Package 'SVEMnet'

November 9, 2025

Type Package

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

Version 2.5.4

Date 2025-11-09

Description Implements Self-Validated Ensemble Models (SVEM; Lemkus et al. (2021) <doi:10.1016/j.chemolab.2021.104439>) using elastic net regression via 'glmnet' (Friedman et al. (2010) <doi:10.18637/jss.v033.i01>). SVEM averages predictions from multiple models fitted to fractionally weighted bootstraps of the data, tuned with anti-correlated validation weights. Supports Gaussian and binomial responses. Also implements the randomized permutation whole-model test for SVEM with Gaussian responses (Karl (2024) <doi:10.1016/j.chemolab.2024.105122>). Some parts of the package code were drafted with assistance from generative AI tools.

Depends R (>= 4.0.0)

Imports glmnet (>= 4.1-6), stats, cluster, ggplot2, rlang, lhs, foreach, doParallel, doRNG, parallel, gamlss, gamlss.dist

Suggests covr, knitr, rmarkdown, testthat (>= 3.0.0), withr, vdiffr, RhpcBLASctl

VignetteBuilder knitr

License GPL-2 | GPL-3

Encoding UTF-8

RoxygenNote 7.3.3

Config/testthat/edition 3

LazyData true

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 2025-11-09 15:40:02 UTC

Contents

SVEMnet-package . . . 2
bigexp_formula . . . 4
bigexp_model_matrix . . . 5
bigexp_prepare . . . 6
bigexp_terms . . . 7
bigexp_train . . . 10
coef.svem_model . . . 11
glmnet_with_cv . . . 13
lipid_screen . . . 17
plot.svem_binomial . . . 21
plot.svem_model . . . 23
plot.svem_significance_test . . . 24
predict.svem_model . . . 25
predict_cv . . . 28
print.bigexp_spec . . . 30
print.svem_significance_test . . . 31
SVEMnet . . . 31
svem_nonzero . . . 37
svem_optimize_random . . . 39
svem_random_table_multi . . . 45
svem_significance_test_parallel . . . 48
with_bigexp_contrasts . . . 53

Index **55**

`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

A typical workflow is to fit models with `SVEMnet()`, optionally run a whole-model test (for example, for response reweighting), and then call `svem_optimize_random()` to propose optimal and exploration candidates for the next experimental round.

Functions

`SVEMnet` Fit an SVEMnet model using Elastic Net regression.

`predict.svem_model` Predict method for SVEM models (optionally debiased, with intervals when available).

`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.

`bigexp_terms` Build a deterministic expanded RHS (polynomials, interactions) with locked levels/ranges.

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

`svem_random_table_multi` Generate one shared random predictor table (with optional mixture constraints) and get predictions from multiple SVEM models at those points.

`svem_optimize_random` Random-search optimizer for multiple responses with goals, weights, optional CIs, and diverse PAM-medoids candidates.

`svem_significance_test_parallel` Parallel whole-model significance test (foreach + doParallel) with optional mixture-group sampling.

`plot.svem_significance_test` Plot helper for visualizing multiple significance-test outputs.

`glmnet_with_cv` Convenience wrapper around repeated `cv.glmnet()` selection.

`lipid_screen` Example dataset for multi-response modeling and mixture-constrained optimization demos.

Families

Supports Gaussian and binomial responses (logit link). For `family = "binomial"`, 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).

Acknowledgments

OpenAI's GPT models (o1-preview and GPT-5 Thinking via ChatGPT) 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)

References

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.

`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.

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_model_matrix` *Build a model matrix using the spec's stored contrasts*

Description

bigexp_model_matrix() combines bigexp_prepare() and model.matrix() so that you can obtain a design matrix that matches the locked spec, including any stored contrast settings.

Usage

```
bigexp_model_matrix(spec, data)
```

Arguments

`spec`	A "bigexp_spec" object.
`data`	A data frame to prepare and encode.

Value

The design matrix returned by model.matrix(), with columns ordered consistently with the spec and its locked factor levels.

Examples

```
set.seed(1)
df3 <- data.frame(
  y  = rnorm(10),
  X1 = rnorm(10),
  X2 = rnorm(10)
)

spec3 <- bigexp_terms(
  y ~ X1 + X2,
  data             = df3,
  factorial_order  = 2,
  polynomial_order = 2
)

MM3 <- bigexp_model_matrix(spec3, df3)
dim(MM3)
head(colnames(MM3))
```

---

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
- 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

The goal is that `model.matrix(spec$formula, data)` (or `bigexp_model_matrix`) 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.
- `data`: a copy of the input data coerced to match the spec.

See Also

`bigexp_terms`, `bigexp_model_matrix`

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,
- 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) string for a large expansion.

Usage

```
bigexp_terms(
  formula,
  data,
  factorial_order = 3L,
  discrete_threshold = 2L,
  polynomial_order = 3L,
  include_pc_2way = TRUE,
  include_pc_3way = FALSE,
  intercept = TRUE
)
```

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), or `I()` powers here. The helper will generate interactions and polynomial terms automatically.
data	Data frame used to decide types and lock factor levels.
factorial_order	Integer >= 1. Maximum order of factorial interactions among the 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.
discrete_threshold	Numeric predictors with at most this many unique finite values are treated as categorical. Default is `2`.
polynomial_order	Integer >= 1. Maximum polynomial degree for continuous 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) and `polynomial_order` >= `2`, include partial-cubic two-way terms of the form `Z:I(X^2)`.
include_pc_3way	Logical. If `TRUE` and `polynomial_order` >= `2`, include partial-cubic three-way terms `I(X^2):Z:W`.
intercept	Logical. If `TRUE` (default), include an intercept in the expansion; if `FALSE`, the generated RHS drops the intercept.

Details

The expansion 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;
- numeric predictors with at most `discrete_threshold` distinct finite values are treated as categorical; and
- all other numeric predictors are treated as continuous, and their observed ranges are stored.

Once built, a `"bigexp_spec"` can be reused to create consistent expansions for new datasets via `bigexp_prepare`, `bigexp_formula`, and `bigexp_model_matrix`. 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 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`, `discrete_threshold`, `include_pc_2way`, `include_pc_3way`, `intercept`, and stored contrast information.

See Also

`bigexp_prepare`, `bigexp_formula`, `bigexp_model_matrix`, `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)

## Inspect the resulting design matrix
MM <- bigexp_model_matrix(spec, df)
dim(MM)
head(colnames(MM))

## 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
)

MM_main <- bigexp_model_matrix(spec_main, df)
head(colnames(MM_main))
```

`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
- 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, discrete_threshold, 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 expanded training data 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 ready for modeling

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
)

str(tr$data)
tr$formula
```

---

coef.svem_model *Coefficients for SVEM Models*

Description

Returns averaged coefficients from an svem_model.

