnot(all(scored$score_table$D %in% D_allowed))
```

`bigexp_train` *Build a spec and prepare training data in one call*

Description

bigexp_train() is a convenience wrapper around `bigexp_terms` and `bigexp_prepare`. It:

- builds a deterministic expansion spec from the training data; and
- immediately prepares that same data to match the locked types and levels.

Usage

```
bigexp_train(formula, data, ...)
```

Arguments

formula	Main-effects formula such as y ~ X1 + X2 + G or y ~ . . Only main effects should appear on the right hand side.
data	Training data frame used to lock types and levels.
...	Additional arguments forwarded to bigexp_terms(), such as factorial_order, polynomial_order, include_pc_2way, include_pc_3way, and intercept.

Details

This is handy when you want a single object that contains both the spec and the training data in a form that is ready to pass into a modeling function. For more control, you can call bigexp_terms() and bigexp_prepare() explicitly instead.

Value

An object of class "bigexp_train" which is a list with components:

- spec: the "bigexp_spec" object returned by bigexp_terms().
- formula: the expanded formula spec$formula.
- data: the prepared training data (predictors coerced to match spec), suitable for passing directly to modeling functions such as lm(), glm(), or SVEMnet().

Examples

```
set.seed(1)
df5 <- data.frame(
  y  = rnorm(20),
  X1 = rnorm(20),
  X2 = rnorm(20)
)

tr <- bigexp_train(
  y ~ X1 + X2,
  data             = df5,
  factorial_order  = 2,
  polynomial_order = 3
)

## Prepared training data and expanded formula:
str(tr$data)
tr$formula

## Example: fit a model using the expanded formula
fit_lm <- lm(tr$formula, data = tr$data)
summary(fit_lm)
```

coef.svem_model *Coefficients for SVEM Models*

Description

Extracts averaged coefficients from an svem_model fitted by SVEMnet.

Usage

```
## S3 method for class 'svem_model'
coef(object, debiased = FALSE, ...)
```

Arguments

object	An object of class svem_model, typically returned by SVEMnet.
debiased	Logical; if TRUE and debiased coefficients are available (Gaussian fits with parms_debiased), return those instead of parms. Default is FALSE.
...	Unused; present for S3 method compatibility.

Details

For Gaussian fits, you can optionally request debiased coefficients (if they were computed and stored) via debiased = TRUE. In that case, the function returns object$parms_debiased. If debiased coefficients are not available, or if debiased = FALSE, the function returns object$parms, which are the ensemble-averaged coefficients across bootstrap members.

For Binomial models, debiased is ignored and the averaged coefficients in object$parms are returned.

This is a lightweight accessor around the stored components of an svem_model:

- parms: ensemble-averaged coefficients over bootstrap members, on the model's link scale;
- parms_debiased: optional debiased coefficients (Gaussian only), if requested at fit time.

Passing debiased = TRUE has no effect if parms_debiased is NULL.

Value

A named numeric vector of coefficients (including the intercept).

See Also

svem_nonzero for bootstrap nonzero percentages and a quick stability plot.

Examples

```
set.seed(1)
n  <- 50
x1 <- rnorm(n)
x2 <- rnorm(n)
eps <- rnorm(n, sd = 0.3)
y_g <- 1 + 2*x1 - 0.5*x2 + eps
dat_g <- data.frame(y_g, x1, x2)

fit_g <- SVEMnet(y_g ~ x1 + x2, data = dat_g,
                 nBoot = 10, glmnet_alpha = 1, relaxed = FALSE)

# Ensemble-averaged coefficients
cc <- coef(fit_g)
head(cc)

# Debiased (only if available for Gaussian fits)
ccd <- coef(fit_g, debiased = TRUE)
head(ccd)
```

`forward_aicc` *Forward Selection by AICc (or AIC / BIC)*

Description

Deterministic greedy forward selection over whole model terms for Gaussian responses, minimizing a least-squares information criterion. This is a single-model benchmark companion to `SVEMnet` (in the same spirit as `glmnet_with_cv`): it shares SVEMnet's formula and `bigexp_terms()` front end and returns an object compatible with `predict()`, `coef()`, and the downstream scoring tools, but fits one deterministic model instead of an SVEM ensemble.

Usage

```
forward_aicc(
  formula,
  data,
  criterion = c("AICc", "AIC", "BIC"),
  response = NULL,
  unseen = c("warn_na", "error")
)
```

Arguments

`formula`	A model formula or a `bigexp_spec` created by `bigexp_terms()`, exactly as in `SVEMnet`.
`data`	A data frame containing the variables in the model.
`criterion`	Character. One of `"AICc"` (default), `"AIC"`, or `"BIC"`.

`response` Optional character. When `formula` is a `bigexp_spec`, names the response column to use on the left-hand side. Defaults to the response stored in the spec.

`unseen` How to treat factor levels not seen in the original `bigexp_spec`: `"warn_na"` (default) or `"error"`.

Details

This benchmark accompanies the newer `svem_forward` extension. The core elastic-net ensemble workflow in Karl (2026), doi:10.1016/j.chemolab.2026.105660, is implemented by `SVEMnet`.

Starting from the intercept-only model, each step evaluates every remaining model term (all columns of a multi-column term, such as a factor's contrast block, enter together), refits by least squares, and adds the term whose augmented model minimizes the criterion. Selection stops when no candidate improves the criterion, when all remaining candidates are collinear with the current model, or when the criterion is undefined for every candidate (for example, AICc requires $n - K - 1 > 0$). Ties are broken by model-matrix term order.

The criteria are computed up to the additive Gaussian constant as

- AICc: $n \log(RSS/n) + 2K + 2K(K+1)/(n-K-1)$,
- AIC: $n \log(RSS/n) + 2K$,
- BIC: $n \log(RSS/n) + K \log(n)$,

where $K = p + 1$ counts the fitted regression coefficients (including the intercept) plus the residual variance. Candidate models that are rank deficient given the current path, or whose criterion is undefined, are skipped.

Only Gaussian responses are supported. The fit is deterministic: no random weights are drawn and repeated calls return identical results. Because `coef_matrix` has one row, `predict(..., se.fit = TRUE)` and `interval = TRUE` are unavailable; use `svem_forward` for ensemble uncertainty with forward-selection base learners.

Value

An object of classes `"svem_forward_ic"` and `"svem_model"`, structured like an `SVEMnet` fit so that `predict()`, `coef()`, and `plot()` work unchanged (`coef_matrix` has a single row, so bootstrap uncertainty options are unavailable). Additional components:

- `criterion`: the criterion used.
- `criterion_value`: its value at the selected model.
- `selected_terms`: character vector of selected term labels, in selection order.
- `selection_path`: data frame with one row per accepted step (`step`, `term`, `k`, `rss`, `criterion`, `improvement`).

See Also

`svem_forward`, `SVEMnet`, `glmnet_with_cv`

Examples

```
set.seed(1)
n  <- 40
X1 <- rnorm(n); X2 <- rnorm(n); X3 <- rnorm(n)
y  <- 1 + 2 * X1 - 1.5 * X2 + rnorm(n, 0, 0.3)
dat <- data.frame(y, X1, X2, X3)

fit <- forward_aicc(y ~ (X1 + X2 + X3)^2, dat)
fit$selected_terms
fit$selection_path
preds <- predict(fit, dat)

## bigexp_spec front end, as in SVEMnet()
spec <- bigexp_terms(y ~ X1 + X2 + X3, dat, factorial_order = 2)
fit2 <- forward_aicc(spec, dat)
```

`glmnet_with_cv` *Fit a glmnet Model with Repeated Cross-Validation*

Description

Repeated K-fold cross-validation over per-alpha lambda paths, with a combined 1-SE rule across repeats. Preserves fields expected by `predict.svem_model()` and internal prediction helpers. When `relaxed = TRUE`, the complete relaxed cross-validation surfaces are aggregated across repeats and alpha, lambda, and gamma are selected jointly. This mirrors `cv.glmnet(relax = TRUE)` rather than choosing lambda from the ordinary (gamma = 1) curve before choosing gamma.

Usage

```
glmnet_with_cv(
  formula,
  data,
  glmnet_alpha = c(0.5, 1),
  standardize = TRUE,
  nfolds = 10,
  repeats = 5,
  choose_rule = c("min", "1se"),
  seed = NULL,
  exclude = NULL,
  relaxed = FALSE,
  relax_gamma = NULL,
  family = c("gaussian", "binomial"),
  ...
)
```

Arguments

formula	Model formula.
data	Data frame containing the variables in the model.
glmnet_alpha	Numeric vector of Elastic Net mixing parameters (alphas) in [0,1]; default c(0.5, 1). When relaxed = TRUE, any alpha = 0 (ridge) is dropped with a warning.
standardize	Logical passed to glmnet() and cv.glmnet() (default TRUE).
nfolds	Requested number of CV folds (default 10). Internally constrained so that there are at least about 3 observations per fold and at least 5 folds when possible.
repeats	Number of independent CV repeats (default 5). Each repeat reuses the same folds across all alphas for paired comparisons.
choose_rule	Character; how to choose the tuning point within each alpha: • "min": tuning point minimizing the cross-validated criterion. • "1se": largest lambda (then largest gamma for relaxed fits) within 1 combined SE of the minimum, where the SE includes both within- and between-repeat variability. Default is "min". In small-mixture simulations, the 1-SE rule tended to increase RMSE on held-out data, so "min" is used as the default here.
seed	Optional integer seed for reproducible fold IDs (and the ridge fallback, if used).
exclude	Optional vector or function for glmnet's exclude= argument. If a function, cv.glmnet() applies it inside each training fold (requires glmnet >= 4.1-2). The callback receives the original design columns, without any internal compatibility column, and the current fit's response and weights. Numerically constant columns remain excluded. When applying this mask or handling a one-column design, callback returns are checked against the original column indices (or an aligned logical vector); other callbacks retain glmnet's validation.
relaxed	Logical; if TRUE, call glmnet() and cv.glmnet() with relax = TRUE and optionally a gamma path (default FALSE). The repeated CV criterion is aggregated over the full lambda-by-gamma surface and both parameters are selected jointly. If cv.glmnet(relax = TRUE) fails for a particular repeat/alpha, the function retries that fit without relaxation; incomplete non-relaxed fallback surfaces are not mixed into a successful relaxed search. The number of fallbacks is recorded in meta$relax_cv_fallbacks.
relax_gamma	Optional numeric vector passed as gamma= to glmnet() and cv.glmnet() when relaxed = TRUE. If NULL, glmnet's internal default gamma grid is used.
family	Model family: either "gaussian" or "binomial", or the corresponding stats::gaussian() or stats::binomial() family objects with canonical links. For Gaussian, y must be numeric. For binomial, y must be 0/1 numeric, logical, or a factor with exactly 2 levels (the second level is treated as 1). Non-canonical links are not supported.
...	Additional arguments forwarded to both cv.glmnet() and glmnet(), for example: parallel, type.measure, intercept, lower.limits, upper.limits, penalty.factor, standardize.response, keep, and so on. Any user-supplied

weights or offset are ignored with a warning: glmnet_with_cv() returns objects meant for side-by-side comparison with SVEMnet() fits, whose weighting is controlled by the SVEM scheme, and downstream prediction helpers do not carry observation weights or offsets. Glmnet algorithm-control values such as maxit, thresh, dfmax, and pmax may be supplied directly for compatibility or via control = list(...) with glmnet versions that support it; they are routed in a version-compatible way when possible. Argument names that the installed glmnet does not recognize are ignored with a warning (misspelling protection).

Details

This function is a convenience wrapper around glmnet() and cv.glmnet() that returns an object in the same structural format as SVEMnet() (class "svem_model"). It is intended for:

- direct comparison of standard cross-validated glmnet fits to SVEMnet models using the same prediction and schema tools, or
- users who want a repeated-cv.glmnet() workflow without any SVEM weighting or bootstrap ensembling.

It is not called internally by the SVEM bootstrap routines.

The basic workflow is:

1. Generate one set of CV fold IDs per repeat. Reuse each repeat's folds across all values in glmnet_alpha.
2. For each alpha, run repeats cv.glmnet() fits, align the lambda paths and, for relaxed fits, the gamma paths, and aggregate the CV surfaces.
3. At each lambda-gamma point, compute a combined SE that accounts for both within-repeat and between-repeat variability.
4. Apply choose_rule ("min" or "1se") to select lambda and gamma for that alpha, then choose the best alpha by comparing these per-alpha scores.

Special cases and fallbacks:

- If there are no predictors after model.matrix() (an intercept-only model), the function returns an intercept-only fit without calling glmnet(), along with a minimal schema for safe prediction.
- If all cv.glmnet() attempts fail for every alpha (a rare edge case), the function falls back to a manual ridge (alpha = 0) CV search over a fixed lambda grid and returns the best ridge solution. For Gaussian models this search uses a mean-squared-error criterion; for binomial models it uses a negative log-likelihood (deviance-equivalent) criterion.

Family-specific behavior:

- For the Gaussian family, an optional calibration lm(y ~ y_pred) is fit on the training data (when there is sufficient variation), and both y_pred and y_pred_debiased are stored.
- For the binomial family, y_pred is always on the probability (response) scale and debiasing is not applied. Both the primary cross-validation and any ridge fallback use deviance-style criteria (binomial negative log-likelihood) rather than squared error.

Design-matrix schema and contrasts:

- The training terms are stored with a compact evaluation environment containing required formula transformations.
- Factor and character levels are recorded in xlevels for safe prediction.
- Per-factor contrasts are stored in contrasts, normalized so that any contrasts recorded as character names are converted back to contrast functions at prediction time.

The returned object inherits classes "svem_cv" and "svem_model" and is designed to be compatible with SVEMnet prediction and schema utilities. It is a standalone, standard glmnet CV workflow that does not use SVEM-style bootstrap weighting or ensembling.

Value

A list of class c("svem_cv","svem_model") with elements:

- parms Named numeric vector of coefficients (including "(Intercept)").
- glmnet_alpha Numeric vector of alphas searched.
- best_alpha Numeric; winning alpha.
- best_lambda Numeric; winning lambda.
- best_gamma Numeric; winning relaxed mixing parameter, or 1 for a non-relaxed fit. It is NA for structural intercept-only fits.
- y_pred In-sample predictions from the returned coefficients (fitted values for Gaussian; probabilities for binomial).
- debias_fit For Gaussian, an optional lm(y ~ y_pred) calibration model; NULL otherwise.
- y_pred_debiased If debias_fit exists, its fitted values; otherwise NULL.
- cv_summary Named list (one element per alpha) of data frames with columns gamma, lambda, mean_cvm, sd_cvm, se_combined, n_repeats, idx_min, idx_1se.
- formula Original modeling formula.
- terms Training terms object with a compact, serializable formula-evaluation environment.
- training_X Training design matrix (without intercept column).
- actual_y Training response vector used for glmnet: numeric y for Gaussian, or 0/1 numeric y for binomial.
- xlevels Factor and character levels seen during training (for safe prediction).
- contrasts Contrasts used for factor predictors during training.
- schema List list(feature_names, terms_str, xlevels, contrasts, terms_hash) for deterministic prediction.
- note Character vector of notes (for example, dropped rows, intercept-only path, ridge fallback, relaxed-coefficient source).
- meta List with fields such as nfolds, repeats, rule, family, relaxed, relax_cv_fallbacks, and cv_object (the final cv.glmnet() object when relaxed = TRUE and keep = TRUE, otherwise NULL).
- diagnostics List of simple diagnostics for the selected model, currently including:
  - k_final: number of coefficients estimated as nonzero *including* the intercept.
  - k_final_no_intercept: number of nonzero slope coefficients (excludes the intercept).
- family Character scalar giving the resolved family ("gaussian" or "binomial"), mirroring meta$family.

Acknowledgments

Generative AI tools were used to assist with coding and roxygen documentation; all content was reviewed and finalized by the author.

References

Karl, A. T. (2026). SVEMnet: An R package for self-validated elastic-net ensembles and multi-response optimization in small-sample mixture-process experiments. *Chemometrics and Intelligent Laboratory Systems*, *271*, 105660. doi:10.1016/j.chemolab.2026.105660

Gotwalt, C., & Ramsey, P. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-Using/ev-p/849873/redirect_from_archived_page/true

Karl, A. T. (2024). A randomized permutation whole-model test heuristic for Self-Validated Ensemble Models (SVEM). *Chemometrics and Intelligent Laboratory Systems*, *249*, 105122. doi:10.1016/j.chemolab.2024.105122

Karl, A., Wisnowski, J., & Rushing, H. (2022). JMP Pro 17 Remedies for Practical Struggles with Mixture Experiments. JMP Discovery Conference. doi:10.13140/RG.2.2.34598.40003/1

Lemkus, T., Gotwalt, C., Ramsey, P., & Weese, M. L. (2021). Self-Validated Ensemble Models for Design of Experiments. *Chemometrics and Intelligent Laboratory Systems*, 219, 104439. doi:10.1016/j.chemolab.2021.104439

Xu, L., Gotwalt, C., Hong, Y., King, C. B., & Meeker, W. Q. (2020). Applications of the Fractional-Random-Weight Bootstrap. *The American Statistician*, 74(4), 345–358. doi:10.1080/00031305.2020.1731599

Ramsey, P., Gaudard, M., & Levin, W. (2021). Accelerating Innovation with Space Filling Mixture Designs, Neural Networks and SVEM. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Accelerating-Innovation-with-Space-Filling-Mixture-Designs/ev-p/756841

Ramsey, P., & Gotwalt, C. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-ev-p/849647/redirect_from_archived_page/true

Ramsey, P., Levin, W., Lemkus, T., & Gotwalt, C. (2021). SVEM: A Paradigm Shift in Design and Analysis of Experiments. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/SVEM-A-Paradigm-Shift-in-Design-and-Analysis-of-Experiments-2021/ev-p/756634

Ramsey, P., & McNeill, P. (2023). CMC, SVEM, Neural Networks, DOE, and Complexity: It's All About Prediction. *JMP Discovery Conference*.

Friedman, J. H., Hastie, T., and Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1), 1-22.

Meinshausen, N. (2007). Relaxed Lasso. Computational Statistics & Data Analysis, 52(1), 374-393.

Kish, L. (1965). *Survey Sampling*. Wiley.

Lumley, T. (2004). Analysis of complex survey samples. *Journal of Statistical Software*, 9(1), 1–19.

Lumley, T. and Scott, A. (2015). AIC and BIC for modelling with complex survey data. *Journal of Survey Statistics and Methodology*, 3(1), 1–18.

Tibshirani, R. J. and Taylor, J. (2012). Degrees of freedom in lasso problems. *The Annals of Statistics*, 40(2), 1198–1232. doi:10.1214/12AOS1003

Examples

```
set.seed(123)
n <- 60; p <- 6
X <- matrix(rnorm(n * p), n, p)
beta <- c(1, -1, rep(0, p - 2))
y <- as.numeric(X %*% beta + rnorm(n))
df_ex <- data.frame(y = y, X)
colnames(df_ex) <- c("y", paste0("x", 1:p))

# Gaussian example, v1-like behavior: choose_rule = "min"
fit_min <- glmnet_with_cv(
  y ~ ., df_ex,
  glmnet_alpha = 1,
  nfolds = 5,
  repeats = 1,
  choose_rule = "min",
  seed = 42,
  family = "gaussian"
)

# Gaussian example, relaxed path with gamma search
fit_relax <- glmnet_with_cv(
  y ~ ., df_ex,
  glmnet_alpha = 1,
  nfolds = 5,
  repeats = 1,
  relaxed = TRUE,
  seed = 42,
  family = "gaussian"
)

# Binomial example (numeric 0/1 response)
set.seed(456)
n2 <- 80; p2 <- 5
X2 <- matrix(rnorm(n2 * p2), n2, p2)
beta2 <- c(1.0, -1.5, rep(0, p2 - 2))
linpred <- as.numeric(X2 %*% beta2)
prob <- plogis(linpred)
y_bin <- rbinom(n2, size = 1, prob = prob)
df_bin <- data.frame(y = y_bin, X2)
colnames(df_bin) <- c("y", paste0("x", 1:p2))

fit_bin <- glmnet_with_cv(
  y ~ ., df_bin,
  glmnet_alpha = 1,
  nfolds = 5,
```

