Package: TRexSelector (via r-universe)

November 20, 2026

Type Package

Title T-Rex Selector: High-Dimensional Variable Selection and FDR Control

Version 2.0.3

Description Variable selection for regression and classification in high-dimensional settings with control of the false discovery rate (FDR) at a user-specified target level. The implementations are primarily based on the T-Rex selector of Machkour, Muma and Palomar (2025) <doi:10.1016/j.sigpro.2025.109894> and its dependency-aware extension for correlated variables by Machkour, Muma and Palomar (2025) <doi:10.1016/j.sigpro.2025.109990>. Since version 2.0.0 the package comes as R bindings to the 'TRexSelector' C++ library, which replaces the pure-R implementation of the 1.x releases.

URL https://github.com/trexselector/TRexSelector

BugReports https://github.com/trexselector/TRexSelector/issues

License GPL (>= 3)

Encoding UTF-8

LazyData true

LinkingTo BH, Rcereal, Rcpp, RcppEigen

Depends R (>= 4.3)

Imports methods, R6, Rcpp

Suggests ggplot2, knitr, Matrix, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

SystemRequirements C++20

Config/roxygen2/version 8.1.1

NeedsCompilation yes

Author Jasin Machkour [aut, cre], Fabian Scheidt [aut], Simon Tien [ctb], Daniel P. Palomar [aut], Michael Muma [aut]

Maintainer Jasin Machkour <jasin.machkour@tu-darmstadt.de>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-20 14:30:36 UTC

RemoteUrl https://github.com/cran/TRexSelector

RemoteRef HEAD

RemoteSha 80f6498da1230465bdfd32591f709ce1e9ba3580

Contents

[.mmap_matrix . . . 3
as.matrix.mmap_matrix . . . 4
dim.mmap_matrix . . . 5
Gauss_data . . . 6
mem_matrix . . . 7
mem_matrix_from . . . 8
ml_agglomerative_cluster . . . 9
ml_cut_tree . . . 10
ml_distance_metric . . . 11
ml_linkage_method . . . 12
MLElasticNet . . . 12
MLElasticNetCV . . . 16
MLLpNormScaler . . . 19
MLPCA . . . 23
MLRidge . . . 25
MLRidgeCV . . . 27
MLSVD . . . 30
MLZScoreScaler . . . 31
print.mmap_matrix . . . 34
TACGP_Solver . . . 35
TAFS_Solver . . . 36
TCDENET_Solver . . . 38
TCDIENET_Solver . . . 39
TCDLASSO_Solver . . . 41
TCDTIKNET_Solver . . . 43
TENET_Solver . . . 45
TENETAug_Solver . . . 47
TGP_Solver . . . 48
TIENET_Solver . . . 50
TIENETAug_Solver . . . 52
TIKNET_Solver . . . 54
TLARS_Solver . . . 56
TLASSO_Solver . . . 58
TMP_Solver . . . 60
TNCGMP_Solver . . . 61
TOMP_Solver . . . 62
TOOLS_Solver . . . 64

trex_biobank_control . . . 65
trex_build_group_centering_k . . . 66
trex_control . . . 67
trex_da_control . . . 72
trex_default_lambda2_grid . . . 73
trex_dummy_distribution . . . 74
trex_fdp . . . 76
trex_fdp_dense . . . 77
trex_gamma_to_k . . . 77
trex_get_num_threads . . . 78
trex_gvs_control . . . 79
trex_gvs_lambda2_power_rule . . . 82
trex_normalize_design . . . 84
trex_precision . . . 85
trex_recall . . . 86
trex_screen_control . . . 87
trex_spca_control . . . 88
trex_tikhonov_control . . . 90
trex_tikhonov_lambda2_power_rule . . . 94
trex_tpp . . . 96
trex_tpp_dense . . . 97
TRexBiobankScreeningSelector . . . 97
TRexDASelector . . . 100
TRexGVSSelector . . . 104
TRexScreeningSelector . . . 106
TRexSelector . . . 109
TRexSPCASelector . . . 112
TRexTikhonovSelector . . . 115
tsolver_options . . . 119
TSTAGEWISE_Solver . . . 122
TSTEPWISE_Solver . . . 124

Index **126**

`[.mmap_matrix` *Extract or assign blocks in an mmap_matrix*

Description

Extract or assign blocks in an mmap_matrix

Usage

```
## S3 method for class 'mmap_matrix'
x[i, j, ..., drop = FALSE]
```

Arguments

x	mmap_matrix object
i	row indices
j	column indices
...	unused arguments
drop	logical. If TRUE the result is coerced to the lowest possible dimension (not supported).

Value

For extraction: a matrix block. For assignment: x invisibly.

See Also

Other memory-mapped matrices: `as.matrix.mmap_matrix()`, `dim.mmap_matrix()`, `mem_matrix()`, `mem_matrix_from()`, `print.mmap_matrix()`

Examples

```
mat <- matrix(as.double(1:20), nrow = 4, ncol = 5)
f <- tempfile(fileext = ".bin")
on.exit(unlink(f), add = TRUE)
mem_matrix_from(mat, f)
mm <- mem_matrix(f, rows = 4, cols = 5)
mm[1:2, 1:3]
mm[2, 3] <- 99.0
mm[1:2, 1:3] <- matrix(as.double(1:6), nrow = 2, ncol = 3)
```

`as.matrix.mmap_matrix` *Get Zero-Copy View as Numeric Vector (Block)*

Description

Get Zero-Copy View as Numeric Vector (Block)

Usage

```
## S3 method for class 'mmap_matrix'
as.matrix(x, ...)
```

Arguments

x	mmap_matrix object
...	extra arguments including row_range and col_range

Value

Numeric matrix (copied)

See Also

Other memory-mapped matrices: `[.mmap_matrix()`, `dim.mmap_matrix()`, `mem_matrix()`, `mem_matrix_from()`, `print.mmap_matrix()`

Examples

```
mat <- matrix(as.double(1:20), nrow = 4, ncol = 5)
f <- tempfile(fileext = ".bin")
on.exit(unlink(f), add = TRUE)
mem_matrix_from(mat, f)
mm <- mem_matrix(f, rows = 4, cols = 5)
as.matrix(mm)
```

---

dim.mmap_matrix *Get mmap_matrix dimensions*

Description

Get mmap_matrix dimensions

Usage

```
## S3 method for class 'mmap_matrix'
dim(x)
```

Arguments

x	mmap_matrix object

Value

an integer vector of length 2

See Also

Other memory-mapped matrices: `[.mmap_matrix()`, `as.matrix.mmap_matrix()`, `mem_matrix()`, `mem_matrix_from()`, `print.mmap_matrix()`

Examples

```
mat <- matrix(as.double(1:20), nrow = 4, ncol = 5)
f <- tempfile(fileext = ".bin")
on.exit(unlink(f), add = TRUE)
mem_matrix_from(mat, f)
mm <- mem_matrix(f, rows = 4, cols = 5)
dim(mm)
```

Gauss_data *Toy data generated from a Gaussian linear model*

Description

A data set containing a predictor matrix X with n = 50 observations and p = 100 variables (predictors), and a sparse parameter vector beta with associated support vector.

Usage

```
Gauss_data
```

Format

A list containing a matrix X and vectors y, beta, and support:

X Predictor matrix, n = 50, p = 100.

y Response vector.

beta Parameter vector.

support Support vector.

See Also

Other T-Rex selectors and controls: TRexBiobankScreeningSelector, TRexDASelector, TRexGVSSelector, TRexSPCASelector, TRexScreeningSelector, TRexSelector, TRexTikhonovSelector, trex_biobank_control(), trex_build_group_centering_k(), trex_control(), trex_da_control(), trex_default_lambda2_grid(), trex_dummy_distribution(), trex_fdp(), trex_fdp_dense(), trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_control(), trex_gvs_lambda2_power_rule(), trex_normalize_design(), trex_precision(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
# Generated as follows:
set.seed(789)
n <- 50
p <- 100
X <- matrix(stats::rnorm(n * p), nrow = n, ncol = p)
beta <- c(rep(5, times = 3), rep(0, times = 97))
```

```
support <- beta > 0
y <- X %*% beta + stats::rnorm(n)
Gauss_data <- list(
  X = X,
  y = y,
  beta = beta,
  support = support
)
```

`mem_matrix` *Memory-Mapped Matrix Object*

Description

S3 object wrapping an external pointer to a Cpp MemoryMappedMatrix.

Usage

```
mem_matrix(filename, rows, cols, mode = "readwrite")
```

Arguments

`filename`	Path to binary mmap file
`rows`	Number of rows
`cols`	Number of columns
`mode`	Access mode: 'readonly' or 'readwrite' (default). Code that writes into the matrix needs 'readwrite': a selector that standardizes the design in place and restores it (`scaling_mode` `"L2"` or `"ZSCORE"`), a T-solver with `normalize` or `intercept` on, `TRexSPCASelector` and the in-place scalers. Code that only reads accepts 'readonly' as well: a selector under `scaling_mode = "NONE"`, a T-solver with `normalize` and `intercept` off, and the fits that say so.

Details

A file serves one writer at a time. Do not run several selectors or T-solvers that standardize the file in place on one file at the same time, neither through a second handle in one session nor in several processes. Each would change the data the other works on, and the results come out wrong. To use one design in several processes at once, pre-normalize it with `trex_normalize_design`, open the file in 'readonly' mode and run every selector with `scaling_mode = "NONE"`.

Value

Object of classes `MemoryMappedMatrix` and `mmap_matrix` (S3 XPtr wrapper)

References

F. Scheidt, J. Machkour, and M. Muma, "Solving FDR-Controlled Sparse Regression Problems with Five Million Variables on a Laptop," in Proc. IEEE 9th Int. Workshop Comput. Adv. Multi-Sensor Adapt. Process. (CAMSAP), 2023, pp. 116-120.

See Also

Other memory-mapped matrices: [.mmap_matrix(), as.matrix.mmap_matrix(), dim.mmap_matrix(), mem_matrix_from(), print.mmap_matrix()

Examples

```
mat <- matrix(as.double(1:20), nrow = 4, ncol = 5)
f <- tempfile(fileext = ".bin")
on.exit(unlink(f), add = TRUE)
mem_matrix_from(mat, f)
mm <- mem_matrix(f, rows = 4, cols = 5)
```

---

mem_matrix_from *Convert R Matrix to System Memory Mapped Matrix*

Description

Writes a numeric matrix to a binary memory-mapped file on disk and returns a MemoryMappedMatrix handle pointing to it.

Usage

```
mem_matrix_from(mat, filename)
```

Arguments

mat	A numeric matrix.
filename	String destination file path.

Value

A MemoryMappedMatrix / mmap_matrix object backed by filename.

References

F. Scheidt, J. Machkour, and M. Muma, "Solving FDR-Controlled Sparse Regression Problems with Five Million Variables on a Laptop," in Proc. IEEE 9th Int. Workshop Comput. Adv. Multi-Sensor Adapt. Process. (CAMSAP), 2023, pp. 116-120.

See Also

Other memory-mapped matrices: [.mmap_matrix(), as.matrix.mmap_matrix(), dim.mmap_matrix(), mem_matrix(), print.mmap_matrix()

Examples

```
mat <- matrix(as.double(1:20), nrow = 4, ncol = 5)
f <- tempfile(fileext = ".bin")
on.exit(unlink(f), add = TRUE)
mm <- mem_matrix_from(mat, f)
print(mm)
```

`ml_agglomerative_cluster` *Agglomerative Clustering*

Description

Performs hierarchical agglomerative clustering on a numeric matrix.

Usage

```
ml_agglomerative_cluster(
  data,
  method = ml_linkage_method$Ward,
  metric = ml_distance_metric$Euclidean,
  use_mmap = FALSE,
  verbose = FALSE,
  n_threads = NULL
)
```

Arguments

`data`	A numeric matrix, or an `mmap_matrix` created by `mem_matrix_from`.
`method`	A value from `ml_linkage_method` (e.g. `ml_linkage_method$Ward`).
`metric`	A value from `ml_distance_metric` (e.g. `ml_distance_metric$Euclidean`). Default is Euclidean.
`use_mmap`	Whether to use memory mapping files for intermediate storage. Default is FALSE. Ignored when `data` is already an `mmap_matrix` (always `TRUE`).
`verbose`	Whether the backend prints its allocation line (the size of the distance matrix in RAM or in the memory-mapped file). Default is FALSE.
`n_threads`	Integer, the thread budget of the distance loops (`NULL`, the default: the library's default budget, `trex_get_num_threads()`, initially 1).

Value

A numeric matrix representing the computed hierarchical linkage.

References

R. Sibson, "SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method," Comput. J., vol. 16, no. 1, pp. 30-34, 1973.

J.-P. Benzécri, "Construction d'une classification ascendante hiérarchique par la recherche en chaîne des voisins réciproques," Les Cahiers de l'Analyse des Données, vol. 7, no. 2, pp. 209-218, 1982.

J. Juan, "Programme de classification hiérarchique par l'algorithme de la recherche en chaîne des voisins réciproques," Les Cahiers de l'Analyse des Données, vol. 7, no. 2, pp. 219-225, 1982.

G. N. Lance and W. T. Williams, "A General Theory of Classificatory Sorting Strategies: I. Hierarchical Systems," Comput. J., vol. 9, pp. 373-380, 1967.

D. Müllner, "Modern Hierarchical, Agglomerative Clustering Algorithms," arXiv:1109.2378, 2011.

F. Murtagh and P. Contreras, "Algorithms for Hierarchical Clustering: An Overview," Wiley Interdiscip. Rev. Data Min. Knowl. Discov., vol. 2, no. 1, pp. 86-97, 2012, doi: 10.1002/widm.53.

F. Murtagh and P. Contreras, "Algorithms for Hierarchical Clustering: An Overview, II," Wiley Interdiscip. Rev. Data Min. Knowl. Discov., vol. 7, no. 6, Art. no. e1219, 2017, doi: 10.1002/widm.1219.

See Also

Other ML methods: `MLElasticNet`, `MLElasticNetCV`, `MLLpNormScaler`, `MLPCA`, `MLRidge`, `MLRidgeCV`, `MLSVD`, `MLZScoreScaler`, `ml_cut_tree()`, `ml_distance_metric`, `ml_linkage_method`

Examples

```
set.seed(1)
data <- matrix(rnorm(30), nrow = 10)
ml_agglomerative_cluster(data, method = ml_linkage_method$Ward)
```

`ml_cut_tree` *Cut Hierarchical Linkage Matrix*

Description

Flatten the dendrogram clustering results based on the number of requested groups.

Usage

```
ml_cut_tree(linkage, num_orig_objs, num_clusters)
```

Arguments

`linkage`	A numeric matrix representing linkage, generated by `ml_agglomerative_cluster`.
`num_orig_objs`	Original number of observations data rows.
`num_clusters`	Target flat group capacity.

Value

An integer vector mapping observation indexes to cluster arrays.

See Also

Other ML methods: MLElasticNet, MLElasticNetCV, MLLpNormScaler, MLPCA, MLRidge, MLRidgeCV, MLSVD, MLZScoreScaler, ml_agglomerative_cluster(), ml_distance_metric, ml_linkage_method

Examples

```
set.seed(1)
data <- matrix(rnorm(30), nrow = 10)
linkage <- ml_agglomerative_cluster(data, method = ml_linkage_method$Ward)
ml_cut_tree(linkage, num_orig_objs = 10, num_clusters = 3)
```

---

ml_distance_metric *Distance Metrics*

Description

Enumeration of supported distance metrics for hierarchical clustering. Supported metrics include: - Euclidean: Standard Euclidean distance: codes as 0. - Correlation: Correlation distance (1 - correlation): codes as 1. - Manhattan: Manhattan (L1) distance: codes as 2. - Correlation_LSH_Filter: Correlation distance with LSH-based filtering: codes as 3. - Correlation_LSH_Approx: Approximate correlation distance using LSH: codes as 4.

Usage

```
ml_distance_metric
```

References

M. S. Charikar, "Similarity Estimation Techniques from Rounding Algorithms," in Proc. ACM Symp. Theory Comput. (STOC), 2002.

P. Indyk and R. Motwani, "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality," in Proc. 30th ACM Symp. Theory Comput. (STOC), 1998, pp. 604-613.

P. Li, T. J. Hastie, and K. W. Church, "Very Sparse Random Projections," in Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD), 2006.

See Also

Other ML methods: MLElasticNet, MLElasticNetCV, MLLpNormScaler, MLPCA, MLRidge, MLRidgeCV, MLSVD, MLZScoreScaler, ml_agglomerative_cluster(), ml_cut_tree(), ml_linkage_method

Examples

```
ml_distance_metric$Euclidean
ml_distance_metric$Correlation
```

ml_linkage_method *Linkage Methods*

Description

Enumeration of supported hierarchical clustering linkage methods. Supported methods include: - `Ward`: Ward's minimum variance method: codes as 0. - `Average` (UPGMA): Average linkage: codes as 1. - `Complete`: Complete linkage: codes as 2. - `Single`: Single linkage: codes as 3. - `WPGMA`: Weighted pair group method with arithmetic mean: codes as 4. - `Median`: Median linkage: codes as 5. - `Centroid`: Centroid linkage: codes as 6.

Usage

```
ml_linkage_method
```

References

G. N. Lance and W. T. Williams, "A General Theory of Classificatory Sorting Strategies: I. Hierarchical Systems," Comput. J., vol. 9, pp. 373-380, 1967.

D. Müllner, "Modern Hierarchical, Agglomerative Clustering Algorithms," arXiv:1109.2378, 2011.

See Also

Other ML methods: `MLElasticNet`, `MLElasticNetCV`, `MLLpNormScaler`, `MLPCA`, `MLRidge`, `MLRidgeCV`, `MLSVD`, `MLZScoreScaler`, `ml_agglomerative_cluster()`, `ml_cut_tree()`, `ml_distance_metric`

Examples

```
ml_linkage_method$Ward
ml_linkage_method$Average
```

MLElasticNet *Elastic Net (coordinate-descent path)*

Description

Fits the elastic-net coefficient path via coordinate descent over a glmnet-style lambda grid (or an explicit grid). Reports coefficients in the original predictor scale, intercepts, the deviance-ratio path, and predictions for new data. The mixing parameter `alpha` interpolates between lasso (`alpha = 1`) and ridge (`alpha = 0`).

Methods

Public methods:

- MLElasticNet$new()
- MLElasticNet$fit()
- MLElasticNet$fit_grid()
- MLElasticNet$get_coef()
- MLElasticNet$get_intercepts()
- MLElasticNet$get_lambdas()
- MLElasticNet$get_dev_ratio()
- MLElasticNet$converged()
- MLElasticNet$n_nonconverged()
- MLElasticNet$predict()
- MLElasticNet$clone()

`MLElasticNet$new()`: Initialize the MLElasticNet object.

Usage:

```
MLElasticNet$new(low_memory = FALSE, n_threads = NULL)
```

Arguments:

`low_memory` Logical (default `FALSE`). `FALSE`: the coordinate descent runs on a centered and standardized working copy of X (one n x p matrix while fitting). `TRUE`: it runs on X itself (an `mmap_matrix` is read in place, `"readonly"` suffices) with no allocation of the size of X; same estimator and grid, the two engines agree to rounding (about 1e-12 in the coefficients).

`n_threads` Integer, the thread budget of the fit's matrix products (`NULL`, the default: the library's default budget, `trex_get_num_threads()`, initially 1).

`MLElasticNet$fit()`: Fit the EN path over an auto-generated glmnet-style grid.

Usage:

```
MLElasticNet$fit(
  X,
  y,
  alpha = 1,
  n_lambda = 100,
  lambda_min_ratio = -1,
  standardize = TRUE,
  intercept = TRUE,
  use_strong_rule = FALSE,
  max_iter = 1e+05,
  tol = 1e-07
)
```

Arguments:

`X` Predictor matrix (n x p), or an `mmap_matrix` (bound zero-copy; `"readonly"` suffices, the fit does not modify X).

`y` Response vector (n).

`alpha` Mixing in `[0, 1]`: 1 = lasso, 0 = ridge (default 1).

`n_lambda` Grid size (default 100).

`lambda_min_ratio` Ratio lambda_min/lambda_max; negative (default) auto-selects glmnet's rule.

`standardize` Standardize columns to unit SD (default TRUE).

`intercept` Fit an unpenalized intercept (default TRUE).

`use_strong_rule` Enable the sequential strong rule (default FALSE).

`max_iter` Max CD sweeps per lambda (default 100000).

`tol` Convergence tolerance (default 1e-7).

Returns: Invisibly returns `self`.

`MLElasticNet$fit_grid()`: Fit the EN path at an explicit lambda grid (sorted descending internally); use to evaluate at exactly glmnet's sequence.

Usage:

```
MLElasticNet$fit_grid(
  X,
  y,
  lambda_grid,
  alpha = 1,
  standardize = TRUE,
  intercept = TRUE,
  use_strong_rule = FALSE,
  max_iter = 1e+05,
  tol = 1e-07
)
```

Arguments:

`X` Predictor matrix (n x p), or an `mmap_matrix` (zero-copy).

`y` Response vector (n).

`lambda_grid` Explicit lambda values (any order, > 0).

`alpha` Mixing in `[0, 1]` (default 1).

`standardize` Standardize columns to unit SD (default TRUE).

`intercept` Fit an unpenalized intercept (default TRUE).

`use_strong_rule` Enable the sequential strong rule (default FALSE).

`max_iter` Max CD sweeps per lambda (default 100000).

`tol` Convergence tolerance (default 1e-7).

Returns: Invisibly returns `self`.

`MLElasticNet$get_coef()`: Coefficient path in original predictor scale (p x n_lambda).

Usage:

```
MLElasticNet$get_coef()
```

Returns: Numeric matrix.

`MLElasticNet$get_intercepts()`: Intercept per lambda (zero if `intercept = FALSE`).

Usage:

```
MLElasticNet$get_intercepts()
```

Returns: Numeric vector.

`MLElasticNet$get_lambdas()`: Lambda grid actually used (descending).

Usage:

```
MLElasticNet$get_lambdas()
```

Returns: Numeric vector.

`MLElasticNet$get_dev_ratio()`: Fraction of null deviance explained per lambda (glmnet %Dev).

Usage:

```
MLElasticNet$get_dev_ratio()
```

Returns: Numeric vector.

`MLElasticNet$converged()`: Whether every lambda reached the convergence tolerance.

Usage:

```
MLElasticNet$converged()
```

Returns: Logical scalar.

`MLElasticNet$n_nonconverged()`: Number of lambdas whose coordinate descent hit `max_iter`.

Usage:

```
MLElasticNet$n_nonconverged()
```

Returns: Integer.

`MLElasticNet$predict()`: Predictions per lambda for new data.

Usage:

```
MLElasticNet$predict(X_new)
```

Arguments:

`X_new` Predictor matrix with the same number of columns as the fit, or an `mmap_matrix` (zero-copy).

Returns: Numeric matrix (nrow(X_new) x n_lambda).

`MLElasticNet$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MLElasticNet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

H. Zou and T. Hastie, "Regularization and Variable Selection via the Elastic Net," J. R. Stat. Soc. Ser. B Stat. Methodol., vol. 67, no. 2, pp. 301-320, 2005.

J. Friedman, T. Hastie, and R. Tibshirani, "Regularization Paths for Generalized Linear Models via Coordinate Descent," J. Stat. Softw., vol. 33, no. 1, pp. 1-22, 2010, doi: 10.18637/jss.v033.i01.

See Also

Other ML methods: MLElasticNetCV, MLLpNormScaler, MLPCA, MLRidge, MLRidgeCV, MLSVD, MLZScoreScaler, ml_agglomerative_cluster(), ml_cut_tree(), ml_distance_metric, ml_linkage_method

Examples

```
set.seed(1)
n <- 50; p <- 8
X <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + X[, 3] - X[, 5] + rnorm(n)
en <- MLElasticNet$new()
en$fit(X, y, alpha = 1.0)
dim(en$get_coef())        # p x n_lambda
en$predict(X[1:5, ])
```

MLElasticNetCV *Elastic Net with Cross-Validation*

Description

Selects the elastic-net penalty by K-fold cross-validation over a shared glmnet-style lambda grid (coordinate descent per fold), returning `lambda.min` and `lambda.1se`. The mixing parameter `alpha` interpolates between lasso (`alpha = 1`) and ridge (`alpha = 0`).

Methods

Public methods:

- MLElasticNetCV$new()
- MLElasticNetCV$fit()
- MLElasticNetCV$cv_min()
- MLElasticNetCV$cv_1se()
- MLElasticNetCV$index_min()
- MLElasticNetCV$index_1se()
- MLElasticNetCV$get_lambdas()
- MLElasticNetCV$get_cv_errors()
- MLElasticNetCV$get_cv_std()
- MLElasticNetCV$clone()

`MLElasticNetCV$new()`: Initialize the MLElasticNetCV object.

Usage:

```
MLElasticNetCV$new(low_memory = FALSE, n_threads = NULL)
```

Arguments:

`low_memory` Logical (default `FALSE`); `TRUE` runs the full-data fit and every fold on X itself with a 0/1 training mask (no allocation of the size of X; an `mmap_matrix` is read in place). See MLElasticNet.

`n_threads` Integer, the thread budget of the matrix products of the fit and its folds (`NULL`, the default: the library's default budget, `trex_get_num_threads()`, initially 1).

`MLElasticNetCV$fit()`: Fit K-fold CV over a glmnet-style grid.

Usage:

```
MLElasticNetCV$fit(
  X,
  y,
  alpha = 0,
  n_folds = 10,
  n_lambda = 100,
  lambda_min_ratio = -1,
  seed = 0,
  standardize = TRUE,
  intercept = TRUE,
  max_iter = 1e+05,
  tol = 1e-07
)
```

Arguments:

`X` Predictor matrix (n x p), or an `mmap_matrix` (bound zero-copy; `"readonly"` suffices, the fit does not modify X).

`y` Response vector (n).

`alpha` Mixing in `[0, 1]`: 1 = lasso, 0 = ridge (default 0).

`n_folds` Number of folds (default 10).

`n_lambda` Grid size (default 100).

`lambda_min_ratio` Ratio lambda_min/lambda_max; negative (default) auto-selects glmnet's rule.

`seed` Fold-permutation RNG seed (default 0).

`standardize` Standardize columns to unit SD (default TRUE).

`intercept` Fit an unpenalized intercept (default TRUE).

`max_iter` Max CD sweeps per lambda (default 100000).

`tol` Convergence tolerance (default 1e-7).

Returns: Invisibly returns `self`.

`MLElasticNetCV$cv_min()`: Lambda that minimizes the mean CV error.

Usage:

```
MLElasticNetCV$cv_min()
```

Returns: Scalar numeric.

`MLElasticNetCV$cv_1se()`: Largest lambda within one SE of the minimum (1SE rule).

Usage:

```
MLElasticNetCV$cv_1se()
```

Returns: Scalar numeric.

`MLElasticNetCV$index_min()`: Index (1-based) of `lambda.min`.

Usage:

```
MLElasticNetCV$index_min()
```

Returns: Integer.

`MLElasticNetCV$index_1se()`: Index (1-based) of `lambda.1se`.

Usage:

```
MLElasticNetCV$index_1se()
```

Returns: Integer.

`MLElasticNetCV$get_lambdas()`: Lambda grid used (descending).

Usage:

```
MLElasticNetCV$get_lambdas()
```

Returns: Numeric vector.

`MLElasticNetCV$get_cv_errors()`: Mean CV MSE per lambda.

Usage:

```
MLElasticNetCV$get_cv_errors()
```

Returns: Numeric vector.

`MLElasticNetCV$get_cv_std()`: Standard error of the CV MSE per lambda.

Usage:

```
MLElasticNetCV$get_cv_std()
```

Returns: Numeric vector.

`MLElasticNetCV$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MLElasticNetCV$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

H. Zou and T. Hastie, "Regularization and Variable Selection via the Elastic Net," J. R. Stat. Soc. Ser. B Stat. Methodol., vol. 67, no. 2, pp. 301-320, 2005.

J. Friedman, T. Hastie, and R. Tibshirani, "Regularization Paths for Generalized Linear Models via Coordinate Descent," J. Stat. Softw., vol. 33, no. 1, pp. 1-22, 2010, doi: 10.18637/jss.v033.i01.

R. Tibshirani, J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R. J. Tibshirani, "Strong Rules for Discarding Predictors in Lasso-Type Problems," J. R. Stat. Soc. Ser. B Stat. Methodol., vol. 74, no. 2, pp. 245-266, 2012, doi: 10.1111/j.1467-9868.2011.01004.x.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, Classification and Regression Trees. Wadsworth, Belmont, CA, 1984.

See Also

Other ML methods: MLElasticNet, MLLpNormScaler, MLPCA, MLRidge, MLRidgeCV, MLSVD, MLZScoreScaler, ml_agglomerative_cluster(), ml_cut_tree(), ml_distance_metric, ml_linkage_method

Examples

```
set.seed(1)
n <- 60; p <- 10
X <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 - X[, 4] + rnorm(n)
cv <- MLElasticNetCV$new()
cv$fit(X, y, alpha = 0.5)
cv$cv_min()
```

MLLpNormScaler *Lp-Norm Scaler*

Description

Scales input features to unit norm based on the chosen L1 or L2 formulation. The center/scale arguments follow R's scale(): the norm is computed around the applied center (around 0 when center = FALSE).

Details

The *_inplace methods mutate X directly (zero-copy) when it is a double-storage matrix; other inputs are coerced to a double matrix first, in which case the coerced copy is modified. The (modified) matrix is returned in both cases. X may also be an mmap_matrix created by mem_matrix / mem_matrix_from: the disk-backed buffer is then bound zero-copy through the same Eigen map interface and modified in place on disk. All scaler methods — including fit() — require the handle to be open in "readwrite" mode, because the scaler core binds a mutable map.

Methods

Public methods:

- MLLpNormScaler$new()
- MLLpNormScaler$fit()
- MLLpNormScaler$transform_inplace()
- MLLpNormScaler$inverse_transform_inplace()
- MLLpNormScaler$fit_transform_inplace()
- MLLpNormScaler$is_fitted()
- MLLpNormScaler$get_dropped_indices()
- MLLpNormScaler$get_centers()
- MLLpNormScaler$get_scales()
- MLLpNormScaler$get_center()

- MLLpNormScaler$get_scale()
- MLLpNormScaler$get_norm_type()
- MLLpNormScaler$save()
- MLLpNormScaler$load()
- MLLpNormScaler$clone()

`MLLpNormScaler$new()`: Initialize the MLLpNormScaler

Usage:

```
MLLpNormScaler$new(
  norm_type = 2,
  center = TRUE,
  scale = TRUE,
  n_threads = NULL
)
```

Arguments:

`norm_type` Integer, 1 for L1 norm, 2 for L2 norm (default: 2)

`center` Logical, whether to center columns to mean 0 (default: TRUE)

`scale` Logical, whether to scale columns to unit Lp norm (default: TRUE)

`n_threads` Integer, the thread budget of the column passes (`NULL`, the default: the library's default budget, `trex_get_num_threads()`, initially 1).

`MLLpNormScaler$fit()`: Compute the norm parameters.

Usage:

```
MLLpNormScaler$fit(X, threshold = 1e-12)
```

Arguments:

`X` A numeric matrix (n x p) or an `mmap_matrix` in `"readwrite"` mode (the scaler core binds a mutable map).

`threshold` Double, lower bound for the scale statistic (default: 1e-12).

Returns: Value.

`MLLpNormScaler$transform_inplace()`: Perform standardization in-place.

Usage:

```
MLLpNormScaler$transform_inplace(X)
```

Arguments:

`X` A numeric matrix (n x p) or an `mmap_matrix` in `"readwrite"` mode.

Returns: The modified matrix `X` (the same handle for an `mmap_matrix`).

`MLLpNormScaler$inverse_transform_inplace()`: Inverse transform to original scale.

Usage:

```
MLLpNormScaler$inverse_transform_inplace(X)
```

Arguments:

`X` A numeric matrix (n x p) or an `mmap_matrix` in `"readwrite"` mode.

Returns: The modified matrix `X` (the same handle for an `mmap_matrix`).

`MLLpNormScaler$fit_transform_inplace()`: Fit and transform in one call, like R's scale().

Usage:

```
MLLpNormScaler$fit_transform_inplace(X, threshold = 1e-12)
```

Arguments:

`X` A numeric matrix (n x p) or an `mmap_matrix` in `"readwrite"` mode.

`threshold` Double, lower bound for the scale statistic (default: 1e-12).

Returns: The modified matrix `X` (the same handle for an `mmap_matrix`).

`MLLpNormScaler$is_fitted()`: Check if scaler is fitted.

Usage:

```
MLLpNormScaler$is_fitted()
```

Returns: Value.

`MLLpNormScaler$get_dropped_indices()`: Get dropped (degenerate) column indices. These columns get scale factor 1 and are still centered (sklearn semantics).

Usage:

```
MLLpNormScaler$get_dropped_indices()
```

Returns: Value.

`MLLpNormScaler$get_centers()`: Get the learned centers (column means).

Usage:

```
MLLpNormScaler$get_centers()
```

Returns: Value.

`MLLpNormScaler$get_scales()`: Get the learned scales.

Usage:

```
MLLpNormScaler$get_scales()
```

Returns: Value.

`MLLpNormScaler$get_center()`: Check if scaler centers data.

Usage:

```
MLLpNormScaler$get_center()
```

Returns: Value.

`MLLpNormScaler$get_scale()`: Check if scaler scales data to unit norm.

Usage:

```
MLLpNormScaler$get_scale()
```

Returns: Value.

`MLLpNormScaler$get_norm_type()`: Get norm type (1 for L1, 2 for L2).

Usage:

```
MLLpNormScaler$get_norm_type()
```

Returns: Value.

`MLLpNormScaler$save()`: Save model state to file.

Usage:

```
MLLpNormScaler$save(filename)
```

Arguments:

`filename` String path to save output.

Returns: Value.

`MLLpNormScaler$load()`: Load model state from file.

Usage:

```
MLLpNormScaler$load(filename)
```

Arguments:

`filename` String path to load input from.

Returns: Value.

`MLLpNormScaler$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MLLpNormScaler$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other ML methods: `MLElasticNet`, `MLElasticNetCV`, `MLPCA`, `MLRidge`, `MLRidgeCV`, `MLSVD`, `MLZScoreScaler`, `ml_agglomerative_cluster()`, `ml_cut_tree()`, `ml_distance_metric`, `ml_linkage_method`

Examples

```
set.seed(1)
X <- matrix(rnorm(50 * 5), 50, 5)
scaler <- MLLpNormScaler$new(norm_type = 2)
scaler$fit(X)
X_norm <- scaler$transform_inplace(X)
```

MLPCA *Principal Component Analysis*

Description

Fits PCA to a data matrix, retaining the top-M principal components. Centering/normalization is applied to X in-place at construction time (the R matrix or the memory-mapped file itself, no copy) and undone by `restore()`, which may be called before or after `fit()` and is idempotent; an object that is garbage-collected without `restore()` writes X back in its finalizer (the same rule as the T-solvers).

Details

Wraps the C++ PCA class. The instance is bound to X at creation (mirroring the C++ constructor); `fit()` is single-use. The fit result is cached internally; use the `get_*()` accessors or pass `X_new` to `transform()` for out-of-sample scoring.

Methods

Public methods:

- `MLPCA$new()`
- `MLPCA$fit()`
- `MLPCA$transform()`
- `MLPCA$restore()`
- `MLPCA$get_scores()`
- `MLPCA$get_loadings()`
- `MLPCA$get_explained_variance()`
- `MLPCA$get_means()`
- `MLPCA$get_norms()`
- `MLPCA$clone()`

`MLPCA$new()`: Initialize PCA bound to X. Centering/normalization is applied to X in-place immediately.

Usage:

```
MLPCA$new(X, M, center = TRUE, normalize = TRUE, n_threads = NULL)
```

Arguments:

`X` A numeric matrix (n x p), or an `mmap_matrix` (`"readwrite"`: the file is modified). Modified in place when centering/normalizing; call `restore()` to undo.

`M` Integer, number of principal components.

`center` Logical, whether to center X in-place (default: TRUE).

`normalize` Logical, whether to scale the columns of X in-place (default: TRUE).

`n_threads` Integer, the thread budget of the matrix products of `fit()` and `transform()` (NULL, the default: the library's default budget, `trex_get_num_threads()`, initially 1).

`MLPCA$fit()`: Fit the PCA bound at construction time (single-use).

Usage:

```
MLPCA$fit()
```

Returns: Invisible `self` for method chaining.

`MLPCA$transform()`: Project new data onto the fitted principal subspace.

Usage:

```
MLPCA$transform(X_new)
```

Arguments:

`X_new` A numeric matrix (n_new x p), or an `mmap_matrix` (zero-copy). The stored training means/norms are applied before projection.

Returns: Score matrix (n_new x M).

`MLPCA$restore()`: Restore `X` to its original values (un-center/un-scale).

Usage:

```
MLPCA$restore()
```

Returns: Invisible `self`.

`MLPCA$get_scores()`: Get the PC score matrix from the `fit()` call.

Usage:

```
MLPCA$get_scores()
```

Returns: Score matrix (n x M), or `NULL` if `fit()` not yet called.

`MLPCA$get_loadings()`: Get loading matrix from the `fit()` call.

Usage:

```
MLPCA$get_loadings()
```

Returns: Loading matrix (p x M), or `NULL` if `fit()` not yet called.

`MLPCA$get_explained_variance()`: Get explained variance per component from the `fit()` call.

Usage:

```
MLPCA$get_explained_variance()
```

Returns: Numeric vector of length M, or `NULL` if `fit()` not yet called.

`MLPCA$get_means()`: Get the column means used for centering.

Usage:

```
MLPCA$get_means()
```

Returns: Numeric vector of length p. Zero-vector when `center = FALSE`.

`MLPCA$get_norms()`: Get the column scaling factors used for normalization.

Usage:

```
MLPCA$get_norms()
```

Returns: Numeric vector of length p. Ones when `normalize = FALSE`.

`MLPCA$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MLPCA$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

G. Guennebaud, B. Jacob et al., Eigen v3, https://libeigen.gitlab.io/, 2010.

See Also

Other ML methods: `MLElasticNet`, `MLElasticNetCV`, `MLLpNormScaler`, `MLRidge`, `MLRidgeCV`, `MLSVD`, `MLZScoreScaler`, `ml_agglomerative_cluster()`, `ml_cut_tree()`, `ml_distance_metric`, `ml_linkage_method`

Examples

```
set.seed(42)
X <- matrix(rnorm(50 * 8), 50, 8)
pca <- MLPCA$new(X, M = 3, center = TRUE)
pca$fit()
pca$get_explained_variance()
X_new <- matrix(rnorm(10 * 8), 10, 8)
pca$transform(X_new)
pca$restore()
```

`MLRidge` *Ridge Regression Solver*

Description

Solves the ridge regression problem $\min_\beta \|y - X\beta\|^2 + \lambda\|\beta\|^2$ for a single regularization value. Dispatches internally between primal ($n \geq p$) and dual ($n < p$) Cholesky paths.

Details

Wraps the C++ `MLRidge::solve(X, y, lambda)` static method.

Methods

Public methods:

- `MLRidge$solve()`
- `MLRidge$clone()`

`MLRidge$solve()`: Solve ridge regression for a single `lambda`.

Usage:

```
MLRidge$solve(X, y, lambda, n_threads = NULL)
```

Arguments:

`X` A numeric matrix (n x p), or an `mmap_matrix` (bound zero-copy; `"readonly"` suffices).

`y` Numeric response vector of length n.

`lambda` Non-negative regularization parameter.

`n_threads` Integer, the thread budget of the matrix products (`NULL`, the default: the library's default budget, `trex_get_num_threads()`, initially 1).

Returns: Coefficient vector of length p.

`MLRidge$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MLRidge$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

A. E. Hoerl and R. W. Kennard, "Ridge Regression: Biased Estimation for Nonorthogonal Problems," Technometrics, vol. 42, no. 1, pp. 80-86, 2000 (Reprint of the 1970 article in the Special 40th Anniversary Issue).

See Also

Other ML methods: `MLElasticNet`, `MLElasticNetCV`, `MLLpNormScaler`, `MLPCA`, `MLRidgeCV`, `MLSVD`, `MLZScoreScaler`, `ml_agglomerative_cluster()`, `ml_cut_tree()`, `ml_distance_metric`, `ml_linkage_method`

Examples

```
set.seed(42)
n <- 40; p <- 5
X <- matrix(rnorm(n * p), n, p)
beta_true <- c(1, -2, 0, 0.5, 0)
y <- X %*% beta_true + rnorm(n, sd = 0.1)
solver <- MLRidge$new()
beta_hat <- solver$solve(X, y, lambda = 0.01)
```

MLRidgeCV *Ridge Regression with Cross-Validation*

Description

Performs explicit K-Fold cross-validation for Ridge Regression model selection.

Methods

Public methods:

- MLRidgeCV$new()
- MLRidgeCV$fit()
- MLRidgeCV$cv_min()
- MLRidgeCV$cv_1se()
- MLRidgeCV$index_min()
- MLRidgeCV$index_1se()
- MLRidgeCV$get_lambdas()
- MLRidgeCV$get_cv_errors()
- MLRidgeCV$get_cv_std()
- MLRidgeCV$interpolates()
- MLRidgeCV$selection_at_grid_edge()
- MLRidgeCV$clone()

`MLRidgeCV$new()`: Initialize the MLRidgeCV object.

Usage:

```
MLRidgeCV$new(low_memory = FALSE, n_threads = NULL)
```

Arguments:

`low_memory` Logical (default `FALSE`). `FALSE`: each fold gathers and standardizes its training rows and takes a thin SVD (about three to four copies of X in flight). `TRUE`: the dual (kernel) form — n x n kernels accumulated from column blocks of X read in place (an `mmap_matrix` included, `"readonly"` suffices), no p-dimensional coefficients; same estimator, grid and rank rule, agreement to numerical precision; memory O(n^2) instead of O(n p), the route for p » n.

`n_threads` Integer, the thread budget of the fit's matrix products (`NULL`, the default: the library's default budget, `trex_get_num_threads()`, initially 1).

`MLRidgeCV$fit()`: Fit the CV model.

Usage:

```
MLRidgeCV$fit(
  X,
  y,
  n_folds = 10,
  n_lambda = 100,
```

```
  lambda_ratio = 1000,
  seed = 0
)
```

Arguments:

`X` Predictor matrix, or an `mmap_matrix` created by `mem_matrix` / `mem_matrix_from` (bound zero-copy; `"readonly"` mode suffices since the fit does not modify `X`).

`y` Response vector.

`n_folds` Number of folds (default 10).

`n_lambda` Number of lambdas in the path (default 100).

`lambda_ratio` Ratio of max to min lambda (default 1000).

`seed` Random seed for CV splits (default 0).

Returns: Value.

`MLRidgeCV$cv_min()`: Get lambda that minimizes CV error.

Usage:

```
MLRidgeCV$cv_min()
```

Returns: Value.

`MLRidgeCV$cv_1se()`: Get 1SE rule lambda.

Usage:

```
MLRidgeCV$cv_1se()
```

Returns: Value.

`MLRidgeCV$index_min()`: Get index of minimum CV error.

Usage:

```
MLRidgeCV$index_min()
```

Returns: Value.

`MLRidgeCV$index_1se()`: Get index of 1SE rule lambda.

Usage:

```
MLRidgeCV$index_1se()
```

Returns: Value.

`MLRidgeCV$get_lambdas()`: Get the sequence of lambdas used.

Usage:

```
MLRidgeCV$get_lambdas()
```

Returns: Value.

`MLRidgeCV$get_cv_errors()`: Get the CV mean squared errors for each lambda.

Usage:

```
MLRidgeCV$get_cv_errors()
```

Returns: Value.

`MLRidgeCV$get_cv_std()`: Get the standard error of the CV errors.

Usage:

```
MLRidgeCV$get_cv_std()
```

Returns: Value.

`MLRidgeCV$interpolates()`: Whether the smallest training fold has p >= n_train, so the lambda -> 0 end of the grid tends to the minimum-norm interpolator instead of an ordinary least-squares fit. The curve still has a genuine interior minimum in that regime; together with `selection_at_grid_edge()` it tells whether `cv_min()` is a real minimum or the grid running out.

Usage:

```
MLRidgeCV$interpolates()
```

Returns: Logical.

`MLRidgeCV$selection_at_grid_edge()`: Whether `cv_min()` or `cv_1se()` landed on the first or the last grid point. The pick is then the grid boundary, not an optimum, and `lambda_ratio` or `n_lambda` need widening.

Usage:

```
MLRidgeCV$selection_at_grid_edge()
```

Returns: Logical.

`MLRidgeCV$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MLRidgeCV$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

A. E. Hoerl and R. W. Kennard, "Ridge Regression: Biased Estimation for Nonorthogonal Problems," Technometrics, vol. 42, no. 1, pp. 80-86, 2000 (Reprint of the 1970 article in the Special 40th Anniversary Issue).

G. Guennebaud, B. Jacob et al., Eigen v3, https://libeigen.gitlab.io/, 2010.

See Also

Other ML methods: `MLElasticNet`, `MLElasticNetCV`, `MLLpNormScaler`, `MLPCA`, `MLRidge`, `MLSVD`, `MLZScoreScaler`, `ml_agglomerative_cluster()`, `ml_cut_tree()`, `ml_distance_metric`, `ml_linkage_method`

Examples

```
set.seed(1)
X <- matrix(rnorm(50 * 5), 50, 5)
y <- X[, 1] * 2 + rnorm(50)
cv <- MLRidgeCV$new()
cv$fit(X, y, n_folds = 5)
cv$cv_min()
cv$selection_at_grid_edge()   # FALSE: the picks are interior grid points
```

MLSVD *SVD Solver*

Description

Computes the top-M truncated SVD of a matrix. Dispatches internally between a direct (tall matrices), Gram (wide matrices), and randomized path based on the shape of `X`.

Details

Wraps the C++ `MLSVD::compute(X, M)` static method.

Methods

Public methods:

- `MLSVD$new()`
- `MLSVD$compute()`
- `MLSVD$clone()`

`MLSVD$new()`: Initialize the solver with its dispatch thresholds.

Usage:

```
MLSVD$new(min_dim_thresh = 10000, block_size = 1024, n_threads = NULL)
```

Arguments:

`min_dim_thresh` Integer (default 10000). The direct path (a Jacobi SVD that copies X and computes every singular triplet) is taken while `min(n, p)` is below it; above it, and when `M` < `min(n, p) / 10`, the randomized path runs two block-wise passes over X without a copy (the route for a large `mmap_matrix`). `p > 2 n` always takes the Gram path.

`block_size` Integer (default 1024): the column block of the block-wise passes.

`n_threads` Integer, the thread budget of the matrix products (`NULL`, the default: the library's default budget, `trex_get_num_threads()`, initially 1).

`MLSVD$compute()`: Compute top-M singular value decomposition of `X`.

Usage:

```
MLSVD$compute(X, M)
```

Arguments:

`X` A numeric matrix (n x p), or an `mmap_matrix` (bound zero-copy; `"readonly"` suffices).

`M` Integer, number of singular components to retain.

Returns: Named list with:

- `U` Left singular vectors (n x M).
- `S` Singular values (M).
- `V` Right singular vectors (p x M).

`MLSVD$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MLSVD$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

G. Guennebaud, B. Jacob et al., Eigen v3, https://libeigen.gitlab.io/, 2010.

See Also

Other ML methods: MLElasticNet, MLElasticNetCV, MLLpNormScaler, MLPCA, MLRidge, MLRidgeCV, MLZScoreScaler, ml_agglomerative_cluster(), ml_cut_tree(), ml_distance_metric, ml_linkage_method

Examples

```
set.seed(42)
X <- matrix(rnorm(50 * 10), 50, 10)
solver <- MLSVD$new()
res <- solver$compute(X, M = 3)
dim(res$U)  # 50 x 3
```

MLZScoreScaler *Z-Score Scaler*

Description

Standardizes features by removing the mean and scaling to unit variance (Bessel-corrected, n - 1). The center/scale arguments follow R's scale(): with center = FALSE, scale = TRUE columns are divided by the root-mean-square around 0.

Details

The *_inplace methods mutate X directly (zero-copy) when it is a double-storage matrix; other inputs are coerced to a double matrix first, in which case the coerced copy is modified. The (modified) matrix is returned in both cases. X may also be an mmap_matrix created by mem_matrix / mem_matrix_from: the disk-backed buffer is then bound zero-copy through the same Eigen map interface and modified in place on disk. All scaler methods — including fit() — require the handle to be open in "readwrite" mode, because the scaler core binds a mutable map.

Methods

Public methods:

- MLZScoreScaler$new()
- MLZScoreScaler$fit()
- MLZScoreScaler$transform_inplace()

- MLZScoreScaler$inverse_transform_inplace()
- MLZScoreScaler$fit_transform_inplace()
- MLZScoreScaler$is_fitted()
- MLZScoreScaler$get_dropped_indices()
- MLZScoreScaler$get_center()
- MLZScoreScaler$get_scale()
- MLZScoreScaler$get_centers()
- MLZScoreScaler$get_scales()
- MLZScoreScaler$save()
- MLZScoreScaler$load()
- MLZScoreScaler$clone()

`MLZScoreScaler$new()`: Initialize the MLZScoreScaler

Usage:

```
MLZScoreScaler$new(center = TRUE, scale = TRUE, n_threads = NULL)
```

Arguments:

`center` Logical, whether to center columns to mean 0 (default: TRUE)

`scale` Logical, whether to scale columns to unit SD (default: TRUE); around 0 instead of the mean when `center = FALSE`, as in R's scale().

`n_threads` Integer, the thread budget of the column passes (`NULL`, the default: the library's default budget, `trex_get_num_threads()`, initially 1).

`MLZScoreScaler$fit()`: Compute the centers and scales to be used for later scaling.

Usage:

```
MLZScoreScaler$fit(X, threshold = 1e-12)
```

Arguments:

`X` A numeric matrix (n x p) or an `mmap_matrix` in `"readwrite"` mode (the scaler core binds a mutable map).

`threshold` Double, lower bound for the scale statistic (default: 1e-12).

Returns: Value.

`MLZScoreScaler$transform_inplace()`: Perform standardization by centering and scaling in-place.

Usage:

```
MLZScoreScaler$transform_inplace(X)
```

Arguments:

`X` A numeric matrix (n x p) or an `mmap_matrix` in `"readwrite"` mode.

Returns: The modified matrix `X` (the same handle for an `mmap_matrix`).

`MLZScoreScaler$inverse_transform_inplace()`: Inverse transform to original scale.

Usage:

```
MLZScoreScaler$inverse_transform_inplace(X)
```

Arguments:

`X` A numeric matrix (n x p) or an `mmap_matrix` in `"readwrite"` mode.

Returns: The modified matrix `X` (the same handle for an `mmap_matrix`).

`MLZScoreScaler$fit_transform_inplace()`: Fit and transform in one call, like R's scale().

Usage:

```
MLZScoreScaler$fit_transform_inplace(X, threshold = 1e-12)
```

Arguments:

`X` A numeric matrix (n x p) or an `mmap_matrix` in `"readwrite"` mode.

`threshold` Double, lower bound for the scale statistic (default: 1e-12).

Returns: The modified matrix `X` (the same handle for an `mmap_matrix`).

`MLZScoreScaler$is_fitted()`: Check if scaler is fitted.

Usage:

```
MLZScoreScaler$is_fitted()
```

Returns: Value.

`MLZScoreScaler$get_dropped_indices()`: Get dropped (degenerate) column indices. These columns get scale factor 1 and are still centered (sklearn semantics).

Usage:

```
MLZScoreScaler$get_dropped_indices()
```

Returns: Value.

`MLZScoreScaler$get_center()`: Check if scaler centers data.

Usage:

```
MLZScoreScaler$get_center()
```

Returns: Value.

`MLZScoreScaler$get_scale()`: Check if scaler scales data to unit variance.

Usage:

```
MLZScoreScaler$get_scale()
```

Returns: Value.

`MLZScoreScaler$get_centers()`: Get the learned centers (column means).

Usage:

```
MLZScoreScaler$get_centers()
```

Returns: Value.

`MLZScoreScaler$get_scales()`: Get the learned scales (standard deviations).

Usage:

```
MLZScoreScaler$get_scales()
```

Returns: Value.

`MLZScoreScaler$save()`: Save model state to file.

Usage:

```
MLZScoreScaler$save(filename)
```

Arguments:

`filename` String path to save output.

Returns: Value.

`MLZScoreScaler$load()`: Load model state from file.

Usage:

```
MLZScoreScaler$load(filename)
```

Arguments:

`filename` String path to load input from.

Returns: Value.

`MLZScoreScaler$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MLZScoreScaler$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other ML methods: `MLElasticNet`, `MLElasticNetCV`, `MLLpNormScaler`, `MLPCA`, `MLRidge`, `MLRidgeCV`, `MLSVD`, `ml_agglomerative_cluster()`, `ml_cut_tree()`, `ml_distance_metric`, `ml_linkage_method`

Examples

```
set.seed(1)
X <- matrix(rnorm(50 * 5), 50, 5)
scaler <- MLZScoreScaler$new()
scaler$fit(X)
X_std <- scaler$transform_inplace(X)
scaler$get_centers()
```

---

print.mmap_matrix *Print mmap_matrix*

Description

Print mmap_matrix

Usage

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

Arguments

x	mmap_matrix object
...	arguments passed to other methods

Value

invisibly returns x

See Also

Other memory-mapped matrices: [.mmap_matrix(), as.matrix.mmap_matrix(), dim.mmap_matrix(), mem_matrix(), mem_matrix_from()

Examples

```
mat <- matrix(as.double(1:20), nrow = 4, ncol = 5)
f <- tempfile(fileext = ".bin")
on.exit(unlink(f), add = TRUE)
mem_matrix_from(mat, f)
mm <- mem_matrix(f, rows = 4, cols = 5)
print(mm)
```

TACGP_Solver *T-ACGP Solver*

Description

Terminating Approximate Conjugate Gradient Pursuit (T-ACGP) solver. Inherits all common methods from TSolver_Base.

Super class

TSolver_Base -> TACGP_Solver

Methods

Public methods:

- TACGP_Solver$new()
- TACGP_Solver$clone()

TACGP_Solver$new(): Create a new TACGP_Solver.

Usage:

```
TACGP_Solver$new(X, D, y, options = tsolver_options())
```

Arguments:

`X` Numeric matrix of predictors in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`D` Numeric matrix of dummies in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`y` Numeric response vector.

`options` Solver options from `tsolver_options` (default: `tsolver_options()`).

`TACGP_Solver$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TACGP_Solver$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

T. Blumensath and M. E. Davies, "Gradient Pursuits," IEEE Trans. Signal Process., vol. 56, no. 6, pp. 2370-2382, 2008.

F. Scheidt and M. Muma, "Terminating Greedy Pursuits for High-Dimensional False Discovery Rate Control," in Proc. Eur. Signal Process. Conf. (EUSIPCO), 2026, pp. 2511-2515.

See Also

Other T-solvers: `TAFS_Solver`, `TCDENET_Solver`, `TCDIENET_Solver`, `TCDLASSO_Solver`, `TCDTIKNET_Solver`, `TENETAug_Solver`, `TENET_Solver`, `TGP_Solver`, `TIENETAug_Solver`, `TIENET_Solver`, `TIKNET_Solver`, `TLARS_Solver`, `TLASSO_Solver`, `TMP_Solver`, `TNCGMP_Solver`, `TOMP_Solver`, `TOOLS_Solver`, `TSTAGEWISE_Solver`, `TSTEPWISE_Solver`, `tsolver_options()`

Examples

```
set.seed(1)
n <- 20; p <- 5
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TACGP_Solver$new(X, D, y)
solver$execute_step(T_stop = 3)
solver$get_beta()
```

`TAFS_Solver` *T-AFS Solver*

Description

Terminating Adaptive Forward Stepwise (T-AFS) solver. The shrinkage parameter `rho` interpolates between Forward Stepwise (`rho = 1`) and LAR/LASSO (`rho -> 0`). Inherits all common methods from `TSolver_Base`.

Super class

TSolver_Base -> TAFS_Solver

Methods

Public methods:

- TAFS_Solver$new()
- TAFS_Solver$clone()

TAFS_Solver$new(): Create a new TAFS_Solver.

Usage:

```
TAFS_Solver$new(X, D, y, options = tsolver_options())
```

Arguments:

X Numeric matrix of predictors in memory, or mmap_matrix (the handle of a memory-mapped file).

D Numeric matrix of dummies in memory, or mmap_matrix (the handle of a memory-mapped file).

y Numeric response vector.

options Solver options from tsolver_options; this solver reads rho_afs, the shrinkage parameter in (0, 1] (default 0.3; 1 is forward stepwise).

TAFS_Solver$clone(): The objects of this class are cloneable with this method.

Usage:

```
TAFS_Solver$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

I. Zhang and R. Tibshirani, "Adaptive Forward Stepwise: A Method for High Sparsity Regression," J. Mach. Learn. Res., vol. 27, no. 35, pp. 1-24, 2026.

F. Scheidt and M. Muma, "Terminating Greedy Pursuits for High-Dimensional False Discovery Rate Control," in Proc. Eur. Signal Process. Conf. (EUSIPCO), 2026, pp. 2511-2515.

See Also

Other T-solvers: TACGP_Solver, TCDENET_Solver, TCDIENET_Solver, TCDLASSO_Solver, TCDTIKNET_Solver, TENETAug_Solver, TENET_Solver, TGP_Solver, TIENETAug_Solver, TIENET_Solver, TIKNET_Solver, TLARS_Solver, TLASSO_Solver, TMP_Solver, TNCGMP_Solver, TOMP_Solver, TOOLS_Solver, TSTAGEWISE_Solver, TSTEPWISE_Solver, tsolver_options()

Examples

```
set.seed(1)
n <- 20; p <- 5
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TAFS_Solver$new(X, D, y, options = tsolver_options(rho_afs = 0.5))
solver$execute_step(T_stop = 3)
solver$get_beta()
```

TCDENET_Solver *T-CDENET Solver (deprecated)*

Description

Deprecated. The coordinate-descent family will be removed in TRexSelector 2.1.0, and constructing this class raises a deprecation warning. Use TENET_Solver, the homotopy solver of the same problem.

Terminating coordinate-descent Elastic Net (diagonal ridge lambda2, the dummies get kappa_dummy * lambda2). The solver used internally by TRexGVSSelector for en_solver = "TCDENET". Inherits the CD getters from TCDSolver_Base and all common methods from TSolver_Base.

Super classes

TSolver_Base -> TCDSolver_Base -> TCDENET_Solver

Methods

Public methods:

- TCDENET_Solver$new()
- TCDENET_Solver$clone()

TCDENET_Solver$new(): Create a new TCDENET_Solver.

Usage:

```
TCDENET_Solver$new(X, D, y, options = tsolver_options())
```

Arguments:

X Numeric matrix of predictors in memory, or mmap_matrix (the handle of a memory-mapped file).

D Numeric matrix of dummies in memory, or mmap_matrix (the handle of a memory-mapped file).

y Numeric response vector.

options Solver options from tsolver_options; this solver reads lambda2, kappa_dummy and the cd_* knobs.

TCDENET_Solver$clone(): The objects of this class are cloneable with this method.

Usage:

```
TCDENET_Solver$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

H. Zou and T. Hastie, "Regularization and Variable Selection via the Elastic Net," J. R. Stat. Soc. Ser. B Stat. Methodol., vol. 67, no. 2, pp. 301-320, 2005.

J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani, "Pathwise Coordinate Optimization," Ann. Appl. Stat., vol. 1, no. 2, pp. 302-332, 2007, doi: 10.1214/07-AOAS131.

J. Friedman, T. Hastie, and R. Tibshirani, "Regularization Paths for Generalized Linear Models via Coordinate Descent," J. Stat. Softw., vol. 33, no. 1, pp. 1-22, 2010, doi: 10.18637/jss.v033.i01.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-solvers: TACGP_Solver, TAFS_Solver, TCDIENET_Solver, TCDLASSO_Solver, TCDTIKNET_Solver, TENETAug_Solver, TENET_Solver, TGP_Solver, TIENETAug_Solver, TIENET_Solver, TIKNET_Solver, TLARS_Solver, TLASSO_Solver, TMP_Solver, TNCGMP_Solver, TOMP_Solver, TOOLS_Solver, TSTAGEWISE_Solver, TSTEPWISE_Solver, tsolver_options()

Examples

```
set.seed(1)
n <- 20; p <- 5
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TCDENET_Solver$new(X, D, y, options = tsolver_options(lambda2 = 0.1))
solver$execute_step(T_stop = 3)
solver$get_beta()
```

TCDIENET_Solver *T-CDIENET Solver (deprecated)*

Description

Deprecated. The coordinate-descent family will be removed in TRexSelector 2.1.0, and constructing this class raises a deprecation warning. Use TIENET_Solver, the homotopy solver of the same problem.

Terminating coordinate-descent Informed Elastic Net: exact fixed-lambda1 IEN minimizers per dummy crossing with O(1) group-penalty hooks (group_penalty = "MEAN", the canonical group-sum penalty, or "CENTERING"). The solver used internally by TRexGVSSelector for solver =

"TCDIENET". Not finally developed: its sweeps grow with lambda2 and on saturating designs, and a solve that does not converge within cd_max_sweeps raises an error. The homotopy solver TIENET_Solver has no such limit. Inherits the CD getters from TCDSolver_Base and all common methods from TSolver_Base.

Super classes

TSolver_Base -> TCDSolver_Base -> TCDIENET_Solver

Methods

Public methods:

- TCDIENET_Solver$new()
- TCDIENET_Solver$get_groups()
- TCDIENET_Solver$get_num_groups()
- TCDIENET_Solver$clone()

TCDIENET_Solver$new(): Create a new TCDIENET_Solver.

Usage:

```
TCDIENET_Solver$new(X, D, y, groups, options = tsolver_options())
```

Arguments:

X Numeric matrix of predictors in memory, or mmap_matrix (the handle of a memory-mapped file).

D Numeric matrix of dummies in memory, or mmap_matrix, the handle of a memory-mapped file (a multiple of ncol(X) columns: one group-aligned layer per p).

y Numeric response vector.

groups Integer vector of 1-based group ids, one per predictor column of X (contiguous ids starting at 1).

options Solver options from tsolver_options; this solver reads lambda2, group_penalty and the cd_* knobs.

TCDIENET_Solver$get_groups(): Get the group ids (1-based, one per predictor).

Usage:

```
TCDIENET_Solver$get_groups()
```

Returns: Integer vector of group ids.

TCDIENET_Solver$get_num_groups(): Get the number of disjoint groups.

Usage:

```
TCDIENET_Solver$get_num_groups()
```

Returns: Integer count of groups.

TCDIENET_Solver$clone(): The objects of this class are cloneable with this method.

Usage:

```
TCDIENET_Solver$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

J. Machkour, M. Muma, and D. P. Palomar, "The Informed Elastic Net for Fast Grouped Variable Selection and FDR Control in Genomics Research," in Proc. IEEE 9th Int. Workshop Comput. Adv. Multi-Sensor Adapt. Process. (CAMSAP), 2023, pp. 466-470.

H. Zou and T. Hastie, "Regularization and Variable Selection via the Elastic Net," J. R. Stat. Soc. Ser. B Stat. Methodol., vol. 67, no. 2, pp. 301-320, 2005.