Usage

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

Arguments

object	An object of class svem_model.
debiased	Logical; if TRUE and available (Gaussian fits), return parms_debiased instead of parms. Default FALSE.
...	Unused (for S3 compatibility).

Details

For Gaussian models, you can optionally return the debiased coefficients (if available) via `debiased = TRUE`. For Binomial models, `debiased` is ignored and the averaged coefficients are returned.

Value

A named numeric vector of coefficients (including intercept).

See Also

`svem_nonzero` for bootstrap nonzero percentages and a quick plot.

Examples

```
set.seed(1)
n  <- 200
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)

# Small nBoot to keep runtime light in examples
fit_g <- SVEMnet(y_g ~ x1 + x2, data = dat_g, nBoot = 30, relaxed = TRUE)

# Averaged coefficients
cc <- coef(fit_g)
head(cc)

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

# Binomial example (0/1 outcome)
set.seed(2)
n  <- 250
x1 <- rnorm(n)
x2 <- rnorm(n)
eta <- -0.4 + 1.1*x1 - 0.7*x2
p   <- 1/(1+exp(-eta))
y_b <- rbinom(n, 1, p)
dat_b <- data.frame(y_b, x1, x2)

fit_b <- SVEMnet(y_b ~ x1 + x2, data = dat_b, family = "binomial",
                 nBoot = 30, relaxed = TRUE)

# Averaged coefficients (binomial)
coef(fit_b)
```

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

Description

Repeated K-fold cross-validation over a per-alpha lambda path, with a combined 1-SE rule across repeats. Preserves fields expected by `predict.svem_model` / internal prediction helpers. Optionally uses glmnet's built-in relaxed elastic net for both the warm-start path and each CV fit. When `relaxed = TRUE`, the final coefficients are taken from a `cv.glmnet()` object at the chosen lambda so that the returned model reflects the relaxed solution (including its chosen γ).

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` / `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 lambda within each alpha: • `"min"`: lambda minimizing the cross-validated criterion. • `"1se"`: largest lambda 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).
relaxed	Logical; if TRUE, call glmnet / cv.glmnet with relax = TRUE and optionally a gamma path (default FALSE). If cv.glmnet(relax=TRUE) fails for a particular repeat/alpha, the function retries that fit without relaxation; the number of such fallbacks is recorded in meta$relax_cv_fallbacks.
relax_gamma	Optional numeric vector passed as gamma= to glmnet / 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() / 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: weights, parallel, type.measure, intercept, maxit, lower.limits, upper.limits, penalty.factor, offset, standardize.response, keep, etc. If family is supplied here, it is ignored in favor of the explicit family argument.

Details

This function is a convenience wrapper around glmnet / 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/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.

For each alpha in glmnet_alpha, the function:

1. Generates a set of CV fold IDs (shared across alphas and repeats).
2. Runs repeats independent cv.glmnet() fits, aligning lambda paths and aggregating the CV curves.
3. Computes a combined SE at each lambda that accounts for both within-repeat and between-repeat variability.
4. Applies choose_rule ("min" or "1se") to pick the lambda for that alpha.

The best alpha is then chosen by comparing these per-alpha scores.

If there are no predictors after model.matrix() (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 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.

The returned object inherits classes `"svem_cv"` and `"svem_model"` and is designed to be compatible with SVEMnet's 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.
- `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 per alpha) of data frames with columns: `lambda`, `mean_cvm`, `sd_cvm`, `se_combined`, `n_repeats`, `idx_min`, `idx_1se`.
- `formula` Original modeling formula.
- `terms` Training `terms` object with environment set to `baseenv()`.
- `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 predict.
- `note` Character vector of notes (e.g., 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`).

Acknowledgments

OpenAI's GPT models (o1-preview and GPT-5 Thinking via ChatGPT) were used to assist with coding and roxygen documentation; all content was reviewed and finalized by the author.

References

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.

Examples

```
set.seed(123)
n <- 100; p <- 10
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 <- 150; p2 <- 8
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 = c(0.5, 1),
  nfolds = 5,
  repeats = 2,
  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]` (e.g., 4.19\ for demonstration of SVEMnet multi-response modeling and desirability-based random-search optimization.

Usage

```
data(lipid_screen)
```

Format

A data frame with N rows and the following columns:

RunID character. Optional identifier.

PEG numeric. Proportion (0–1).

Helper numeric. Proportion (0–1).

Ionizable numeric. Proportion (0–1).

Cholesterol numeric. Proportion (0–1).

Ionizable_Lipid_Type factor.

N_P_ratio numeric.

flow_rate numeric.

Potency numeric. Response.

Size numeric. Response (e.g., nm).

PDI numeric. Response (polydispersity index).

Notes character. Optional free-text notes.

Details

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 optimization with `svem_optimize_random()` using Derringer–Suich desirabilities and weighted combining (`combine = "geom"` or `combine = "mean"`),
- returning both high-score *optimal* candidates and high-uncertainty *exploration* candidates,
- optional whole-model reweighting (WMT) of response weights via p-values, and exporting per-row scores to the original data with `original_data_scored`.

Acknowledgments

OpenAI's GPT models (o1-preview and GPT-5 Thinking via ChatGPT) 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

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.

Examples

```
# 1) Load the bundled dataset
data(lipid_screen)
str(lipid_screen)

# 2) Build a deterministic expansion using bigexp_terms()
#    Provide main effects only on the RHS; expansion width is controlled via arguments.
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, # include partial cubic terms
  include_pc_3way  = FALSE
)

# 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/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 1, 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) Run random-search optimization (DS + optimal & exploration candidates)
set.seed(2)
opt_out <- svem_optimize_random(
  objects                  = objs,
  goals                    = goals,
  n                        = 25000,
  mixture_groups           = mix,
  level                    = 0.95,
  k_candidates             = 5,
  top_frac                 = 0.02,
  k_exploration_candidates = 5,
  exploration_top_frac     = 0.05,
  numeric_sampler          = "random",
  verbose                  = TRUE
)

# Inspect optimal solution and candidates (scores and uncertainty included)
```

```
opt_out$best
opt_out$best_pred
opt_out$best_ci
head(opt_out$candidates)

# Inspect exploration target and candidates
opt_out$exploration_best
head(opt_out$exploration_candidates)

# 8) Repeat with WMT reweighting using the original data (requires 'data')
#    Choose either "neglog10" (aggressive) or "one_minus_p" (conservative).
set.seed(3)
opt_wmt <- svem_optimize_random(
  objects                 = objs,
  goals                   = goals,
  data                    = lipid_screen,  # used for WMT and original_data_scored
  n                       = 25000,
  mixture_groups          = mix,
  level                   = 0.95,
  k_candidates            = 5,
  top_frac                = 0.02,
  k_exploration_candidates = 5,
  exploration_top_frac    = 0.05,
  numeric_sampler         = "random",
  reweight_by_wmt         = TRUE,
  wmt_transform           = "neglog10",
  verbose                 = TRUE
)