```
  repeats = 1,
  seed = 99,
  family = "binomial"
)
```

`lipid_screen` *Lipid formulation screening data*

Description

An example dataset for modeling Potency, Size, and PDI as functions of formulation and process settings. Percent composition columns are stored as proportions in `[0, 1]` (for example, 4.19 percent is 0.0419). This table is intended for demonstration of SVEMnet multi-response modeling, desirability-based random-search optimization, and probabilistic design-space construction.

Usage

```
data(lipid_screen)
```

Format

A data frame with one row per experimental run and the following columns:

RunID character. Optional identifier for each run.

PEG numeric. Proportion (0-1).

Helper numeric. Proportion (0-1).

Ionizable numeric. Proportion (0-1).

Cholesterol numeric. Proportion (0-1).

Ionizable_Lipid_Type factor. Categorical identity of the ionizable lipid.

N_P_ratio numeric. Molar or mass $N : P$ ratio (unitless).

flow_rate numeric. Process flow rate (arbitrary units).

Operator factor. Categorical blocking factor.

Potency numeric. Response (for example, normalized activity).

Size numeric. Response (for example, particle size in nm).

PDI numeric. Response (polydispersity index).

Notes character. Optional free-text notes.

Details

The four composition columns `PEG`, `Helper`, `Ionizable`, and `Cholesterol` are stored as proportions in `[0,1]` and sum to exactly 1 in every row (so the mixture columns are exactly collinear), making them natural candidates for mixture constraints in optimization and design-space examples.

This dataset accompanies examples showing:

- fitting three SVEM models (Potency, Size, PDI) on a shared expanded factor space via `bigexp_terms` and `bigexp_formula`,
- random design generation using SVEM random-table helpers (for use with multi-response optimization),
- multi-response scoring and candidate selection with `svem_score_random` (Derringer–Suich desirabilities, weights, uncertainty) and `svem_select_from_score_table` (optimal and high-uncertainty medoid candidates),
- returning both high-score optimal candidates and high-uncertainty exploration candidates from the same scored table by changing the target column (for example `score` vs `uncertainty_measure` or `wmt_score`),
- optional whole-model reweighting (WMT) of response weights via `svem_wmt_multi` (for p-values and multipliers) together with `svem_score_random` (via its `wmt` argument),
- constructing a probabilistic design space in one step by passing process-mean specifications via the `specs` argument of `svem_score_random` (internally using `svem_append_design_space_cols` when needed).

Acknowledgments

Generative AI tools were used to assist with coding and roxygen documentation; all content was reviewed and finalized by the author.

Source

Simulated screening table supplied by the package author.

References

Karl, A. T. (2026). SVEMnet: An R package for self-validated elastic-net ensembles and multi-response optimization in small-sample mixture-process experiments. *Chemometrics and Intelligent Laboratory Systems*, *271*, 105660. doi:10.1016/j.chemolab.2026.105660

Gotwalt, C., & Ramsey, P. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-Using/ev-p/849873/redirect_from_archived_page/true

Karl, A. T. (2024). A randomized permutation whole-model test heuristic for Self-Validated Ensemble Models (SVEM). *Chemometrics and Intelligent Laboratory Systems*, *249*, 105122. doi:10.1016/j.chemolab.2024.105122

Karl, A., Wisnowski, J., & Rushing, H. (2022). JMP Pro 17 Remedies for Practical Struggles with Mixture Experiments. JMP Discovery Conference. doi:10.13140/RG.2.2.34598.40003/1

Lemkus, T., Gotwalt, C., Ramsey, P., & Weese, M. L. (2021). Self-Validated Ensemble Models for Design of Experiments. *Chemometrics and Intelligent Laboratory Systems*, 219, 104439. doi:10.1016/j.chemolab.2021.104439

Xu, L., Gotwalt, C., Hong, Y., King, C. B., & Meeker, W. Q. (2020). Applications of the Fractional-Random-Weight Bootstrap. *The American Statistician*, 74(4), 345–358. doi:10.1080/00031305.2020.1731599

Ramsey, P., Gaudard, M., & Levin, W. (2021). Accelerating Innovation with Space Filling Mixture Designs, Neural Networks and SVEM. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Accelerating-Innovation-with-Space-Filling-Mixture-Designs/ev-p/756841

Ramsey, P., & Gotwalt, C. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-ev-p/849647/redirect_from_archived_page/true

Ramsey, P., Levin, W., Lemkus, T., & Gotwalt, C. (2021). SVEM: A Paradigm Shift in Design and Analysis of Experiments. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/SVEM-A-Paradigm-Shift-in-Design-and-Analysis-of-Experiments-2021/ev-p/756634

Ramsey, P., & McNeill, P. (2023). CMC, SVEM, Neural Networks, DOE, and Complexity: It's All About Prediction. *JMP Discovery Conference*.

Friedman, J. H., Hastie, T., and Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1), 1-22.

Meinshausen, N. (2007). Relaxed Lasso. Computational Statistics & Data Analysis, 52(1), 374-393.

Kish, L. (1965). *Survey Sampling*. Wiley.

Lumley, T. (2004). Analysis of complex survey samples. *Journal of Statistical Software*, 9(1), 1–19.

Lumley, T. and Scott, A. (2015). AIC and BIC for modelling with complex survey data. *Journal of Survey Statistics and Methodology*, 3(1), 1–18.

Tibshirani, R. J. and Taylor, J. (2012). Degrees of freedom in lasso problems. *The Annals of Statistics*, 40(2), 1198–1232. doi:10.1214/12AOS1003

Examples

```
data(lipid_screen)
str(lipid_screen)

## Not run:
# Full production-scale modeling workflow. It is not run automatically
# because it fits wide ensembles, runs permutation WMTs, and scores a large
# constrained candidate table.
#2) Build a deterministic expansion using bigexp_terms()
#  Provide main effects only on the right-hand side; expansion width
#  is controlled via arguments. Here Operator is treated as a blocking
#  factor: additive only, no interactions or polynomial terms.
spec <- bigexp_terms(
  Potency ~ PEG + Helper + Ionizable + Cholesterol +
    Ionizable_Lipid_Type + N_P_ratio + flow_rate,
```

```
  data             = lipid_screen,
  factorial_order  = 3,   # up to 3-way interactions
  polynomial_order = 3,   # include up to cubic terms I(X^2), I(X^3)
  include_pc_2way  = TRUE,  # partial-cubic two-way terms Z:I(X^2)
  include_pc_3way  = FALSE,  # no partial-cubic three-way terms I(X^2):Z:W
  blocking         = "Operator",
  discrete_numeric = c("N_P_ratio","flow_rate")
)

# 3) Reuse the same locked expansion for other responses
form_pot <- bigexp_formula(spec, "Potency")
form_siz <- bigexp_formula(spec, "Size")
form_pdi <- bigexp_formula(spec, "PDI")

# 4) Fit SVEM models with the shared factor space and expansion
set.seed(1)
fit_pot <- SVEMnet(form_pot, lipid_screen)
fit_siz <- SVEMnet(form_siz, lipid_screen)
fit_pdi <- SVEMnet(form_pdi, lipid_screen)

# 5) Collect models in a named list by response
objs <- list(Potency = fit_pot, Size = fit_siz, PDI = fit_pdi)

# 6) Define multi-response goals and weights (DS desirabilities under the hood)
#    Maximize Potency (0.6), minimize Size (0.3), minimize PDI (0.1)
goals <- list(
  Potency = list(goal = "max", weight = 0.6),
  Size    = list(goal = "min", weight = 0.3),
  PDI     = list(goal = "min", weight = 0.1)
)

# Mixture constraints: components sum to total, with bounds
mix <- list(list(
  vars  = c("PEG", "Helper", "Ionizable", "Cholesterol"),
  lower = c(0.01, 0.10, 0.10, 0.10),
  upper = c(0.05, 0.60, 0.60, 0.60),
  total = 1.0
))

# 7) Optional: whole-model tests and WMT multipliers via svem_wmt_multi()
#    This wrapper runs svem_significance_test_parallel() for each response,
#    plots the distance distributions, and prints p-values and multipliers.

set.seed(123)
wmt_out <- svem_wmt_multi(
  formulas       = list(Potency = form_pot,
                        Size    = form_siz,
                        PDI     = form_pdi),
  data           = lipid_screen,
  mixture_groups = mix,
  wmt_control    = list(seed = 123, nCore = 1L),
  plot           = TRUE
)
```

```
# Inspect WMT p-values and multipliers (also printed by the wrapper)
wmt_out$p_values
wmt_out$multipliers

# 8) Optional: define process-mean specifications for a joint design space.
#    Potency at least 78, Size no more than 100, PDI less than 0.25.
#    Here we only specify the bounded side; the unbounded side defaults to
#    lower = -Inf or upper = Inf inside svem_score_random().
specs_ds <- list(
  Potency = list(lower = 78),
  Size    = list(upper = 100),
  PDI     = list(upper = 0.25)
)

# 9) Random-search scoring in one step via svem_score_random()
#    This draws a random candidate table, computes DS desirabilities,
#    a combined multi-response score, WMT-adjusted wmt_score (if `wmt`
#    is supplied), CI-based uncertainty, and (when `specs` is supplied)
#    appends mean-level design-space columns.
#
#    The `wmt` and `specs` arguments are optional:
#      - Omit `wmt` for no whole-model reweighting.
#      - Omit `specs` if you do not need design-space probabilities.

set.seed(3)
scored <- svem_score_random(
  objects         = objs,
  goals           = goals,
  data            = lipid_screen,  # scored and returned as original_data_scored
  n               = 25000,
  mixture_groups  = mix,
  level           = 0.95,
  combine         = "geom",
  numeric_sampler = "random",
  wmt             = wmt_out,   # optional: NULL for no WMT
  specs           = specs_ds,  # optional: NULL for no design-space columns
  verbose         = TRUE
)

# 10) Select optimal and exploration sets from the same scored table

# Optimal medoid candidates (maximizing DS score)
opt_sel <- svem_select_from_score_table(
  score_table = scored$score_table,
  target      = "score",      # score column is maximized
  direction   = "max",
  k           = 5,
  top_type    = "frac",
  top         = 0.1,
  label       = "round1_score_optimal"
)
```

```
# Optimal medoid candidates (maximizing WMT-adjusted wmt_score)
opt_sel_wmt <- svem_select_from_score_table(
  score_table = scored$score_table,
  target      = "wmt_score",  # wmt_score column is maximized
  direction   = "max",
  k           = 5,
  top_type    = "frac",
  top         = 0.1,
  label       = "round1_wmt_optimal"
)

# Exploration medoid candidates (highest uncertainty_measure)
explore_sel <- svem_select_from_score_table(
  score_table = scored$score_table,
  target      = "uncertainty_measure",  # uncertainty_measure column is maximized
  direction   = "max",
  k           = 5,
  top_type    = "frac",
  top         = 0.1,
  label       = "round1_explore"
)

# In-spec medoid candidates (highest joint mean-level assurance)
inspec_sel <- svem_select_from_score_table(
  score_table = scored$score_table,
  target      = "p_joint_mean",         # p_joint_mean column is maximized
  direction   = "max",
  k           = 5,
  top_type    = "frac",
  top         = 0.10,
  label       = "round1_inspec"
)

# Single best by score, including per-response CIs
opt_sel$best

# Single best by WMT-adjusted score, including per-response CIs
opt_sel_wmt$best

# Diverse high-score candidates (medoids)
head(opt_sel_wmt$candidates)

# Highest-uncertainty setting and its medoid candidates
explore_sel$best
head(explore_sel$candidates)

# Highest probability mean-in-spec setting and its medoid candidates
inspec_sel$best
head(inspec_sel$candidates)

# 11) Scored original data (predictions, desirabilities, score, wmt_score, uncertainty)
head(scored$original_data_scored)
```

```
# 12) Example: combine new candidate settings with the best existing run
#     and (optionally) export a CSV for the next experimental round.

# Best existing run from the original scored data (no new medoids; k = 0)
best_existing <- svem_select_from_score_table(
  score_table = scored$original_data_scored,
  target      = "score",
  direction   = "max",
  k           = 0,               # k <= 0 => only the best row, no medoids
  top_type    = "frac",
  top         = 1.0,
  label       = "round1_existing_best"
)

# 13) Prepare candidate export tables for the next experimental round.
#     svem_export_candidates_csv() accepts individual selection objects.
#     Calls are commented out so examples/tests do not write files.

# Export a design-style candidate table (factors + responses + predictions)
# out.df <- svem_export_candidates_csv(
#   opt_sel,
#   opt_sel_wmt,
#   explore_sel,
#   inspec_sel,
#   best_existing,
#   file      = "lipid_screen_round1_candidates.csv",
#   overwrite = TRUE
# )
# head(out.df)

# Export all columns including desirabilities, CI widths, and design-space columns
# out.df2 <- svem_export_candidates_csv(
#   opt_sel,
#   opt_sel_wmt,
#   explore_sel,
#   inspec_sel,
#   best_existing,
#   file      = "lipid_screen_round1_candidates_all.csv",
#   overwrite = TRUE
# )
# head(out.df2)

## End(Not run)
```

`plot.svem_binomial` *Plot Method for SVEM Binomial Models*

Description

Diagnostics for `svem_binomial` fits from `SVEMnet(..., family = "binomial")`. Produces one of:

- type = "calibration": Reliability curve (binned average predicted probability vs observed rate), with jittered raw points for context.
- type = "roc": ROC curve with trapezoidal AUC in the title.
- type = "pr": Precision–Recall curve with step-wise Average Precision (AP).

Usage

```
## S3 method for class 'svem_binomial'
plot(
  x,
  type = c("calibration", "roc", "pr"),
  bins = 10,
  jitter_width = 0.05,
  ...
)
```

Arguments

x	An object of class svem_binomial.
type	One of "calibration", "roc", or "pr" (default "calibration").
bins	Integer number of equal-frequency bins for calibration (default 10).
jitter_width	Vertical jitter amplitude for raw points in calibration (default 0.05).
...	Additional aesthetics passed to ggplot2::geom_line() or ggplot2::geom_point().

Details

For ROC/PR, simple one-class guards are used (returns a diagonal ROC and trivial PR). The function assumes binomial models store x$y_pred on the *probability* scale.

Value

A ggplot2 object.

Examples

```
## --- Binomial example: simulate, fit, and plot --------------------------
set.seed(2025)
n  <- 100
x1 <- rnorm(n); x2 <- rnorm(n); x3 <- rnorm(n)
eta    <- -0.4 + 1.1*x1 - 0.8*x2 + 0.5*x3
p_true <- plogis(eta)
y      <- rbinom(n, 1, p_true)
dat_b  <- data.frame(y, x1, x2, x3)

fit_b <- SVEMnet(
  y ~ x1 + x2 + x3 + I(x1^2) + (x1 + x2 + x3)^2,
  data          = dat_b,
  family        = "binomial",
  glmnet_alpha  = 1,
```

```
  nBoot         = 10,
  objective     = "auto",
  weight_scheme = "SVEM",
  relaxed       = FALSE
)

# Calibration / ROC / PR
plot(fit_b, type = "calibration", bins = 12)
plot(fit_b, type = "roc")
plot(fit_b, type = "pr")
```

`plot.svem_model` *Plot Method for SVEM Models (Gaussian / Generic)*

Description

Plots actual versus predicted values for an `svem_model`. This is the default plot for models fit with `SVEMnet(..., family = "gaussian")` and any other non-binomial models that share the `svem_model` class.

Usage

```
## S3 method for class 'svem_model'
plot(x, plot_debiased = FALSE, ...)
```

Arguments

`x`	An object of class `svem_model`.
`plot_debiased`	Logical; if `TRUE`, include debiased predictions (when available) as an additional series. Default `FALSE`.
`...`	Additional aesthetics passed to `ggplot2::geom_point()`.

Details

Points show fitted values against observed responses; the dashed line is the 45-degree identity. If available and requested, debiased predictions are included as a second series.

This method assumes the fitted object stores the training response in `$actual_y` and in-sample predictions in `$y_pred`, as produced by `SVEMnet()` and `glmnet_with_cv()`.

Value

A `ggplot2` object.

Examples

```
## --- Gaussian example: simulate, fit, and plot --------------------------
set.seed(2026)
n  <- 60
X1 <- rnorm(n); X2 <- rnorm(n); X3 <- rnorm(n)
eps <- rnorm(n, sd = 0.4)
y_g <- 1.2 + 2*X1 - 0.7*X2 + 0.3*X3 + 1.1*(X1*X2) + 0.8*(X1^2) + eps
dat_g <- data.frame(y_g, X1, X2, X3)

fit_g <- SVEMnet(
  y_g ~ (X1 + X2 + X3)^2 + I(X1^2) + I(X2^2),
  data          = dat_g,
  family        = "gaussian",
  glmnet_alpha  = 1,
  nBoot         = 10,
  objective     = "auto",
  weight_scheme = "SVEM",
  relaxed       = FALSE
)

# Actual vs predicted (with and without debias overlay)
plot(fit_g, plot_debiased = FALSE)
plot(fit_g, plot_debiased = TRUE)
```

---

plot.svem_significance_test

Plot SVEM significance test results for one or more responses

---

Description

Plots the Mahalanobis-like distances for original and permuted data from one or more SVEM significance test results returned by `svem_significance_test_parallel()`.

Usage

```
## S3 method for class 'svem_significance_test'
plot(x, ..., labels = NULL)
```

Arguments

`x`	An object of class `svem_significance_test`.
`...`	Optional additional `svem_significance_test` objects to include in the same plot.
`labels`	Optional character vector of labels for the responses. If not provided, the function uses inferred response names (from `data_d$Response` or `x$response`) and ensures uniqueness.

Details

If additional `svem_significance_test` objects are provided via `...`, their distance tables (`$data_d`) are stacked and plotted together using a shared x-axis grouping of `"Response / Source"` and a fill aesthetic indicating `"Original"` vs `"Permutation"`.

Value

A `ggplot2` object showing the distributions of distances for original vs. permuted data, grouped by response.

`predict.svem_model` *Predict Method for SVEM Models (Gaussian and Binomial)*

Description

Generate predictions from a fitted SVEM model (Gaussian or binomial), with optional bootstrap uncertainty and family-appropriate output scales.

Usage