J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani, "Pathwise Coordinate Optimization," Ann. Appl. Stat., vol. 1, no. 2, pp. 302-332, 2007, doi: 10.1214/07-AOAS131.

J. Friedman, T. Hastie, and R. Tibshirani, "Regularization Paths for Generalized Linear Models via Coordinate Descent," J. Stat. Softw., vol. 33, no. 1, pp. 1-22, 2010, doi: 10.18637/jss.v033.i01.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-solvers: `TACGP_Solver`, `TAFS_Solver`, `TCDENET_Solver`, `TCDLASSO_Solver`, `TCDTIKNET_Solver`, `TENETAug_Solver`, `TENET_Solver`, `TGP_Solver`, `TIENETAug_Solver`, `TIENET_Solver`, `TIKNET_Solver`, `TLARS_Solver`, `TLASSO_Solver`, `TMP_Solver`, `TNCGMP_Solver`, `TOMP_Solver`, `TOOLS_Solver`, `TSTAGEWISE_Solver`, `TSTEPWISE_Solver`, `tsolver_options()`

Examples

```
set.seed(1)
n <- 20; p <- 4
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TCDIENET_Solver$new(X, D, y, groups = c(1, 1, 2, 2),
                              options = tsolver_options(lambda2 = 0.1))
solver$execute_step(T_stop = 3)
solver$get_beta()
```

TCDLASSO_Solver *T-CDLASSO Solver (deprecated)*

Description

Deprecated. The coordinate-descent family will be removed in TRexSelector 2.1.0, and constructing this class raises a deprecation warning. Use `TLASSO_Solver`, the homotopy solver of the same problem.

Terminating coordinate-descent LASSO: exact penalized-lasso minimizers at every dummy crossing, localized with certificates (duality gap and coordinate statuses; `cd_certified_localization = FALSE` in the options selects the LEGACY probe bisection). The base of the CD family (`TCDENET_Solver`, `TCDIENET_Solver`, `TCDTIKNET_Solver` inherit its getters). Inherits all common methods from `TSolver_Base`.

Super classes

TSolver_Base -> TCDSolver_Base -> TCDLASSO_Solver

Methods

Public methods:

- TCDLASSO_Solver$new()
- TCDLASSO_Solver$clone()

`TCDLASSO_Solver$new()`: Create a new `TCDLASSO_Solver`.

Usage:

```
TCDLASSO_Solver$new(X, D, y, options = tsolver_options())
```

Arguments:

`X` Numeric matrix of predictors in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`D` Numeric matrix of dummies in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`y` Numeric response vector.

`options` Solver options from `tsolver_options`; this solver reads the `cd_*` knobs.

`TCDLASSO_Solver$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TCDLASSO_Solver$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

R. Tibshirani, "Regression Shrinkage and Selection via the Lasso," J. R. Stat. Soc. Ser. B Stat. Methodol., vol. 58, no. 1, pp. 267-288, 1996.

J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani, "Pathwise Coordinate Optimization," Ann. Appl. Stat., vol. 1, no. 2, pp. 302-332, 2007, doi: 10.1214/07-AOAS131.

J. Friedman, T. Hastie, and R. Tibshirani, "Regularization Paths for Generalized Linear Models via Coordinate Descent," J. Stat. Softw., vol. 33, no. 1, pp. 1-22, 2010, doi: 10.18637/jss.v033.i01.

W. J. Fu, "Penalized Regression: The Bridge Versus the Lasso," J. Comput. Graph. Statist., vol. 7, pp. 397-416, 1998.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-solvers: TACGP_Solver, TAFS_Solver, TCDENET_Solver, TCDIENET_Solver, TCDTIKNET_Solver, TENETAug_Solver, TENET_Solver, TGP_Solver, TIENETAug_Solver, TIENET_Solver, TIKNET_Solver, TLARS_Solver, TLASSO_Solver, TMP_Solver, TNCGMP_Solver, TOMP_Solver, TOOLS_Solver, TSTAGEWISE_Solver, TSTEPWISE_Solver, tsolver_options()

Examples

```
set.seed(1)
n <- 20; p <- 5
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TCDLASSO_Solver$new(X, D, y)
solver$execute_step(T_stop = 3)
solver$get_beta()
solver$is_certified_localization()
```

TCDTIKNET_Solver *T-CDTIKNET Solver (deprecated)*

Description

Deprecated. The coordinate-descent family will be removed in TRexSelector 2.1.0, and constructing this class raises a deprecation warning. Use TIKNET_Solver, the homotopy solver of the same problem.

Terminating coordinate-descent Tikhonov Net: exact fixed-lambda1 minimizers per dummy crossing under the general quadratic penalty lambda2 * t(beta) K beta with a sparse PSD K. Dummy coupling per fold_dummies in the options (FALSE: independent kappa_dummy ridge; TRUE: FOLDED layered convention). TRexTikhonovSelector no longer runs it. Not finally developed: under the FOLDED coupling its sweeps grow with lambda2, and on saturating designs as well, and a solve that does not converge within cd_max_sweeps raises an error. The homotopy solver TIKNET_Solver has no such limit. Inherits the CD getters from TCDSolver_Base and all common methods from TSolver_Base.

Super classes

TSolver_Base -> TCDSolver_Base -> TCDTIKNET_Solver

Methods

Public methods:

- TCDTIKNET_Solver$new()
- TCDTIKNET_Solver$is_folded_dummy_coupling()
- TCDTIKNET_Solver$clone()

TCDTIKNET_Solver$new(): Create a new TCDTIKNET_Solver.

Usage:

```
TCDTIKNET_Solver$new(X, D, y, tikhonov_K, options = tsolver_options())
```

Arguments:

X Numeric matrix of predictors in memory, or mmap_matrix (the handle of a memory-mapped file).

`D` Numeric matrix of dummies in memory, or `mmap_matrix`, the handle of a memory-mapped file (a multiple of `ncol(X)` columns when `fold_dummies = TRUE`).

`y` Numeric response vector.

`tikhonov_K` p x p PSD Tikhonov matrix: a `Matrix::dgCMatrix` or a dense matrix.

`options` Solver options from `tsolver_options`; this solver reads `lambda2`, `kappa_dummy`, `fold_dummies` and the `cd_*` knobs.

`TCDTIKNET_Solver$is_folded_dummy_coupling()`: Is the FOLDED dummy coupling active?

Usage:

```
TCDTIKNET_Solver$is_folded_dummy_coupling()
```

Returns: Logical.

`TCDTIKNET_Solver$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TCDTIKNET_Solver$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

A. N. Tikhonov, "Solution of Incorrectly Formulated Problems and the Regularization Method," Soviet Math. Dokl., vol. 4, pp. 1035-1038, 1963 (English translation of Dokl. Akad. Nauk SSSR 151 (1963), no. 3, 501-504).

Y. Park, L. Reichel, G. Rodriguez, and X. Yu, "Parameter Determination for Tikhonov Regularization Problems in General Form," J. Comput. Appl. Math., vol. 343, pp. 12-25, 2018, doi: 10.1016/j.cam.2018.04.049.

J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani, "Pathwise Coordinate Optimization," Ann. Appl. Stat., vol. 1, no. 2, pp. 302-332, 2007, doi: 10.1214/07-AOAS131.

J. Friedman, T. Hastie, and R. Tibshirani, "Regularization Paths for Generalized Linear Models via Coordinate Descent," J. Stat. Softw., vol. 33, no. 1, pp. 1-22, 2010, doi: 10.18637/jss.v033.i01.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-solvers: `TACGP_Solver`, `TAFS_Solver`, `TCDENET_Solver`, `TCDIENET_Solver`, `TCDLASSO_Solver`, `TENETAug_Solver`, `TENET_Solver`, `TGP_Solver`, `TIENETAug_Solver`, `TIENET_Solver`, `TIKNET_Solver`, `TLARS_Solver`, `TLASSO_Solver`, `TMP_Solver`, `TNCGMP_Solver`, `TOMP_Solver`, `TOOLS_Solver`, `TSTAGEWISE_Solver`, `TSTEPWISE_Solver`, `tsolver_options()`

Examples

```
set.seed(1)
n <- 20; p <- 4
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TCDTIKNET_Solver$new(X, D, y, tikhonov_K = diag(p),
                               options = tsolver_options(lambda2 = 0.1))
solver$execute_step(T_stop = 3)
solver$get_beta()
```

TENET_Solver *T-ENET Solver*

Description

Terminating Elastic Net solver (L1 + L2 regularization). Inherits all common methods from TSolver_Base.

Super class

TSolver_Base -> TENET_Solver

Methods

Public methods:

- TENET_Solver$new()
- TENET_Solver$get_num_removals()
- TENET_Solver$get_cycling_ratio()
- TENET_Solver$get_lambda()
- TENET_Solver$get_alpha()
- TENET_Solver$clone()

TENET_Solver$new(): Create a new TENET_Solver.

Usage:

```
TENET_Solver$new(X, D, y, options = tsolver_options())
```

Arguments:

X Numeric matrix of predictors in memory, or mmap_matrix (the handle of a memory-mapped file).

D Numeric matrix of dummies in memory, or mmap_matrix (the handle of a memory-mapped file).

y Numeric response vector.

options Solver options from tsolver_options; this solver reads lambda2, the L2 (ridge) penalty (>= 0; the default 0 is the plain lasso path).

`TENET_Solver$get_num_removals()`: Get the number of coefficient removals.

Usage:

```
TENET_Solver$get_num_removals()
```

Returns: Integer count of removals.

`TENET_Solver$get_cycling_ratio()`: Get the cycling ratio (removals / total steps).

Usage:

```
TENET_Solver$get_cycling_ratio()
```

Returns: Scalar numeric cycling ratio.

`TENET_Solver$get_lambda()`: Get the regularization (lambda) path.

Usage:

```
TENET_Solver$get_lambda()
```

Returns: Numeric vector of lambda values, one per step.

`TENET_Solver$get_alpha()`: The elastic-net mixing alpha implied by lambda2.

Usage:

```
TENET_Solver$get_alpha()
```

Returns: Numeric scalar.

`TENET_Solver$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TENET_Solver$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

H. Zou and T. Hastie, "Regularization and Variable Selection via the Elastic Net," J. R. Stat. Soc. Ser. B Stat. Methodol., vol. 67, no. 2, pp. 301-320, 2005.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least Angle Regression," Ann. Statist., vol. 32, no. 2, pp. 407-499, 2004.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-solvers: `TACGP_Solver`, `TAFS_Solver`, `TCDENET_Solver`, `TCDIENET_Solver`, `TCDLASSO_Solver`, `TCDTIKNET_Solver`, `TENETAug_Solver`, `TGP_Solver`, `TIENETAug_Solver`, `TIENET_Solver`, `TIKNET_Solver`, `TLARS_Solver`, `TLASSO_Solver`, `TMP_Solver`, `TNCGMP_Solver`, `TOMP_Solver`, `TOOLS_Solver`, `TSTAGEWISE_Solver`, `TSTEPWISE_Solver`, `tsolver_options()`

Examples

```
set.seed(1)
n <- 20; p <- 5
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TENET_Solver$new(X, D, y, options = tsolver_options(lambda2 = 0.1))
solver$execute_step(T_stop = 3)
solver$get_beta()
```

TENETAug_Solver *T-ENET-Aug Solver*

Description

Terminating Elastic Net solver via row augmentation (TENETAug). The ridge penalty is encoded in an augmented data system solved by an inner T-LASSO (or pure T-LARS) path, as the T-Rex+GVS selector of the 1.x versions does (`trex(method = "trex+GVS")`). Inherits all common methods from `TSolver_Base`.

Super class

`TSolver_Base` -> `TENETAug_Solver`

Methods

Public methods:

- `TENETAug_Solver$new()`
- `TENETAug_Solver$clone()`

`TENETAug_Solver$new()`**:** Create a new `TENETAug_Solver`.

Usage:

```
TENETAug_Solver$new(X, D, y, options = tsolver_options())
```

Arguments:

`X` Numeric matrix of predictors in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`D` Numeric matrix of dummies in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`y` Numeric response vector.

`options` Solver options from `tsolver_options`; this solver reads `lambda2` (the ridge penalty, >= 0) and `tenet_aug_use_lars` (run the inner path with a pure-LARS solver that never drops variables).

`TENETAug_Solver$clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TENETAug_Solver$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

H. Zou and T. Hastie, "Regularization and Variable Selection via the Elastic Net," J. R. Stat. Soc. Ser. B Stat. Methodol., vol. 67, no. 2, pp. 301-320, 2005.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least Angle Regression," Ann. Statist., vol. 32, no. 2, pp. 407-499, 2004.

J. Machkour, M. Muma, and D. P. Palomar, "False Discovery Rate Control for Grouped Variable Selection in High-Dimensional Linear Models Using the T-Knock Filter," in Proc. 30th Eur. Signal Process. Conf. (EUSIPCO), 2022, pp. 892-896.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-solvers: TACGP_Solver, TAFS_Solver, TCDENET_Solver, TCDIENET_Solver, TCDLASSO_Solver, TCDTIKNET_Solver, TENET_Solver, TGP_Solver, TIENETAug_Solver, TIENET_Solver, TIKNET_Solver, TLARS_Solver, TLASSO_Solver, TMP_Solver, TNCGMP_Solver, TOMP_Solver, TOOLS_Solver, TSTAGEWISE_Solver, TSTEPWISE_Solver, tsolver_options()

Examples

```
set.seed(1)
n <- 20; p <- 5
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TENETAug_Solver$new(X, D, y, options = tsolver_options(lambda2 = 0.1))
solver$execute_step(T_stop = 3)
solver$get_beta()
```

TGP_Solver *T-GP Solver*

Description

Terminating Gradient Pursuit (T-GP) solver. Inherits all common methods from TSolver_Base.

Super class

TSolver_Base -> TGP_Solver

Methods

Public methods:

- TGP_Solver$new()
- TGP_Solver$clone()

`TGP_Solver$new()`: Create a new `TGP_Solver`.

Usage:

```
TGP_Solver$new(X, D, y, options = tsolver_options())
```

Arguments:

`X` Numeric matrix of predictors in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`D` Numeric matrix of dummies in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`y` Numeric response vector.

`options` Solver options from `tsolver_options` (default: `tsolver_options()`).

`TGP_Solver$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TGP_Solver$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

T. Blumensath and M. E. Davies, "Gradient Pursuits," IEEE Trans. Signal Process., vol. 56, no. 6, pp. 2370-2382, 2008.

F. Scheidt and M. Muma, "Terminating Greedy Pursuits for High-Dimensional False Discovery Rate Control," in Proc. Eur. Signal Process. Conf. (EUSIPCO), 2026, pp. 2511-2515.

See Also

Other T-solvers: TACGP_Solver, TAFS_Solver, TCDENET_Solver, TCDIENET_Solver, TCDLASSO_Solver, TCDTIKNET_Solver, TENETAug_Solver, TENET_Solver, TIENETAug_Solver, TIENET_Solver, TIKNET_Solver, TLARS_Solver, TLASSO_Solver, TMP_Solver, TNCGMP_Solver, TOMP_Solver, TOOLS_Solver, TSTAGEWISE_Solver, TSTEPWISE_Solver, tsolver_options()

Examples

```
set.seed(1)
n <- 20; p <- 5
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TGP_Solver$new(X, D, y)
solver$execute_step(T_stop = 3)
solver$get_beta()
```

TIENET_Solver *T-IENET Solver (native pathwise)*

Description

Terminating Informed Elastic Net solver, native LARS homotopy on the group-ridge penalty (no augmented copies). The solver used internally by `TRexGVSSelector` for `gvs_type = "IEN"` with `solver = "TIENET"`. Inherits all common methods from `TSolver_Base`.

Super class

`TSolver_Base` -> `TIENET_Solver`

Methods

Public methods:

- `TIENET_Solver$new()`
- `TIENET_Solver$get_lambda()`
- `TIENET_Solver$get_lambda2()`
- `TIENET_Solver$get_groups()`
- `TIENET_Solver$get_num_groups()`
- `TIENET_Solver$clone()`

`TIENET_Solver$new()`: Create a new `TIENET_Solver`.

Usage:

```
TIENET_Solver$new(X, D, y, groups, options = tsolver_options())
```

Arguments:

`X` Numeric matrix of predictors in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`D` Numeric matrix of dummies in memory, or `mmap_matrix`, the handle of a memory-mapped file (a multiple of `ncol(X)` columns when `lambda2 > 0`: one group-aligned layer per p).

`y` Numeric response vector.

`groups` Integer vector of 1-based group ids, one per predictor column of `X` (contiguous ids starting at 1).

`options` Solver options from `tsolver_options`; this solver reads `lambda2` (the group-ridge penalty, >= 0).

`TIENET_Solver$get_lambda()`: Get the regularization (lambda) path.

Usage:

```
TIENET_Solver$get_lambda()
```

Returns: Numeric vector of lambda values, one per step.

`TIENET_Solver$get_lambda2()`: Get the group-ridge penalty lambda2 used.

Usage:

```
TIENET_Solver$get_lambda2()
```

Returns: Scalar numeric lambda2.

`TIENET_Solver$get_groups()`: Get the group ids (1-based, one per predictor).

Usage:

```
TIENET_Solver$get_groups()
```

Returns: Integer vector of group ids.

`TIENET_Solver$get_num_groups()`: Get the number of disjoint groups.

Usage:

```
TIENET_Solver$get_num_groups()
```

Returns: Integer count of groups.

`TIENET_Solver$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TIENET_Solver$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

J. Machkour, M. Muma, and D. P. Palomar, "The Informed Elastic Net for Fast Grouped Variable Selection and FDR Control in Genomics Research," in Proc. IEEE 9th Int. Workshop Comput. Adv. Multi-Sensor Adapt. Process. (CAMSAP), 2023, pp. 466-470.

H. Zou and T. Hastie, "Regularization and Variable Selection via the Elastic Net," J. R. Stat. Soc. Ser. B Stat. Methodol., vol. 67, no. 2, pp. 301-320, 2005.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least Angle Regression," Ann. Statist., vol. 32, no. 2, pp. 407-499, 2004.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-solvers: `TACGP_Solver`, `TAFS_Solver`, `TCDENET_Solver`, `TCDIENET_Solver`, `TCDLASSO_Solver`, `TCDTIKNET_Solver`, `TENETAug_Solver`, `TENET_Solver`, `TGP_Solver`, `TIENETAug_Solver`, `TIKNET_Solver`, `TLARS_Solver`, `TLASSO_Solver`, `TMP_Solver`, `TNCGMP_Solver`, `TOMP_Solver`, `TOOLS_Solver`, `TSTAGEWISE_Solver`, `TSTEPWISE_Solver`, `tsolver_options()`

Examples

```
set.seed(1)
n <- 20; p <- 4
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
groups <- c(1, 1, 2, 2)  # 1-based group id per predictor
solver <- TIENET_Solver$new(X, D, y, groups = groups,
                            options = tsolver_options(lambda2 = 0.1))
solver$execute_step(T_stop = 3)
solver$get_beta()
```

TIENETAug_Solver *T-IENET-Aug Solver*

Description

Terminating Informed Elastic Net solver via row augmentation (TIENETAug). The group-ridge penalty (one augmented row per group, scaled by `sqrt(lambda2)`) is encoded in an augmented data system solved by an inner T-LASSO (or pure T-LARS) path. This is the solver used internally by `TRexGVSSelector` for `gvs_type = "IEN"`. Inherits all common methods from `TSolver_Base`.

Super class

`TSolver_Base` -> `TIENETAug_Solver`

Methods

Public methods:

- `TIENETAug_Solver$new()`
- `TIENETAug_Solver$get_lambda2()`
- `TIENETAug_Solver$get_groups()`
- `TIENETAug_Solver$get_num_groups()`
- `TIENETAug_Solver$clone()`

`TIENETAug_Solver$new()`**:** Create a new `TIENETAug_Solver`.

Usage:

```
TIENETAug_Solver$new(X, D, y, groups, options = tsolver_options())
```

Arguments:

`X` Numeric matrix of predictors in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`D` Numeric matrix of dummies in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`y` Numeric response vector.

groups Integer vector of 1-based group ids, one per predictor column of X (contiguous ids starting at 1).

options Solver options from tsolver_options; this solver reads lambda2 (the group-ridge penalty, >= 0) and tenet_aug_use_lars (run the inner path with a pure-LARS solver that never drops variables).

`TIENETAug_Solver$get_lambda2()`: Get the group-ridge penalty lambda2 used.

Usage:

```
TIENETAug_Solver$get_lambda2()
```

Returns: Scalar numeric lambda2.

`TIENETAug_Solver$get_groups()`: Get the group ids (1-based, one per predictor).

Usage:

```
TIENETAug_Solver$get_groups()
```

Returns: Integer vector of group ids.

`TIENETAug_Solver$get_num_groups()`: Get the number of disjoint groups.

Usage:

```
TIENETAug_Solver$get_num_groups()
```

Returns: Integer group count.

`TIENETAug_Solver$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TIENETAug_Solver$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

J. Machkour, M. Muma, and D. P. Palomar, "The Informed Elastic Net for Fast Grouped Variable Selection and FDR Control in Genomics Research," in Proc. IEEE 9th Int. Workshop Comput. Adv. Multi-Sensor Adapt. Process. (CAMSAP), 2023, pp. 466-470.