# Compare weights and look at candidates under WMT
opt_wmt$weights_original
opt_wmt$weights_final
opt_wmt$wmt_p_values
head(opt_wmt$candidates)
head(opt_wmt$exploration_candidates)

# Scored original data (predictions, desirabilities, score, uncertainty)
head(opt_wmt$original_data_scored)
```

`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

```
## Not run:
  ## --- Binomial example: simulate, fit, and plot --------------------------
  set.seed(2025)
  n  <- 600
  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  = c(1, 0.5),
    nBoot         = 60,
    objective     = "auto",
    weight_scheme = "SVEM",
    relaxed       = TRUE
  )
```

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

## End(Not run)
```

`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

```
## Not run:
  ## --- Gaussian example: simulate, fit, and plot --------------------------
  set.seed(2026)
  n  <- 300
  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  = c(1, 0.5),
    nBoot         = 60,
    objective     = "auto",
    weight_scheme = "SVEM",
    relaxed       = TRUE
  )

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

## End(Not run)
```

---

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,
  agg = c("parms", "mean"),
  ...
)
```

Arguments

object	A fitted SVEM model (class svem_model; binomial models typically also inherit 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). Ignored for Gaussian models.
debias	(Gaussian only) Logical; default FALSE. If TRUE, apply the linear calibration fit (y ~ y_pred) learned at training when available. Ignored (and internally set to FALSE) for binomial.

se.fit	Logical; if TRUE, return bootstrap standard errors computed from member predictions (requires coef_matrix). Not available for type = "class".
interval	Logical; if TRUE, return percentile confidence limits from member predictions (requires coef_matrix). Not available for type = "class".
level	Confidence level for percentile intervals. Default 0.95.
agg	Aggregation method for ensemble predictions. One of "parms" (default) or "mean"; see *Aggregation modes*. For binomial models, predict() always uses "mean" regardless of the input.
...	Currently unused.

Details

This unified 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" | "class". No debiasing is applied; "response" returns probabilities in $[0, 1]$.

Uncertainty summaries (se.fit, interval) and all binomial predictions (and Gaussian agg = "mean") 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 supported.

Value

If se.fit = FALSE and interval = FALSE:

- **Gaussian**: a numeric vector of predictions (response scale).
- **Binomial**: a numeric vector for type = "response" (probabilities) or type = "link" (log-odds), or an integer vector 0/1 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: percentile confidence 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 exactly match training:

- Uses the training terms (with environment set to baseenv()).
- Harmonizes factor/character columns to training xlevels.
- Reuses stored per-factor contrasts when available; otherwise falls back to the current global contrasts options.

- Zero-fills any columns present at training but absent in `newdata`, and reorders columns to match training.

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

Aggregation modes

`agg = "parms"` (Gaussian only) Use the aggregated coefficients saved at fit time (`object$parms`; for Gaussian with `debias = TRUE`, use `object$parms_debiased`).

`agg = "mean"` Average per-bootstrap member predictions (on the requested scale) and, for Gaussian with `debias = TRUE`, apply the calibration to member predictions before aggregation. Requires `coef_matrix`.

For **binomial** SVEM models, `predict()` always behaves as if `agg = "mean"`: predictions are aggregated from member predictions on the requested scale (probability or link), and any user-specified `agg` is ignored with a warning. The stored coefficient averages (`parms`, `parms_debiased`) are retained for diagnostics but are not used in prediction.

Debiasing

For **Gaussian** fits only, 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. Binomial fits are never debiased, even if `debias = TRUE` is requested.

Uncertainty

When `se.fit = TRUE`, the returned standard errors are 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 `coef_matrix`. For `type = "class"` (binomial), uncertainty summaries are not available.

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 = 40, glmnet_alpha = c(1, 0.5), relaxed = TRUE, family = "gaussian"
)

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

```
## Mean-aggregation with uncertainty (requires coef_matrix)
out_g <- predict(fit_g, dat, debias = TRUE, agg = "mean",
                 se.fit = TRUE, interval = TRUE, level = 0.9)
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 = 50, glmnet_alpha = c(1, 0.5), 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/interval)

## 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)
```

---

predict_cv *Predict for* svem_cv *objects (and convenience wrapper)*

Description

Generate predictions from a fitted object returned by glmnet_with_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, require an exact column-name match with the training design (including intercept position) after alignment. Default FALSE.
...	Additional arguments (currently unused).

Details

The design matrix for newdata is rebuilt using the training terms (with environment set to baseenv()), along with the saved factor xlevels and contrasts (stored on the object and cached in object$schema). Columns are aligned robustly to the training order:

- Any training columns that model.matrix() drops for newdata (e.g., a factor collapses to a single level) are added back as zero columns.
- Columns are reordered to exactly match the training order.
- Rows with unseen factor levels are warned about and return NA predictions.

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

If debias = TRUE and a calibration fit lm(y ~ y_pred) exists with a finite slope, predictions are linearly transformed as 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.

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 = 2,
```

```
  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 = c(0.5, 1),
  nfolds = 5,
  repeats = 2,
  seed = 11,
  family = "binomial"
)

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

`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.

Usage

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

Arguments

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

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).

SVEMnet *Fit an SVEMnet Model (with optional relaxed elastic net)*

Description

Wrapper for glmnet (Friedman et al. 2010) to fit an ensemble of Elastic Net models using the Self-Validated Ensemble Model method (SVEM; Lemkus et al. 2021), with an option to use glmnet's built-in relaxed elastic net (Meinshausen 2007). Supports searching over multiple alpha values in the Elastic Net penalty.

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"),
  auto_ratio_cutoff = 1.3,
  relaxed = TRUE,
  response = NULL,
  unseen = c("warn_na", "error"),
  family = c("gaussian", "binomial"),
  ...
)
```