```
## S3 method for class 'svem_model'
predict(
  object,
  newdata,
  type = c("response", "link", "class"),
  debias = FALSE,
  se.fit = FALSE,
  interval = FALSE,
  level = 0.95,
  ...
)
```

Arguments

`object`	A fitted SVEM model (class `svem_model`; binomial models typically also inherit class `svem_binomial`). Created by `SVEMnet()`.
`newdata`	A data frame of new predictor values.
`type`	(Binomial only) One of: • `"response"` (default): predicted probabilities in $[0, 1]$. • `"link"`: linear predictor (log-odds). • `"class"`: 0/1 class labels (threshold 0.5). Uncertainty summaries are not available for this type. Ignored for Gaussian models.
`debias`	(Gaussian only) Logical; default `FALSE`. If `TRUE`, apply the linear calibration fit `lm(y ~ y_pred)` learned at training when available. Ignored (and internally set to `FALSE`) for binomial models.

se.fit	Logical; if TRUE, return bootstrap standard errors computed from member predictions (requires coef_matrix). Not available for type = "class". For Gaussian models, this forces use of bootstrap member predictions instead of aggregate coefficients.
interval	Logical; if TRUE, return percentile limits from member predictions (requires coef_matrix). Not available for type = "class". For Gaussian models, this forces use of bootstrap member predictions instead of aggregate coefficients.
level	Central probability mass for member-percentile intervals. Default 0.95.
...	Currently unused.

Details

This method dispatches on object$family:

- **Gaussian**: returns predictions on the response (identity) scale. Optional linear calibration ("debias") learned at training may be applied.
- **Binomial**: supports glmnet-style type = "link", "response", or "class" predictions. No debiasing is applied; type = "response" returns probabilities in $[0, 1]$.

Uncertainty summaries (se.fit, interval) and all binomial predictions are based on per-bootstrap member predictions obtained from the coefficient matrix stored in object$coef_matrix. If coef_matrix is NULL, these options are not available (and binomial prediction will fail). For Gaussian models with se.fit = FALSE and interval = FALSE, predictions are computed directly from the aggregated coefficients.

Value

If se.fit = FALSE and interval = FALSE:

- **Gaussian**: a numeric vector of predictions on the response (identity) scale.
- **Binomial**: a numeric vector for type = "response" (probabilities) or type = "link" (log-odds), or an integer vector of 0/1 labels for type = "class".

If se.fit and/or interval are TRUE (and type != "class"), a list with components:

- fit: predictions on the requested scale.
- se.fit: bootstrap standard errors (when se.fit = TRUE).
- lwr, upr: member-percentile limits (when interval = TRUE).

Rows containing unseen or missing factor levels produce NA predictions (and NA SEs/intervals), with a warning.

Design-matrix reconstruction

The function rebuilds the design matrix for newdata to match the training design:

- Uses the training terms and its compact formula-evaluation environment.
- Harmonizes factor and character predictors to the training xlevels.

- Reuses stored per-factor `contrasts` when available; otherwise falls back to saved global contrast options.
- Zero-fills any columns present at training but absent in `newdata`, and reorders columns to match the training order.

Rows containing unseen factor levels yield `NA` predictions (with a warning).

Aggregation and debiasing

For Gaussian SVEM models:

Point predictions When `se.fit = FALSE` and `interval = FALSE`, predictions are computed from the aggregated coefficients saved at fit time (`object$parms`; or `object$parms_debiased` when `debias = TRUE`). This is algebraically equivalent to averaging member predictions when the coefficients were formed as bootstrap means.

Bootstrap-based summaries When `se.fit = TRUE` and/or `interval = TRUE`, predictions are computed from per-bootstrap member predictions using `object$coef_matrix`. For `debias = TRUE`, the linear calibration is applied to member predictions before summarizing.

For binomial SVEM models, predictions are always aggregated from member predictions on the requested scale (probability or link) using `coef_matrix`; the stored coefficient averages (`parms`, `parms_debiased`) are retained for diagnostics but are not used in prediction. The `debias` argument is ignored and treated as `FALSE` for binomial models.

For Gaussian fits, if `debias = TRUE` and a calibration model `lm(y ~ y_pred)` was learned at training, predictions (and, when applicable, member predictions) are transformed by that calibration. This debiasing is never applied for binomial fits.

Uncertainty

When `se.fit = TRUE`, standard errors are computed as the row-wise standard deviations of member predictions on the requested scale. When `interval = TRUE`, percentile intervals are computed from member predictions on the requested scale, using the requested `level`. Both require a non-null `coef_matrix`. For `type = "class"` (binomial), uncertainty summaries are not available.

These summaries describe variation among bootstrap member predictions. They do not include new-observation noise, and the percentile limits alone do not establish calibrated confidence or prediction coverage.

See Also

`SVEMnet`

Examples

```
## ---- Gaussian example ------------------------------------------------
set.seed(1)
n  <- 60
X1 <- rnorm(n); X2 <- rnorm(n); X3 <- rnorm(n)
y  <- 1 + 0.8 * X1 - 0.6 * X2 + 0.2 * X3 + rnorm(n, 0, 0.4)
dat <- data.frame(y, X1, X2, X3)
```

```
fit_g <- SVEMnet(
  y ~ (X1 + X2 + X3)^2, dat,
  nBoot = 10, glmnet_alpha = 1,
  relaxed = FALSE, family = "gaussian"
)

## Aggregate-coefficient predictions (with and without debiasing)
p_g  <- predict(fit_g, dat)                # debias = FALSE (default)
p_gd <- predict(fit_g, dat, debias = TRUE) # apply calibration, if available

## Bootstrap-based uncertainty (requires coef_matrix)
out_g <- predict(
  fit_g, dat,
  debias   = TRUE,
  se.fit   = TRUE,
  interval = TRUE,
  level    = 0.90
)
str(out_g)

## ---- Binomial example ------------------------------------------------
set.seed(2)
n  <- 120
X1 <- rnorm(n); X2 <- rnorm(n); X3 <- rnorm(n)
eta <- -0.3 + 1.1 * X1 - 0.8 * X2 + 0.5 * X1 * X3
p   <- plogis(eta)
yb  <- rbinom(n, 1, p)
db  <- data.frame(yb = yb, X1 = X1, X2 = X2, X3 = X3)

fit_b <- SVEMnet(
  yb ~ (X1 + X2 + X3)^2, db,
  nBoot = 10, glmnet_alpha = 1,
  relaxed = FALSE, family = "binomial"
)

## Probabilities, link, and classes
p_resp <- predict(fit_b, db, type = "response")
p_link <- predict(fit_b, db, type = "link")
y_hat  <- predict(fit_b, db, type = "class")  # 0/1 labels (no SE or interval)

## Bootstrap-based uncertainty on the probability scale
out_b <- predict(
  fit_b, db,
  type     = "response",
  se.fit   = TRUE,
  interval = TRUE,
  level    = 0.90
)
str(out_b)
```

predict_cv *Predict from glmnet_with_cv Fits (svem_cv Objects)*

Description

Generate predictions from a fitted object returned by `glmnet_with_cv()` (class `"svem_cv"`).

Usage

```
predict_cv(object, newdata, debias = FALSE, strict = FALSE, ...)

## S3 method for class 'svem_cv'
predict(object, newdata, debias = FALSE, strict = FALSE, ...)
```

Arguments

`object`	A fitted object returned by `glmnet_with_cv()` (class `"svem_cv"`).
`newdata`	A data frame of new predictor values.
`debias`	Logical; if `TRUE` and a debiasing fit is available, apply it. Has an effect only for Gaussian models where `debias_fit` is present.
`strict`	Logical; if `TRUE`, enforce a strict column-name match between the aligned design for `newdata` and the training design (including the intercept position). Default `FALSE`.
`...`	Additional arguments (currently unused).

Details

The design matrix for `newdata` is rebuilt using the stored training `terms` (with a compact formula-evaluation environment), together with the saved factor `xlevels` and `contrasts` (cached in `object$schema`). Columns are then aligned back to the training design in a robust way:

- Any training columns that `model.matrix()` drops for `newdata` (for example, a factor collapsing to a single level) are added back as zero columns.
- Columns are reordered to exactly match the training order.
- Rows containing unseen factor/character levels are warned about and their predictions are set to `NA`.

For Gaussian fits (`family = "gaussian"`), the returned values are on the original response (identity-link) scale. For binomial fits (`family = "binomial"`), the returned values are probabilities in `[0,1]` (logit-link inverted via `plogis()`).

If `debias = TRUE` and a calibration model `lm(y ~ y_pred)` is present with a finite slope, predictions are adjusted via `a + b * pred`. Debiasing is only fitted and used for Gaussian models; for binomial models the `debias` argument has no effect.

`predict_cv()` is a small convenience wrapper that simply calls the underlying S3 method `predict.svem_cv()`, keeping a single code path for prediction from `glmnet_with_cv()` objects.

Value

A numeric vector of predictions on the response scale: numeric fitted values for Gaussian models; probabilities in `[0,1]` for binomial models. Rows with unseen factor/character levels return `NA`.

Examples

```
set.seed(1)
n <- 50; p <- 5
X <- matrix(rnorm(n * p), n, p)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(n)
df_ex <- data.frame(y = as.numeric(y), X)
colnames(df_ex) <- c("y", paste0("x", 1:p))

fit <- glmnet_with_cv(
  y ~ ., df_ex,
  glmnet_alpha = 1,
  nfolds = 5,
  repeats = 1,
  seed = 9,
  family = "gaussian"
)

preds_raw <- predict_cv(fit, df_ex)
preds_db  <- predict_cv(fit, df_ex, debias = TRUE)
cor(preds_raw, df_ex$y)

# Binomial example (probability predictions on [0,1] scale)
set.seed(2)
n2 <- 80; p2 <- 4
X2 <- matrix(rnorm(n2 * p2), n2, p2)
eta2 <- X2[, 1] - 0.8 * X2[, 2]
pr2 <- plogis(eta2)
y2 <- rbinom(n2, size = 1, prob = pr2)
df_bin <- data.frame(y = y2, X2)
colnames(df_bin) <- c("y", paste0("x", 1:p2))

fit_bin <- glmnet_with_cv(
  y ~ ., df_bin,
  glmnet_alpha = 1,
  nfolds = 5,
  repeats = 1,
  seed = 11,
  family = "binomial"
)

prob_hat <- predict_cv(fit_bin, df_bin)
summary(prob_hat)
```

print.bigexp_formula *Print a formula that carries a* bigexp_spec *attribute*

Description

Prints the formula itself and a one-line note, instead of the default formula print, which would dump the entire attached spec.

Usage

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

Arguments

x	A formula with an attached bigexp_spec (from bigexp_terms or bigexp_formula).
...	Passed to the default formula print method.

Value

x, invisibly.

print.bigexp_spec *Print method for bigexp_spec objects*

Description

This print method shows a compact summary of the expansion settings and the predictors that are treated as continuous or categorical. It also reports any variables that were designated as blocking factors and therefore enter the model only additively (no interactions, no polynomials).

Usage

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

Arguments

x	A "bigexp_spec" object.
...	Unused.

Examples

```
set.seed(1)
df4 <- data.frame(
  y  = rnorm(10),
  X1 = rnorm(10),
  G  = factor(sample(c("A", "B"), 10, replace = TRUE))
)

spec4 <- bigexp_terms(
  y ~ X1 + G,
  data             = df4,
  factorial_order  = 2,
  polynomial_order = 2
)

print(spec4)

## Example with a blocking factor:
set.seed(2)
df_block2 <- data.frame(
  y           = rnorm(12),
  X1          = rnorm(12),
  G           = factor(sample(c("A", "B"), 12, replace = TRUE)),
  Operator    = factor(sample(letters[1:3], 12, replace = TRUE)),
  AmbientTemp = rnorm(12, mean = 22, sd = 1.5)
)

spec_block2 <- bigexp_terms(
  y ~ X1 + G,
  data             = df_block2,
  factorial_order  = 2,
  polynomial_order = 3,
  blocking         = c("Operator", "AmbientTemp")
)

print(spec_block2)
```

---

print.svem_significance_test

Print Method for SVEM Significance Test

Description

Prints the median p-value from an object of class `svem_significance_test`.

Usage

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

Arguments

x	An object of class svem_significance_test.
...	Additional arguments (unused).

---

print.svem_thompson_batch *Print method for svem_thompson_batch objects*

Description

Print method for svem_thompson_batch objects

Usage

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

Arguments

x	An object of class "svem_thompson_batch".
...	Unused.

Value

x, invisibly.

SVEMnet *Fit an SVEMnet model (Self-Validated Ensemble Elastic Net)*

Description

Fit a Self-Validated Ensemble Model (SVEM) with elastic net or relaxed elastic net base learners using glmnet. Fractional random-weight (FRW) train/validation weights are drawn on each bootstrap replicate, a validation-weighted information criterion (wAIC, wBIC, or wSSE) is minimized to select the penalty, and predictions are ensembled across replicates. Gaussian and binomial responses are supported.

Usage