H. Zou and T. Hastie, "Regularization and Variable Selection via the Elastic Net," J. R. Stat. Soc. Ser. B Stat. Methodol., vol. 67, no. 2, pp. 301-320, 2005.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least Angle Regression," Ann. Statist., vol. 32, no. 2, pp. 407-499, 2004.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-solvers: TACGP_Solver, TAFS_Solver, TCDENET_Solver, TCDIENET_Solver, TCDLASSO_Solver, TCDTIKNET_Solver, TENETAug_Solver, TENET_Solver, TGP_Solver, TIENET_Solver, TIKNET_Solver, TLARS_Solver, TLASSO_Solver, TMP_Solver, TNCGMP_Solver, TOMP_Solver, TOOLS_Solver, TSTAGEWISE_Solver, TSTEPWISE_Solver, tsolver_options()

Examples

```
set.seed(1)
n <- 20; p <- 4
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
groups <- c(1, 1, 2, 2)  # 1-based group id per predictor
solver <- TIENETAug_Solver$new(X, D, y, groups = groups,
                               options = tsolver_options(lambda2 = 0.1))
solver$execute_step(T_stop = 3)
solver$get_beta()
```

TIKNET_Solver *T-TIKNET Solver*

Description

Terminating Tikhonov Net solver: native LARS homotopy with a general quadratic penalty `lambda2 * t(beta) K beta` on the folded sums (a sparse PSD `K = t(Gamma) Gamma`, see `trex_gamma_to_k`). FOLDED dummy coupling only. The solver used internally by `TRexTikhonovSelector` for `solver = "TIKNET"`. Inherits all common methods from `TSolver_Base`.

Super class

`TSolver_Base` -> `TIKNET_Solver`

Methods

Public methods:

- `TIKNET_Solver$new()`
- `TIKNET_Solver$get_lambda()`
- `TIKNET_Solver$get_lambda2()`
- `TIKNET_Solver$get_tikhonov_k()`
- `TIKNET_Solver$clone()`

`TIKNET_Solver$new()`**:** Create a new `TIKNET_Solver`.

Usage:

```
TIKNET_Solver$new(X, D, y, tikhonov_K, options = tsolver_options())
```

Arguments:

`X` Numeric matrix of predictors in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`D` Numeric matrix of dummies in memory, or `mmap_matrix`, the handle of a memory-mapped file (a multiple of `ncol(X)` columns when `lambda2 > 0`: FOLDED layered convention).

`y` Numeric response vector.

`tikhonov_K` p x p PSD Tikhonov matrix: a `Matrix::dgCMatrix` or a dense matrix.

`options` Solver options from `tsolver_options`; this solver reads `lambda2` (the quadratic penalty weight, >= 0).

`TIKNET_Solver$get_lambda()`: Get the regularization (lambda) path.

Usage:

```
TIKNET_Solver$get_lambda()
```

Returns: Numeric vector of lambda values, one per step.

`TIKNET_Solver$get_lambda2()`: Get the quadratic penalty weight lambda2 used.

Usage:

```
TIKNET_Solver$get_lambda2()
```

Returns: Scalar numeric lambda2.

`TIKNET_Solver$get_tikhonov_k()`: The Tikhonov matrix K used (sparse `dgCMatrix`).

Usage:

```
TIKNET_Solver$get_tikhonov_k()
```

Returns: A `Matrix::dgCMatrix` (p x p).

`TIKNET_Solver$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TIKNET_Solver$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

A. N. Tikhonov, "Solution of Incorrectly Formulated Problems and the Regularization Method," Soviet Math. Dokl., vol. 4, pp. 1035-1038, 1963 (English translation of Dokl. Akad. Nauk SSSR 151 (1963), no. 3, 501-504).

Y. Park, L. Reichel, G. Rodriguez, and X. Yu, "Parameter Determination for Tikhonov Regularization Problems in General Form," J. Comput. Appl. Math., vol. 343, pp. 12-25, 2018, doi: 10.1016/j.cam.2018.04.049.

A. E. Hoerl and R. W. Kennard, "Ridge Regression: Biased Estimation for Nonorthogonal Problems," Technometrics, vol. 42, no. 1, pp. 80-86, 2000 (Reprint of the 1970 article in the Special 40th Anniversary Issue).

Y. Miche, M. van Heeswijk, P. Bas, O. Simula, and A. Lendasse, "TROP-ELM: A Double-Regularized ELM Using LARS and Tikhonov Regularization," Neurocomputing, vol. 74, pp. 2413-2421, 2011, doi: 10.1016/j.neucom.2010.12.042.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least Angle Regression," Ann. Statist., vol. 32, no. 2, pp. 407-499, 2004.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-solvers: TACGP_Solver, TAFS_Solver, TCDENET_Solver, TCDIENET_Solver, TCDLASSO_Solver, TCDTIKNET_Solver, TENETAug_Solver, TENET_Solver, TGP_Solver, TIENETAug_Solver, TIENET_Solver, TLARS_Solver, TLASSO_Solver, TMP_Solver, TNCGMP_Solver, TOMP_Solver, TOOLS_Solver, TSTAGEWISE_Solver, TSTEPWISE_Solver, tsolver_options()

Examples

```
set.seed(1)
n <- 20; p <- 4
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
K <- diag(p)
solver <- TIKNET_Solver$new(X, D, y, tikhonov_K = K,
                            options = tsolver_options(lambda2 = 0.1))
solver$execute_step(T_stop = 3)
solver$get_beta()
```

TLARS_Solver *T-LARS Solver*

Description

Terminating Least Angle Regression (T-LARS) solver. Inherits all common methods from TSolver_Base.

Super class

TSolver_Base -> TLARS_Solver

Methods

Public methods:

- TLARS_Solver$new()
- TLARS_Solver$get_lambda()
- TLARS_Solver$get_kahan_refresh_interval()
- TLARS_Solver$set_kahan_refresh_interval()
- TLARS_Solver$clone()

TLARS_Solver$new(): Create a new TLARS_Solver.

Usage:

```
TLARS_Solver$new(X, D, y, options = tsolver_options())
```

Arguments:

X Numeric matrix of predictors in memory, or mmap_matrix (the handle of a memory-mapped file).

`D` Numeric matrix of dummies in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`y` Numeric response vector.

`options` Solver options from `tsolver_options` (default: `tsolver_options()`).

`TLARS_Solver$get_lambda()`: Get the regularization (lambda) path.

Usage:

```
TLARS_Solver$get_lambda()
```

Returns: Numeric vector of lambda values, one per step.

`TLARS_Solver$get_kahan_refresh_interval()`: Kahan refresh interval of the LARS-family solver (steps between exact refreshes of the compensated sums).

Usage:

```
TLARS_Solver$get_kahan_refresh_interval()
```

Returns: Integer.

`TLARS_Solver$set_kahan_refresh_interval()`: Set the Kahan refresh interval.

Usage:

```
TLARS_Solver$set_kahan_refresh_interval(interval)
```

Arguments:

`interval` Positive integer.

Returns: Invisibly returns `self`.

`TLARS_Solver$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TLARS_Solver$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least Angle Regression," Ann. Statist., vol. 32, no. 2, pp. 407-499, 2004.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-solvers: `TACGP_Solver`, `TAFS_Solver`, `TCDENET_Solver`, `TCDIENET_Solver`, `TCDLASSO_Solver`, `TCDTIKNET_Solver`, `TENETAug_Solver`, `TENET_Solver`, `TGP_Solver`, `TIENETAug_Solver`, `TIENET_Solver`, `TIKNET_Solver`, `TLASSO_Solver`, `TMP_Solver`, `TNCGMP_Solver`, `TOMP_Solver`, `TOOLS_Solver`, `TSTAGEWISE_Solver`, `TSTEPWISE_Solver`, `tsolver_options()`

Examples

```
set.seed(1)
n <- 20; p <- 5
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TLARS_Solver$new(X, D, y)
solver$execute_step(T_stop = 3)
solver$get_beta()
solver$get_actives()
```

TLASSO_Solver *T-LASSO Solver*

Description

Terminating LASSO solver (T-LARS with variable removal). Inherits all common methods from TSolver_Base.

Super class

TSolver_Base -> TLASSO_Solver

Methods

Public methods:

- TLASSO_Solver$new()
- TLASSO_Solver$get_num_removals()
- TLASSO_Solver$get_cycling_ratio()
- TLASSO_Solver$get_lambda()
- TLASSO_Solver$clone()

TLASSO_Solver$new(): Create a new TLASSO_Solver.

Usage:

```
TLASSO_Solver$new(X, D, y, options = tsolver_options())
```

Arguments:

X Numeric matrix of predictors in memory, or mmap_matrix (the handle of a memory-mapped file).

D Numeric matrix of dummies in memory, or mmap_matrix (the handle of a memory-mapped file).

y Numeric response vector.

options Solver options from tsolver_options (default: tsolver_options()).

TLASSO_Solver$get_num_removals(): Get the number of coefficient removals.

Usage:

```
TLASSO_Solver$get_num_removals()
```

Returns: Integer count of removals.

`TLASSO_Solver$get_cycling_ratio()`: Get the cycling ratio (removals / total steps).

Usage:

```
TLASSO_Solver$get_cycling_ratio()
```

Returns: Scalar numeric cycling ratio.

`TLASSO_Solver$get_lambda()`: Get the regularization (lambda) path.

Usage:

```
TLASSO_Solver$get_lambda()
```

Returns: Numeric vector of lambda values, one per step.

`TLASSO_Solver$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TLASSO_Solver$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

R. Tibshirani, "Regression Shrinkage and Selection via the Lasso," J. R. Stat. Soc. Ser. B Stat. Methodol., vol. 58, no. 1, pp. 267-288, 1996.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least Angle Regression," Ann. Statist., vol. 32, no. 2, pp. 407-499, 2004.

M. R. Osborne, B. Presnell, and B. A. Turlach, "On the LASSO and Its Dual," J. Comput. Graph. Statist., vol. 9, no. 2, pp. 319-337, 2000.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-solvers: `TACGP_Solver`, `TAFS_Solver`, `TCDENET_Solver`, `TCDIENET_Solver`, `TCDLASSO_Solver`, `TCDTIKNET_Solver`, `TENETAug_Solver`, `TENET_Solver`, `TGP_Solver`, `TIENETAug_Solver`, `TIENET_Solver`, `TIKNET_Solver`, `TLARS_Solver`, `TMP_Solver`, `TNCGMP_Solver`, `TOMP_Solver`, `TOOLS_Solver`, `TSTAGEWISE_Solver`, `TSTEPWISE_Solver`, `tsolver_options()`

Examples

```
set.seed(1)
n <- 20; p <- 5
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
```

```
solver <- TLASSO_Solver$new(X, D, y)
solver$execute_step(T_stop = 3)
solver$get_beta()
```

`TMP_Solver` *T-MP Solver*

Description

Terminating Matching Pursuit (T-MP) solver. Inherits all common methods from `TSolver_Base`.

Super class

`TSolver_Base` -> `TMP_Solver`

Methods

Public methods:

- `TMP_Solver$new()`
- `TMP_Solver$clone()`

`TMP_Solver$new()`: Create a new `TMP_Solver`.

Usage:

```
TMP_Solver$new(X, D, y, options = tsolver_options())
```

Arguments:

`X` Numeric matrix of predictors in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`D` Numeric matrix of dummies in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`y` Numeric response vector.

`options` Solver options from `tsolver_options` (default: `tsolver_options()`).

`TMP_Solver$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TMP_Solver$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

S. G. Mallat and Z. Zhang, "Matching Pursuits with Time-Frequency Dictionaries," IEEE Trans. Signal Process., vol. 41, no. 12, pp. 3397-3415, 1993, doi: 10.1109/78.258082.

F. Scheidt and M. Muma, "Terminating Greedy Pursuits for High-Dimensional False Discovery Rate Control," in Proc. Eur. Signal Process. Conf. (EUSIPCO), 2026, pp. 2511-2515.

See Also

Other T-solvers: TACGP_Solver, TAFS_Solver, TCDENET_Solver, TCDIENET_Solver, TCDLASSO_Solver, TCDTIKNET_Solver, TENETAug_Solver, TENET_Solver, TGP_Solver, TIENETAug_Solver, TIENET_Solver, TIKNET_Solver, TLARS_Solver, TLASSO_Solver, TNCGMP_Solver, TOMP_Solver, TOOLS_Solver, TSTAGEWISE_Solver, TSTEPWISE_Solver, tsolver_options()

Examples

```
set.seed(1)
n <- 20; p <- 5
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TMP_Solver$new(X, D, y)
solver$execute_step(T_stop = 3)
solver$get_beta()
```

TNCGMP_Solver *T-NCGMP Solver*

Description

Terminating Norm-Corrected Gradient Matching Pursuit (T-NCGMP) solver. Two variants are available: "line_search" (standard, 1-D projection) and "fully_corrective" (OMP-style least squares over the active set). Inherits all common methods from TSolver_Base.

Super class

TSolver_Base -> TNCGMP_Solver

Methods

Public methods:

- TNCGMP_Solver$new()
- TNCGMP_Solver$clone()

TNCGMP_Solver$new(): Create a new TNCGMP_Solver.

Usage:

```
TNCGMP_Solver$new(X, D, y, options = tsolver_options())
```

Arguments:

X Numeric matrix of predictors in memory, or mmap_matrix (the handle of a memory-mapped file).

D Numeric matrix of dummies in memory, or mmap_matrix (the handle of a memory-mapped file).

y Numeric response vector.

options Solver options from tsolver_options; this solver reads ncgmp_variant: "line_search" (default) or "fully_corrective".

TNCGMP_Solver$clone(): The objects of this class are cloneable with this method.

Usage:

```
TNCGMP_Solver$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

F. Locatello, R. Khanna, M. Tschannen, and M. Jaggi, "A Unified Optimization View on Generalized Matching Pursuit and Frank-Wolfe," in Proc. 20th Int. Conf. Artif. Intell. Statist. (AISTATS), Proc. Mach. Learn. Res. vol. 54, 2017, pp. 860-868.

F. Scheidt and M. Muma, "Terminating Greedy Pursuits for High-Dimensional False Discovery Rate Control," in Proc. Eur. Signal Process. Conf. (EUSIPCO), 2026, pp. 2511-2515.

See Also

Other T-solvers: TACGP_Solver, TAFS_Solver, TCDENET_Solver, TCDIENET_Solver, TCDLASSO_Solver, TCDTIKNET_Solver, TENETAug_Solver, TENET_Solver, TGP_Solver, TIENETAug_Solver, TIENET_Solver, TIKNET_Solver, TLARS_Solver, TLASSO_Solver, TMP_Solver, TOMP_Solver, TOOLS_Solver, TSTAGEWISE_Solver, TSTEPWISE_Solver, tsolver_options()

Examples

```
set.seed(1)
n <- 20; p <- 5
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TNCGMP_Solver$new(X, D, y,
                            options = tsolver_options(ncgmp_variant = "line_search"))
solver$execute_step(T_stop = 3)
solver$get_beta()
```

TOMP_Solver *T-OMP Solver*

Description

Terminating Orthogonal Matching Pursuit (T-OMP) solver. Inherits all common methods from TSolver_Base.

Super class

TSolver_Base -> TOMP_Solver

Methods

Public methods:

- TOMP_Solver$new()
- TOMP_Solver$clone()

`TOMP_Solver$new()`: Create a new `TOMP_Solver`.

Usage:

```
TOMP_Solver$new(X, D, y, options = tsolver_options())
```

Arguments:

`X` Numeric matrix of predictors in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`D` Numeric matrix of dummies in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`y` Numeric response vector.

`options` Solver options from `tsolver_options` (default: `tsolver_options()`).

`TOMP_Solver$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TOMP_Solver$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad, "Orthogonal Matching Pursuit: Recursive Function Approximation with Applications to Wavelet Decomposition," in Proc. 27th Asilomar Conf. Signals, Syst., Comput., 1993, pp. 40-44.

J. A. Tropp, "Greed is Good: Algorithmic Results for Sparse Approximation," IEEE Trans. Inf. Theory, vol. 50, no. 10, pp. 2231-2242, 2004, doi: 10.1109/TIT.2004.834793.

T. Blumensath and M. E. Davies, "On the Difference Between Orthogonal Matching Pursuit and Orthogonal Least Squares," unpublished note, draft of March 29, 2007.

F. Scheidt and M. Muma, "Terminating Greedy Pursuits for High-Dimensional False Discovery Rate Control," in Proc. Eur. Signal Process. Conf. (EUSIPCO), 2026, pp. 2511-2515.

See Also

Other T-solvers: `TACGP_Solver`, `TAFS_Solver`, `TCDENET_Solver`, `TCDIENET_Solver`, `TCDLASSO_Solver`, `TCDTIKNET_Solver`, `TENETAug_Solver`, `TENET_Solver`, `TGP_Solver`, `TIENETAug_Solver`, `TIENET_Solver`, `TIKNET_Solver`, `TLARS_Solver`, `TLASSO_Solver`, `TMP_Solver`, `TNCGMP_Solver`, `TOOLS_Solver`, `TSTAGEWISE_Solver`, `TSTEPWISE_Solver`, `tsolver_options()`

Examples

```
set.seed(1)
n <- 20; p <- 5
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TOMP_Solver$new(X, D, y)
solver$execute_step(T_stop = 3)
solver$get_beta()
```

TOOLS_Solver *T-OLS Solver*

Description

Terminating Orthogonal Least Squares (T-OLS) solver. Inherits all common methods from TSolver_Base.

Super class

TSolver_Base -> TOOLS_Solver

Methods

Public methods:

- TOOLS_Solver$new()
- TOOLS_Solver$clone()

TOOLS_Solver$new(): Create a new TOOLS_Solver.

Usage:

```
TOOLS_Solver$new(X, D, y, options = tsolver_options())
```

Arguments:

X Numeric matrix of predictors in memory, or mmap_matrix (the handle of a memory-mapped file).

D Numeric matrix of dummies in memory, or mmap_matrix (the handle of a memory-mapped file).

y Numeric response vector.

options Solver options from tsolver_options (default: tsolver_options()).

TOOLS_Solver$clone(): The objects of this class are cloneable with this method.

Usage:

```
TOOLS_Solver$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

S. Chen, S. A. Billings, and W. Luo, "Orthogonal Least Squares Methods and Their Application to Non-Linear System Identification," Int. J. Control, vol. 50, no. 5, pp. 1873-1896, 1989, doi: 10.1080/00207178908953472.

T. Blumensath and M. E. Davies, "On the Difference Between Orthogonal Matching Pursuit and Orthogonal Least Squares," unpublished note, draft of March 29, 2007.

F. Scheidt and M. Muma, "Terminating Greedy Pursuits for High-Dimensional False Discovery Rate Control," in Proc. Eur. Signal Process. Conf. (EUSIPCO), 2026, pp. 2511-2515.

See Also

Other T-solvers: TACGP_Solver, TAFS_Solver, TCDENET_Solver, TCDIENET_Solver, TCDLASSO_Solver, TCDTIKNET_Solver, TENETAug_Solver, TENET_Solver, TGP_Solver, TIENETAug_Solver, TIENET_Solver, TIKNET_Solver, TLARS_Solver, TLASSO_Solver, TMP_Solver, TNCGMP_Solver, TOMP_Solver, TSTAGEWISE_Solver, TSTEPWISE_Solver, tsolver_options()

Examples

```
set.seed(1)
n <- 20; p <- 5
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TOOLS_Solver$new(X, D, y)
solver$execute_step(T_stop = 3)
solver$get_beta()
```

---

trex_biobank_control *T-Rex Biobank Screening Control Parameters*

Description

Build a control list for TRexBiobankScreeningSelector: the accepted range of the FDR estimate of the screening and the target FDR level of the fallback T-Rex selector.

Usage

```
trex_biobank_control(
  lower_bound_FDR = 0.05,
  upper_bound_FDR = 0.15,
  target_FDR_trex = 0.1
)
```

Arguments

lower_bound_FDR
: Lower bound of the accepted range of the FDR estimate of the screening (default: 0.05). An estimate below it suggests too little power.

upper_bound_FDR
: Upper bound of the accepted range of the FDR estimate of the screening (default: 0.15). An estimate above it suggests too many false discoveries.

target_FDR_trex
: Target FDR level of the fallback T-Rex selector (default: 0.1).

Value

A named list for use in TRexBiobankScreeningSelector.