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	Number of bootstrap iterations (default 200).
glmnet_alpha	Elastic Net mixing parameter(s). May be a vector with entries in the range between 0 and 1, inclusive, where alpha = 1 is Lasso and alpha = 0 is Ridge. Defaults to c(0.5, 1).
weight_scheme	Character; weighting scheme used to generate bootstrap training and validation weights (default "SVEM"). One of: • "SVEM": Self-Validated Ensemble Model weights. For each bootstrap, independent $U_i \sim \mathrm{Uniform}(0,1)$ are drawn and converted to anti-correlated FRW copies $w_i^{\mathrm{train}} = -\log U_i$ and $w_i^{\mathrm{valid}} = -\log(1-U_i)$, each rescaled to have mean 1. This is the default and implements the SVEM scheme of Lemkus et al. • "FRW_plain": Fractional–random–weight (FRW) regression without self-validation. A single FRW weight vector $w_i = -\log U_i$ (rescaled to mean 1) is used for both training and validation. This reproduces the fractional-random-weight bootstrap regression of Xu et al. (2020) and related work, with one weighted fit and no self-validation split. • "Identity": Uses unit weights for both training and validation (no resampling). This is primarily useful with nBoot = 1 when you want a single glmnet fit whose penalty is chosen via the selected information criterion (wAIC/wBIC/wSSE), while still using SVEMnet's formula expansion and diagnostics.
objective	Objective used to pick lambda on each bootstrap path (default "auto"). One of "auto", "wAIC", "wBIC", or "wSSE".
auto_ratio_cutoff	Single cutoff for the automatic rule when objective = "auto" (default 1.3). Let r = n_X / p_X, where n_X is the number of training rows and p_X is the number of predictors in the model matrix after dropping the intercept column. If r >= auto_ratio_cutoff, SVEMnet uses wAIC; otherwise it uses wBIC.
relaxed	Logical, TRUE or FALSE (default TRUE). When TRUE (for family = "gaussian"), use glmnet's relaxed elastic net path and select both lambda and relaxed gamma on each bootstrap. When FALSE, fit the standard glmnet path. Note: if relaxed = TRUE and glmnet_alpha includes 0 (ridge), alpha = 0 is dropped. For family = "binomial", relaxed fits are currently disabled for stability: SVEMnet behaves as if relaxed = FALSE, and a warning is issued if relaxed = TRUE is requested.
response	Optional character. When formula is a bigexp_spec, this names the response column to use on the LHS; defaults to the response stored in the spec.
unseen	How to treat unseen factor levels when formula is a bigexp_spec: "warn_na" (default; convert to NA with a warning) or "error" (stop).

`family` Character scalar specifying the model family. One of `"gaussian"` (default) or `"binomial"`. SVEMnet currently assumes the canonical identity link for Gaussian and the canonical logit link for binomial. For `"binomial"`, the response must be numeric 0/1, logical, or a factor with exactly two levels (the second level is treated as 1).

`...` Additional args passed to `glmnet()` (e.g., `penalty.factor`, `lower.limits`, `upper.limits`, `offset`, `standardize.response`, etc.). Any user-supplied `weights` are ignored so SVEM can supply its own bootstrap weights. Any user-supplied `standardize` is ignored; SVEMnet always uses `standardize = TRUE`.

Details

You can pass either:

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

In many applications, `SVEMnet()` is used as part of a closed-loop optimization workflow: models are fit on current experimental data, whole-model tests (WMT) are optionally used to reweight response goals, and then `svem_optimize_random()` proposes both optimal and exploration candidates for the next experimental round. See the `lipid_screen` help page for a worked example.

SVEM applies fractional bootstrap weights to training data and anti-correlated weights for validation when `weight_scheme = "SVEM"`. For each bootstrap, glmnet paths are fit for each alpha in `glmnet_alpha`, and the lambda (and, if `relaxed = TRUE`, relaxed gamma) minimizing a weighted validation criterion is selected.

Weighting schemes. With `weight_scheme = "SVEM"` (the default), SVEMnet uses the fractional-random-weight (FRW) construction with an explicit self-validation split: for each bootstrap replicate and observation i, a shared $U_i \sim \mathrm{Uniform}(0,1)$ is drawn and converted into anti-correlated train/validation copies $w_i^{\mathrm{train}} = -\log U_i$ and $w_i^{\mathrm{valid}} = -\log(1-U_i)$, each rescaled so that their mean is 1. This keeps all rows in every fit while inducing a stable out-of-bag–style validation set for selecting λ (and, if used, the relaxed γ).

With `weight_scheme = "FRW_plain"`, SVEMnet instead uses a single FRW weight vector $w_i = -\log U_i$ for *both* training and validation (one weighted fit, no self-validation split). It is included mainly for historical comparison and method-teaching; for small-sample prediction we recommend the default `"SVEM"` scheme.

Finally, `weight_scheme = "Identity"` sets both training and validation weights to 1. In combination with `nBoot = 1`, this effectively wraps a single `glmnet` fit and chooses λ (and, for Gaussian models, the relaxed γ) by the chosen information criterion (wAIC or wBIC), without any bootstrap variation. This can be useful when you want classical AIC/BIC selection on top of a deterministic expansion, but do not want or need ensembling.

Predictors are always standardized internally via `glmnet::glmnet(..., standardize = TRUE)`.

When `relaxed = TRUE` and `family = "gaussian"`, `coef(fit, s = lambda, gamma = g)` is used to obtain the coefficient path at each relaxed gamma in the internal grid (by default `c(0.2, 0.6, 1)`). Metrics are computed from validation-weighted errors and model size is taken as the number of nonzero coefficients including the intercept (support size), keeping selection consistent between

standard and relaxed paths. For family = "binomial", relaxed fits are currently disabled for numerical stability, so only the standard glmnet path is used even if relaxed = TRUE.

Automatic objective rule ("auto"): This function uses a single ratio cutoff, auto_ratio_cutoff, applied to r = n_X / p_X, where p_X is computed from the model matrix with the intercept column removed. If r >= auto_ratio_cutoff the selection criterion is wAIC; otherwise it is wBIC.

Implementation notes for safety:

- The training terms object is stored with environment set to baseenv() to avoid accidental lookups in the calling environment.
- A compact schema (feature names, xlevels, contrasts) is stored to let predict() reconstruct the design matrix deterministically.
- A lightweight sampling schema (numeric ranges and factor levels for raw predictors) is cached to enable random-table generation without needing the original data.

For family = "gaussian", the loss used in validation is a weighted SSE, and wAIC/wBIC are computed from a Gaussian log-likelihood proxy.

For family = "binomial", the validation loss is the weighted binomial deviance, and wAIC/wBIC are computed as deviance + 2 * k or deviance + log(n_eff) * k, where k is the number of active parameters (1 for the intercept plus the number of nonzero parameters) and n_eff is the effective validation size. The response must be numeric 0/1 or a two-level factor; internally it is converted to 0/1.

Value

An object of class svem_model with elements:

- parms: averaged coefficients (including intercept).
- parms_debiased: averaged coefficients adjusted by the calibration fit.
- debias_fit: lm(y ~ y_pred) calibration model used for debiasing (or NULL).
- coef_matrix: per-bootstrap coefficient matrix.
- nBoot, glmnet_alpha, best_alphas, best_lambdas, weight_scheme, relaxed.
- best_relax_gammas: per-bootstrap relaxed gamma chosen (NA if relaxed = FALSE).
- objective_input, objective_used, objective (same as objective_used), auto_used, auto_decision, auto_rule.
- dropped_alpha0_for_relaxed: whether alpha = 0 was removed because relaxed = TRUE.
- schema: list(feature_names, terms_str, xlevels, contrasts, terms_hash) for safe predict.
- sampling_schema: list(predictor_vars, var_classes, num_ranges = rbind(min=..., max=...) for numeric raw predictors, factor_levels = list(...) for factor/character raw predictors).
- diagnostics: list with k_summary (median and IQR of selected size), fallback_rate, n_eff_summary, alpha_freq, relax_gamma_freq.
- actual_y, training_X, y_pred, y_pred_debiased, nobs, nparm, formula, terms, xlevels, contrasts.
- used_bigexp_spec: logical flag indicating whether a bigexp_spec was used.