```
SVEMnet(
  formula,
  data,
  nBoot = 200,
  glmnet_alpha = c(0.5, 1),
  weight_scheme = c("SVEM", "FRW_plain", "Identity"),
  objective = c("auto", "wAIC", "wBIC", "wSSE"),
  relaxed = "auto",
  response = NULL,
  unseen = c("warn_na", "error"),
  family = c("gaussian", "binomial"),
  complexity = c("support", "edf"),
  ...,
  store_member_weights = FALSE
)
```

Arguments

formula	A formula specifying the model to be fitted, or a bigexp_spec created by bigexp_terms().
data	A data frame containing the variables in the model.
nBoot	Integer. Number of bootstrap replicates (default 200). Each replicate draws FRW weights and fits a glmnet path.
glmnet_alpha	Numeric vector of elastic net mixing parameters alpha in [0, 1]. alpha = 1 is lasso, alpha = 0 is ridge. Defaults to c(0.5, 1). When relaxed = TRUE, alpha = 0 is automatically dropped (ridge + relaxed is not used).
weight_scheme	Character. Weighting scheme for train/validation copies. One of: • "SVEM" (default): Self-Validated Ensemble Model weights. For each replicate and row, a shared uniform draw U_i ~ Unif(0, 1) is converted to anti-correlated FRW weights w_train_i = -log(U_i) and w_valid_i = -log(1 - U_i). Each weight vector is then rescaled to have mean 1 (sum n). • "FRW_plain": Fractional random-weight regression without a separate validation copy. A single FRW vector w_i = -log(U_i) is used for both training and validation and rescaled to have mean 1 (sum n). This reproduces the FRW bootstrap regression of Xu et al. (2020) and related work. • "Identity": Uses unit weights for both training and validation (no resampling). In combination with nBoot = 1 this wraps a single glmnet fit and selects the penalty by the chosen information criterion, while still using SVEMnet's expansion and diagnostics.
objective	Character. One of "auto", "wAIC", "wBIC", or "wSSE". • "wAIC": Gaussian AIC-like criterion based on the weighted SSE. • "wBIC": Gaussian BIC-like criterion based on the weighted SSE. See Details for the exact criteria in the Gaussian and binomial cases.

`relaxed` Logical or character. Default `"auto"`. If `TRUE`, use glmnet's relaxed elastic-net path and select both the penalty `lambda` and the relaxed refit parameter `gamma` on each bootstrap. If `FALSE`, fit the standard glmnet path without the relaxed step. If `"auto"` (default), SVEMnet uses `relaxed = TRUE` for `family = "gaussian"` and `relaxed = FALSE` for `family = "binomial"`.

`response` Optional character. When `formula` is a `bigexp_spec`, this names the response column to use on the left-hand side. Defaults to the response stored in the spec.

`unseen` How to treat factor levels not seen in the original `bigexp_spec` when `formula` is a `bigexp_spec`. One of `"warn_na"` (default; convert unseen levels to `NA` with a warning) or `"error"` (stop with an error).

`family` Character. One of `"gaussian"` (default) or `"binomial"`. For Gaussian models SVEMnet uses the identity link; for binomial it uses the canonical logit link. The binomial response must be numeric 0/1, logical, or a factor with exactly two levels (the second level is treated as 1).

`complexity` Character. Complexity measure used in the wAIC/wBIC penalty term. One of:

- `"support"` (default): the number of nonzero coefficients (including the intercept), as described in the package references. It is a model-size proxy. For a fixed lasso tuning parameter it estimates degrees of freedom under the usual full-rank assumptions; collinearity and adaptive tuning require additional care.
- `"edf"` (experimental, provided for further study): a gamma-blended effective degrees of freedom. For an elastic-net path point with active set A, penalty λ, and mixing parameter $\alpha < 1$, the penalized fit's effective df is the ridge-type trace $\sum_j e_j/(e_j + n(\lambda/s_{y,w})(1-\alpha))$, where e_j are the eigenvalues of the weighted, standardized Gram matrix of the active columns, and $s_{y,w}$ is the training-weighted response standard deviation. This conditional trace holds the active set and tuning choices fixed; it does not include their adaptation cost. Nonuniform penalty factors enter the corresponding diagonal ridge matrix. Finite coefficient bounds are not supported for EDF selection; support-selected bounded fits report EDF as unavailable. The relaxed refit's df is the rank of the active design. The blended complexity for relaxed mixing parameter γ is $1 + \gamma\, df_{enet} + (1-\gamma)\, rank(A)$. For $\alpha = 1$ (lasso) this reduces to one plus the rank of the centered active design (Tibshirani and Taylor, 2012), equal to support when those columns are linearly independent. The numerical rank uses Gram eigenvalues exceeding 10^{-10} times the largest eigenvalue; very ill-conditioned columns can therefore count as dependent. Support also overstates the conditional trace for $\alpha < 1$. Implemented for `family = "gaussian"` only; binomial fits fall back to `"support"` with a warning.

The admissibility guardrail (the nonzero non-intercept count must be strictly less than $n_{eff,adm}$) and the reported model-size diagnostics always use the support count, regardless of this setting. `"edf"` changes only the complexity value inserted into the wAIC/wBIC penalty and is intended for simulation study of the alpha/gamma comparison; the published SVEM results use `"support"`. This separate conservative constraint can exclude elastic-net models with many nonzeros even when their conditional EDF is small.

`...` Additional arguments passed to `glmnet()`, such as `penalty.factor`, `lower.limits`, `upper.limits`, or `standardize.response`. An `exclude` callback receives the original design columns and the current fit's response and weights; an internal compatibility column is never passed to it. Numerically constant columns remain excluded in addition to user exclusions. When applying this mask or handling a one-column design, the package checks that callback returns index the original columns (or form an aligned logical vector); other callbacks retain `glmnet`'s validation. Any user-supplied `weights` are ignored (SVEM supplies its own fractional random weights), any user-supplied `offset` is ignored (SVEMnet does not support offsets), and any user-supplied `standardize` is ignored; SVEMnet always calls `glmnet` with `standardize = TRUE`. Glmnet algorithm-control values (for example `maxit` or `thresh`) supplied directly or through `control = list(...)` are routed in a version-compatible way when possible. The arguments `nlambda`, `lambda`, and `lambda.min.ratio` are reserved and managed by SVEMnet (`nlambda` is fixed at 500); user-supplied values are ignored with a warning. Argument names that the installed `glmnet` does not recognize are also ignored with a warning (misspelling protection).

`store_member_weights` Logical; default `FALSE`. If `TRUE`, retain the exact per-member uniforms and the mean-one training and validation weights in `member_weights`. This can materially increase object size. The stored weights support reproducibility and experimental ensemble-mean diagnostics; they do not make the member-percentile limits calibrated confidence or prediction intervals.

Details

This is the core ensemble fitter described in Karl (2026), doi:10.1016/j.chemolab.2026.105660. The published package workflow uses the default `complexity = "support"`; the optional `complexity = "edf"` selector is an experimental extension.

You can pass either:

- a standard model formula, e.g., `y ~ X1 + X2 + X3 + I(X1^2) + (X1 + X2 + X3)^2`, or
- a `bigexp_spec` created by `bigexp_terms()`, in which case SVEMnet will build the design matrix deterministically (locked types, levels, and contrasts) and, if requested, swap the response to fit multiple independent responses over the same expansion.

SVEMnet implements Self-Validated Ensemble Models using elastic net and relaxed elastic net base learners from glmnet. Each bootstrap replicate draws fractional random weights, builds a train and validation copy, fits a path over lambda (and optionally over alpha and relaxed gamma), and selects a path point by minimizing a validation-weighted criterion. Final predictions are obtained by averaging replicate predictions on the chosen scale.

By default, `relaxed = "auto"` resolves to `TRUE` for Gaussian fits and `FALSE` for binomial fits.

The function is typically used in small-n design-of-experiments (DOE) workflows where classical train/validation splits and cross-validation can be unstable. A common pattern is: (1) build a deterministic expansion with `bigexp_terms()`, (2) fit SVEM models via `SVEMnet()`, (3) perform whole-model significance testing, and (4) call `svem_score_random()` for constrained multi-response optimization.

Weighting schemes:

- With `weight_scheme = "SVEM"`, SVEMnet uses a pair of anti-correlated FRW vectors for train and validation. All rows appear in every replicate, but train and validation contributions are separated through the shared uniform draws.
- With `weight_scheme = "FRW_plain"`, a single FRW vector is used for both train and validation, which reproduces FRW regression without a self-validation split. This is mainly provided for method comparison and teaching.
- With `weight_scheme = "Identity"`, both train and validation weights are 1. Setting `nBoot = 1` in this mode yields a single glmnet fit whose penalty is chosen by the selected information criterion, without any bootstrap variation.

Selection criteria (Gaussian): For `family = "gaussian"`, the validation loss is a weighted sum of squared errors on the validation copy. Let $\mathrm{SSE}_w = \sum_i w_i^{valid} r_i^2$ denote the weighted SSE. The criteria are:

- `"wSSE"`: loss-only selector that minimizes the weighted SSE SSE_w,
- `"wAIC"`: Gaussian AIC analog $C(\lambda) = n \log\{\mathrm{SSE}_w(\lambda)/n\} + 2k$,
- `"wBIC"`: Gaussian BIC analog $C(\lambda) = n \log\{\mathrm{SSE}_w(\lambda)/n\} + \log(n_e f f_a dm)\, k$.

The FRW validation weights are rescaled to have mean one, so that their sum is always $\sum_i w_i^{valid} = n$. The AIC/BIC analogs therefore use $n \log(\mathrm{SSE}_w/n)$ as the Gaussian loss term, while the `"wSSE"` selector uses SSE_w directly.

The effective validation size is computed from the FRW weights using Kish's effective sample size $n_e f f = (\sum_i w_i^{valid})^2 / \sum_i (w_i^{valid})^2$ and then truncated to lie between 2 and `n` to form $n_e f f_a dm$. The AIC-style selector uses a $2k$ penalty; the BIC-style selector uses a $\log(n_e f f_a dm) k$ penalty, so that the loss term is scaled by total validation weight while the complexity penalty uses a weight-concentration adjustment. For `"wAIC"` and `"wBIC"`, the number of non-intercept coefficients must be strictly less than $n_e f f_a dm$; equality is inadmissible.

Because the FRW validation weights are random rather than fixed design weights, these information-criterion scores are used heuristically for relative model comparison within each FRW replicate, rather than as exact AIC/BIC values. The training and validation copies reuse the same responses; they are not independent held-out samples. Kish size measures weight concentration for a weighted mean, not regression rank, leverage or selection complexity. The name `wAIC` does not refer to Bayesian WAIC. Also, `wBIC` has a smaller per-coefficient penalty than `wAIC` when $n_e f f_a dm < \exp(2)$. Normalizing weights to total `n` defines the loss-to-penalty scale; substituting Kish size in the loss would be a different selection rule.

For diagnostics, SVEMnet reports the raw Kish effective sizes across bootstraps (see `diagnostics$n_eff_summary`), while $n_e f f_a dm$ is used internally in the penalty and model-size guardrail. Near-interpolating path points are screened out via a simple model size guardrail before minimization. When `objective = "auto"`, Gaussian fits use `"wAIC"` and binomial fits use `"wBIC"`. This guard is a heuristic cap, not a guarantee of positive residual degrees of freedom. With `weight_scheme = "Identity"`, support may reach `n`; shrunken fits can still have positive SSE, while exact interpolation makes the Gaussian profile likelihood singular. Nonpositive support-based residual denominators are flagged in `member_diagnostics`.

This structure (pseudo-likelihood using total weight and BIC penalty using a Kish-type effective sample size) parallels survey-weighted information criteria as in Lumley and Scott (2015) and Kish (1965).

Selection criteria (binomial): For `family = "binomial"`, the validation loss is the weighted negative log-likelihood on the FRW validation copy (equivalently, proportional to the binomial deviance up to a constant factor). Let NLL denote the weighted negative log-likelihood. The same labels are used:

- `"wSSE"`: loss-only selector based on NLL (the name is retained for backward compatibility),
- `"wAIC"`: deviance-style criterion $C(\lambda) = 2\,\mathrm{NLL}(\lambda) + 2k$,
- `"wBIC"`: deviance-style criterion $C(\lambda) = 2\,\mathrm{NLL}(\lambda) + \log(n_e f f_a dm)\,k$.

The effective validation size $n_e f f_a dm$ and the model size guardrail are handled as in the Gaussian case: for `"wAIC"` and `"wBIC"` we compute a Kish effective size from the FRW validation weights, truncate it to lie between 2 and `n`, and require the number of nonzero coefficients (excluding the intercept) to be less than this effective size when evaluating the criterion.

Auto rule: When `objective = "auto"`, SVEMnet selects the criterion by family:

- `family = "gaussian"` -> `"wAIC"`
- `family = "binomial"` -> `"wBIC"`

Relaxed elastic net: When `relaxed = TRUE`, SVEMnet calls glmnet with `relax = TRUE` and traverses a small grid of relaxed refit values (gamma). For each alpha and gamma, SVEMnet evaluates all lambda path points on the validation copy and records the combination that minimizes the selected criterion. Model size is always defined as the number of nonzero coefficients including the intercept, so standard and relaxed paths are scored on the same scale.

Gaussian debiasing: For Gaussian models, SVEMnet optionally performs a simple linear calibration of ensemble predictions on the training data. When there is sufficient variation in the fitted values and `nBoot` is at least 10, the function fits `lm(y ~ y_pred)` and uses the coefficients to construct debiased coefficients and debiased fitted values. Binomial fits do not use debiasing; predictions are ensembled on the probability or link scale directly.

Implementation notes:

- Predictors are always standardized internally via `glmnet(..., standardize = TRUE)`.
- The terms object is stored with a compact evaluation environment so that prediction supports standard and local transformation functions without accidentally capturing the full calling environment.
- A compact schema (feature names, factor levels, contrasts, and a simple hash) is stored to allow `predict()` and companion functions to rebuild model matrices deterministically, even when the original data frame is not available.
- A separate sampling schema stores raw predictor ranges and factor levels for use in random candidate generation for optimization.

Value

An object of class `"svem_model"` (and `"svem_binomial"` when `family = "binomial"`) with components:

- `parms`: Vector of ensemble-averaged coefficients, including the intercept.
- `parms_debiased`: Vector of coefficients after optional debiasing (see Details; Gaussian only).

- debias_fit: If debiasing was performed, the calibration model lm(y ~ y_pred); otherwise NULL.
- coef_matrix: Matrix of per-bootstrap coefficients (rows = bootstraps, columns = intercept and predictors).
- nBoot: Number of bootstrap replicates actually used.
- glmnet_alpha: Vector of alpha values considered.
- best_alphas: Per-bootstrap alpha selected by the criterion.
- best_lambdas: Per-bootstrap lambda selected by the criterion.
- best_relax_gammas: Per-bootstrap relaxed gamma selected when relaxed = TRUE; NA otherwise.
- complexity: The complexity measure actually used ("support" or "edf").
- best_edfs: Per-bootstrap blended effective df of the selected Gaussian path point. This diagnostic is computed even when complexity = "support"; it is NA for binomial fits.
- member_diagnostics: Per-member Gaussian validation SSE, selected objective value, support size, effective df, raw and bounded Kish validation size, support-based residual df and scale, fallback status, and admissibility warnings. Gaussian residual-scale fields are NA for binomial fits.
- member_weights: If store_member_weights = TRUE, a list containing matrices uniforms, training, and validation, with rows aligned to coef_matrix; otherwise NULL.
- weight_scheme: The weighting scheme that was used.
- relaxed: Logical flag indicating whether relaxed paths were used.
- relaxed_input: The user-supplied value for relaxed (one of TRUE, FALSE, or "auto"). The resolved flag actually used is reported in relaxed.
- dropped_alpha0_for_relaxed: Logical; TRUE if alpha = 0 was dropped because relaxed = TRUE.
- objective_input: The objective requested by the user.
- objective_used: The objective actually used after applying the "auto" rule (for example "wAIC" or "wBIC").
- objective: Same as objective_used (for convenience).
- auto_used: Logical; TRUE if objective = "auto".
- auto_decision: The objective selected by the auto rule (wAIC or wBIC) when auto_used = TRUE.
- diagnostics: List with summary information, including:
  - k_summary: Median and IQR of selected model size (number of nonzero coefficients including intercept).
  - fallback_rate: Proportion of bootstraps that fell back to an intercept-only fit.
  - fallback_reasons: Named counts summarizing why any non-structural bootstrap fits required that fallback.
  - n_eff_summary: Summary of raw Kish effective validation sizes $n_eff = (\sum_i w_i^{valid})^2 / \sum_i (w_i^{valid})^2$ across bootstraps (before truncation to form n_eff_adm).
  - alpha_freq: Relative frequency of selected alpha values (if any).

  - relax_gamma_freq: Relative frequency of selected relaxed gamma values (if relaxed = TRUE and any were selected).
- actual_y: Numeric response vector used in fitting (0/1 for binomial).
- training_X: Numeric model matrix without the intercept column used for training.
- y_pred: Fitted values from the ensemble on the training data. For Gaussian this is on the response scale; for binomial it is on the probability scale.
- y_pred_debiased: Debiased fitted values on the training data (Gaussian only); NULL otherwise.
- nobs: Number of observations used in fitting.
- nparm: Number of parameters in the full expansion (intercept plus predictors).
- formula: The formula used for fitting (possibly derived from a bigexp_spec).
- terms: terms object used for building the design matrix, with a compact evaluation environment containing any required formula transformations.
- xlevels: Factor levels recorded at training time.
- contrasts: Contrasts used for building the design matrix.
- schema: Compact description for safe prediction, including feature_names, terms_str, xlevels, contrasts, contrasts_options, and a simple hash.
- sampling_schema: Schema used to generate random candidate tables, including predictor names, variable classes, numeric ranges, and factor levels.
- used_bigexp_spec: Logical flag indicating whether a bigexp_spec was used.
- family: The fitted family ("gaussian" or "binomial").

Acknowledgments

Generative AI tools were used to assist with coding and roxygen documentation; all content was reviewed and finalized by the author.

References

Karl, A. T. (2026). SVEMnet: An R package for self-validated elastic-net ensembles and multi-response optimization in small-sample mixture-process experiments. *Chemometrics and Intelligent Laboratory Systems*, *271*, 105660. doi:10.1016/j.chemolab.2026.105660

Gotwalt, C., & Ramsey, P. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-Using/ev-p/849873/redirect_from_archived_page/true

Karl, A. T. (2024). A randomized permutation whole-model test heuristic for Self-Validated Ensemble Models (SVEM). *Chemometrics and Intelligent Laboratory Systems*, *249*, 105122. doi:10.1016/j.chemolab.2024.105122

Karl, A., Wisnowski, J., & Rushing, H. (2022). JMP Pro 17 Remedies for Practical Struggles with Mixture Experiments. JMP Discovery Conference. doi:10.13140/RG.2.2.34598.40003/1

Lemkus, T., Gotwalt, C., Ramsey, P., & Weese, M. L. (2021). Self-Validated Ensemble Models for Design of Experiments. *Chemometrics and Intelligent Laboratory Systems*, 219, 104439. doi:10.1016/j.chemolab.2021.104439

Xu, L., Gotwalt, C., Hong, Y., King, C. B., & Meeker, W. Q. (2020). Applications of the Fractional-Random-Weight Bootstrap. *The American Statistician*, 74(4), 345–358. doi:10.1080/00031305.2020.1731599

Ramsey, P., Gaudard, M., & Levin, W. (2021). Accelerating Innovation with Space Filling Mixture Designs, Neural Networks and SVEM. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Accelerating-Innovation-with-Space-Filling-Mixture-Designs/ev-p/756841

Ramsey, P., & Gotwalt, C. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-ev-p/849647/redirect_from_archived_page/true

Ramsey, P., Levin, W., Lemkus, T., & Gotwalt, C. (2021). SVEM: A Paradigm Shift in Design and Analysis of Experiments. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/SVEM-A-Paradigm-Shift-in-Design-and-Analysis-of-Experiments-2021/ev-p/756634

Ramsey, P., & McNeill, P. (2023). CMC, SVEM, Neural Networks, DOE, and Complexity: It's All About Prediction. *JMP Discovery Conference*.

Friedman, J. H., Hastie, T., and Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1), 1-22.

Meinshausen, N. (2007). Relaxed Lasso. Computational Statistics & Data Analysis, 52(1), 374-393.

Kish, L. (1965). *Survey Sampling*. Wiley.

Lumley, T. (2004). Analysis of complex survey samples. *Journal of Statistical Software*, 9(1), 1–19.

Lumley, T. and Scott, A. (2015). AIC and BIC for modelling with complex survey data. *Journal of Survey Statistics and Methodology*, 3(1), 1–18.

Tibshirani, R. J. and Taylor, J. (2012). Degrees of freedom in lasso problems. *The Annals of Statistics*, 40(2), 1198–1232. doi:10.1214/12AOS1003

Zou, H., Hastie, T., & Tibshirani, R. (2007). On the "degrees of freedom" of the lasso. *The Annals of Statistics*, 35(5), 2173–2192. doi:10.1214/009053607000000127

Examples

```
set.seed(42)

n  <- 30
X1 <- rnorm(n)
X2 <- rnorm(n)
X3 <- rnorm(n)
eps <- rnorm(n, sd = 0.5)
y <- 1 + 2 * X1 - 1.5 * X2 + 0.5 * X3 + 1.2 * (X1 * X2) +
  0.8 * (X1^2) + eps
dat <- data.frame(y, X1, X2, X3)

# Minimal hand-written expansion
mod_relax <- SVEMnet(
  y ~ (X1 + X2 + X3)^2 + I(X1^2) + I(X2^2),
  data          = dat,
  glmnet_alpha  = 1,
```

```
  nBoot         = 10,
  objective     = "auto",
  weight_scheme = "SVEM",
  relaxed       = FALSE
)

pred_in_raw <- predict(mod_relax, dat, debias = FALSE)
pred_in_db  <- predict(mod_relax, dat, debias = TRUE)

# ------------------------------------------------------------------------------
# Big expansion (full factorial + polynomial surface + partial-cubic crosses)
# Build once, reuse for one or more responses
# ------------------------------------------------------------------------------
spec <- bigexp_terms(
  y ~ X1 + X2 + X3,
  data             = dat,
  factorial_order  = 3,
  polynomial_order = 3,
  include_pc_3way  = FALSE
)

# Fit using the spec (auto-prepares data)
fit_y <- SVEMnet(
  spec, dat,
  glmnet_alpha  = 1,
  nBoot         = 8,
  objective     = "auto",
  weight_scheme = "SVEM"
)

# A second, independent response over the same expansion
set.seed(99)
dat$y2 <- 0.5 + 1.4 * X1 - 0.6 * X2 + 0.2 * X3 + rnorm(n, 0, 0.4)
fit_y2 <- SVEMnet(
  spec, dat, response = "y2",
  glmnet_alpha  = 1,
  nBoot         = 8,
  objective     = "auto",
  weight_scheme = "SVEM"
)

svem_nonzero(fit_y2)

p1 <- predict(fit_y,  dat)
p2 <- predict(fit_y2, dat, debias = TRUE)

# Show that a new batch expands identically under the same spec
newdat <- data.frame(
  y  = y,
  X1 = X1 + rnorm(n, 0, 0.05),
  X2 = X2 + rnorm(n, 0, 0.05),
  X3 = X3 + rnorm(n, 0, 0.05)
)
```

```
prep_new <- bigexp_prepare(spec, newdat)
stopifnot(identical(
  colnames(model.matrix(spec$formula, bigexp_prepare(spec, dat)$data)),
  colnames(model.matrix(spec$formula, prep_new$data))
))
preds_new <- predict(fit_y, prep_new$data)

## Binomial example
set.seed(2)
n  <- 80
X1 <- rnorm(n); X2 <- rnorm(n); X3 <- rnorm(n)
eta <- -0.3 + 1.1 * X1 - 0.8 * X2 + 0.5 * X1 * X3
p   <- plogis(eta)
yb  <- rbinom(n, 1, p)
db  <- data.frame(yb = yb, X1 = X1, X2 = X2, X3 = X3)

fit_b <- SVEMnet(
  yb ~ (X1 + X2 + X3)^2, db,
  nBoot        = 10,
  glmnet_alpha = 1,
  family       = "binomial"
)

## Probabilities, link, and classes
p_resp <- predict(fit_b, db, type = "response")
p_link <- predict(fit_b, db, type = "link")
y_hat  <- predict(fit_b, db, type = "class")  # 0/1 labels

## Mean aggregation with uncertainty on probability scale
out_b <- predict(
  fit_b, db,
  type     = "response",
  se.fit   = TRUE,
  interval = TRUE,
  level    = 0.9
)
str(out_b)

## Example with blocking (requires SVEMnet to store sampling_schema$blocking)
set.seed(2)
df_block <- data.frame(
  y1         = rnorm(40),
  y2         = rnorm(40),
  X1         = runif(40),
  X2         = runif(40),
  Operator   = factor(sample(paste0("Op", 1:3), 40, TRUE)),
  AmbientTmp = rnorm(40, mean = 22, sd = 2)
)

spec_block <- bigexp_terms(
  y1 ~ X1 + X2,
  data             = df_block,
  factorial_order  = 2,
```

```
  polynomial_order = 2,
  blocking         = c("Operator", "AmbientTmp")
)

fit_b1 <- SVEMnet(spec_block, df_block, response = "y1", nBoot = 8,
                  glmnet_alpha = 1, relaxed = FALSE)
fit_b2 <- SVEMnet(spec_block, df_block, response = "y2", nBoot = 8,
                  glmnet_alpha = 1, relaxed = FALSE)

tab_block <- svem_random_table_multi(list(fit_b1, fit_b2), n = 200)
```

`svem_export_candidates_csv` *Export SVEM candidate sets to CSV*

Description

Given one or more selection objects returned by `svem_select_from_score_table`, concatenate their `$best` rows and `$candidates` and export a CSV suitable for planning new experimental runs.

Each row is tagged with:

- `candidate_type`: `"best"` or `"medoid"`.
- `selection_label`: derived from the `label` argument used in `svem_select_from_score_table()` when available.

The function does not modify any response or prediction columns (for example, `Potency`, `Potency_pred`); it simply harmonizes columns across inputs (adding `NA`-filled columns where necessary), concatenates rows, and reorders a few metadata columns for readability.

Any columns named `candidate_type`, `selection_label`, or `Notes_from_SVEMnet` that are present in the final data frame are moved to the leftmost positions in that order.

Usage

```
svem_export_candidates_csv(
  ...,
  file = NULL,
  overwrite = FALSE,
  write_file = TRUE
)
```

Arguments

`...`	One or more objects returned by `svem_select_from_score_table`. You may also pass a single list of such objects.
`file`	Character scalar; path to the CSV file to be written. Required only when `write_file = TRUE`.

overwrite	Logical; if FALSE (default) and file already exists, an error is thrown. If TRUE, any existing file at file is overwritten. Only used when write_file = TRUE.
write_file	Logical; if TRUE (default), write the combined table to file as CSV and print the full path. If FALSE, no file is written and file may be NULL; the concatenated data.frame is still returned (invisibly).

Value

Invisibly, the data.frame that was written to CSV (or would be written, when write_file = FALSE).

Examples

```
# Fabricated scored candidates keep this example independent of model fitting.
scored <- data.frame(
  x1 = c(0.1, 0.4, 0.7, 0.9),
  x2 = factor(c("A", "B", "A", "B")),
  score = c(0.55, 0.82, 0.91, 0.73),
  uncertainty_measure = c(0.15, 0.31, 0.20, 0.44)
)

optimal <- svem_select_from_score_table(
  scored, target = "score", direction = "max", k = 0,
  label = "round1_optimal"
)
exploratory <- svem_select_from_score_table(
  scored, target = "uncertainty_measure", direction = "max", k = 0,
  label = "round1_exploratory"
)

# Export to disk when desired:
# svem_export_candidates_csv(
#   optimal, exploratory, file = "round1_candidates.csv",
#   overwrite  = FALSE,
#   write_file = TRUE
# )