See Also

Other T-Rex selectors and controls: Gauss_data, TRexBiobankScreeningSelector, TRexDASelector, TRexGVSSelector, TRexSPCASelector, TRexScreeningSelector, TRexSelector, TRexTikhonovSelector, trex_build_group_centering_k(), trex_control(), trex_da_control(), trex_default_lambda2_grid(), trex_dummy_distribution(), trex_fdp(), trex_fdp_dense(), trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_control(), trex_gvs_lambda2_power_rule(), trex_normalize_design(), trex_precision(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
trex_biobank_control()
trex_biobank_control(lower_bound_FDR = 0.02, upper_bound_FDR = 0.10)
trex_biobank_control(target_FDR_trex = 0.2)
```

---

trex_build_group_centering_k

Build the centering penalty matrix of a group assignment

Description

Computes the block-diagonal matrix `K = blockdiag(I - J / p_m)` over the groups, with `p_m` the size of group m and J the matrix of ones: the penalty under which the members of a group share a coefficient. It is the matrix that `penalty_structure = "GROUP_CENTERING"` of trex_tikhonov_control stands for, so a selector with that structure needs the groups alone. A group with one member contributes nothing.

Usage

```
trex_build_group_centering_k(prior_groups)
```

Arguments

prior_groups	Integer vector with the group of each variable, numbered 1 to M without gaps (length p).

Value

A sparse `Matrix::dgCMatrix` (p x p). Requires the Matrix package (Suggests) for the sparse return class.

See Also

Other T-Rex selectors and controls: Gauss_data, TRexBiobankScreeningSelector, TRexDASelector, TRexGVSSelector, TRexSPCASelector, TRexScreeningSelector, TRexSelector, TRexTikhonovSelector, trex_biobank_control(), trex_control(), trex_da_control(), trex_default_lambda2_grid(), trex_dummy_distribution(), trex_fdp(), trex_fdp_dense(), trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_control(), trex_gvs_lambda2_power_rule(), trex_normalize_design(), trex_precision(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
if (requireNamespace("Matrix", quietly = TRUE)) {
  K <- trex_build_group_centering_k(rep(1:2, each = 3))
}
```

---

trex_control *T-Rex Control Parameters*

Description

Build a control list for the T-Rex selectors.

Usage

```
trex_control(
  solver = "TLARS",
  K = 20,
  tloop_stagnation_stop = NULL,
  tloop_max_stagnant_steps = 5,
  parallel_rnd_experiments = FALSE,
  use_memory_mapping = FALSE,
  max_outer_threads = 1,
  max_inner_threads = 1,
  max_dummy_multiplier = 10,
  use_max_T_stop = TRUE,
  opt_threshold = 0.75,
```

```
  lloop_strategy = "STANDARD",
  tol = 1e-06,
  lambda2 = 0.1,
  rho_afs = 0.3,
  ncgmp_variant = 0,
  exch_tie_alpha = -1,
  exch_tie_floor = 0.5,
  tenet_aug_use_lars = FALSE,
  cd_lambda_rel_tol = 1e-06,
  cd_tol_certify = -1,
  cd_tol_probe = -1,
  cd_gram_cap = 0,
  cd_max_sweeps = 0,
  cd_certified_localization = TRUE,
  dummy_distribution = trex_dummy_distribution(),
  scaling_mode = c("L2", "ZSCORE", "NONE")
)
```

Arguments

`solver` Solver algorithm: "TLARS", "TLASSO", "TENET", "TENET_AUG", "TSTAGEWISE", "TSTEPWISE", "TOMP", "TGP", "TACGP", "TMP", "TNCGMP", "TOOLS", "TAFS", "TCDLASSO", "TCDENET" (default: "TLARS"). "TENET_AUG" is the row-augmented formulation of the TENET elastic net (proved equivalent for `lambda2 >= 0`) and is accepted wherever TENET runs; it solves with the standard LASSO inner path unless the optional `tenet_aug_use_lars` is set. The coordinate-descent (CD) family (TCDLASSO/TCDENET) records exact penalized-lasso minimizers per dummy crossing. The IEN-family names "TIENET", "TIENET_AUG", and "TCDIENET" are accepted by the parser but only dispatchable inside `TRexGVSSelector` with `gvs_type = "IEN"`; "TIKNET" needs a penalty matrix and is dispatchable only inside `TRexTikhonovSelector`. The plain selector rejects both families with an informative error. The coordinate-descent names "TCDLASSO", "TCDENET" and "TCDIENET" are deprecated and will be removed in TRexSelector 2.1.0: a selector that receives one of them raises a deprecation warning. "TCDTIKNET" is deprecated as well, and no selector accepts it any more. Use "TLASSO", "TENET", "TIENET" and "TIKNET", the homotopy solvers of the same problems.

`K` Number of random experiments (default: 20, at least 2).

`tloop_stagnation_stop` Whether to stop the T-loop on stagnation. NULL (default) leaves the choice to the C++ core, the same rule on every surface: FALSE for the path solvers (TLARS, TLASSO, TENET, TENET_AUG, TIENET, TIENET_AUG, TIKNET and the CD family TCDLASSO, TCDENET, TCDIENET, TCDTIKNET), TRUE for the greedy ones (the OMP family, TSTEPWISE, TSTAGEWISE, TAFS, TNCGMP, TOOLS).

`tloop_max_stagnant_steps` Number of T steps in a row without a growing selection that ends the T-loop under the stagnation stop (default: 5, at least 1). The selection is counted at the

highest voting level whose FDP estimate does not exceed `tFDR`.

`parallel_rnd_experiments`
: Run the K random experiments in parallel (default: `FALSE`, the experiments run in sequence). Inside a parallel run every solver runs serially; the lean L-loop strategies (`"SEEDED"`, `"PERMUTATION"`, `"PERMUTATION_SEEDED"`) always run their experiments in sequence. Was `use_openmp`.

`use_memory_mapping`
: Keep the dummy matrices in memory-mapped files instead of in memory (default: FALSE).

`max_outer_threads`
: Threads of the parallel K-loop (default: 1; a whole number of at least 1), capped at `K`. Ignored while `parallel_rnd_experiments` is `FALSE`.

`max_inner_threads`
: Thread budget of the solvers' own passes over the design (default: 1, serial; a whole number of at least 1). Live when the experiments run in sequence; inside a parallel run the solvers stay serial. A pass splits only above about a million entries of the design, so small problems run serially whatever the budget.

`max_dummy_multiplier`
: Cap of the dummy multiplier: the L-loop adds dummies up to this many times the number of variables (default: 10).

`use_max_T_stop`
: Whether the T-loop ends at `ceiling(n / 2)` at the latest (default: TRUE). Without the bound it can run up to the number of dummies.

`opt_threshold`
: Reference voting level of the L-loop (default: 0.75, in `[0, 1]`). The L-loop raises L until the FDP estimate at the largest voting level that does not exceed `opt_threshold` is at or below `tFDR`.

`lloop_strategy`
: L-loop strategy (default: "STANDARD", as in C++ and Python). "STANDARD" draws fresh dummies at every L step, as the 1.x versions do. "HCONCAT" keeps the dummies of the earlier steps and appends p columns per step, which is faster. The two gave the same FDR in simulations and can select differently. "PERMUTATION" stores one base dummy matrix and permutes its rows per experiment. "SKIPL" skips the L-loop and runs with `max_dummy_multiplier` times p dummies at once. The seeded variants re-derive the dummies from the seed at every step and store nothing: "SEEDED" is the storage-free twin of "STANDARD" (bit-identical to "STANDARD" for the same seed) and "PERMUTATION_SEEDED" the storage-free twin of "PERMUTATION".

`tol`
: Numerical tolerance of the selector (default: 1e-6): the threshold of its own comparisons and the tolerance of the solvers in its random experiments. A stand-alone solver defaults to machine epsilon (`tsolver_options`). The two tolerances serve different purposes and differ on purpose.

`lambda2`
: Elastic-net L2 penalty (TENET hyperparameter, default: 0.1).

`rho_afs`
: Shrinkage step of TAFS in (0, 1] (default: 0.3); 1 is forward stepwise.

`ncgmp_variant`
: NCGMP variant: 0 = line_search, 1 = fully_corrective (default: 0).

`exch_tie_alpha`
: Exchangeable-tie band width of the greedy solvers (TOMP, TOOLS, TGP, TACGP, TMP, TNCGMP and TAFS), in units of the pairwise ranking-noise sd. When > 0, statistically indistinguishable top candidates within highly correlated clusters

are picked uniformly at random per step, restoring the within-experiment occurrence spread that the dependency-aware (DA) deflation's FDR control relies on. `-1` (default): the selector decides. `TRexDASelector` runs with 0.25, every other selector without the band. `0` turns the band off, also under T-Rex+DA. The path solvers ignore the option.

`exch_tie_floor` Minimum absolute correlation for exchangeable-tie candidates, in (0, 1) (default: 0.5). Ignored unless `exch_tie_alpha > 0`.

`tenet_aug_use_lars`
: Logical, for `solver = "TENET_AUG"` in the plain/DA/screening selectors. The default `FALSE` runs the STANDARD LASSO inner path (variables can be dropped); `TRUE` opts into the pure-LARS inner path that never drops variables. Inside `TRexGVSSelector` use the `gvs_control` knob of the same name instead. Ignored by every other solver.

`cd_lambda_rel_tol`
: Relative lambda step tolerance of the CCD family's crossing search (default: 1e-6, as in C++ and Python). Values <= 0 keep the solver's internal default. Ignored by the non-CCD solvers. Deprecated: it will be removed with the coordinate-descent family in TRexSelector 2.1.0.

`cd_tol_certify` KKT certification tolerance of the CCD family. Must be set together with `cd_tol_probe`; the default -1 keeps the solver's own tolerances. Ignored by the non-CCD solvers. Deprecated: it will be removed with the coordinate-descent family in TRexSelector 2.1.0.

`cd_tol_probe` Probing tolerance of the CCD family's coordinate sweeps. Must be set together with `cd_tol_certify`; the default -1 keeps the solver's own tolerances. Ignored by the non-CCD solvers. Deprecated: it will be removed with the coordinate-descent family in TRexSelector 2.1.0.

`cd_gram_cap` Column cap for the CCD family's cached Gram matrix; above it the solver falls back to the naive updates (default: 0 = keep the solver's internal default). Ignored by the non-CCD solvers. Deprecated: it will be removed with the coordinate-descent family in TRexSelector 2.1.0.

`cd_max_sweeps` Maximum coordinate sweeps per CCD subproblem (default: 0 = keep the solver's internal default of 2000). A solve that stops at this budget far from a solution (KKT violation above 0.001 lambda_1) raises an error instead of continuing on an unconverged path. Ignored by the non-CCD solvers. Deprecated: it will be removed with the coordinate-descent family in TRexSelector 2.1.0.

`cd_certified_localization`
: Logical (default `TRUE`): the CD family localizes every dummy crossing with certificates — a duality gap and coordinate statuses at every bracket endpoint, closed-form continuation between them, the record placed at the knot. `FALSE` selects the LEGACY probe bisection with the support-nesting stop (the pre-2026-08 behaviour, kept as the A/B baseline). Ignored by the non-CD solvers. Deprecated: it will be removed with the coordinate-descent family in TRexSelector 2.1.0.

`dummy_distribution`
: Dummy variable distribution, created with `trex_dummy_distribution` (default: standard Normal).

`scaling_mode` Working-design convention, one of `"L2"` (default), `"ZSCORE"` or `"NONE"`. `"L2"` centers `X` and `y` inside the selector and divides each column of `X` by its L2 norm; `"ZSCORE"` centers and divides by the sample SD (glmnet `standardize = TRUE` convention); the dummies follow the same convention. `"NONE"` takes `X` and `y` exactly as supplied (no centering, no scaling): pre-center and pre-scale as the model demands (`trex_normalize_design` reproduces the selector's own preprocessing), or supply an intercept-free dictionary (steering vectors, Vandermonde, Fourier atoms) whose columns must not be centered. The dummies keep the unit-L2 standard, so a `"NONE"` design should be unit-L2-commensurate (a warning names columns whose L2 norm leaves [0.25, 4]). Non-finite entries and dead columns (constant under `"L2"` / `"ZSCORE"`, all-zero under `"NONE"`) are rejected in every mode with the column named. `"L2"` and `"ZSCORE"` standardize the design in place and restore it, so a memory-mapped design must be open in `"readwrite"` mode; under `"NONE"` nothing is written and `"readonly"` mode suffices.

Value

A named list for the constructors of the T-Rex selectors.

Note

Seeds. The seed is an argument of the selector constructors, for example `TRexSelector$new()`. A seed of 0 or more makes a run deterministic and reproducible: the same seed draws the same dummies. A negative seed (the default -1) draws them from the hardware entropy source, and the run is not reproducible bit by bit. A Monte Carlo study gives every trial a seed of its own, or uses -1.

See Also

Other T-Rex selectors and controls: `Gauss_data`, `TRexBiobankScreeningSelector`, `TRexDASelector`, `TRexGVSSelector`, `TRexSPCASelector`, `TRexScreeningSelector`, `TRexSelector`, `TRexTikhonovSelector`, `trex_biobank_control()`, `trex_build_group_centering_k()`, `trex_da_control()`, `trex_default_lambda2_grid()`, `trex_dummy_distribution()`, `trex_fdp()`, `trex_fdp_dense()`, `trex_gamma_to_k()`, `trex_get_num_threads()`, `trex_gvs_control()`, `trex_gvs_lambda2_power_rule()`, `trex_normalize_design()`, `trex_precision()`, `trex_recall()`, `trex_screen_control()`, `trex_spca_control()`, `trex_tikhonov_control()`, `trex_tikhonov_lambda2_power_rule()`, `trex_tpp()`, `trex_tpp_dense()`

Examples

```
trex_control()
trex_control(solver = "TLASSO", K = 10)
trex_control(scaling_mode = "NONE")  # user-managed centering and scaling
```

`trex_da_control` *T-Rex+DA Control Parameters*

Description

Build a control list for the dependency-aware correction of TRexDASelector.

Usage

```
trex_da_control(
  da_method = "BT",
  cor_coef = -2,
  rho_thr_DA = 0.02,
  hc_linkage = "single",
  hc_grid_length = 0,
  prior_groups = NULL,
  bt_selection_mode = "FeasibleOnly"
)
```

Arguments

`da_method`	Dependency model (default: "BT"). "BT", the binary tree, is made for disjoint groups of highly correlated variables. "NN", nearest neighbours, is made for overlapping groups of highly correlated variables and sweeps the correlation thresholds of a rho grid of length `hc_grid_length`. "AR1" is made for a first-order autoregressive dependency across the columns. "EQUI" is made for variables that all share one correlation.
`cor_coef`	Correlation coefficient for AR1/EQUI. -2.0 triggers auto-estimation (default: -2.0). The estimate is the mean lag-1 autocorrelation of the rows for AR1 and the mean pairwise correlation of the variables for EQUI. Both methods use the magnitude of a supplied coefficient. Any other value outside `[-1, 1]` gives a warning and is replaced by the estimate.
`rho_thr_DA`	Correlation threshold of the AR1 and EQUI methods (default: 0.02). AR1: the window around a variable reaches `ceiling(log(rho_thr_DA) / log(abs(cor_coef)))` columns to each side, up to the first lag whose correlation `abs(cor_coef)^lag` has fallen to the threshold, range `[0, 1)`. EQUI: the correction is skipped when `abs(cor_coef)` does not exceed the threshold, range `[0, 1]`.
`hc_linkage`	Hierarchical clustering linkage: "single", "complete", "average", "wpgma" (default: "single"). Used by BT and by prior groups. "ward" is rejected, because its update is valid for squared Euclidean distances only and the correlation distance `1 - abs(cor)` is not Euclidean. The capitalized names of TRexSelector 2.0.1 ("Single") are accepted as well.
`hc_grid_length`	Number of correlation thresholds on the calibration grid of BT, NN and prior groups. `0` (default) takes `min(20, p)`.

prior_groups
: Optional prior grouping constraints: a list of integer vectors, each of length p, giving non-negative group labels per variable. When supplied (non-NULL, non-empty), the prior-groups dependency-aware path is used and da_method is ignored: the finest common refinement of all supplied levels acts as a hard merge constraint, hierarchical clustering (hc_linkage, correlation distance) runs within each constraint group, and the calibration rho grid (length hc_grid_length) is built from the pooled within-group dendrogram heights plus the conservative rho = 1 singleton anchor. The deflation therefore acts on tight data-driven neighbourhoods inside the known groups, never on the raw groups themselves. (default: NULL).

bt_selection_mode
: Which cells of the calibration grid may win the selection of the methods BT, NN and prior groups. "FeasibleOnly" (default) admits only cells whose FDP estimate is at or below the target, which controls the realized FDR. "RFaithful" admits every cell with the largest selection count, as the 1.x versions of the package do, and can exceed the target at high within-group correlation. Ignored by AR1 and EQUI.

Value

A named list for use in TRexDASelector.

See Also

Other T-Rex selectors and controls: Gauss_data, TRexBiobankScreeningSelector, TRexDASelector, TRexGVSSelector, TRexSPCASelector, TRexScreeningSelector, TRexSelector, TRexTikhonovSelector, trex_biobank_control(), trex_build_group_centering_k(), trex_control(), trex_default_lambda2_grid(), trex_dummy_distribution(), trex_fdp(), trex_fdp_dense(), trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_control(), trex_gvs_lambda2_power_rule(), trex_normalize_design(), trex_precision(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
trex_da_control()
trex_da_control(da_method = "BT", hc_linkage = "average")
# Prior-groups path (constraints over p = 4 variables):
trex_da_control(prior_groups = list(c(1, 1, 2, 2)))
```

---

trex_default_lambda2_grid

The default lambda_2 grid of the pilot rule and of the power rules

---

Description

Returns the grid that the C++ core holds once: 0, 1, 3, 10, 30, 100, 300, 1000, 3000. It is the default of pilot_grid in trex_tikhonov_control and of grid in trex_tikhonov_lambda2_power_rule and trex_gvs_lambda2_power_rule. The values are stated for columns of unit norm.

Usage

```
trex_default_lambda2_grid()
```

Value

A numeric vector, ascending.

See Also

Other T-Rex selectors and controls: Gauss_data, TRexBiobankScreeningSelector, TRexDASelector, TRexGVSSelector, TRexSPCASelector, TRexScreeningSelector, TRexSelector, TRexTikhonovSelector, trex_biobank_control(), trex_build_group_centering_k(), trex_control(), trex_da_control(), trex_dummy_distribution(), trex_fdp(), trex_fdp_dense(), trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_control(), trex_gvs_lambda2_power_rule(), trex_normalize_design(), trex_precision(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
trex_default_lambda2_grid()
```

---

trex_dummy_distribution

T-Rex Dummy Variable Distribution

Description

Build a distribution specification for the dummy variables injected by the T-Rex Selector. Pass the result to the `dummy_distribution` argument of `trex_control`.

Usage

```
trex_dummy_distribution(
  type = "Normal",
  a = NULL,
  df = NULL,
  location = NULL,
  scale = NULL,
  b = NULL,
  c = NULL,
  dim = NULL,
  s = NULL,
  p1 = NULL,
  p2 = NULL
)
```

Arguments

type	Character string naming the distribution. One of: "Normal", "Uniform", "Rademacher", "StudentT", "Laplace", "Gumbel", "Triangle", "UniformSphere", "Mammen", "ConstrainedSparseRademacher", "Logistic". Defaults to "Normal".
a	Half-range of "Uniform", which draws from U(-a, a) (default: sqrt(3), unit variance), or lower bound of "Triangle" (default: -sqrt(6)).
df	Degrees of freedom of "StudentT" (default: 5).
location	Location parameter of "Laplace", "Gumbel" and "Logistic" (default: 0).
scale	Scale parameter of "Laplace" (default: 1/sqrt(2)), "Gumbel" (default: 1) and "Logistic" (default: sqrt(3)/pi).
b	Mode of "Triangle" (default: 0).
c	Upper bound of "Triangle" (default: sqrt(6)).
dim	Dimension of the sphere of "UniformSphere" (default: 3).
s	Sparsity parameter of "ConstrainedSparseRademacher" in (0, 1] (default: 0.1). An entry is +1 or -1 with probability s/2 each and 0 otherwise, and s = 1 gives the Rademacher distribution.
p1, p2	Probabilities of the two points of "Mammen", -(sqrt(5) - 1)/2 and (sqrt(5) + 1)/2 (defaults: (sqrt(5) + 1)/(2 sqrt(5)) and (sqrt(5) - 1)/(2 sqrt(5))).

Value

A named list describing the dummy distribution, for use in trex_control.

References

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

E. Mammen, "Bootstrap and Wild Bootstrap for High Dimensional Linear Models," Ann. Statist., vol. 21, no. 1, pp. 255-285, 1993, doi: 10.1214/aos/1176349025.