Acknowledgments

OpenAI's GPT models (o1-preview and GPT-5 Thinking via ChatGPT) were used to assist with coding and roxygen documentation; all content was reviewed and finalized by the author.

References

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.

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  = c(1, 0.5),
  nBoot         = 75,
  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,  # up to 3-way interactions among X1, X2, X3
  polynomial_order = 3, # include I(X^3) for each continuous predictor
  include_pc_3way = FALSE
)

# Fit using the spec (auto-prepares data)
fit_y <- SVEMnet(
  spec, dat,
  glmnet_alpha  = c(1, 0.5),
  nBoot         = 50,
  objective     = "auto",
  weight_scheme = "SVEM",
  relaxed       = FALSE
)

# 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  = c(1, 0.5),
  nBoot         = 50,
  objective     = "auto",
  weight_scheme = "SVEM",
  relaxed       = FALSE
)
```

```
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  <- 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 = 50, glmnet_alpha = c(1, 0.5), 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/interval)

## 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)
```

---

svem_nonzero *Coefficient Nonzero Percentages (SVEM)*

Description

Calculates the percentage of bootstrap iterations in which each coefficient (excluding the intercept) is nonzero, using a small tolerance. Optionally draws 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`.
`tol`	Numeric tolerance for "nonzero" (default `1e-7`).
`plot`	Logical; if `TRUE`, draws a quick ggplot summary (default `TRUE`).
`print_table`	Logical; if `TRUE`, prints a compact table (default `TRUE`).
`...`	Unused.

Details

This function summarizes variable selection stability across SVEM bootstrap refits. It expects `object$coef_matrix` to contain the per-bootstrap coefficients (including an intercept column).

Value

Invisibly returns a data frame with columns:

- `Variable`
- `Percent of Bootstraps Nonzero`

See Also

`coef.svem_model` for averaged (optionally debiased) coefficients.

Examples

```
## ---------- Gaussian demo ----------
set.seed(10)
n  <- 220
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)^2, data = dat, nBoot = 40, relaxed = TRUE)

# Table + plot
nz <- svem_nonzero(fit, tol = 1e-7, plot = TRUE, print_table = TRUE)
```

```
head(nz)

## ---------- Binomial demo ----------
set.seed(11)
n  <- 260
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- rnorm(n)
lp <- -0.3 + 0.9*x1 - 0.6*x2 + 0.2*x3
p  <- 1/(1+exp(-lp))
y  <- rbinom(n, 1, p)
dat_b <- data.frame(y, x1, x2, x3)

fit_b <- SVEMnet(y ~ x1 + x2 + x3, data = dat_b,
                 family = "binomial", nBoot = 40, relaxed = TRUE)

# Plot optional; still summarizes the bootstrap selection frequencies
svem_nonzero(fit_b, plot = TRUE, print_table = TRUE)
```

`svem_optimize_random` *Random-search optimizer with desirabilities, WMT reweighting, CIs, optimal + exploration candidates, and scored originals*

Description

Draw random points via `svem_random_table_multi`, map each response to a desirability in `[0,1]` using Derringer–Suich (DS) curves, combine them into a single score, optionally reweight response importances by whole-model test (WMT) p-values, and return: the best design, diverse high-score optimal candidates (PAM medoids of the top fraction), and a second set of exploration candidates that target high predicted uncertainty. Medoids are rows of the sampled table, so all candidates are feasible under sampling and mixture constraints. If `data` is provided, the function also returns that same table augmented with per-row predictions, DS terms, the combined score, and an uncertainty measure.

Usage

```
svem_optimize_random(
  objects,
  goals,
  data = NULL,
  n = 50000,
  mixture_groups = NULL,
  level = 0.95,
  top_frac = 0.02,
  k_candidates = 5,
  verbose = TRUE,
  combine = c("geom", "mean"),
```