# Inspect the combined table without writing a file.
cand_tbl <- svem_export_candidates_csv(optimal, exploratory, write_file = FALSE)
head(cand_tbl)
```

---

svem_forward *Fit an SVEM Model with Forward-Selection Base Learners*

Description

Fit a Self-Validated Ensemble Model in which each bootstrap replicate's base learner is greedy forward selection on training-weighted least squares, instead of the glmnet elastic-net path used by

SVEMnet. The forward path (intercept-only model up to the full greedy path) plays the lambda-path role: every path point is scored on the fractional-random-weight validation copy with the same validation-weighted criteria as SVEMnet() ("wAIC" by default, or "wBIC" / "wSSE"), and the minimizing path point wins that replicate. Winning coefficient vectors are averaged across bootstrap replicates exactly as in SVEMnet().

Usage

```
svem_forward(
  formula,
  data,
  nBoot = 200,
  objective = c("auto", "wAIC", "wBIC", "wSSE"),
  weight_scheme = c("SVEM", "FRW_plain", "Identity"),
  response = NULL,
  unseen = c("warn_na", "error"),
  store_member_weights = FALSE
)
```

Arguments

formula	A model formula or a bigexp_spec created by bigexp_terms(), exactly as in SVEMnet.
data	A data frame containing the variables in the model.
nBoot	Integer. Number of bootstrap replicates (default 200).
objective	Character. One of "auto" (resolves to "wAIC"), "wAIC", "wBIC", or "wSSE", with the same definitions as in SVEMnet (Gaussian case): the model size k counts the fitted coefficients of the path point including the intercept.
weight_scheme	Character. "SVEM" (default), "FRW_plain", or "Identity", as in SVEMnet. "Identity" with nBoot = 1 yields a single deterministic forward-selection fit tuned by the chosen information criterion on the training data. The support guard does not guarantee positive residual degrees of freedom. In particular, "Identity" gives $n_{eff,adm} = n$, so the wAIC/wBIC rule admits saturated path points with $k = n$. A forward path can then interpolate the training data and be selected by the log-scale criterion. Such members are flagged in member_diagnostics. By comparison, forward_aicc() skips models with nonpositive residual degrees of freedom for AIC/BIC; its AICc criterion requires at least three residual degrees of freedom.
response	Optional character. When formula is a bigexp_spec, names the response column to use on the left-hand side.
unseen	How to treat factor levels not seen in the original bigexp_spec: "warn_na" (default) or "error".
store_member_weights	Logical; default FALSE. If TRUE, retain the exact per-member uniforms and mean-one training and validation weights in member_weights. This can materially increase object size.

Details

This is an experimental alternative to the elastic-net base learners in the package workflow described by Karl (2026), doi:10.1016/j.chemolab.2026.105660. The package paper's results do not establish the performance of this implementation.

Per replicate and row, a shared uniform draw is converted to anti-correlated train/validation weights (`weight_scheme = "SVEM"`), the forward path is grown on rows scaled by `sqrt(w_train)`, greedy steps add the whole model term (multi-column factor blocks move as a unit) that most reduces training RSS, and each path point is refit and scored by validation-weighted residuals on the original (unweighted) rows. For `"wAIC"`/`"wBIC"`, path growth stops once the slope count reaches the Kish admissible effective sample size $n_{eff,adm}$, because all further path points would be inadmissible; `"wSSE"` paths are never truncated. Replicates with no admissible finite path point fall back to a weighted intercept-only model, as in `SVEMnet()`.

Only Gaussian responses are supported. There is no debias argument at fit time: as in `SVEMnet()`, a linear calibration `lm(y ~ y_pred)` is stored when `nBoot >= 10` and the fitted values vary, and `predict(..., debias = TRUE)` applies it.

Unlike the elastic net, forward selection contains no shrinkage, so exactly collinear candidates (for example the third mixture component after the first two have entered) are skipped as rank deficient rather than entering with shrunk coefficients.

Value

An object of classes `"svem_forward"` and `"svem_model"`, structured like an `SVEMnet` fit: `parms`, `parms_debiased`, `debias_fit`, `coef_matrix` (rows = bootstrap members), `diagnostics`, `schema`, `sampling_schema`, and so on, so that `predict()` (including `se.fit`/`interval`), `coef()`, `plot()`, `svem_nonzero()`, `svem_random_table_multi()`, and `svem_score_random()` work unchanged. Forward-selection-specific components:

- `best_ks`: per-bootstrap selected model size (coefficients including the intercept).
- `best_edfs`: per-bootstrap effective df; for the full-rank weighted least-squares refits used here this equals `best_ks`.
- `member_diagnostics`: per-member validation SSE, objective value, support/effective df, Kish validation size, residual scale, and admissibility warnings.
- `member_weights`: optional stored FRW draws and weights, when `store_member_weights = TRUE`.
- `diagnostics$path_length_max`: longest grown path (points, including the intercept-only point).
- `diagnostics$admissibility_truncation_rate`: fraction of replicates whose path growth was stopped by the wAIC/wBIC admissibility ceiling.
- `diagnostics$selection_frequencies`: fraction of replicates in which each model term was part of the winning model.

See Also

`forward_aicc` for the deterministic single-model version, `SVEMnet` for the elastic-net ensemble.

Examples

```
set.seed(42)
n  <- 30
X1 <- rnorm(n); X2 <- rnorm(n); X3 <- rnorm(n)
y  <- 1 + 2 * X1 - 1.5 * X2 + 0.8 * X1 * X2 + rnorm(n, 0, 0.5)
dat <- data.frame(y, X1, X2, X3)

fit <- svem_forward(y ~ (X1 + X2 + X3)^2, dat, nBoot = 8)
preds <- predict(fit, dat)

## bigexp_spec front end, as in SVEMnet()
spec <- bigexp_terms(y ~ X1 + X2 + X3, dat, factorial_order = 2)
fit2 <- svem_forward(spec, dat, nBoot = 8)
```

---

svem_ij_variance *Experimental FRW Infinitesimal-Jackknife Variance Diagnostic*

Description

Estimate the sampling-variance component of the raw SVEM ensemble-mean prediction from covariances between stored fractional-random training weights and member predictions. This adapts the infinitesimal-jackknife calculations for bootstrap-smoothed estimators in Efron (2014) and Wager, Hastie, and Efron (2014) by replacing ordinary-bootstrap counts with the mean-one FRW training weights used by SVEM.

Usage

```
svem_ij_variance(object, newdata, finite_B = TRUE)
```

Arguments

`object`	A Gaussian `svem_model` fitted with `store_member_weights = TRUE`.
`newdata`	A data frame of predictor values.
`finite_B`	Logical; default `TRUE`. Subtract the ordinary- bootstrap-inspired finite-ensemble Monte Carlo correction $nB^{-2}\sum_b\{T_b(x)-\bar{T}(x)\}^2$. The untruncated raw estimate and the correction are always returned.

Details

For member prediction $T_b(x)$ and stored mean-one training weight W_{bi}, the raw diagnostic is

$$V_{IJ}(x)=\sum_i \mathrm{cov}_b\{W_{bi},T_b(x)\}^2.$$

When `finite_B = TRUE`, the finite-ensemble correction is subtracted and negative corrected values are truncated at zero for the reported variance and standard error.

This is an experimental diagnostic, not a calibrated confidence interval. The cited IJ results are for ordinary bootstrap counts; the FRW substitution used here is an adaptation. In particular, this function does not correct prediction bias, selection nonregularity, or observation noise, and it must not be used as a prediction interval for a new observation.

Value

A data frame with one row per row of `newdata` and columns `fit`, `variance`, `se`, `variance_raw`, `finite_B_correction`, `truncated_at_zero`, and `calibrated` (always `FALSE`). Attributes record the number of observations, ensemble members, weighting scheme, method, and caution.

References

Efron, B. (2014). Estimation and Accuracy After Model Selection. *Journal of the American Statistical Association*, 109, 991–1007.

Wager, S., Hastie, T., and Efron, B. (2014). Confidence Intervals for Random Forests: The Jackknife and the Infinitesimal Jackknife. *Journal of Machine Learning Research*, 15, 1625–1651.

See Also

`SVEMnet`, `svem_forward`

---

svem_nonzero *Coefficient Nonzero Percentages (SVEM)*

Description

Summarizes variable-selection stability across SVEM bootstrap refits by computing the percentage of bootstrap iterations in which each coefficient (excluding the intercept) is nonzero, using a small tolerance. Optionally produces a quick **ggplot2** summary and/or prints a compact table.

Usage

```
svem_nonzero(object, tol = 1e-07, plot = TRUE, print_table = TRUE, ...)
```

Arguments

`object`	An object of class `svem_model` with a non-empty `$coef_matrix` component. `svem_nonzero()` is not defined for `svem_cv` objects.
`tol`	Numeric tolerance for "nonzero". Coefficients with `|beta| > tol` are counted as nonzero. Default is `1e-7`. This is an absolute threshold in the raw coefficient units, so reports using a positive tolerance depend on predictor and response units. Use `tol = 0` for exact selection frequencies, matching the internal support counts used for model selection.
`plot`	Logical; if `TRUE`, draws a quick **ggplot2** summary plot of the nonzero percentages (default `TRUE`).

print_table	Logical; if TRUE, prints a compact table of nonzero percentages to the console (default TRUE).
...	Unused; included for future extension.

Details

This function expects object$coef_matrix to contain the per-bootstrap coefficients (including an intercept column), which SVEMnet always stores in its fitted objects. Rows correspond to bootstrap fits; columns correspond to coefficients.

Internally, svem_nonzero():

- checks for and drops rows of coef_matrix that contain any non-finite values, to keep summaries stable;
- drops an "(Intercept)" column if present;
- computes 100 * mean(|beta_j| > tol) across bootstrap rows for each remaining coefficient.

The plot is a simple line + point chart with labels, ordered by decreasing nonzero percentage. It is intended as a quick diagnostic; for publication graphics, you may want to customize the output data frame with your own plotting code.

Value

Invisibly returns a data frame with columns:

- Variable: coefficient name (excluding the intercept).
- Percent of Bootstraps Nonzero: percentage (0–100) of bootstrap fits in which |beta| > tol.

If no non-intercept coefficients are found (for example, if only the intercept is present), an empty data frame is returned and a message is issued.

See Also

coef.svem_model for ensemble-averaged (optionally debiased) coefficients.

Examples

```
set.seed(10)
n  <- 60
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- rnorm(n)
eps <- rnorm(n, sd = 0.4)
y   <- 0.7 + 1.5*x1 - 0.8*x2 + 0.05*x3 + eps
dat <- data.frame(y, x1, x2, x3)

fit <- SVEMnet(y ~ x1 + x2 + x3, data = dat,
               nBoot = 10, glmnet_alpha = 1, relaxed = FALSE)

nz <- svem_nonzero(fit, tol = 1e-7, plot = FALSE, print_table = TRUE)
head(nz)
```

svem_random_table_multi *Generate a Random Prediction Table from Multiple SVEMnet Models (no refit)*

Description

Samples the original predictor factor space cached in fitted `svem_model` objects and computes predictions from each model at the same random points. This is intended for multiple responses built over the same factor space and a deterministic factor expansion (for example via a shared `bigexp_terms`), so that a shared sampling schema is available.

Usage

```
svem_random_table_multi(
  objects,
  n = 1000,
  mixture_groups = NULL,
  debias = FALSE,
  range_tol = 1e-08,
  numeric_sampler = c("random", "uniform")
)
```

Arguments

`objects`	A list of fitted `svem_model` objects returned by `SVEMnet()`. Each object must contain a valid `$sampling_schema` produced by `SVEMnet()`. A single model is also accepted and treated as a length-one list.
`n`	Number of random points to generate (rows in the output tables). Default is `1000`.
`mixture_groups`	Optional list of mixture constraint groups. Each group is a list with a character vector `vars`, numeric vectors `lower` and `upper` in the same order as `vars`, and a numeric scalar `total`. Bounds default to 0 and 1, and the total defaults to 1. Mixture variables must be numeric-like and must also appear in the models' `predictor_vars` (that is, they must be used as predictors in all models).
`debias`	Logical; if `TRUE`, apply each model's calibration during prediction when available (for Gaussian fits). This is passed to `predict.svem_model()`. Default is `FALSE`.
`range_tol`	Numeric tolerance for comparing numeric ranges across models (used when checking that all `$sampling_schema$num_ranges` agree). Default is `1e-8`.
`numeric_sampler`	Sampler for non-mixture numeric predictors: `"random"` (default), or `"uniform"`. • `"random"`: random Latin hypercube when the **lhs** package is available; otherwise independent uniforms via `runif()`. • `"uniform"`: independent uniform draws within numeric ranges (fastest; no **lhs** dependency).

Details

No refitting is performed. Predictions are obtained by averaging per-bootstrap member predictions on the requested scale. Intercept-only models return n rows with no predictor columns and constant predictions.

All models must share an identical predictor schema. Specifically, their `$sampling_schema` entries must agree on:

- The same `predictor_vars` in the same order.
- The same `var_classes` for each predictor.
- Identical factor `levels` and level order for all categorical predictors.
- Numeric `num_ranges` that match within `range_tol` for all continuous predictors.
- When present, the same `blocking` set (up to order).

The function stops with an informative error message if any of these checks fail.

Discrete numeric predictors (automatic). If any supplied model stores discrete-numeric sampling information in its `$sampling_schema`, this function will automatically respect it (no separate user argument).

This information is stored as:

- `$sampling_schema$discrete_numeric`: a character vector of discrete numeric variable names; and
- `$sampling_schema$discrete_levels`: a named list mapping those names to allowed numeric values.

(Older objects may use `$sampling_schema$discrete_values` instead of `discrete_levels`; this function accepts both for backward compatibility.)

Discrete numeric variables are sampled independently (uniform over allowed values) and are excluded from Latin hypercube sampling; LHS (when used) is applied only to the remaining continuous numeric predictors. Discrete numeric variables are not allowed to be mixture variables.

Models may be Gaussian or binomial. For binomial fits, predictions are returned on the probability scale (that is, on the response scale) by default, consistent with the default behavior of `predict.svem_model()`.

Value

A list with three data frames:

- `data`: the sampled predictor settings, one row per random point.
- `pred`: one column per response, aligned to `data` rows.
- `all`: `cbind(data, pred)` for convenience.

Each prediction column is named by the model's response (left-hand side) with a "_pred" suffix (for example, "y1_pred"). If that name would collide with a predictor name or with another prediction column, the function stops with an error and asks the user to rename the response or predictor.

Typical workflow

1. Build a deterministic expansion (for example with bigexp_terms) and fit several SVEMnet() models for different responses on the same factor space, using the same expansion / sampling settings.
2. Ensure that the fitted models were created with a version of SVEMnet() that populates $sampling_schema.
3. Collect the fitted models in a list and pass them to svem_random_table_multi().
4. Use $data (predictors), $pred (response columns), or $all (cbind(data, pred)) for downstream plotting, summarization, or cross-response comparison.

Blocking variables

If the models were fit using a bigexp_spec that included blocking variables (for example blocking = c("Operator", "Plate_ID")) and SVEMnet() stored these in $sampling_schema$blocking, then svem_random_table_multi() will:

- treat those variables as blocking factors; and
- hold them fixed at a single value across the sampled table.

Specifically:

- For blocking numeric variables, the function uses the midpoint of the recorded numeric range, (min + max) / 2, for all rows. If the variable also has stored discrete support, the midpoint is snapped deterministically to the nearest allowed discrete value.
- For blocking categorical variables, the function uses a single reference level equal to the most frequent observed level (mode) in the training data, with ties broken deterministically; if the mode is unavailable, it falls back to the first stored level.

Blocking variables are not allowed to appear in mixture_groups. If any mixture group tries to use a blocking variable, the function stops with an error.

When no blocking information is present in $sampling_schema (for example for models fit without a bigexp_spec or without blocking), all predictors are sampled using their stored numeric ranges, discrete supports, or categorical levels and the chosen numeric_sampler.

---

svem_score_random *Random-search scoring for SVEM models*

Description

Draw random points from the SVEM sampling schema, compute multi-response desirability scores and (optionally) whole-model-test (WMT) reweighted scores, and attach a scalar uncertainty measure based on percentile CI widths. This function does *not* choose candidates; see svem_select_from_score_table for selection and clustering.

Predictions used inside this scorer are always generated with debiasing disabled (i.e., debias = FALSE) regardless of whether the underlying SVEM fits support calibration.

When specs is supplied, the function also attempts to append mean-level "in spec" probabilities and related joint indicators using the SVEM bootstrap ensemble via svem_append_design_space_cols.

These quantities reflect uncertainty on the *process mean* at each sampled setting under the fitted SVEM models, not unit-level predictive probabilities. If any error occurs in this spec-limit augmentation, it is caught with a warning regardless of `verbose`, and the affected table(s) are returned without the spec-related columns.

Usage