See Also

Other T-Rex selectors and controls: Gauss_data, TRexBiobankScreeningSelector, TRexDASelector, TRexGVSSelector, TRexSPCASelector, TRexScreeningSelector, TRexSelector, TRexTikhonovSelector, trex_biobank_control(), trex_build_group_centering_k(), trex_control(), trex_da_control(), trex_default_lambda2_grid(), trex_fdp(), trex_fdp_dense(), trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_control(), trex_gvs_lambda2_power_rule(), trex_normalize_design(), trex_precision(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
# Standard Normal (default)
trex_dummy_distribution()
```

```
# Student-t with 3 degrees of freedom
trex_dummy_distribution("StudentT", df = 3)
```

`trex_fdp` *Compute False Discovery Proportion (FDP) using index sets*

Description

Compute False Discovery Proportion (FDP) using index sets

Usage

```
trex_fdp(selected_indices, true_support)
```

Arguments

`selected_indices`	Integer vector of selected indices.
`true_support`	Integer vector of true active indices.

Value

Numeric FDP value.

See Also

Other T-Rex selectors and controls: `Gauss_data`, `TRexBiobankScreeningSelector`, `TRexDASelector`, `TRexGVSSelector`, `TRexSPCASelector`, `TRexScreeningSelector`, `TRexSelector`, `TRexTikhonovSelector`, `trex_biobank_control()`, `trex_build_group_centering_k()`, `trex_control()`, `trex_da_control()`, `trex_default_lambda2_grid()`, `trex_dummy_distribution()`, `trex_fdp_dense()`, `trex_gamma_to_k()`, `trex_get_num_threads()`, `trex_gvs_control()`, `trex_gvs_lambda2_power_rule()`, `trex_normalize_design()`, `trex_precision()`, `trex_recall()`, `trex_screen_control()`, `trex_spca_control()`, `trex_tikhonov_control()`, `trex_tikhonov_lambda2_power_rule()`, `trex_tpp()`, `trex_tpp_dense()`

Examples

```
trex_fdp(c(1L, 3L, 5L), c(1L, 2L, 3L))
```

trex_fdp_dense *Compute False Discovery Proportion (FDP) using dense vectors*

Description

Compute False Discovery Proportion (FDP) using dense vectors

Usage

```
trex_fdp_dense(beta_hat, beta, eps = 1e-15)
```

Arguments

beta_hat	Numeric vector of estimated coefficients.
beta	Numeric vector of true coefficients.
eps	Numeric threshold for counting non-zeros (default: 1e-15).

Value

Numeric FDP value.

See Also

Other T-Rex selectors and controls: Gauss_data, TRexBiobankScreeningSelector, TRexDASelector, TRexGVSSelector, TRexSPCASelector, TRexScreeningSelector, TRexSelector, TRexTikhonovSelector, trex_biobank_control(), trex_build_group_centering_k(), trex_control(), trex_da_control(), trex_default_lambda2_grid(), trex_dummy_distribution(), trex_fdp(), trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_control(), trex_gvs_lambda2_power_rule(), trex_normalize_design(), trex_precision(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
trex_fdp_dense(c(1.5, 0, 0.8, 0, 0), c(2.0, 0, 1.0, 0, 0))
```

trex_gamma_to_k *Build a Tikhonov penalty matrix K from an operator Gamma*

Description

Computes K = t(Gamma) %*% Gamma (pruned and compressed) for use as tikhonov_K in trex_tikhonov_control. For example, a first-difference Gamma yields the smoothness-encoding graph Laplacian of a chain.

Usage

```
trex_gamma_to_k(gamma)
```

Arguments

gamma	Regularization operator (m x p): a base `matrix` or a sparse `Matrix::dgCMatrix`.

Value

A sparse `Matrix::dgCMatrix` (p x p). Requires the Matrix package (Suggests) for the sparse return class.

See Also

Other T-Rex selectors and controls: `Gauss_data`, `TRexBiobankScreeningSelector`, `TRexDASelector`, `TRexGVSSelector`, `TRexSPCASelector`, `TRexScreeningSelector`, `TRexSelector`, `TRexTikhonovSelector`, `trex_biobank_control()`, `trex_build_group_centering_k()`, `trex_control()`, `trex_da_control()`, `trex_default_lambda2_grid()`, `trex_dummy_distribution()`, `trex_fdp()`, `trex_fdp_dense()`, `trex_get_num_threads()`, `trex_gvs_control()`, `trex_gvs_lambda2_power_rule()`, `trex_normalize_design()`, `trex_precision()`, `trex_recall()`, `trex_screen_control()`, `trex_spca_control()`, `trex_tikhonov_control()`, `trex_tikhonov_lambda2_power_rule()`, `trex_tpp()`, `trex_tpp_dense()`

Examples

```
# First-difference operator over 5 variables -> chain Laplacian
if (requireNamespace("Matrix", quietly = TRUE)) {
  p <- 5
  gamma <- diff(diag(p))
  K <- trex_gamma_to_k(gamma)
}
```

---

trex_get_num_threads *The library's thread budget*

Description

The library is single-threaded by default. Every parallel region of the C++ backend runs on an explicit budget: a selector's random experiments and its inner kernels on `trex_control( max_outer_threads, max_inner_threads)`, a T-solver on `tsolver_options(n_threads)`, and every method without an option of its own (the scalers, the clustering, the ML regressions, the dummy and data generators, `trex_normalize_design`) on the library's default budget, which starts at 1. `trex_get_num_threads()` reads that default budget and `trex_set_num_threads()` sets it for this session; the `n_threads` argument of those methods overrides it per object. `trex_get_max_threads()` reports the threads OpenMP can run on this machine (1 in a build without OpenMP), an upper bound for a budget.

Usage

```
trex_get_num_threads()

trex_get_max_threads()

trex_set_num_threads(n)
```

Arguments

n	Positive integer, the default thread budget.

Value

`trex_get_num_threads()`, `trex_get_max_threads()`: an integer. `trex_set_num_threads()`: invisibly, the previous budget.

See Also

Other T-Rex selectors and controls: `Gauss_data`, `TRexBiobankScreeningSelector`, `TRexDASelector`, `TRexGVSSelector`, `TRexSPCASelector`, `TRexScreeningSelector`, `TRexSelector`, `TRexTikhonovSelector`, `trex_biobank_control()`, `trex_build_group_centering_k()`, `trex_control()`, `trex_da_control()`, `trex_default_lambda2_grid()`, `trex_dummy_distribution()`, `trex_fdp()`, `trex_fdp_dense()`, `trex_gamma_to_k()`, `trex_gvs_control()`, `trex_gvs_lambda2_power_rule()`, `trex_normalize_design()`, `trex_precision()`, `trex_recall()`, `trex_screen_control()`, `trex_spca_control()`, `trex_tikhonov_control()`, `trex_tikhonov_lambda2_power_rule()`, `trex_tpp()`, `trex_tpp_dense()`

Examples

```
trex_get_num_threads()   # 1
old <- trex_set_num_threads(min(2L, trex_get_max_threads()))
trex_get_num_threads()
trex_set_num_threads(old)
```

---

trex_gvs_control *T-Rex+GVS Control Parameters*

Description

Build a control list for the grouped variable selection of `TRexGVSSelector`. The solver follows `gvs_type`. For "EN" `en_solver` chooses it: TENET (default), TENET_AUG or the deprecated TCDENET. For "IEN" the `solver` of `trex_control` names it, TIENET (native pathwise), TIENET_AUG or the deprecated TCDIENET, so the IEN track needs a base control with one of them. The selector refuses a base control whose solver does not fit the track. TCDIENET is not finally developed: its sweeps grow with `lambda_2` and on saturating designs, and a solve that does not converge within `cd_max_sweeps` raises an error. TCDENET and TCDIENET are deprecated and will be removed with the coordinate-descent family in TRexSelector 2.1.0: the selector raises a deprecation warning for them. Use TENET and TIENET.

Usage

```
trex_gvs_control(
  gvs_type = "EN",
  corr_max = 0.5,
  hc_linkage = "single",
  lambda_2 = -1,
```

```
  lambda2_method = NULL,
  en_solver = "TENET",
  tenet_aug_use_lars = FALSE,
  groups = NULL,
  cv_n_folds = 10,
  cv_n_lambda = 1000,
  cv_n_lambda1 = 40,
  cv_n_lambda2 = 100,
  cv_dfmax_ratio = 0.8,
  cv_seed = -1,
  group_labels = NULL
)
```

Arguments

gvs_type	Penalty of the grouped selection: "EN" (elastic net) or "IEN" (informed elastic net) (default: "EN").
corr_max	Correlation threshold of the automatic clustering (default: 0.5). The dendrogram is cut at the height `1 - corr_max`. With "single" linkage (default) no two variables of different clusters correlate above `corr_max`. With "complete" linkage every two variables of one cluster correlate at least at `corr_max`, and variables of different clusters can exceed it.
hc_linkage	Hierarchical clustering linkage: "single", "complete", "average", "wpgma" (default: "single"). Used when no `groups` are supplied. "ward" is rejected, because its update is valid for squared Euclidean distances only and the correlation distance `1 - abs(cor)` is not Euclidean. The capitalized names of TRexSelector 2.0.1 ("Single") are accepted as well.
lambda_2	L2 penalty weight for group structure. A negative value (default `-1`) triggers automatic determination via the method selected by `lambda2_method`; `0` is the degenerate pure T-LASSO case (no ridge); a positive value is used as-is. A supplied value refers to the scale of the standardized columns. Under `scaling_mode = "ZSCORE"` the columns have the squared norm `n - 1`, so the same shrinkage needs the value times `n - 1`.
lambda2_method	Auto-determination strategy for `lambda_2` when `lambda_2 < 0`. `NULL` (default) resolves by track: `"CV_1SE_CCD"` for `gvs_type = "EN"` and `"CV_1SE_IEN_CCD"` for `gvs_type = "IEN"` (the IEN-geometry tuner consumes the group structure; the 1SE rule is preferred because the MIN rule tends to pin to the grid floor). The C++ core applies the same rule at construction on every surface, so an explicit `"CV_1SE_CCD"` on the IEN track also runs the IEN tuner. Explicit choices: EN-shaped ridge CV `"CV_1SE_CCD"`, `"CV_MIN_CCD"`, `"CV_1SE_SVD"`, `"CV_MIN_SVD"` (allowed on both tracks); IEN-geometry `"CV_1SE_IEN_CCD"`, `"CV_MIN_IEN_CCD"`, `"CV_1SE_TIK_SVD"`, `"CV_MIN_TIK_SVD"` (require `gvs_type = "IEN"` — they consume the group structure).
en_solver	Elastic Net solver variant for `gvs_type = "EN"`: `"TENET"` (default, Gram-based), `"TENET_AUG"` (row-augmented LASSO), or `"TCDENET"` (coordinate descent; exact fixed-lambda1 elastic-net minimizers per dummy crossing; deprecated, it will be removed in TRexSelector 2.1.0, use `"TENET"`). All solve the same EN

problem and select the same variables up to path-vs-minimizer differences; the switch exists to validate solver equivalence, **not** to choose a LARS vs LASSO path (for that, see `tenet_aug_use_lars`).

`tenet_aug_use_lars`
: Logical. For the augmented solvers, `en_solver = "TENET_AUG"` or `gvs_type = "IEN"` with the solver `"TIENET_AUG"`: run the augmented system with a pure-LARS inner path that never drops variables (`TRUE`) instead of the default LASSO inner path that can (`FALSE`, default). This is the LARS-vs-LASSO distinction. Ignored for `"TENET"` and `"TIENET"`.

`groups`
: Integer vector of manually specified group assignments, one entry per variable, numbered 1 to M without gaps (default: NULL).

`cv_n_folds`
: Number of folds for the k-fold cross-validation used by the CV `lambda2_method` paths (default: 10, matching `cv.glmnet`). Ignored when `lambda_2 >= 0`.

`cv_n_lambda`
: Number of points on the `lambda_2` grid of the rules `"CV_1SE_SVD"`, `"CV_MIN_SVD"`, `"CV_1SE_CCD"`, `"CV_MIN_CCD"`, `"CV_1SE_TIK_SVD"` and `"CV_MIN_TIK_SVD"` (default: 1000). The rules `"CV_1SE_IEN_CCD"` and `"CV_MIN_IEN_CCD"` use `cv_n_lambda2`. Ignored when `lambda_2 >= 0`.

`cv_n_lambda1`
: IEN-geometry coordinate-descent tuners (`"CV_1SE_IEN_CCD"`, `"CV_MIN_IEN_CCD"`) only: points on the profiled lambda_1 path per lambda_2 value (default: 40, the shortest path that still reproduces the 100-point picks; the cost is U-shaped in this number). Ignored by the other rules and when `lambda_2 >= 0`.

`cv_n_lambda2`
: IEN-geometry coordinate-descent tuners only: points on the lambda_2 grid (default: 100). Ignored by the other rules and when `lambda_2 >= 0`.

`cv_dfmax_ratio`
: IEN-geometry coordinate-descent tuners only: support cap as a fraction of the smallest fold's training size; each lambda_1 path stops once more than `floor(cv_dfmax_ratio * n_train)` coefficients are nonzero (default: 0.8). A value `<= 0` disables the cap. Larger values allow larger supports at a higher tuner cost. Must be finite. Ignored by the other rules and when `lambda_2 >= 0`.

`cv_seed`
: Seed for the deterministic fold-permutation RNG used by the CV `lambda2_method` paths. `-1` (default) derives it from the T-Rex `seed` (deterministic when `seed >= 0`, hardware entropy when `seed < 0`); a value `>= 0` overrides `seed` for CV folds only, giving per-trial fold reproducibility.

`group_labels`
: Optional character vector with a name per group of `groups`, in the order of the group numbers. Its length must equal the number of groups. The names are returned in the field `group_labels` of the selector (default: NULL).

Value

A named list for use in TRexGVSSelector.

See Also

Other T-Rex selectors and controls: `Gauss_data`, `TRexBiobankScreeningSelector`, `TRexDASelector`, `TRexGVSSelector`, `TRexSPCASelector`, `TRexScreeningSelector`, `TRexSelector`, `TRexTikhonovSelector`, `trex_biobank_control()`, `trex_build_group_centering_k()`, `trex_control()`, `trex_da_control()`, `trex_default_lambda2_grid()`, `trex_dummy_distribution()`, `trex_fdp()`, `trex_fdp_dense()`,

trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_lambda2_power_rule(), trex_normalize_design(), trex_precision(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
trex_gvs_control()
trex_gvs_control(gvs_type = "IEN", corr_max = 0.3)
```

---

trex_gvs_lambda2_power_rule

The knee of the full lambda_2 profile of the T-Rex+GVS selector

---

Description

Runs one FDR-controlled selection of TRexGVSSelector per value of grid for lambda_2, the ridge part of the elastic net or of the informed elastic net, with the same seed, so the probes share their dummies and differ by the penalty weight alone. The pick is the smallest lambda_2 whose selection is within knee_tolerance selections of the largest (the knee of the profile). The result holds every probe's selection. The pick depends on y and lies outside the fixed-lambda_2 FDR guarantee that each probe carries. trex_tikhonov_lambda2_power_rule is the same rule on the Tikhonov selector.

Usage

```
trex_gvs_lambda2_power_rule(
  X,
  y,
  tFDR = 0.1,
  gvs_control = trex_gvs_control(),
  control = NULL,
  seed = 0,
  grid = trex_default_lambda2_grid(),
  knee_tolerance = 1,
  verbose = FALSE
)
```

Arguments

X	Feature matrix (n x p) in memory, or an mmap_matrix, the handle of a memory-mapped file. Every probe normalizes it in place and hands it back, so X is unchanged on return and never copied. A memory-mapped design needs "readwrite" mode for that, and "readonly" mode suffices under scaling_mode = "NONE".
y	Response vector (length n).
tFDR	Target FDR level (default: 0.1).

gvs_control	A GVS control list from trex_gvs_control() (default: trex_gvs_control()); its lambda_2 is replaced per probe, everything else is shared by every probe.
control	A control list from trex_control(), or NULL (default), taken as by TRexGVSSelector: with NULL the base control of the C++ class applies, and a control that is passed has to name a solver of the track.
seed	Selector seed, the same for every probe (default: 0). A negative seed would draw fresh dummies per probe and is refused.
grid	lambda_2 candidates, stated for columns of unit norm, sorted and de-duplicated. The default is trex_default_lambda2_grid(): 0, 1, 3, 10, 30, 100, 300, 1000, 3000. Under scaling_mode = "ZSCORE" the probes run at these values times n - 1. The value 0 is the T-Rex with the lasso.
knee_tolerance	Selections below the largest count that still count as "on the plateau" (default: 1); 0 reads the largest selection, ties toward the smallest lambda_2.
verbose	Whether the probes print their progress (default: FALSE).

Value

A list with the fields of trex_tikhonov_lambda2_power_rule: lambda2 (the pick, on the working scale: supplied as lambda_2 it reproduces the selection), index (1-based, into grid), selected (the 1-based selection at the pick), grid (the values the probes ran at, ascending and de-duplicated), grid_scale (the factor between the supplied grid and grid, n - 1 under "ZSCORE" and 1 otherwise), counts (selection sizes per grid value), probes (one list per grid value with lambda2, selected, T_stop and seconds), max_selected, n_ties, the boundary flags at_lower_end and at_upper_end (the pick sits on the first or on the last grid value, and a pick on the last one asks for a wider grid) and seconds.

See Also

Other T-Rex selectors and controls: Gauss_data, TRexBiobankScreeningSelector, TRexDASelector, TRexGVSSelector, TRexSPCASelector, TRexScreeningSelector, TRexSelector, TRexTikhonovSelector, trex_biobank_control(), trex_build_group_centering_k(), trex_control(), trex_da_control(), trex_default_lambda2_grid(), trex_dummy_distribution(), trex_fdp(), trex_fdp_dense(), trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_control(), trex_normalize_design(), trex_precision(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
data(Gauss_data)
X <- Gauss_data$X
y <- Gauss_data$y
prof <- trex_gvs_lambda2_power_rule(X, y, tFDR = 0.1, seed = 1,
                                    grid = c(0, 1, 10))
prof$lambda2
prof$counts
```

`trex_normalize_design` *Apply the T-Rex data normalization to a design*

Description

The selectors' own preprocessing as a user-callable helper: the same code path a selector runs at construction for the given `scaling_mode`. Use it to pre-normalize a design and then run a selector with `trex_control(scaling_mode = "NONE")` (the selection is identical to running the raw design under `"L2"`), to inspect the centering and scaling parameters, or to validate a design before a long run. The inputs are copied and left untouched.

Usage

```
trex_normalize_design(
  X,
  y,
  scaling_mode = c("L2", "ZSCORE", "NONE"),
  eps = 1e-06,
  n_threads = NULL
)
```

Arguments

`X`	Numeric design matrix (n x p).
`y`	Numeric response vector (length n).
`scaling_mode`	One of `"L2"` (default; center, divide by the column L2 norm), `"ZSCORE"` (center, divide by the sample SD) or `"NONE"` (copies equal the inputs; identity parameters).
`eps`	Dead-column threshold on the working L2 norm (default 1e-6, the selectors' default solver tolerance).
`n_threads`	Integer, the thread budget of the column passes (`NULL`, the default: the library's default budget, `trex_get_num_threads()`, initially 1).

Details

Errors on a non-finite entry or a dead column (constant under `"L2"` / `"ZSCORE"`, all-zero under `"NONE"`), naming the column (0-based, as reported by the C++ core).

Value

A list with `X` and `y` (the normalized copies), `X_means`, `X_scales` (the per-column divisors: L2 norms for `"L2"`, sample SDs for `"ZSCORE"`, ones for `"NONE"`) and `y_mean`.

See Also

Other T-Rex selectors and controls: Gauss_data, TRexBiobankScreeningSelector, TRexDASelector, TRexGVSSelector, TRexSPCASelector, TRexScreeningSelector, TRexSelector, TRexTikhonovSelector, trex_biobank_control(), trex_build_group_centering_k(), trex_control(), trex_da_control(), trex_default_lambda2_grid(), trex_dummy_distribution(), trex_fdp(), trex_fdp_dense(), trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_control(), trex_gvs_lambda2_power_rule(), trex_precision(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
set.seed(1)
X <- matrix(rnorm(200), 40, 5)
y <- X[, 1] + rnorm(40)
d <- trex_normalize_design(X, y)
colMeans(d$X)            # ~ 0
sqrt(colSums(d$X^2))     # 1
```

---

trex_precision *Compute Precision using index sets*

Description

Compute Precision using index sets

Usage

```
trex_precision(selected_indices, true_support)
```

Arguments

selected_indices	Integer vector of selected indices.
true_support	Integer vector of true active indices.

Value

Numeric precision value.

See Also

Other T-Rex selectors and controls: Gauss_data, TRexBiobankScreeningSelector, TRexDASelector, TRexGVSSelector, TRexSPCASelector, TRexScreeningSelector, TRexSelector, TRexTikhonovSelector, trex_biobank_control(), trex_build_group_centering_k(), trex_control(), trex_da_control(), trex_default_lambda2_grid(), trex_dummy_distribution(), trex_fdp(), trex_fdp_dense(), trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_control(), trex_gvs_lambda2_power_rule(), trex_normalize_design(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
trex_precision(c(1L, 3L, 5L), c(1L, 2L, 3L))
```

`trex_recall` *Compute Recall using index sets*

Description

Compute Recall using index sets

Usage

```
trex_recall(selected_indices, true_support)
```

Arguments

`selected_indices`	Integer vector of selected indices.
`true_support`	Integer vector of true active indices.

Value

Numeric recall value.

See Also

Other T-Rex selectors and controls: `Gauss_data`, `TRexBiobankScreeningSelector`, `TRexDASelector`, `TRexGVSSelector`, `TRexSPCASelector`, `TRexScreeningSelector`, `TRexSelector`, `TRexTikhonovSelector`, `trex_biobank_control()`, `trex_build_group_centering_k()`, `trex_control()`, `trex_da_control()`, `trex_default_lambda2_grid()`, `trex_dummy_distribution()`, `trex_fdp()`, `trex_fdp_dense()`, `trex_gamma_to_k()`, `trex_get_num_threads()`, `trex_gvs_control()`, `trex_gvs_lambda2_power_rule()`, `trex_normalize_design()`, `trex_precision()`, `trex_screen_control()`, `trex_spca_control()`, `trex_tikhonov_control()`, `trex_tikhonov_lambda2_power_rule()`, `trex_tpp()`, `trex_tpp_dense()`

Examples

```
trex_recall(c(1L, 3L, 5L), c(1L, 2L, 3L))
```

`trex_screen_control` *T-Rex+Screening Control Parameters*

Description

Build a control list for the screening step in TRexScreeningSelector and TRexBiobankScreeningSelector.