```
  numeric_sampler = c("random", "maximin", "uniform"),
  reweight_by_wmt = FALSE,
  wmt_transform = c("neglog10", "one_minus_p"),
  wmt_control = list(seed = 123),
  k_exploration_candidates = 5,
  exploration_top_frac = 0.05
)
```

Arguments

objects	Named list of svem_model objects (from SVEMnet()). Names are treated as response identifiers (typically matching the left-hand sides of the model formulas).
goals	Named list per response. Each entry must include: goal (one of "max", "min", "target") and a nonnegative weight. For goal = "target", also provide target. Optional per-response DS controls: for "max"/"min": lower_acceptable (L), upper_acceptable (U), shape (>= 0); for "target": tol (symmetric) or tol_left/tol_right, and shape_left/shape_right. If anchors or tolerances are not provided, robust defaults are inferred from the sampled responses using the 2nd-98th percentile range.
data	Optional data frame used when reweight_by_wmt = TRUE and to produce original_data_scored. If reweight_by_wmt = TRUE and data is not supplied (or is not a data frame), the function stops. Each model's stored formula is evaluated on data for the WMT via svem_significance_test_parallel(). Any mixture_groups are forwarded.
n	Number of random samples to draw.
mixture_groups	Optional mixture constraints forwarded to svem_random_table_multi(). Each group may include vars, lower, upper, and total.
level	Confidence level for percentile intervals. Default: 0.95.
top_frac	Fraction in (0, 1] of highest-score rows to cluster for optimal candidates. Default: 0.02.
k_candidates	Number of diverse optimal candidates (medoids) to return. If 0, no optimality clustering is performed. Default: 5.
verbose	Logical; print a compact summary of the run and results.
combine	How to combine per-response desirabilities. Use "geom" for weighted geometric mean (default) or "mean" for weighted arithmetic mean. For combine = "geom", a small floor is applied before logging to avoid log(0).
numeric_sampler	Sampler for non-mixture numeric predictors passed to svem_random_table_multi(). One of "random" (default), "maximin", or "uniform". "random" uses lhs::randomLHS() when available, otherwise plain runif().
reweight_by_wmt	Logical; if TRUE, compute whole-model p-values (WMT) for each response on data and downweight responses with weak factor relationships before scoring. Requires data. **Not allowed if any responses are binomial.**

`wmt_transform` Transformation used to turn p-values into multipliers when `reweight_by_wmt = TRUE`. One of `"neglog10"`, `"one_minus_p"`. Multipliers are floored internally to avoid zeroing weights and then renormalized to sum to one with the user weights.

`wmt_control` Optional list to override WMT defaults passed to `svem_significance_test_parallel()`. Recognized entries: `nPoint`, `nSVEM`, `nPerm`, `percent`, `nBoot`, `glmnet_alpha`, `weight_scheme`, `objective`, `auto_ratio_cutoff`, `relaxed`, `verbose`, `nCore`, `seed`, `spec`, `response`, `use_spec_contrasts`. By default, `svem_optimize_random()` uses `wmt_control = list(seed = 123)`, so WMT calls are reproducible; you can override this by passing your own `wmt_control` (with or without a `seed`).

`k_exploration_candidates` Number of diverse exploration candidates (medoids) to return. If `0`, no exploration clustering is performed. Default: `5`.

`exploration_top_frac` Fraction in `(0, 1]` of rows with the largest uncertainty measure to cluster for exploration candidates. Default: `0.05`.

Details

A typical closed-loop workflow for formulation or process optimization is:

1. Fit one or more `SVEMnet()` models for responses of interest.
2. Optionally run whole-model testing (WMT) to reweight response goals.
3. Call `svem_optimize_random()` to generate:
   - high-scoring "optimal" candidates for immediate testing, and
   - high-uncertainty exploration candidates to improve the models.
4. Run these candidates in the lab, append the new data, refit the models, and repeat as needed.

See the package vignette for a full worked example of this closed-loop workflow.

Multi-response scoring. Each response is mapped to a DS desirability $d_r \in [0, 1]$. Anchors L and U (and target-band tolerances) default to robust values derived from the sampled 2nd-98th percentile range when not supplied. Desirabilities are combined across responses using either a weighted arithmetic mean (`combine = "mean"`) or a weighted geometric mean (`combine = "geom"`), with a small fixed floor applied inside the log to avoid `log(0)`.

Whole-model reweighting (WMT). When `reweight_by_wmt = TRUE`, each response receives a multiplier from its whole-model p-value computed by `svem_significance_test_parallel()` on `data`. Final weights are proportional to the product of the user weight and the multiplier, then renormalized to sum to one. Supported transforms: `"neglog10"` (aggressive) and `"one_minus_p"` (conservative). Multipliers are floored internally.

Uncertainty and exploration. The `uncertainty_measure` is the weighted sum of robustly normalized CI widths across responses (each width normalized using the sampled 2nd-98th percentile range, then weighted by the final weights). The largest row is the exploration target; PAM medoids over the top `exploration_top_frac` by this measure are returned as exploration candidates. Both optimal and exploration candidate tables include `score` and `uncertainty_measure`.

Implementation notes. Point predictions use ensemble-mean aggregation (`agg = "mean"`) with `debias = FALSE`, both inside `svem_random_table_multi()` and in the candidate summaries. Percentile CIs use `agg = "mean"`. The geometric combiner uses a fixed floor of `1e-6`; the WMT multiplier floor is `1e-3`. For binomial responses, fits and CI bounds are clamped to `[0,1]`.

Value

A list with the following components:

best One-row data frame at the winning design with predictors, per-response predictions, per-response percentile CIs (if available), the combined `score`, and `uncertainty_measure`.

best_idx Row index of the selected best design in the sampled table.

best_x Predictors at the best design.

best_pred Named numeric vector of predicted responses at `best_x`.

best_ci Data frame of percentile limits at `best_x`.

candidates Data frame of `k_candidates` diverse optimal candidates (medoids; existing rows) with predictors, predictions, percentile CIs, the combined `score`, and `uncertainty_measure`; `NULL` if `k_candidates = 0`.

exploration_best One-row data frame at the exploration target, with predictors, per-response predictions, percentile CIs, `score`, and `uncertainty_measure`.

exploration_best_idx Row index with the largest `uncertainty_measure`.

exploration_best_x Predictors at the exploration target.

exploration_best_pred Predicted responses at `exploration_best_x`.

exploration_best_ci Percentile CIs at `exploration_best_x`.

exploration_candidates Data frame of `k_exploration_candidates` diverse high-uncertainty candidates (medoids; existing rows) with predictors, predictions, percentile CIs, `uncertainty_measure`, and `score`; `NULL` if `k_exploration_candidates = 0`.

score_table Sampled table with responses, per-response desirabilities, weighted terms, optional log-weighted terms (when `combine = "geom"`), CI widths, normalized CI widths, weighted CI widths, the `uncertainty_measure`, and final `score`.

original_data_scored If `data` is provided: `data` augmented with predicted responses, per-response desirabilities, combined `score`, and `uncertainty_measure`. Otherwise `NULL`.

weights_original User-provided weights normalized to sum to one before WMT reweighting.

weights_final Final weights after WMT multipliers and renormalization.

wmt_p_values Named vector of per-response whole-model p-values when `reweight_by_wmt = TRUE`; otherwise `NULL`.

wmt_multipliers Named vector of per-response WMT multipliers when `reweight_by_wmt = TRUE`; otherwise `NULL`.

goals Data frame describing each response goal, weight, target, and echoing original and final weights and (when applicable) WMT information.

Binomial handling

For responses fit with `family = "binomial"`, this function expects predictions on the probability scale. Predicted fits and percentile CI bounds (when available) are internally clamped to `[0,1]` before desirability and uncertainty calculations. To protect current Gaussian behavior, no link-scale transforms are applied. **Reweighting via WMT is not supported when any responses are binomial**; if `reweight_by_wmt = TRUE` and at least one response is binomial, the function stops with an informative error.

Acknowledgments

OpenAI's GPT models (o1-preview and GPT-5 Thinking via ChatGPT) were used to assist with coding and roxygen documentation; all content was reviewed and finalized by the author.

References

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.

See Also

`SVEMnet()`, `svem_random_table_multi()`, `predict.svem_model()`

Examples

```
## --- Larger Gaussian-only example ---
set.seed(1)
n  <- 120
X1 <- runif(n); X2 <- runif(n)
F  <- factor(sample(c("lo","hi"), n, TRUE))
y1 <- 1 + 1.5*X1 - 0.8*X2 + 0.4*(F=="hi") + rnorm(n, 0, 0.2)
y2 <- 0.7 + 0.4*X1 + 0.4*X2 - 0.2*(F=="hi") + rnorm(n, 0, 0.2)
dat <- data.frame(y1, y2, X1, X2, F)

m1 <- SVEMnet(y1 ~ X1 + X2 + F, dat, nBoot = 30, family = "gaussian")
m2 <- SVEMnet(y2 ~ X1 + X2 + F, dat, nBoot = 30, family = "gaussian")
objs <- list(y1 = m1, y2 = m2)

goals <- list(
  y1 = list(goal = "max",    weight = 0.6),
  y2 = list(goal = "target", weight = 0.4, target = 0.9)
)

out <- svem_optimize_random(
  objects      = objs,
  goals        = goals,
  n            = 5000,
  level        = 0.95,
  k_candidates = 5,
  top_frac     = 0.02,
  k_exploration_candidates = 5,
  exploration_top_frac     = 0.01,
  numeric_sampler = "random",
  verbose      = TRUE
)
out$best
head(out$candidates)
out$exploration_best
head(out$exploration_candidates)