```
svem_score_random(
  objects,
  goals,
  data = NULL,
  n = 50000,
  mixture_groups = NULL,
  level = 0.95,
  combine = c("geom", "mean"),
  numeric_sampler = c("random", "uniform"),
  wmt = NULL,
  verbose = TRUE,
  specs = NULL
)
```

Arguments

`objects`	List of `svem_model` objects (from `SVEMnet`). When unnamed, `svem_score_random()` attempts to infer response names from the left-hand sides of the model formulas. Names (when present) are treated as response identifiers and should typically match the model response names. All models must share a common sampling schema (predictor set, factor levels, numeric ranges) compatible with `svem_random_table_multi`.
`goals`	List of per-response goal specifications. Either: • a named list, where names may be either `names(objects)` or the left-hand-side response names from the fitted models; or • an unnamed list with the same length as `objects`, in which case entries are matched to models by position. Each `goals[[response]]` must be a list with at least: • `goal`: one of `"max"`, `"min"`, `"target"`; • `weight`: nonnegative numeric weight. For `goal = "target"`, also provide `target`. Optional Derringer–Suich controls: • For `"max"` or `"min"`: `lower_acceptable`, `upper_acceptable`, `shape`. • For `"target"`: `tol` (symmetric), or `tol_left` / `tol_right`, and `shape_left` / `shape_right`. When anchors/tolerances are not supplied, robust defaults are inferred from the sampled table using the q0.02–q0.98 span.
`data`	Optional data frame. When supplied (regardless of whether `wmt` is used), it is scored and returned as `original_data_scored`, with predictions (in `<resp>_pred` columns), per-response desirabilities, `score` (and `wmt_score` if applicable), and

uncertainty_measure appended. When specs is supplied and the spec-limit augmentation succeeds, the same mean-level spec columns as in score_table (per-response <resp>_p_in_spec_mean, <resp>_in_spec_point, and joint p_joint_mean, joint_in_spec_point) are appended as well. Recognized columns from an earlier scoring round are replaced with a warning. A generated output name that collides with a model predictor or an observed response variable is rejected before sampling.

n — Number of random samples to draw in the predictor space. This is the number of rows in the sampled table used for scoring.

mixture_groups — Optional mixture and simplex constraints passed to svem_random_table_multi. Each group typically specifies mixture variable names, bounds, and a total.

level — Confidence level for percentile intervals used in the CI width and uncertainty calculations. Default 0.95.

combine — How to combine per-response desirabilities into a scalar score. One of:

- "geom": weighted geometric mean (default);
- "mean": weighted arithmetic mean.

numeric_sampler — Character string controlling how numeric predictors are sampled inside svem_random_table_multi. One of:

- "random": Latin hypercube sampling when **lhs** is available, otherwise independent uniforms;
- "uniform": independent uniforms over stored numeric ranges.

wmt — Optional object returned by svem_wmt_multi. When non-NULL, its multipliers (and p_values, if present) are aligned to names(objects) and used to define WMT weights, wmt_score. When NULL, only user weights are used and no WMT reweighting is applied.

verbose — Logical; if TRUE, print a compact summary of the run (and any WMT diagnostics from upstream) to the console.

specs — Optional named list of specification objects, one per response in objects for which you want to define a mean-level spec constraint. Each entry should be either NULL (no specs for that response) or a list with components:

- lower: numeric lower limit (may be -Inf, NA, or NULL for a one-sided upper spec);
- upper: numeric upper limit (may be Inf, NA, or NULL for a one-sided lower spec).

Names of specs, when provided, should be a subset of names(objects) or of the model response names (left-hand sides). The specification structure matches that used by svem_append_design_space_cols.

Details

Typical workflow: A common pattern is:

1. Fit one or more SVEMnet() models for the responses of interest.
2. Call svem_score_random() to:

- draw candidate settings in factor space,
- compute Derringer–Suich (DS) desirabilities and a combined multi-response score, and
- attach a scalar uncertainty measure derived from percentile CI widths.

3. Optionally provide `specs` to append mean-level "in spec" probabilities and joint indicators based on the SVEM bootstrap ensemble (process-mean assurance).
4. Use `svem_select_from_score_table` to:
   - select one "best" row (e.g., maximizing `score` or `wmt_score`), and
   - pick a small, diverse set of medoid candidates for optimality or exploration (e.g. high `uncertainty_measure`).

   The robust bounded summaries `score`, per-response desirabilities, per-response weighted CI-width summaries, and `uncertainty_measure` may contain ties at 0 or 1 because of the percentile clipping described below. If the intended single "best" row is the largest raw predicted response, select on the corresponding `<response>_pred` column. If the intended single exploration row is the widest interval, create and select on a raw width column such as `<response>_ciw = <response>_upr - <response>_lwr`.
5. Run selected candidates, append the new data, refit the SVEM models, and repeat as needed.

Multi-response desirability scoring: Each response is mapped to a Derringer–Suich desirability $d_r \in [0, 1]$ according to its goal:

- `goal = "max"`: larger values are better;
- `goal = "min"`: smaller values are better;
- `goal = "target"`: values near a target are best.

Per-response anchors (acceptable lower/upper limits or target-band tolerances) can be supplied in `goals`; when not provided, robust defaults are inferred from the sampled responses using the q0.02–q0.98 span. As a result, the largest few percent of predictions may share desirability 1 (and the smallest few percent may share desirability 0) when default anchors are used. This is intentional for robust desirability scoring. Use `<response>_pred` directly when selecting the single largest or smallest predicted response.

Per-response desirabilities are combined into a single scalar `score` using either:

- a weighted arithmetic mean (`combine = "mean"`), or
- a weighted geometric mean (`combine = "geom"`), with a small floor applied inside the log to avoid `log(0)`.

User-provided weights in `goals[[resp]]$weight` are normalized to sum to one and always define `weights_original` and the user-weighted `score`.

Whole-model reweighting (WMT): When a WMT object from `svem_wmt_multi` is supplied via the `wmt` argument, each response receives a multiplier derived from its whole-model p-value. Final WMT weights are proportional to the product of the user weight and the multiplier, then renormalized to sum to one:

$$w_r^{(\text{final})} \propto w_r^{(\text{user})} \times m_r,$$

where m_r comes from `wmt$multipliers`. The user weights always define `score`; the WMT-adjusted weights define `wmt_score`. The uncertainty measure is always weighted using the user weights, even when WMT is supplied.

Binomial responses. If any responses are fitted with `family = "binomial"`, supplying a non-`NULL` `wmt` object is not allowed and the function stops with a clear error. Predictions and CI bounds for binomial responses are interpreted on the probability (response) scale and clamped to `[0, 1]` before desirability and uncertainty calculations.

Uncertainty measure: The `uncertainty_measure` is a weighted sum of robustly normalized percentile CI widths across responses. For each response, we compute the bootstrap percentile CI width $\mathrm{CIwidth}_r(x) = u_r(x) - \ell_r(x)$ and then map it to the unit interval using an affine rescaling based on the empirical q0.02 and q0.98 quantiles of the CI widths for that response. The quantile anchors are computed once from the sampled `score_table` and reused when scoring `data`, so `uncertainty_measure` in `original_data_scored` is on the same scale as in `score_table`:

$$\tilde{W}_r(x) = \frac{\min\{\max(\mathrm{CIwidth}_r(x), q_{0.02}(r)), q_{0.98}(r)\} - q_{0.02}(r)}{q_{0.98}(r) - q_{0.02}(r)}.$$

The scalar `uncertainty_measure` is then

$$\text{uncertainty}(x) = \sum_r w_r \, \tilde{W}_r(x),$$

where w_r are the user-normalized response weights derived from `goals[[resp]]$weight`. Larger values of `uncertainty_measure` indicate settings where the ensemble CI is relatively wide compared to the response's typical scale and are natural targets for exploration. Because the normalization clips at empirical q0.02 and q0.98, the widest few percent of intervals may share `uncertainty_measure = 1`. Use a raw interval width column such as `<response>_ciw = <response>_upr - <response>_lwr` when selecting the single widest interval. If a model has fewer than two bootstrap members or its interval prediction fails, the CI columns are `NA`. Its uncertainty contribution is the neutral value `0.5` times its user weight, and a warning identifies the affected response. These fallback values do not measure fitted uncertainty.

Spec-limit mean-level probabilities: If `specs` is provided, `svem_score_random()` attempts to pass the scored table and models to `svem_append_design_space_cols` to compute, for each response with an active spec:

- `<resp>_p_in_spec_mean`: estimated probability (under the SVEM bootstrap ensemble) that the process mean at a setting lies within the specified interval;
- `<resp>_in_spec_point`: 0/1 indicator that the point prediction lies within the same interval.

and joint quantities:

- `p_joint_mean`: product of per-response mean-level probabilities over responses with active specs;
- `joint_in_spec_point`: 0/1 indicator that all point predictions are in spec across responses with active specs.

Names in `specs` may refer either to `names(objects)` or to the model response names; they are automatically aligned to the fitted models.

These probabilities are defined on the *conditional means* at each sampled setting, not on individual units or lots, and are best interpreted as ensemble-based assurance measures under the SVEM + FRW pipeline. If the augmentation step fails for any reason (for example, missing predictor columns or incompatible models), the error is caught with a warning regardless of `verbose`, and `score_table` and/or `original_data_scored` are returned without the spec-related columns.

Value

A list with components:

`score_table` Data frame with predictors, predicted responses (columns `<resp>_pred` for each `resp` in `names(objects)`), per-response desirabilities, `score`, optional `wmt_score`, and `uncertainty_measure`. For each response `r` in `names(objects)`, additional columns `r_lwr`, `r_upr` (percentile CI bounds at level `level`) and `r_ciw_w` (weighted, normalized CI width contribution to `uncertainty_measure`) are appended. When `specs` is supplied and the spec-limit augmentation succeeds, additional columns `<resp>_p_in_spec_mean`, `<resp>_in_spec_point`, `p_joint_mean`, and `joint_in_spec_point` are appended.

`original_data_scored` If `data` is supplied, that data augmented with prediction columns `<resp>_pred`, per-response desirabilities, `score`, optional `wmt_score`, and `uncertainty_measure`; otherwise `NULL`. When `specs` is supplied and the spec-limit augmentation succeeds, the same mean-level spec columns as in `score_table` are appended to `original_data_scored` as well.

`weights_original` User-normalized response weights.

`weights_final` Final weights after WMT, if `wmt` is supplied; otherwise equal to `weights_original`.

`wmt_p_values` Named vector of per-response whole-model p-values when `wmt` is supplied and contains `p_values`; otherwise `NULL`.

`wmt_multipliers` Named vector of per-response WMT multipliers when `wmt` is supplied; otherwise `NULL`.

Acknowledgments

Generative AI tools were used to assist with coding and roxygen documentation; all content was reviewed and finalized by the author.

References

Karl, A. T. (2026). SVEMnet: An R package for self-validated elastic-net ensembles and multi-response optimization in small-sample mixture-process experiments. *Chemometrics and Intelligent Laboratory Systems*, *271*, 105660. doi:10.1016/j.chemolab.2026.105660

Gotwalt, C., & Ramsey, P. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-Using/ev-p/849873/redirect_from_archived_page/true

Karl, A. T. (2024). A randomized permutation whole-model test heuristic for Self-Validated Ensemble Models (SVEM). *Chemometrics and Intelligent Laboratory Systems*, *249*, 105122. doi:10.1016/j.chemolab.2024.105122

Karl, A., Wisnowski, J., & Rushing, H. (2022). JMP Pro 17 Remedies for Practical Struggles with Mixture Experiments. JMP Discovery Conference. doi:10.13140/RG.2.2.34598.40003/1

Lemkus, T., Gotwalt, C., Ramsey, P., & Weese, M. L. (2021). Self-Validated Ensemble Models for Design of Experiments. *Chemometrics and Intelligent Laboratory Systems*, 219, 104439. doi:10.1016/j.chemolab.2021.104439

Xu, L., Gotwalt, C., Hong, Y., King, C. B., & Meeker, W. Q. (2020). Applications of the Fractional-Random-Weight Bootstrap. *The American Statistician*, 74(4), 345–358. doi:10.1080/00031305.2020.1731599

Ramsey, P., Gaudard, M., & Levin, W. (2021). Accelerating Innovation with Space Filling Mixture Designs, Neural Networks and SVEM. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Accelerating-Innovation-with-Space-Filling-Mixture-Designs/ev-p/756841

Ramsey, P., & Gotwalt, C. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-ev-p/849647/redirect_from_archived_page/true

Ramsey, P., Levin, W., Lemkus, T., & Gotwalt, C. (2021). SVEM: A Paradigm Shift in Design and Analysis of Experiments. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/SVEM-A-Paradigm-Shift-in-Design-and-Analysis-of-Experiments-2021/ev-p/756634

Ramsey, P., & McNeill, P. (2023). CMC, SVEM, Neural Networks, DOE, and Complexity: It's All About Prediction. *JMP Discovery Conference*.

Friedman, J. H., Hastie, T., and Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1), 1-22.

Meinshausen, N. (2007). Relaxed Lasso. Computational Statistics & Data Analysis, 52(1), 374-393.

Kish, L. (1965). *Survey Sampling*. Wiley.

Lumley, T. (2004). Analysis of complex survey samples. *Journal of Statistical Software*, 9(1), 1–19.

Lumley, T. and Scott, A. (2015). AIC and BIC for modelling with complex survey data. *Journal of Survey Statistics and Methodology*, 3(1), 1–18.

Tibshirani, R. J. and Taylor, J. (2012). Degrees of freedom in lasso problems. *The Annals of Statistics*, 40(2), 1198–1232. doi:10.1214/12AOS1003

See Also

SVEMnet, svem_random_table_multi, svem_select_from_score_table, svem_append_design_space_cols(), svem_wmt_multi

Examples

```
## Small runnable two-response example
set.seed(11)
n_toy <- 36
toy <- data.frame(x1 = runif(n_toy), x2 = runif(n_toy))
toy$yield <- 1 + 2 * toy$x1 - toy$x2 + rnorm(n_toy, sd = 0.15)
toy$impurity <- 1.5 - toy$x1 + 0.5 * toy$x2 + rnorm(n_toy, sd = 0.10)

fit_yield <- SVEMnet(yield ~ x1 + x2, toy, nBoot = 5,
                     glmnet_alpha = 1, relaxed = FALSE)
fit_impurity <- SVEMnet(impurity ~ x1 + x2, toy, nBoot = 5,
                        glmnet_alpha = 1, relaxed = FALSE)

toy_scored <- svem_score_random(
  objects = list(yield = fit_yield, impurity = fit_impurity),
  goals = list(
    yield = list(goal = "max", weight = 0.7),
    impurity = list(goal = "min", weight = 0.3)
  ),
  n = 100,
  numeric_sampler = "uniform",
```

```
  verbose = FALSE
)
head(toy_scored$score_table)

## Not run:
## Production-scale lipid workflow. This is not run automatically because
## it fits three wide ensembles and optionally performs permutation WMTs.
## ---------------------------------------------------------------------------
## Multi-response SVEM scoring with Derringer-Suich desirabilities
## ---------------------------------------------------------------------------

data(lipid_screen)

# Build a deterministic expansion once and reuse for all responses
spec <- bigexp_terms(
  Potency ~ PEG + Helper + Ionizable + Cholesterol +
    Ionizable_Lipid_Type + N_P_ratio + flow_rate,
  data             = lipid_screen,
  factorial_order  = 3,
  polynomial_order = 3,
  include_pc_2way  = TRUE,
  include_pc_3way  = FALSE
)

form_pot <- bigexp_formula(spec, "Potency")
form_siz <- bigexp_formula(spec, "Size")
form_pdi <- bigexp_formula(spec, "PDI")

set.seed(1)
fit_pot <- SVEMnet(form_pot, lipid_screen)
fit_siz <- SVEMnet(form_siz, lipid_screen)
fit_pdi <- SVEMnet(form_pdi, lipid_screen)

# Collect SVEM models in a named list by response
objs <- list(Potency = fit_pot, Size = fit_siz, PDI = fit_pdi)

# Targets and user weights for Derringer-Suich desirabilities
goals <- list(
  Potency = list(goal = "max", weight = 0.6),
  Size    = list(goal = "min", weight = 0.3),
  PDI     = list(goal = "min", weight = 0.1)
)

# Optional mixture constraints (composition columns sum to 1)
mix <- list(list(
  vars  = c("PEG", "Helper", "Ionizable", "Cholesterol"),
  lower = c(0.01, 0.10, 0.10, 0.10),
  upper = c(0.05, 0.60, 0.60, 0.60),
  total = 1.0
))

# Basic random-search scoring without WMT or design-space specs
set.seed(3)
```

```
scored_basic <- svem_score_random(
  objects         = objs,
  goals           = goals,
  n               = 10000,          # number of random candidates
  mixture_groups  = mix,
  combine         = "geom",
  numeric_sampler = "random",
  verbose         = FALSE
)

# Scored candidate table: predictors, <resp>_pred, <resp>_des, score, uncertainty
names(scored_basic$score_table)
head(scored_basic$score_table)

# Scored original data (if 'data' is supplied)
# scored_basic$original_data_scored contains predictions + desirabilities

## ---------------------------------------------------------------------------
## With whole-model tests (WMT) and process-mean specifications
## ---------------------------------------------------------------------------

set.seed(123)
wmt_out <- svem_wmt_multi(
  formulas       = list(Potency = form_pot,
                        Size    = form_siz,
                        PDI     = form_pdi),
  data           = lipid_screen,
  mixture_groups = mix,
  wmt_control    = list(seed = 123, nCore = 1L),
  plot           = FALSE,
  verbose        = FALSE
)

# Simple process-mean specs for a joint design space:
#   Potency >= 78, Size <= 100, PDI <= 0.25
specs_ds <- list(
  Potency = list(lower = 78),
  Size    = list(upper = 100),
  PDI     = list(upper = 0.25)
)

set.seed(4)
scored_full <- svem_score_random(
  objects         = objs,
  goals           = goals,
  data            = lipid_screen,  # score the original runs as well
  n               = 25000,
  mixture_groups  = mix,
  level           = 0.95,
  combine         = "geom",
  numeric_sampler = "random",
  wmt             = wmt_out,       # optional: WMT reweighting
  specs           = specs_ds,      # optional: design-space columns
```

```
  verbose         = TRUE
)

# The scored table now includes:
#  * score, wmt_score, uncertainty_measure
#  * per-response CIs: <resp>_lwr, <resp>_upr
#  * design-space columns, e.g. Potency_p_in_spec_mean, p_joint_mean
names(scored_full$score_table)

## ----------------------------------------------------------------------
## Positional (unnamed) goals matched to objects by position
## ----------------------------------------------------------------------

data(lipid_screen)

# Build a deterministic expansion once and reuse for all responses
spec <- bigexp_terms(
  Potency ~ PEG + Helper + Ionizable + Cholesterol +
    Ionizable_Lipid_Type + N_P_ratio + flow_rate,
  data             = lipid_screen,
  factorial_order  = 3,
  polynomial_order = 3,
  include_pc_2way  = TRUE,
  include_pc_3way  = FALSE
)

form_pot <- bigexp_formula(spec, "Potency")
form_siz <- bigexp_formula(spec, "Size")
form_pdi <- bigexp_formula(spec, "PDI")

set.seed(1)
fit_pot <- SVEMnet(form_pot, lipid_screen)
fit_siz <- SVEMnet(form_siz, lipid_screen)
fit_pdi <- SVEMnet(form_pdi, lipid_screen)

# Collect SVEM models in a list.
# Here goals will be matched by position: Potency, Size, PDI.
objs <- list(fit_pot, fit_siz, fit_pdi)

# Positional goals (unnamed list): must have same length as 'objects'
goals_positional <- list(
  list(goal = "max", weight = 0.6),  # for Potency (objs[[1]])
  list(goal = "min", weight = 0.3),  # for Size    (objs[[2]])
  list(goal = "min", weight = 0.1)   # for PDI     (objs[[3]])
)

set.seed(5)
scored_pos <- svem_score_random(
  objects         = objs,
  goals           = goals_positional,
  n               = 5000,
  numeric_sampler = "random",
  verbose         = FALSE
```

```
)

names(scored_pos$score_table)


## End(Not run)
```

`svem_select_from_score_table` *Select best row and diverse candidates from an SVEM score table*

Description

Given a scored random-search table (e.g. from `svem_score_random()`), pick a single "best" row under a chosen objective column and sample a small, diverse set of medoid candidates from the top of that ranking. Any per-response CI columns (e.g. `*_lwr` / `*_upr`) present in `score_table` are carried through unchanged.

Optionally, a string `label` can be supplied to annotate the returned `best` row and `candidates` by appending that label to a `"Notes_from_SVEMnet"` column. If `"Notes_from_SVEMnet"` is missing, it is created. If it exists and is nonempty, the label is appended with `"; "` as a separator.

Usage

```
svem_select_from_score_table(
  score_table,
  target = "score",
  direction = c("max", "min"),
  k = 5,
  top_type = c("frac", "n"),
  top = 0.1,
  predictor_cols = NULL,
  label = NULL
)
```

Arguments

`score_table`	Data frame with predictors, responses, scores, and `uncertainty_measure`, typically `scored$score_table` from `svem_score_random`. When medoids are requested (`k > 0`), the predictor columns used for clustering are taken from the `"svem_predictor_cols"` attribute by default. If that attribute is missing, a heuristic is used. If you accidentally pass the full `scored` list, a helpful error is thrown reminding you to use `scored$score_table`.
`target`	Character scalar naming the column in `score_table` to optimize (e.g. `"score"`, `"wmt_score"`, `"uncertainty_measure"`, or a response-specific column such as `"Y_pred"`). The robust bounded columns created by `svem_score_random` may

	contain ties at 0 or 1 because of percentile clipping. For a single largest predicted response, use <response>_pred; for a single widest interval, create a raw width column such as <response>_ciw = <response>_upr - <response>_lwr and use that as target.
direction	Either "max" or "min" indicating whether larger or smaller values of target are preferred.
k	Integer; desired number of medoid candidates to return. If k <= 0, only the best row is returned and no clustering is performed.
top_type	Either "frac" or "n" specifying whether top is a fraction of rows or an integer count.
top	Value for the top set: a fraction in (0,1] if top_type = "frac", or an integer >= 1 if top_type = "n".
predictor_cols	Optional character vector of predictor column names used to measure diversity in the PAM step when k > 0. When NULL (default), the function first tries attr(score_table, "svem_predictor_cols"). If that is unavailable, it falls back to a heuristic that prefers non-derived predictor columns (excluding e.g. *_pred, *_des, *_lwr, *_upr, *_ciw_w, *_p_in_spec_mean, *_in_spec_point, score, wmt_score, uncertainty_measure, p_joint_mean, joint_in_spec_point, candidate_type, selection_label, Notes_from_SVEMnet). If no usable predictor columns can be inferred, a warning is issued and only best is returned.
label	Optional character scalar. When non-NULL, this label is appended into a "Notes_from_SVEMnet" column for the returned best row and candidates. If "Notes_from_SVEMnet" is missing, it is created; if present and nonempty, the label is appended using "; " as separator.

Details

svem_score_random() returns both response prediction columns (for example, Y_pred) and robust bounded summaries such as score, per-response desirabilities, per-response weighted CI-width summaries, and uncertainty_measure. The bounded summaries are useful for defining high-desirability or high-uncertainty regions for medoid selection, but they may not identify a unique single best row when several rows are clipped to 1. In that case, select on a raw column such as Y_pred, or add a raw CI-width column such as Y_ciw = Y_upr - Y_lwr, before calling this function.

Value

A list with components:

best One-row data frame at the optimum of target under the specified direction, including any columns present in score_table (e.g. *_lwr / *_upr).

candidates Data frame of medoid candidates (possibly empty or NULL) drawn from the top top of the ranking on target, with all columns carried through from score_table.

label The label value supplied to this call (or NULL); used by svem_export_candidates_csv for the selection_label column.

call The matched call, including all arguments used to create this selection object.

See Also

svem_score_random, svem_select_candidates()

---

svem_significance_test_parallel *SVEM whole-model significance test with mixture support (parallel)*

Description

Perform a permutation-based whole-model significance test for a continuous (Gaussian) SVEM fit, with optional mixture-factor groups and parallel SVEM refits.

Usage