## Optional: reweight goals by whole-model p-values (Gaussian-only).
out_wmt <- svem_optimize_random(
  objects         = objs,
  goals           = goals,
  data            = dat,
  n               = 5000,
  level           = 0.95,
  k_candidates    = 5,
  top_frac        = 0.02,
  k_exploration_candidates = 5,
  exploration_top_frac     = 0.01,
  numeric_sampler = "random",
  reweight_by_wmt = TRUE,
  wmt_transform   = "neglog10",
  verbose         = TRUE
)
```

```
out_wmt$weights_original
out_wmt$weights_final
out_wmt$wmt_p_values
head(out_wmt$candidates)
head(out_wmt$exploration_candidates)
head(out_wmt$original_data_scored)

## --- Smaller mixed example: one Gaussian + one Binomial (probability scale) ---
set.seed(42)
n  <- 80
X1 <- runif(n); X2 <- runif(n); G <- factor(sample(c("lo","hi"), n, TRUE))

# Gaussian response
yg <- 2 + 1.1*X1 - 0.7*X2 + 0.5*(G=="hi") + rnorm(n, 0, 0.25)

# Binomial response (probability via logistic link)
eta <- -0.3 + 1.2*X1 - 0.4*X2 + 0.6*(G=="hi")
p   <- 1/(1 + exp(-eta))
yb  <- rbinom(n, 1, p)

dmix <- data.frame(yg, yb, X1, X2, G)

mg <- SVEMnet(yg ~ X1 + X2 + G, dmix, nBoot = 30, family = "gaussian")
mb <- SVEMnet(yb ~ X1 + X2 + G, dmix, nBoot = 30, family = "binomial", relaxed = FALSE)

objs_mix <- list(yg = mg, yb = mb)
goals_mix <- list(
  yg = list(goal = "max", weight = 0.5),
  yb = list(goal = "max", weight = 0.5)  # maximize event probability
)

out_mix <- svem_optimize_random(
  objects      = objs_mix,
  goals        = goals_mix,
  n            = 3000,
  level        = 0.95,
  k_candidates = 3,
  top_frac     = 0.03,
  numeric_sampler = "random",
  reweight_by_wmt = FALSE,  # required when any response is binomial
  verbose      = TRUE
)
out_mix$best
head(out_mix$candidates)
```

---

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. Intended for multiple responses built over the same factor space and a deterministic factor expansion (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", "maximin", "uniform")
)
```

Arguments

`objects`	A list of fitted `svem_model` objects returned by `SVEMnet()`. Each object must contain `$sampling_schema` produced by the updated `SVEMnet()` implementation. A single model is also accepted and treated as a length-one list.
`n`	Number of random points to generate. Default is `1000`.
`mixture_groups`	Optional list of mixture constraint groups. Each group is a list with elements `vars`, `lower`, `upper`, `total` (see *Notes on mixtures*).
`debias`	Logical; if `TRUE`, apply each model's calibration during prediction when available (for Gaussian fits). Default is `FALSE`.
`range_tol`	Numeric tolerance for comparing numeric ranges across models. Default is `1e-8`.
`numeric_sampler`	Sampler for non-mixture numeric predictors: `"random"` (default), `"maximin"`, or `"uniform"`. If `"random"` is selected and the **lhs** package is available, `lhs::randomLHS()` is used; otherwise plain `runif()`.

Details

Predictions are computed via `predict.svem_model(..., agg = "mean")`, i.e. by averaging per-bootstrap member predictions on the requested scale. No refitting is performed.

All models must share an identical predictor schema:

- The same `predictor_vars` in the same order
- The same `var_classes` for each predictor
- Identical factor `levels` and level order
- Numeric `ranges` that match within `range_tol`

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

Models may be Gaussian or binomial; binomial predictions are returned on the probability scale by default.

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). If a response name would collide with a predictor name, `".pred"` is appended.

Sampling strategy

Non-mixture numeric variables are sampled using a selectable method:

- `"random"`: random Latin hypercube when **lhs** is available, else independent uniforms on each range.
- `"maximin"`: maximin Latin hypercube (more space-filling; slower).
- `"uniform"`: independent uniform draws within numeric ranges (fastest).

Mixture variables (if any) are sampled jointly within each specified group using a truncated Dirichlet so that elementwise bounds and the total sum are satisfied. Categorical variables are sampled from cached factor levels. The same random predictor table is fed to each model so response columns are directly comparable.

Notes on mixtures

Each mixture group should list only numeric-like variables. Bounds are interpreted on the original scale of those variables. If `total` equals the sum of lower bounds, the sampler returns the lower-bound corner for that group. Infeasible constraints (i.e., `sum(lower) > total` or `sum(upper) < total`) produce an error.

See Also

`SVEMnet`, `predict.svem_model`

Examples

```
set.seed(1)
n <- 60
X1 <- runif(n); X2 <- runif(n)
A <- runif(n); B <- runif(n); C <- pmax(0, 1 - A - B)
F <- factor(sample(c("lo","hi"), n, TRUE))

## Gaussian responses
y1 <- 1 + 2*X1 - X2 + 3*A + 1.5*B + 0.5*C + (F=="hi") + rnorm(n, 0, 0.3)
y2 <- 0.5 + 0.8*X1 + 0.4*X2 + rnorm(n, 0, 0.2)

## Binomial response (probability via logistic link)
eta  <- -0.5 + 1.2*X1 - 0.7*X2 + 0.8*(F=="hi") + 0.6*A
p    <- 1 / (1 + exp(-eta))
```

```
yb   <- rbinom(n, size = 1, prob = p)

d  <- data.frame(y1, y2, yb, X1, X2, A, B, C, F)

fit1 <- SVEMnet(y1 ~ X1 + X2 + A + B + C + F, d, nBoot = 40, family = "gaussian")
fit2 <- SVEMnet(y2 ~ X1 + X2 + A + B + C + F, d, nBoot = 40, family = "gaussian")
fitb <- SVEMnet(yb ~ X1 + X2 + A + B + C + F, d, nBoot = 40, family = "binomial")

# Mixture constraint for A, B, C that sum to 1
mix <- list(list(vars  = c("A","B","C"),
                 lower = c(0,0,0),
                 upper = c(1,1,1),
                 total = 1))

# Fast random sampler (shared predictor table; predictions bound as columns)
tab_fast <- svem_random_table_multi(
  objects         = list(y1 = fit1, y2 = fit2, yb = fitb),
  n               = 2000,
  mixture_groups  = mix,
  debias          = FALSE,
  numeric_sampler = "random"
)
head(tab_fast$all)

# Check that the binomial predictions are on [0,1]
range(tab_fast$pred$yb)

# Uniform sampler (fastest)
tab_uni <- svem_random_table_multi(
  objects         = list(y1 = fit1, y2 = fit2, yb = fitb),
  n               = 2000,
  debias          = FALSE,
  numeric_sampler = "uniform"
)
head(tab_uni$all)
```