```
svem_significance_test_parallel(
  formula,
  data,
  mixture_groups = NULL,
  nPoint = 2000,
  nSVEM = 10,
  nPerm = 150,
  percent = 90,
  nBoot = 100,
  glmnet_alpha = c(1),
  weight_scheme = c("SVEM"),
  objective = c("wAIC", "wBIC", "wSSE", "auto"),
  relaxed = FALSE,
  verbose = TRUE,
  nCore = 1L,
  seed = NULL,
  rng_sample_kind = c("Rejection", "Rounding"),
  spec = NULL,
  response = NULL,
  use_spec_contrasts = TRUE,
  ...
)
```

Arguments

formula	A model formula. If spec is provided, the right-hand side is ignored and replaced by the locked expansion in spec. A bigexp_spec attached by bigexp_formula() or stored on spec$formula is also honored. Response transformations are retained unless response explicitly overrides the response column.
data	A data frame containing the variables in the model.
mixture_groups	Optional list describing one or more mixture-factor groups. Each element should be a list with components:

- `vars`: character vector of column names;
- `lower`: numeric vector of lower bounds (same length as `vars`);
- `upper`: numeric vector of upper bounds (same length as `vars`);
- `total`: scalar specifying the sum of the mixture variables.

All mixture variables must appear in exactly one group. Defaults to `NULL`.

`nPoint` Number of random evaluation points in the factor space (default `2000`); must be a positive integer.

`nSVEM` Number of SVEM fits on the original (unpermuted) data used to summarize the observed surface (default `10`); must be at least one.

`nPerm` Number of SVEM fits on permuted responses used to build the null reference distribution (default `150`); must be at least five. Values below 20 are permitted for tests but trigger a warning because they are too small for production inference.

`percent` Percentage of variance to capture in the SVD of the permutation surfaces (default `90`); must lie strictly between zero and 100.

`nBoot` Number of bootstrap iterations within each inner SVEM fit (default `100`); must be at least two because member standard deviations are required.

`glmnet_alpha` Numeric vector of `glmnet` alpha values (default `c(1)`).

`weight_scheme` Weighting scheme for the inner SVEM fits. Only `"SVEM"` is accepted: the whole-model test in Karl (2024) is defined for the anti-correlated fractional-random-weight scheme, so the other `SVEMnet()` weighting schemes are intentionally not offered here.

`objective` Objective used inside `SVEMnet()` to pick the bootstrap path solution. One of `"wAIC"`, `"wBIC"`, or `"wSSE"` (default `"wAIC"`).

`relaxed` Logical; default `FALSE`. When `TRUE`, inner `SVEMnet()` fits use `glmnet`'s relaxed elastic-net path and select both `lambda` and relaxed `gamma` on each bootstrap. When `FALSE`, the standard `glmnet` path is used. If `relaxed = TRUE` and `glmnet_alpha` includes `0`, ridge (`alpha = 0`) is dropped by `SVEMnet()` for relaxed fits.

`verbose` Logical; if `TRUE`, display progress messages (default `TRUE`).

`nCore` Number of CPU cores for processing. Default is `1`, which runs in-process and is suitable for CRAN and other shared systems. Values greater than one use a private cluster local to this call.

`seed` Optional integer seed for reproducible RNG (default `NULL`). Deterministic per-fit seeds yield reproducibility regardless of parallel scheduling and core count.

`rng_sample_kind`

Sampling algorithm used with the `"L'Ecuyer-CMRG"` RNG. The default `"Rejection"` uses R's modern unbiased discrete sampler. Use `"Rounding"` to reproduce legacy WMT streams from SVEMnet 3.3.1 and earlier.

`spec` Optional `bigexp_spec` created by `bigexp_terms()`. If provided, the test reuses its locked expansion while preserving the left-hand side of `formula`, unless `response` is supplied. If omitted, a valid spec attached to `formula` is used automatically. Categorical sampling uses `spec$levels`, and numeric sampling

prefers spec$num_range when available. Discrete numeric predictors recorded by bigexp_terms() (spec$settings$discrete_numeric + spec$settings$discrete_levels) are sampled only from their recorded allowed levels when building the evaluation grid. Blocking variables recorded in spec$settings$blocking are treated as nuisance factors and held fixed at a single reference value across the evaluation grid (numeric blocks at the range midpoint, snapped to the discrete support when recorded; categorical blocks at the most frequent training level), matching svem_random_table_multi(). They are never swept, so between-block variation does not contribute to the whole-model distances.

response
: Optional character name for the response variable to use when spec is supplied. If omitted, the response is taken from the left-hand side of formula.

use_spec_contrasts
: Logical; default TRUE. When spec is supplied and use_spec_contrasts = TRUE, the function replays spec$settings$contrasts_options on the parallel workers for deterministic factor coding.

...
: Additional arguments passed to SVEMnet() and then to glmnet() (for example: penalty.factor, lower.limits, upper.limits, standardize.response, etc.). The relaxed setting is controlled by the relaxed argument of this function. User weights and offset are not supported (SVEM controls its own weighting). Argument names that neither SVEMnet() nor the installed glmnet recognizes are ignored with a warning (misspelling protection).

Details

The procedure follows Karl (2024): it generates a space-filling evaluation grid in the factor space, fits multiple SVEM models on the original data and on permuted responses, standardizes grid predictions, reduces them via an SVD-based low-rank representation, and summarizes each fit by a Mahalanobis-type distance in the reduced space. A flexible SHASHo distribution is then fit to the permutation distances and used to obtain a whole-model p-value for the observed surface.

Because the test is based on a finite number of permutations and a fitted null distribution, the reported p-values are approximate and are intended as a diagnostic measure of global factor signal, not as exact hypothesis tests.

With nCore = 1, SVEM refits run in-process. With more than one core, the function uses a private PSOCK cluster and parallel::parLapplyLB(). It does not register or change a foreach backend.

Reproducible parallel RNG (Windows/macOS/Linux): when seed is supplied, the function sets its private computation stream to RNGkind("L'Ecuyer-CMRG", sample.kind = rng_sample_kind) and generates a deterministic, per-iteration seed schedule on the master. Each fit slot calls set.seed() with its assigned seed before performing random draws (including permutations and the bootstrap randomness inside SVEMnet()). This makes results reproducible regardless of worker scheduling and independent of the number of cores. The caller's RNG kind and .Random.seed are restored on exit, including error exits. A failed fit slot is retried twice with deterministic sub-seeds while its response permutation remains fixed. If all three attempts fail, the test aborts with a failure summary rather than silently reducing nSVEM or nPerm. Likewise, a non-converged SHASHo null-distribution fit is reported as an error rather than being used to calculate a whole-model p-value. The SHASHo null distribution is fitted by independently written maximum-likelihood code based on the sinh-arcsinh-normal transformation of Jones and Pewsey (2009), doi:10.1093/biomet/asp053. It uses stats::nlminb(), deterministic multiple starts, and numerical derivatives. No source code from

the former GAMLSS dependencies is used. The distribution and distance statistic are unchanged, but fitted parameters and p-values can differ from versions before 3.6.1, which used a different fitting algorithm. A fit must satisfy both the optimizer's convergence criterion and a small numerical likelihood gradient in the internally normalized coordinates (location, log(scale/tail), skew, log(tail)). If these checks fail, no p-value is returned; increasing `nPerm` can help resolve a small reference sample.

The function can optionally reuse a deterministic, locked expansion built with `bigexp_terms()`. Supply `spec` (and optionally `response`) to ensure that categorical levels, contrasts, and the polynomial/interaction structure are identical across repeated calls and across multiple responses sharing the same factor space.

Although the implementation calls `SVEMnet()` internally and will technically run for any supported `family`, the significance test is *designed* for continuous (Gaussian) responses and should be interpreted in that setting.

Value

An object of class `"svem_significance_test"`, a list with components:

- `p_value`: median whole-model p-value across the `nSVEM` original SVEM fits.
- `p_values`: numeric vector of length `nSVEM` with the per-fit p-values.
- `d_Y`: numeric vector of distances for the original SVEM fits.
- `d_pi_Y`: numeric vector of distances for the permutation fits.
- `distribution_fit`: an `svem_shasho_fit` list with `parameters` (`mu`, `sigma`, `nu`, `tau` on their natural scales), `loglik`, `n`, `converged`, `method`, and optimizer diagnostics. This is not a GAMLSS object. `coef(distribution_fit, what = "mu")` (or `"sigma"`, `"nu"`, `"tau"`) returns the intercept on the link scale, as in earlier versions: identity for mu/nu, log for sigma/tau.
- `data_d`: data frame of distances and source labels (original vs permutation), suitable for plotting.
- `diagnostics`: RNG mode, deterministic retry counts and reasons, SHASHo convergence information, removed/retained evaluation-grid columns, and the retained SVD component count.

Acknowledgments

Generative AI tools were used to assist with coding and roxygen documentation; all content was reviewed and finalized by the author.

References

Karl, A. T. (2026). SVEMnet: An R package for self-validated elastic-net ensembles and multi-response optimization in small-sample mixture-process experiments. *Chemometrics and Intelligent Laboratory Systems*, *271*, 105660. doi:10.1016/j.chemolab.2026.105660

Gotwalt, C., & Ramsey, P. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-Using/ev-p/849873/redirect_from_archived_page/true

Karl, A. T. (2024). A randomized permutation whole-model test heuristic for Self-Validated Ensemble Models (SVEM). *Chemometrics and Intelligent Laboratory Systems*, *249*, 105122. doi:10.1016/j.chemolab.2024.105122

Karl, A., Wisnowski, J., & Rushing, H. (2022). JMP Pro 17 Remedies for Practical Struggles with Mixture Experiments. JMP Discovery Conference. doi:10.13140/RG.2.2.34598.40003/1

Lemkus, T., Gotwalt, C., Ramsey, P., & Weese, M. L. (2021). Self-Validated Ensemble Models for Design of Experiments. *Chemometrics and Intelligent Laboratory Systems*, 219, 104439. doi:10.1016/j.chemolab.2021.104439

Xu, L., Gotwalt, C., Hong, Y., King, C. B., & Meeker, W. Q. (2020). Applications of the Fractional-Random-Weight Bootstrap. *The American Statistician*, 74(4), 345–358. doi:10.1080/00031305.2020.1731599

Ramsey, P., Gaudard, M., & Levin, W. (2021). Accelerating Innovation with Space Filling Mixture Designs, Neural Networks and SVEM. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Accelerating-Innovation-with-Space-Filling-Mixture-Designs/ev-p/756841

Ramsey, P., & Gotwalt, C. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-ev-p/849647/redirect_from_archived_page/true

Ramsey, P., Levin, W., Lemkus, T., & Gotwalt, C. (2021). SVEM: A Paradigm Shift in Design and Analysis of Experiments. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/SVEM-A-Paradigm-Shift-in-Design-and-Analysis-of-Experiments-2021/ev-p/756634

Ramsey, P., & McNeill, P. (2023). CMC, SVEM, Neural Networks, DOE, and Complexity: It's All About Prediction. *JMP Discovery Conference*.

Friedman, J. H., Hastie, T., and Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1), 1-22.

Meinshausen, N. (2007). Relaxed Lasso. Computational Statistics & Data Analysis, 52(1), 374-393.

Kish, L. (1965). *Survey Sampling*. Wiley.

Lumley, T. (2004). Analysis of complex survey samples. *Journal of Statistical Software*, 9(1), 1–19.

Lumley, T. and Scott, A. (2015). AIC and BIC for modelling with complex survey data. *Journal of Survey Statistics and Methodology*, 3(1), 1–18.

Tibshirani, R. J. and Taylor, J. (2012). Degrees of freedom in lasso problems. *The Annals of Statistics*, 40(2), 1198–1232. doi:10.1214/12AOS1003

See Also

SVEMnet, bigexp_terms, bigexp_formula

Examples

```
## Not run:
  ## Production WMT inference is intentionally not run in examples because
  ## it requires repeated ensemble fits for many response permutations.
  set.seed(1)
```

```
# Small toy data with a 3-component mixture A, B, C
n <- 40
sample_trunc_dirichlet <- function(n, lower, upper, total) {
  k <- length(lower)
  stopifnot(length(upper) == k, total >= sum(lower), total <= sum(upper))
  avail <- total - sum(lower)
  if (avail <= 0) return(matrix(rep(lower, each = n), nrow = n))
  out <- matrix(NA_real_, n, k)
  i <- 1L
  while (i <= n) {
    g <- rgamma(k, 1, 1)
    w <- g / sum(g)
    x <- lower + avail * w
    if (all(x <= upper + 1e-12)) { out[i, ] <- x; i <- i + 1L }
  }
  out
}

lower <- c(0.10, 0.20, 0.05)
upper <- c(0.60, 0.70, 0.50)
total <- 1.0
ABC   <- sample_trunc_dirichlet(n, lower, upper, total)
A <- ABC[, 1]; B <- ABC[, 2]; C <- ABC[, 3]
X <- runif(n)
F <- factor(sample(c("red", "blue"), n, replace = TRUE))
y <- 2 + 3*A + 1.5*B + 1.2*C + 0.5*X + 1*(F == "red") + rnorm(n, sd = 0.3)
dat <- data.frame(y = y, A = A, B = B, C = C, X = X, F = F)

mix_spec <- list(list(
  vars  = c("A", "B", "C"),
  lower = lower,
  upper = upper,
  total = total
))

## Example 1: direct formula interface (no locked expansion spec)
res1 <- svem_significance_test_parallel(
  y ~ A + B + C + X + F,
  data           = dat,
  mixture_groups = mix_spec,
  glmnet_alpha   = 1,
  weight_scheme  = "SVEM",
  objective      = "auto",
  relaxed        = FALSE,   # default, shown for clarity
  nCore          = 1,
  seed           = 123,
  verbose        = FALSE
)
res1$p_value

## Example 2: using a deterministic bigexp expansion spec
## Build a wide expansion once and reuse it via `spec`
```

```
spec <- bigexp_terms(
  y ~ A + B + C + X + F,
  data             = dat,
  factorial_order  = 2,  # up to 2-way interactions
  polynomial_order = 2   # up to quadratic terms in continuous vars
)

## Run the same significance test, but with the locked expansion:
## - `formula` is still required, but its RHS is ignored when `spec` is given
## - `response` tells the helper which LHS to use with `spec`
res2 <- svem_significance_test_parallel(
  y ~ A + B + C + X + F,
  data               = dat,
  mixture_groups     = mix_spec,
  glmnet_alpha       = 1,
  weight_scheme      = "SVEM",
  objective          = "auto",
  relaxed            = FALSE,
  nCore              = 1,
  seed               = 123,
  spec               = spec,
  response           = "y",
  use_spec_contrasts = TRUE,
  verbose            = FALSE
)
res2$p_value

## End(Not run)
```

`svem_thompson_batch` *Propose a batch of runs by Thompson sampling from SVEM bootstrap ensembles*

Description

Experimental sequential-design proposer for multi-response SVEM workflows. This extends the core workflow described in Karl (2026), doi:10.1016/j.chemolab.2026.105660; its performance is not established by the results in that package paper. Given fitted `SVEMnet` models and per-response desirability goals, `svem_thompson_batch()` proposes `batch_size` new candidate runs by parallel Thompson sampling (Thompson, 1933; Kandasamy et al., 2018) from the SVEM bootstrap ensembles: for each batch slot, one bootstrap member is drawn at random for each response, the drawn members' predictions are mapped to Derringer–Suich desirabilities and combined into a single score, and the candidate maximizing that score is added to the batch (previously selected candidates are excluded).

Each slot re-draws the bootstrap members, allowing candidates preferred by different members to enter the batch without an explicit exploration parameter. Bootstrap members are used as heuristic function draws; they are not samples from a fitted Bayesian posterior. This complements the score/uncertainty workflow of `svem_score_random` and `svem_select_from_score_table`: CI-width exploration targets regions of large global uncertainty, while Thompson sampling refines the current contenders for the optimum.

Usage