`svem_significance_test_parallel` *SVEM Significance Test with Mixture Support (Parallel Version)*

Description

Whole-model significance test for continuous (Gaussian) SVEM fits, with support for 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("auto", "wAIC", "wBIC", "wSSE"),
  auto_ratio_cutoff = 1.3,
  relaxed = FALSE,
  verbose = TRUE,
  nCore = parallel::detectCores() - 1,
  seed = NULL,
  spec = NULL,
  response = NULL,
  use_spec_contrasts = TRUE,
  ...
)
```

Arguments

formula	A formula specifying the model to be tested. If spec is provided, the right-hand side is ignored and replaced by the locked expansion in spec.
data	A data frame containing the variables in the model.
mixture_groups	Optional list describing one or more mixture factor groups. Each element of the list should be a list with components vars (character vector of column names), lower (numeric vector of lower bounds of the same length as vars), upper (numeric vector of upper bounds of the same length), and total (scalar specifying the sum of the mixture variables). All mixture variables must be included in vars, and no variable can appear in more than one mixture group. Defaults to NULL.
nPoint	Number of random points in the factor space (default: 2000).
nSVEM	Number of SVEM fits on the original (unpermuted) data used to summarize the observed surface (default: 10).
nPerm	Number of SVEM fits on permuted responses used to build the null reference distribution (default: 150).
percent	Percentage of variance to capture in the SVD (default: 90).
nBoot	Number of bootstrap iterations within each SVEM fit (default: 100).
glmnet_alpha	The alpha parameter(s) for glmnet (default: c(1)).
weight_scheme	Weighting scheme for SVEM (default: "SVEM"). Passed to SVEMnet().
objective	Objective used inside SVEMnet() to pick the bootstrap path solution. One of "auto", "wAIC", "wBIC", or "wSSE" (default: "auto"). Note: "wGIC" is no longer supported.

auto_ratio_cutoff
: Single cutoff for the automatic rule when objective = "auto" (default 1.3). With r = n_X / p_X, if r >= auto_ratio_cutoff wAIC is used; otherwise wBIC. Passed through to SVEMnet().

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. This value is passed through to SVEMnet(). Note: 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 parallel processing. Default is parallel::detectCores() - 1, with a floor of 1.

seed
: Optional integer seed for reproducible parallel RNG (default: NULL). When supplied, the master RNG kind is set to "L'Ecuyer-CMRG" with sample.kind = "Rounding", and doRNG::registerDoRNG() is used so that the %dorng% loops are reproducible regardless of scheduling.

spec
: Optional bigexp_spec created by bigexp_terms(). If provided, the test reuses its locked expansion. The working formula becomes bigexp_formula(spec, response_name), where response_name is taken from response if supplied, otherwise from the left-hand side of formula. Categorical sampling uses spec$levels and numeric sampling prefers spec$num_range when available.

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 coding.

...
: Additional arguments passed to SVEMnet() and then to glmnet() (for example: penalty.factor, offset, lower.limits, upper.limits, standardize.response, etc.). The relaxed setting is controlled by the relaxed argument of this function and any relaxed value passed via ... is ignored with a warning.

Details

The test follows Karl (2024): it generates a space-filling grid in the factor space, fits multiple SVEM models on the original data and on permuted responses, standardizes predictions on the grid, 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.

All SVEM refits (for the original and permuted responses) are run in parallel using foreach + doParallel. Random draws (including permutations and evaluation-grid sampling) are made reproducible across workers using doRNG together with RNGkind("L'Ecuyer-CMRG", sample.kind = "Rounding") when a seed is supplied.

The function can optionally reuse a deterministic, locked expansion built with bigexp_terms(). Provide 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

A list of class `svem_significance_test` with components:

- `p_value`: the median whole-model p-value over original SVEM fits.
- `p_values`: vector of p-values for each original SVEM fit.
- `d_Y`: distances for the original SVEM fits.
- `d_pi_Y`: distances for the permutation fits.
- `distribution_fit`: the fitted SHASHo distribution object.
- `data_d`: data frame of distances and source labels, suitable for plotting.

Acknowledgments

OpenAI's GPT models (o1-preview and GPT-5 Thinking via ChatGPT) were used to assist with coding and roxygen documentation; all content was reviewed and finalized by the author.

References

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.

See Also

bigexp_terms, bigexp_formula

Examples

```
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
))
```

```
# Parallel significance test (default relaxed = FALSE)
res <- svem_significance_test_parallel(
  y ~ A + B + C + X + F,
  data           = dat,
  mixture_groups = mix_spec,
  glmnet_alpha   = c(1),
  weight_scheme  = "SVEM",
  objective      = "auto",
  auto_ratio_cutoff = 1.3,
  relaxed        = FALSE,   # default, shown for clarity
  nCore          = 2,
  seed           = 123,
  verbose        = FALSE
)
print(res$p_value)
```

`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.

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, 25
∗ **datasets**
lipid_screen, 17
∗ **package**
SVEMnet-package, 2

bigexp_formula, *3*, 4, *9*, *52*
bigexp_model_matrix, 5, *6*, *7*, *9*
bigexp_prepare, 6, *9*
bigexp_terms, *3*, *6*, *7*, 7, *52*
bigexp_train, *9*, 10

coef.svem_model, *3*, 11, *38*

glmnet_with_cv, *3*, 13

lipid_screen, *3*, 17

plot.svem_binomial, 21
plot.svem_model, *3*, 23
plot.svem_significance_test, *3*, 24
predict.svem_cv *(*predict_cv*)*, 28
predict.svem_model, *3*, 25, *47*
predict.svem_model(), *43*
predict_cv, 28
print.bigexp_spec, 30
print.svem_significance_test, 31

svem_nonzero, *3*, *12*, 37
svem_optimize_random, *3*, 39
svem_random_table_multi, *3*, 45
svem_random_table_multi(), *43*
svem_significance_test_parallel, *3*, 48
SVEMnet, *3*, *27*, 31, *47*
SVEMnet(), *43*
SVEMnet-package, 2

with_bigexp_contrasts, 53