```
svem_thompson_batch(
  objects,
  goals,
  batch_size = 4,
  candidates = NULL,
  n = 5000,
  mixture_groups = NULL,
  numeric_sampler = c("random", "uniform"),
  combine = c("geom", "mean"),
  verbose = TRUE
)
```

Arguments

objects	List of svem_model objects (from SVEMnet), one per response, as in svem_score_random. When unnamed, response names are inferred from the model formulas. All models must have bootstrap coefficients (coef_matrix; fit with nBoot >= 2) and, when candidates = NULL, a shared sampling schema.
goals	List of per-response goal specifications in the same format as svem_score_random (see Details).
batch_size	Integer number of runs to propose. Default 4.
candidates	Optional data frame of candidate settings containing the predictor columns required by the models. When NULL (default), n candidates are drawn with svem_random_table_multi. Supplied columns must not use generated output names (slot, ts_score, or response _draw, _des, _pred columns); collisions are rejected to preserve candidate settings.
n	Number of random candidates to draw when candidates is NULL. Default 5000.
mixture_groups	Optional mixture and simplex constraints passed to svem_random_table_multi (ignored when candidates is supplied).
numeric_sampler	Character; how numeric predictors are sampled when candidates = NULL: "random" (Latin hypercube when **lhs** is available) or "uniform". As in svem_score_random.
combine	How to combine per-response desirabilities into the slot score: "geom" (weighted geometric mean; default) or "mean" (weighted arithmetic mean).
verbose	Logical; if TRUE (default), print a compact summary of the proposed batch.

Details

Candidate table: Candidates may be supplied directly via candidates (a data frame containing at least the predictor columns used by the models). When candidates = NULL, n random candidates are drawn from the models' shared sampling schema with svem_random_table_multi, honoring mixture_groups and numeric_sampler exactly as in svem_score_random.

Goals, desirabilities, and combination: goals uses the same specification as svem_score_random: each response gets goal ("max", "min", or "target"), a nonnegative weight, and optional

Derringer–Suich anchors (`lower_acceptable`/`upper_acceptable`/`shape`, or `target`/`tol`/`tol_left`/`tol_right`/`shape_left`/`shape_right`). When anchors are not supplied, robust defaults are inferred from the ensemble-mean predictions over the candidate table using the q0.02–q0.98 span, mirroring `svem_score_random`. Fixing anchors explicitly is recommended for sequential use so that the desirability scale does not drift between iterations. Desirabilities are combined by a weighted geometric mean (`combine = "geom"`, with a small floor inside the log) or a weighted arithmetic mean (`combine = "mean"`); weights are normalized to sum to one.

Thompson draws: For each response the full member-by-candidate prediction matrix is formed once from `object$coef_matrix` (all models must be fitted with `nBoot >= 2` so that `coef_matrix` is available). For slot $b = 1, \ldots,$ `batch_size` and response r, a member index $m_{b,r}$ is drawn uniformly at random and the slot score at candidate x is

$$s_b(x) = \prod_r d_r\{\hat{y}_r^{(m_{b,r})}(x)\}^{w_r}$$

(or the weighted arithmetic mean when `combine = "mean"`). The proposal for slot b is the eligible candidate maximizing $s_b(x)$; ties keep the earliest candidate row. Draws are controlled by the R random number generator, so use `set.seed()` for reproducibility.

Predictions are always computed with debiasing disabled (`debias = FALSE`), matching `svem_score_random`. Binomial responses are handled on the probability scale.

Typical sequential workflow:

1. Fit one `SVEMnet()` model per response on the current data.
2. Call `svem_thompson_batch()` to propose the next batch of runs (optionally reserving some slots for medoids from `svem_select_from_score_table`, e.g. top-score medoids for exploitation or widest-CI medoids for global gap-filling).
3. Run the proposed settings, append the new data, refit, repeat.

Value

An object of class `"svem_thompson_batch"`: a list with components

`proposals` Data frame with `batch_size` rows: the selected candidate settings (all columns of the candidate table), plus `slot` (batch position), `ts_score` (the slot's Thompson score at the selected candidate), `<resp>_draw` (the drawn member's prediction at the selected candidate), `<resp>_des` (its desirability), and `<resp>_pred` (the ensemble-mean prediction, for reference).

`members` Integer matrix (`batch_size` by number of responses) of the bootstrap member indices drawn for each slot.

`candidate_index` Integer vector: row indices of the proposals in the candidate table.

`candidates` The candidate table used (predictor columns).

`weights` Normalized response weights.

`ds_params` Per-response Derringer–Suich parameters actually used (after anchor inference), useful for keeping anchors fixed across sequential iterations.

`call` The matched call.

Acknowledgments

Generative AI tools were used to assist with coding and roxygen documentation; all content was reviewed and finalized by the author.

References

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3–4), 285–294. doi:10.1093/biomet/25.34.285

Kandasamy, K., Krishnamurthy, A., Schneider, J., and Poczos, B. (2018). Parallelised Bayesian optimisation via Thompson sampling. *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, PMLR 84, 133–142.

Russo, D. J., Van Roy, B., Kazerouni, A., Osband, I., and Wen, Z. (2018). A tutorial on Thompson sampling. *Foundations and Trends in Machine Learning*, 11(1), 1–96. doi:10.1561/2200000070

Derringer, G., and Suich, R. (1980). Simultaneous optimization of several response variables. *Journal of Quality Technology*, 12(4), 214–219. doi:10.1080/00224065.1980.11980968

Karl, A. T. (2026). SVEMnet: An R package for self-validated elastic-net ensembles and multi-response optimization in small-sample mixture-process experiments. *Chemometrics and Intelligent Laboratory Systems*, *271*, 105660. doi:10.1016/j.chemolab.2026.105660

Gotwalt, C., & Ramsey, P. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-Using/ev-p/849873/redirect_from_archived_page/true

Karl, A. T. (2024). A randomized permutation whole-model test heuristic for Self-Validated Ensemble Models (SVEM). *Chemometrics and Intelligent Laboratory Systems*, *249*, 105122. doi:10.1016/j.chemolab.2024.105122

Karl, A., Wisnowski, J., & Rushing, H. (2022). JMP Pro 17 Remedies for Practical Struggles with Mixture Experiments. JMP Discovery Conference. doi:10.13140/RG.2.2.34598.40003/1

Lemkus, T., Gotwalt, C., Ramsey, P., & Weese, M. L. (2021). Self-Validated Ensemble Models for Design of Experiments. *Chemometrics and Intelligent Laboratory Systems*, 219, 104439. doi:10.1016/j.chemolab.2021.104439

Xu, L., Gotwalt, C., Hong, Y., King, C. B., & Meeker, W. Q. (2020). Applications of the Fractional-Random-Weight Bootstrap. *The American Statistician*, 74(4), 345–358. doi:10.1080/00031305.2020.1731599

Ramsey, P., Gaudard, M., & Levin, W. (2021). Accelerating Innovation with Space Filling Mixture Designs, Neural Networks and SVEM. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Accelerating-Innovation-with-Space-Filling-Mixture-Designs/ev-p/756841

Ramsey, P., & Gotwalt, C. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-ev-p/849647/redirect_from_archived_page/true

Ramsey, P., Levin, W., Lemkus, T., & Gotwalt, C. (2021). SVEM: A Paradigm Shift in Design and Analysis of Experiments. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/SVEM-A-Paradigm-Shift-in-Design-and-Analysis-of-Experiments-2021/ev-p/756634

Ramsey, P., & McNeill, P. (2023). CMC, SVEM, Neural Networks, DOE, and Complexity: It's All About Prediction. *JMP Discovery Conference*.

Friedman, J. H., Hastie, T., and Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1), 1-22.

Meinshausen, N. (2007). Relaxed Lasso. Computational Statistics & Data Analysis, 52(1), 374-393.

Kish, L. (1965). *Survey Sampling*. Wiley.

Lumley, T. (2004). Analysis of complex survey samples. *Journal of Statistical Software*, 9(1), 1–19.

Lumley, T. and Scott, A. (2015). AIC and BIC for modelling with complex survey data. *Journal of Survey Statistics and Methodology*, 3(1), 1–18.

Tibshirani, R. J. and Taylor, J. (2012). Degrees of freedom in lasso problems. *The Annals of Statistics*, 40(2), 1198–1232. doi:10.1214/12AOS1003

See Also

SVEMnet, svem_score_random, svem_select_from_score_table, svem_random_table_multi

Examples

```
## Small runnable two-response proposal
set.seed(21)
n_toy <- 36
toy <- data.frame(x1 = runif(n_toy), x2 = runif(n_toy))
toy$yield <- 1 + 1.5 * toy$x1 - toy$x2 + rnorm(n_toy, sd = 0.15)
toy$impurity <- 1.2 - toy$x1 + 0.4 * toy$x2 + rnorm(n_toy, sd = 0.10)

fit_yield <- SVEMnet(yield ~ x1 + x2, toy, nBoot = 5,
                     glmnet_alpha = 1, relaxed = FALSE)
fit_impurity <- SVEMnet(impurity ~ x1 + x2, toy, nBoot = 5,
                        glmnet_alpha = 1, relaxed = FALSE)

toy_prop <- svem_thompson_batch(
  objects = list(yield = fit_yield, impurity = fit_impurity),
  goals = list(
    yield = list(goal = "max", weight = 0.7),
    impurity = list(goal = "min", weight = 0.3)
  ),
  batch_size = 2,
  n = 100,
  verbose = FALSE
)
toy_prop$proposals

## Not run:
## Production-scale lipid workflow. This is not run automatically because
## it fits wide ensembles and searches thousands of constrained candidates.
## ------------------------------------------------------------------------
## Thompson-sampling batch proposals for the lipid screen
## ------------------------------------------------------------------------
```

```
data(lipid_screen)

spec <- bigexp_terms(
  Potency ~ PEG + Helper + Ionizable + Cholesterol +
    Ionizable_Lipid_Type + N_P_ratio + flow_rate,
  data             = lipid_screen,
  factorial_order  = 2,
  polynomial_order = 2
)

set.seed(1)
fit_pot <- SVEMnet(bigexp_formula(spec, "Potency"), lipid_screen)
fit_siz <- SVEMnet(bigexp_formula(spec, "Size"),    lipid_screen)

objs <- list(Potency = fit_pot, Size = fit_siz)

goals <- list(
  Potency = list(goal = "max", weight = 0.7),
  Size    = list(goal = "min", weight = 0.3)
)

mix <- list(list(
  vars  = c("PEG", "Helper", "Ionizable", "Cholesterol"),
  lower = c(0.01, 0.10, 0.10, 0.10),
  upper = c(0.05, 0.60, 0.60, 0.60),
  total = 1.0
))

set.seed(2)
prop <- svem_thompson_batch(
  objects        = objs,
  goals          = goals,
  batch_size     = 4,
  n              = 3000,
  mixture_groups = mix
)

prop$proposals[, c("PEG", "Helper", "Ionizable", "Cholesterol",
                   "ts_score", "Potency_draw", "Size_draw")]

## Keep the inferred desirability anchors fixed at later iterations by
## passing explicit anchors extracted from prop$ds_params, e.g.:
p <- prop$ds_params$Potency
goals$Potency <- list(goal = "max", weight = 0.7,
                      lower_acceptable = p$L, upper_acceptable = p$U)

## ------------------------------------------------------------------------
## Hybrid batch: exploit medoids + Thompson-sampling slots
## ------------------------------------------------------------------------

set.seed(3)
scored <- svem_score_random(objs, goals, n = 3000,
                            mixture_groups = mix, verbose = FALSE)
```

```
sel <- svem_select_from_score_table(scored$score_table,
                                    target = "score", k = 2)

set.seed(4)
ts <- svem_thompson_batch(objs, goals, batch_size = 2,
                          n = 3000, mixture_groups = mix,
                          verbose = FALSE)

pred_cols <- attr(scored$score_table, "svem_predictor_cols")
batch <- rbind(sel$candidates[, pred_cols],
               ts$proposals[, pred_cols])
batch

## End(Not run)
```

`svem_wmt_multi` *Whole-model tests for multiple SVEM responses (WMT wrapper)*

Description

Convenience wrapper around `svem_significance_test_parallel` for running whole-model tests (WMT) on multiple responses that share the same dataset and mixture constraints. This helper:

- takes a formula or a list of formulas and a single `data` frame,
- calls `svem_significance_test_parallel()` for each response,
- extracts per-response p-values and converts them to WMT multipliers via a chosen transform, and
- optionally plots the WMT objects together using `plot.svem_significance_test` and prints a compact summary of p-values and multipliers.

The resulting `multipliers` vector is designed to be passed directly to downstream scoring functions (for example, as an optional WMT argument to `svem_score_random()`), with response names matched by `names()`.

Usage

```
svem_wmt_multi(
  formulas,
  data,
  mixture_groups = NULL,
  wmt_transform = c("neglog10", "one_minus_p"),
  wmt_control = list(seed = 123),
  plot = TRUE,
  verbose = TRUE
)
```

Arguments

formulas	A single formula or a (preferably named) list of formulas, one per response. If unnamed, response names are inferred from the left-hand side of each formula; non-unique names are made unique.
data	Data frame containing the predictors and responses referenced in formulas.
mixture_groups	Optional mixture and simplex constraints passed to svem_significance_test_parallel.
wmt_transform	Character; transformation used to convert WMT p-values into multipliers. One of: • "neglog10": $f(p) = [-\log_{10}(p)]^{\text{strength}}$, • "one_minus_p": $f(p) = (1-p)^{\text{strength}}$. Currently, strength = 1 is used internally.
wmt_control	Optional list of extra arguments passed directly to svem_significance_test_parallel. By default this is list(seed = 123) so that WMT calls are reproducible; you may override or extend this (e.g. list(seed = 999, nPerm = 300)). A nonempty list must be fully named. Arguments formula, data, mixture_groups, and response are controlled by this wrapper and ignored with a warning; each response, including any transformation, comes from its formula. Other duplicated names are rejected. A shared spec is allowed: it supplies the expansion and sampling schema for every formula, takes precedence over attached specifications, and preserves each formula's left-hand side. The WMT is serial by default. Pass nCore explicitly to opt into parallel execution, and use rng_sample_kind = "Rounding" only when reproducing the legacy pre-3.4.0 random stream. Any entries not recognized by svem_significance_test_parallel are ignored by that function.
plot	Logical; if TRUE (default), attempt to plot all successfully computed WMT objects together via plot.svem_significance_test.
verbose	Logical; if TRUE (default), print progress and a compact summary of p-values and multipliers.

Value

A list of class "svem_wmt_multi" with components:

wmt_objects Named list of WMT objects (one per response), as returned by svem_significance_test_parallel(). Entries are NULL where a WMT call failed.

p_values Named numeric vector of per-response p-values (bounded away from 0/1), or NA when unavailable.

multipliers Named numeric vector of per-response WMT multipliers derived from the p-values using wmt_transform. Failed tests retain multiplier 1 and emit a warning regardless of verbose; this fallback is not evidence of model significance.

wmt_transform The transformation used.

wmt_control The remaining arguments passed through to svem_significance_test_parallel() after removing the wrapper-controlled entries.

Acknowledgments

Generative AI tools were used to assist with coding and roxygen documentation; all content was reviewed and finalized by the author.

References

Karl, A. T. (2026). SVEMnet: An R package for self-validated elastic-net ensembles and multi-response optimization in small-sample mixture-process experiments. *Chemometrics and Intelligent Laboratory Systems*, *271*, 105660. doi:10.1016/j.chemolab.2026.105660

Gotwalt, C., & Ramsey, P. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-Using/ev-p/849873/redirect_from_archived_page/true

Karl, A. T. (2024). A randomized permutation whole-model test heuristic for Self-Validated Ensemble Models (SVEM). *Chemometrics and Intelligent Laboratory Systems*, *249*, 105122. doi:10.1016/j.chemolab.2024.105122

Karl, A., Wisnowski, J., & Rushing, H. (2022). JMP Pro 17 Remedies for Practical Struggles with Mixture Experiments. JMP Discovery Conference. doi:10.13140/RG.2.2.34598.40003/1

Lemkus, T., Gotwalt, C., Ramsey, P., & Weese, M. L. (2021). Self-Validated Ensemble Models for Design of Experiments. *Chemometrics and Intelligent Laboratory Systems*, 219, 104439. doi:10.1016/j.chemolab.2021.104439

Xu, L., Gotwalt, C., Hong, Y., King, C. B., & Meeker, W. Q. (2020). Applications of the Fractional-Random-Weight Bootstrap. *The American Statistician*, 74(4), 345–358. doi:10.1080/00031305.2020.1731599

Ramsey, P., Gaudard, M., & Levin, W. (2021). Accelerating Innovation with Space Filling Mixture Designs, Neural Networks and SVEM. *JMP Discovery Conference*. https://community.jmp.com/t5/Abstracts/Accelerating-Innovation-with-Space-Filling-Mixture-Designs/ev-p/756841

Ramsey, P., & Gotwalt, C. (2018). Model Validation Strategies for Designed Experiments Using Bootstrapping Techniques With Applications to Biopharmaceuticals. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/Model-Validation-Strategies-for-Designed-Experiments-ev-p/849647/redirect_from_archived_page/true

Ramsey, P., Levin, W., Lemkus, T., & Gotwalt, C. (2021). SVEM: A Paradigm Shift in Design and Analysis of Experiments. *JMP Discovery Conference - Europe*. https://community.jmp.com/t5/Abstracts/SVEM-A-Paradigm-Shift-in-Design-and-Analysis-of-Experiments-2021/ev-p/756634

Ramsey, P., & McNeill, P. (2023). CMC, SVEM, Neural Networks, DOE, and Complexity: It's All About Prediction. *JMP Discovery Conference*.

Friedman, J. H., Hastie, T., and Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1), 1-22.

Meinshausen, N. (2007). Relaxed Lasso. Computational Statistics & Data Analysis, 52(1), 374-393.

Kish, L. (1965). *Survey Sampling*. Wiley.

Lumley, T. (2004). Analysis of complex survey samples. *Journal of Statistical Software*, 9(1), 1–19.

Lumley, T. and Scott, A. (2015). AIC and BIC for modelling with complex survey data. *Journal of Survey Statistics and Methodology*, 3(1), 1–18.

Tibshirani, R. J. and Taylor, J. (2012). Degrees of freedom in lasso problems. *The Annals of Statistics*, 40(2), 1198–1232. doi:10.1214/12AOS1003

See Also

svem_significance_test_parallel, plot.svem_significance_test

Examples

```
## Not run:
## Production WMT workflow: not run automatically because each response
## requires repeated SVEM fits under many response permutations.
data(lipid_screen)

spec <- bigexp_terms(
  Potency ~ PEG + Helper + Ionizable + Cholesterol +
    Ionizable_Lipid_Type + N_P_ratio + flow_rate,
  data             = lipid_screen,
  factorial_order  = 3,
  polynomial_order = 3,
  include_pc_2way  = TRUE,
  include_pc_3way  = FALSE
)

form_pot <- bigexp_formula(spec, "Potency")
form_siz <- bigexp_formula(spec, "Size")
form_pdi <- bigexp_formula(spec, "PDI")

mix <- list(list(
  vars  = c("PEG", "Helper", "Ionizable", "Cholesterol"),
  lower = c(0.01, 0.10, 0.10, 0.10),
  upper = c(0.05, 0.60, 0.60, 0.60),
  total = 1.0
))

set.seed(123)
wmt_out <- svem_wmt_multi(
  formulas       = list(Potency = form_pot,
                        Size    = form_siz,
                        PDI     = form_pdi),
  data           = lipid_screen,
  mixture_groups = mix,
  wmt_transform  = "neglog10",
  wmt_control    = list(seed = 123, nCore = 1L),
  plot           = TRUE
)

wmt_out$p_values
wmt_out$multipliers
```

```
## later: pass wmt_out$multipliers into svem_score_random()

## End(Not run)
```

`with_bigexp_contrasts` *Evaluate code with the spec's recorded contrast options*

Description

with_bigexp_contrasts() temporarily restores the contrasts options that were active when the spec was built, runs a block of code, and then restores the original options. This is useful when a modeling function uses the global `options("contrasts")` to decide how to encode factors (for example, `lm()`, `glm()`, or other modeling functions that call `model.matrix()` internally).

Usage

```
with_bigexp_contrasts(spec, code)
```

Arguments

spec	A "bigexp_spec" object with stored contrasts_options in settings.
code	Code to evaluate with temporarily restored options.

Examples

```
set.seed(1)
df4 <- data.frame(
  y  = rnorm(10),
  X1 = rnorm(10),
  G  = factor(sample(c("A", "B"), 10, replace = TRUE))
)

spec4 <- bigexp_terms(
  y ~ X1 + G,
  data             = df4,
  factorial_order  = 2,
  polynomial_order = 2
)

with_bigexp_contrasts(spec4, {
  mm4 <- model.matrix(spec4$formula, df4)
  head(mm4)
})
```

Index

∗ **SVEM methods**
predict.svem_model, 36
∗ **datasets**
lipid_screen, 26
∗ **package**
SVEMnet-package, 3

bigexp_formula, *3, 4*, 7, *12, 13*, *27*, *42*, *80*
bigexp_prepare, *4*, 8, *12, 13*, *15*
bigexp_terms, *3–5*, *8, 9*, 9, *15*, *27*, *42*, *62*, *64*, *80*
bigexp_train, *4*, *13*, 15

coef.svem_model, *3*, 17, *61*

forward_aicc, *5*, 18, *58*

glmnet_with_cv, *5*, *18, 19*, 20

lipid_screen, *5*, 26

plot.svem_binomial, 32
plot.svem_model, *3*, 34
plot.svem_significance_test, *4*, 35, *88, 89*, *91*
predict.svem_cv *(*predict_cv*)*, 40
predict.svem_model, *3*, 36
predict_cv, 40
print.bigexp_formula, 42
print.bigexp_spec, 42
print.svem_significance_test, 43
print.svem_thompson_batch, 44

svem_export_candidates_csv, *3, 4*, 55, *75*
svem_forward, *5*, *19*, 56, *60*
svem_ij_variance, *5*, 59
svem_nonzero, *3*, *17*, 60
svem_random_table_multi, *4*, 62, *65, 66*, *70*, *83*, *86*
svem_score_random, *3–5*, *27*, 64, *74*, *76*, *82–84*, *86*
svem_select_from_score_table, *3, 4*, *27*, *55*, *64*, *67*, *70*, 74, *82*, *84*, *86*
svem_significance_test_parallel, *3–5*, *76*, *88, 89*, *91*
svem_thompson_batch, *5*, 82
svem_wmt_multi, *3–5*, *27*, *66, 67*, *70*, 88
SVEMnet, *3*, *5*, *17–19*, *38*, 44, *57, 58*, *60, 61*, *65*, *70*, *80*, *82, 83*, *86*
SVEMnet-package, 3

with_bigexp_contrasts, *4*, 92