Usage

```
trex_screen_control(
  use_bootstrap_CI = FALSE,
  R_boot = 1000,
  ci_grid_step = 0.001,
  trex_method = "TREX",
  cor_coef = -2,
  rho_thr_DA = 0.02,
  n_blocks = 5
)
```

Arguments

`use_bootstrap_CI`
: Whether the confidence-based screening (bootstrap) makes the selection (default: FALSE). The FDR estimate stays one over the number of selected variables. With a dependency-aware variant it mostly selects nothing: the filter lowers the ordinary count that the confidence level has to reach, and in simulations no level reached it in 90 % to 100 % of the runs.

`R_boot`
: Number of bootstrap resamples, at least 2 (default: 1000).

`ci_grid_step`
: Step of the grid over the confidence level, in `(0, 1]` (default: 0.001).

`trex_method`
: Variant of the screening (default: "TREX"). "TREX" screens without a dependency-aware filter. The other variants apply a dependency-aware filter to the relative occurrences. "TREX_DA_AR1", for a first-order autoregressive dependency, compares a variable with the neighbors inside the AR(1) window. "TREX_DA_EQUI", for equi-correlated variables, compares a variable with all other variables. "TREX_DA_BLOCK_EQUI", for block equi-correlated variables, makes the same comparison inside contiguous blocks. "TREX_DA_NN" compares a variable with the neighbors inside the AR(1) window whose sample correlation reaches `rho_thr_DA`, and a variable without such a neighbor keeps its relative occurrence. With a threshold at or above the correlation of the data no neighbor is left, and the variant selects as "TREX".

`cor_coef`
: Correlation coefficient of the dependency-aware variants. -2.0 triggers autoestimation (default: -2.0). The variants use the magnitude of a supplied coefficient. Any other value outside `[-1, 1]` gives a warning and is replaced by the estimate.

`rho_thr_DA` Correlation threshold of the dependency-aware variants (default: 0.02). "TREX_DA_AR1": the window around a variable reaches `ceiling(log(rho_thr_DA) / log(abs(cor_coef)))` columns to each side, up to the first lag whose correlation `abs(cor_coef)^lag` has fallen to the threshold, range `[0, 1)`. "TREX_DA_NN": the same window, of which the variables stay whose sample correlation with the variable reaches the threshold, range `[0, 1)`. A threshold at or above the correlation of the data leaves no neighbor, the variant then selects as "TREX" and the filter protects no longer, so the threshold stays below the correlation that the filter is meant to guard against. At the default the variant selected as "TREX_DA_AR1" in simulations. "TREX_DA_EQUI" and "TREX_DA_BLOCK_EQUI": the filter is skipped when `abs(cor_coef)` does not exceed the threshold, range `[0, 1]`.

`n_blocks` Number of blocks of "TREX_DA_BLOCK_EQUI", between 1 and the number of variables (default: 5).

Value

A named list for use in TRexScreeningSelector or TRexBiobankScreeningSelector.

See Also

Other T-Rex selectors and controls: `Gauss_data`, `TRexBiobankScreeningSelector`, `TRexDASelector`, `TRexGVSSelector`, `TRexSPCASelector`, `TRexScreeningSelector`, `TRexSelector`, `TRexTikhonovSelector`, `trex_biobank_control()`, `trex_build_group_centering_k()`, `trex_control()`, `trex_da_control()`, `trex_default_lambda2_grid()`, `trex_dummy_distribution()`, `trex_fdp()`, `trex_fdp_dense()`, `trex_gamma_to_k()`, `trex_get_num_threads()`, `trex_gvs_control()`, `trex_gvs_lambda2_power_rule()`, `trex_normalize_design()`, `trex_precision()`, `trex_recall()`, `trex_spca_control()`, `trex_tikhonov_control()`, `trex_tikhonov_lambda2_power_rule()`, `trex_tpp()`, `trex_tpp_dense()`

Examples

```
trex_screen_control()
trex_screen_control(use_bootstrap_CI = TRUE, R_boot = 500)
```

`trex_spca_control` *T-Rex+SPCA Control Parameters*

Description

Build the control list of the T-Rex sparse PCA selector (T-Rex+SPCA), `TRexSPCASelector`. The list holds the parameters below. Every other parameter of the T-Rex+GVS run of each component (the clustering, the cross-validation grid, the L-loop strategy, the scaling mode, the threads) runs at its default, see `trex_gvs_control()` and `trex_control()`.

Usage

```
trex_spca_control(
  mode = "ActiveSet",
  lambda2_ridge_loadings = 1e-06,
  gvs_type = "EN",
  lambda_2 = -1,
  lambda2_method = "CV_1SE_CCD",
  en_solver = "TENET_AUG",
  tenet_aug_use_lars = FALSE,
  K = 20L,
  max_dummy_multiplier = 10L
)
```

Arguments

mode	Assembly of the sparse loading vectors. "ActiveSet" (default) is the ridge regression of the component on the selected variables, normalized to unit length (the T-Rex PCA of the paper). "Thresholded" keeps the largest ordinary PCA loadings in absolute value, as many as variables were selected, normalized to unit length (the T-Rex Thresholded PCA of the paper).
lambda2_ridge_loadings	Ridge penalty of the loading assembly in "ActiveSet" mode (default: 1e-6). Not used in "Thresholded" mode.
gvs_type	Penalty of the T-Rex+GVS runs. Only "EN" (default, elastic net) is accepted: the method is defined on the elastic net, and "IEN" is refused.
lambda_2	Ridge part of the elastic net of the T-Rex+GVS runs. A negative value (default: -1) lets the rule lambda2_method choose it, 0 is the T-LASSO without the ridge part, and a positive value is used as it is. A supplied value refers to the scale of the standardized columns.
lambda2_method	Rule that chooses lambda_2 when lambda_2 < 0: "CV_1SE_CCD" (default), "CV_MIN_CCD", "CV_1SE_SVD" or "CV_MIN_SVD". The rules of the informed elastic net are refused.
en_solver	Elastic net solver of the T-Rex+GVS runs. "TENET_AUG" (default) is the elastic net as a LASSO on row-augmented data, the form of the paper. "TENET" is the elastic net on the Gram matrix. Both solve the same elastic net problem.
tenet_aug_use_lars	Logical. For en_solver = "TENET_AUG": run the augmented system with a pure-LARS inner path that never drops variables (TRUE) instead of the LASSO inner path that can (FALSE, default). Ignored for "TENET".
K	Number of random experiments of each T-Rex+GVS run (default: 20).
max_dummy_multiplier	Cap of the dummy multiplier of each T-Rex+GVS run: the L-loop adds dummies up to this many times the number of variables (default: 10).

Value

A named list for passing to TRexSPCASelector$new().

See Also

Other T-Rex selectors and controls: Gauss_data, TRexBiobankScreeningSelector, TRexDASelector, TRexGVSSelector, TRexSPCASelector, TRexScreeningSelector, TRexSelector, TRexTikhonovSelector, trex_biobank_control(), trex_build_group_centering_k(), trex_control(), trex_da_control(), trex_default_lambda2_grid(), trex_dummy_distribution(), trex_fdp(), trex_fdp_dense(), trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_control(), trex_gvs_lambda2_power_rule(), trex_normalize_design(), trex_precision(), trex_recall(), trex_screen_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
trex_spca_control()
trex_spca_control(mode = "Thresholded", K = 10)
trex_spca_control(lambda_2 = 0.01)
```

---

trex_tikhonov_control *T-Rex+Tikhonov Control Parameters*

Description

Build the control list of TRexTikhonovSelector, the T-Rex selector with a general Tikhonov penalty (T-Rex+Tikhonov). The list carries the penalty, the rule that chooses its weight lambda_2, and how the dummies are drawn. The help of the class describes the method.

Usage

```
trex_tikhonov_control(
  tikhonov_K = NULL,
  penalty_structure = "SPARSE_GENERAL",
  lambda_2 = -1,
  lambda2_method = "PILOT_KNEE",
  cv_n_folds = 10,
  cv_n_lambda = 100,
  cv_n_lambda1 = 40,
  cv_n_lambda2 = 100,
  cv_dfmax_ratio = 0.8,
  cv_seed = -1,
  pilot_grid = trex_default_lambda2_grid(),
  pilot_K = 10,
  pilot_T = 5,
  pilot_L_multiplier = 1,
  pilot_knee_tolerance = 1,
  pilot_statistic = "MAJORITY",
  pilot_group_fraction = 0.5,
  pilot_shared_dummies = FALSE,
  fold_dummy_coupling = TRUE,
```

```
  cluster_dummies = TRUE,
  prior_groups = NULL,
  corr_max = 0.5,
  hc_linkage = "single"
)
```

Arguments

tikhonov_K	Penalty matrix K (p x p), symmetric and positive semi-definite: a base matrix or a sparse Matrix::dgCMatrix. trex_gamma_to_k builds it from a regularization operator Gamma. Required for penalty_structure = "SPARSE_GENERAL", where the selector checks its content once at construction (finite entries, symmetry, positive semi-definiteness). Optional under "GROUP_CENTERING": left out, the selector builds the centering matrix from prior_groups. A supplied matrix is compared with that centering matrix, and another matrix is an error.
penalty_structure	Structural form of the penalty. "SPARSE_GENERAL" (default): the supplied tikhonov_K is the penalty. "GROUP_CENTERING": K is the block-diagonal centering matrix I - J / p_m over the groups of prior_groups, which this structure requires, so the members of a group share a coefficient (trex_build_group_centering_k returns the matrix). A partition into single variables is allowed: K is then zero and the path is the lasso path.
lambda_2	Weight of the penalty. A negative value (default: -1) lets lambda2_method choose it, 0 is no penalty (the lasso path of the solver), and a positive value is used as it is. The value must be finite. A supplied value refers to the scale of the standardized columns. Under scaling_mode = "ZSCORE" the columns have the squared norm n - 1, so the same shrinkage needs the value times n - 1. The rules report their pick on this scale.
lambda2_method	Rule that chooses lambda_2 when lambda_2 < 0. "PILOT_KNEE" (default), the pilot rule, reads it off a few random experiments per value of pilot_grid: the smallest lambda_2 whose pilot selection count is within pilot_knee_tolerance of the largest, the knee of the profile. The field lambda2_pilot of the selector holds its record. The cross-validation rules tune on the normalized data before the random experiments. "CV_1SE_IEN_CCD" and "CV_MIN_IEN_CCD" profile the lambda_1 path out per lambda_2 candidate and are well-posed for every p and n. "CV_1SE_TIK_SVD" and "CV_MIN_TIK_SVD" are an analytic ridge cross-validation (lambda_1 = 0), exact for n > p and refused when the unpenalized null space interpolates the training folds. The knee of the full selection profile, one selection per grid value, is trex_tikhonov_lambda2_power_rule.
cv_n_folds	Folds of the cross-validation rules, 2 to n (default: 10).
cv_n_lambda	"CV_1SE_TIK_SVD" and "CV_MIN_TIK_SVD": points on the lambda_2 grid (default: 100).
cv_n_lambda1	"CV_1SE_IEN_CCD" and "CV_MIN_IEN_CCD": points on the profiled lambda_1 path per lambda_2 value (default: 40).
cv_n_lambda2	"CV_1SE_IEN_CCD" and "CV_MIN_IEN_CCD": points on the lambda_2 grid (default: 100).

cv_dfmax_ratio "CV_1SE_IEN_CCD" and "CV_MIN_IEN_CCD": support cap as a fraction of the smallest fold's training size. Each lambda_1 path stops once more than floor(cv_dfmax_ratio * n_train) coefficients are nonzero (default: 0.8). A value <= 0 disables the cap. Larger values allow larger supports at a higher cost. Must be finite.

cv_seed Seed of the fold permutation of the cross-validation rules. -1 (default) derives it from the seed of the selector, a value >= 0 replaces it for the folds only.

pilot_grid Pilot rule: lambda_2 candidates, stated for columns of unit norm, sorted and de-duplicated internally. The default is the grid of the C++ core, trex_default_lambda2_grid(): 0, 1, 3, 10, 30, 100, 300, 1000, 3000. Under scaling_mode = "ZSCORE" the pilot runs at these values times n - 1, as the cross-validation rules return their pick, so one grid means the same shrinkage under every scaling mode. The pick is reported on the working scale and can be supplied as lambda_2. Keep 0 on the grid so that "no penalty" is an admissible answer: a penalty that misfits the truth loses power as lambda_2 grows, and the rule then switches it off.

pilot_K Pilot rule: pilot experiments per grid value (default: 10; at least 2, capped at K of trex_control). The majority count has the resolution 1 / pilot_K.

pilot_T Pilot rule: stopping time of the pilot paths (default: 5; at least 1, capped at min(L_0, ceiling(n / 2)), as the T-loop caps its stopping time). The pick reads the count at this stopping time.

pilot_L_multiplier
Pilot rule: pilot dummies L_0 = pilot_L_multiplier * p (default: 1; at most max_dummy_multiplier of trex_control).

pilot_knee_tolerance
Pilot rule: selections below the largest pilot count that still count as "on the plateau" (default: 1). 0 reads the largest count, ties toward the smallest lambda_2.

pilot_statistic
Pilot rule: the count the pick is read off, per grid value. "MAJORITY" (default) is the number of variables ahead of the last pilot dummy (the pilot_T-th) in more than half of the pilot experiments, the count of the T-Rex selector at the voting level 1/2. It is zero under the global null. "CALIBRATED" is the T-Rex selection rule applied to the pilot data, the largest selection whose estimated FDP stays within the target. It is coarser at a small pilot_K. "GROUP_MAJORITY" is the majority count restricted to the groups of prior_groups in which at least pilot_group_fraction of the members are ahead. A group that enters as a whole counts with its members, a group that enters as a minority counts nothing. It needs prior_groups. The record lambda2_pilot holds all three counts.

pilot_group_fraction
"GROUP_MAJORITY": the share of a group's members that must be ahead for the group to count, in (0, 1] (default: 0.5).

pilot_shared_dummies
Logical (default FALSE): the pilot draws its dummies from a seed stream of its own, so the dummies of the main run do not depend on the pick, and the main run equals a run at the picked lambda_2 with the same seed. TRUE: the pilot uses the first dummy block of the main strategy (not with the PERMUTATION strategies).

`fold_dummy_coupling`
: Logical (default `TRUE`): in the penalty every dummy takes the place of the variable it was drawn for (folded coupling), so dummies and null variables stay exchangeable for every K. `FALSE`, a ridge block of the dummies decoupled from K, is an error: it existed with the solver `"TCDTIKNET"` only, which the selector no longer accepts. Deprecated: the argument will be removed with the coordinate-descent family in TRexSelector 2.1.0.

`cluster_dummies`
: Logical (default `TRUE`): draw the dummies per cluster of correlated variables, as in `TRexGVSSelector`. `FALSE` gives i.i.d. dummies, for validation runs only: with an informed K they deform the exchangeability of dummies and null variables that the FDR calibration relies on.

`prior_groups`
: Groups of the variables: an integer vector with the group of each variable, numbered 1 to M without gaps (length p). They define the clusters of the dummies (with `cluster_dummies`), the penalty under `"GROUP_CENTERING"` and the count `"GROUP_MAJORITY"` of the pilot rule. `NULL` (default): the clusters of the dummies are found by clustering the correlations with `corr_max` and `hc_linkage`.

`corr_max`
: Correlation threshold of the automatic clustering, between 0 and 1 (default: 0.5). The dendrogram is cut at the height `1 - corr_max`. With "single" linkage (default) no two variables of different clusters correlate above `corr_max`. With "complete" linkage every two variables of one cluster correlate at least at `corr_max`, and variables of different clusters can exceed it. Ignored when `prior_groups` is set or `cluster_dummies = FALSE`.

`hc_linkage`
: Hierarchical clustering linkage: "single", "complete", "average", "wpgma" (default: "single"). "ward" is rejected, because its update is valid for squared Euclidean distances only and the correlation distance `1 - abs(cor)` is not Euclidean. Ignored when `prior_groups` is set or `cluster_dummies = FALSE`. The capitalized names of TRexSelector 2.0.1 ("Single") are accepted as well.

Value

A named list for use in `TRexTikhonovSelector`.

See Also

Other T-Rex selectors and controls: `Gauss_data`, `TRexBiobankScreeningSelector`, `TRexDASelector`, `TRexGVSSelector`, `TRexSPCASelector`, `TRexScreeningSelector`, `TRexSelector`, `TRexTikhonovSelector`, `trex_biobank_control()`, `trex_build_group_centering_k()`, `trex_control()`, `trex_da_control()`, `trex_default_lambda2_grid()`, `trex_dummy_distribution()`, `trex_fdp()`, `trex_fdp_dense()`, `trex_gamma_to_k()`, `trex_get_num_threads()`, `trex_gvs_control()`, `trex_gvs_lambda2_power_rule()`, `trex_normalize_design()`, `trex_precision()`, `trex_recall()`, `trex_screen_control()`, `trex_spca_control()`, `trex_tikhonov_lambda2_power_rule()`, `trex_tpp()`, `trex_tpp_dense()`

Examples

```
trex_tikhonov_control(tikhonov_K = diag(10))
trex_tikhonov_control(penalty_structure = "GROUP_CENTERING",
                      prior_groups = rep(1:2, each = 5),
                      pilot_grid = c(0, 1, 10, 100))
```

```
trex_tikhonov_control(tikhonov_K = diag(10),
                      lambda2_method = "CV_1SE_IEN_CCD")
```

`trex_tikhonov_lambda2_power_rule`
The knee of the full lambda_2 profile of the T-Rex+Tikhonov selector

---

Description

Runs one FDR-controlled selection of `TRexTikhonovSelector` per value of `grid` with the same seed, so the probes share their dummies and differ by the penalty weight alone, and picks the smallest lambda_2 whose selection is within `knee_tolerance` selections of the largest (the knee of the profile). This is the reference rule behind the default `lambda2_method = "PILOT_KNEE"` of `trex_tikhonov_control`, which reads the same knee off a few random experiments per grid value at a fraction of the cost. The full profile costs one selection per grid value and returns every probe's selection. The pick depends on y and lies outside the fixed-lambda_2 FDR guarantee that each probe carries. `trex_gvs_lambda2_power_rule` is the same rule on the T-Rex+GVS selector.

Usage

```
trex_tikhonov_lambda2_power_rule(
  X,
  y,
  tFDR = 0.1,
  tik_control,
  control = NULL,
  seed = 0,
  grid = trex_default_lambda2_grid(),
  knee_tolerance = 1,
  verbose = FALSE
)
```

Arguments

`X`	Feature matrix (n x p) in memory, or an `mmap_matrix`, the handle of a memory-mapped file. Every probe normalizes it in place and hands it back, so `X` is unchanged on return and never copied. A memory-mapped design needs `"readwrite"` mode for that, and `"readonly"` mode suffices under `scaling_mode = "NONE"`.
`y`	Response vector (length n).
`tFDR`	Target FDR level (default: 0.1).
`tik_control`	A Tikhonov control list from `trex_tikhonov_control()`. Its `lambda_2` is replaced per probe, everything else is shared by every probe.
`control`	A control list from `trex_control()`, or `NULL` (default), taken as by `TRexTikhonovSelector`: with `NULL` the base control of the C++ class applies, and a control that is passed must name the solver `"TIKNET"`.

`seed` Selector seed, the same for every probe (default: 0). A negative seed would draw fresh dummies per probe and is refused.

`grid` lambda_2 candidates, stated for columns of unit norm, sorted and de-duplicated. The default is the grid of the pilot rule, `trex_default_lambda2_grid()`: 0, 1, 3, 10, 30, 100, 300, 1000, 3000. Under `scaling_mode = "ZSCORE"` the probes run at these values times `n - 1`, as the pilot rule does. Keep 0 on the grid so that "no penalty" is an admissible answer.

`knee_tolerance` Selections below the largest count that still count as "on the plateau" (default: 1). `0` reads the largest selection, ties toward the smallest lambda_2.

`verbose` Whether the probes print their progress (default: `FALSE`).

Value

A list:

`lambda2` The pick, on the working scale of the runs. Supplied as `lambda_2`, it reproduces the selection at the pick.

`index` Index of the pick into `grid`, counted from 1.

`selected` The selection at the pick (indices from 1, ascending).

`grid` The values the probes ran at, ascending and without duplicates: the supplied grid times `grid_scale`.

`grid_scale` The factor between the supplied grid and `grid`: `n - 1` under `scaling_mode = "ZSCORE"`, 1 otherwise.

`counts` The selection sizes per grid value.

`probes` One list per grid value with `lambda2`, `selected`, `T_stop` (the stopping time of the run) and `seconds`.

`max_selected` The largest selection on the grid.

`n_ties` Grid values within the tolerance of the largest selection.

`at_lower_end` The pick sits on the first grid value.

`at_upper_end` The pick sits on the last grid value, a boundary solution: widen the grid before trusting it.

`seconds` Wall-clock seconds of the whole profile.

See Also

Other T-Rex selectors and controls: `Gauss_data`, `TRexBiobankScreeningSelector`, `TRexDASelector`, `TRexGVSSelector`, `TRexSPCASelector`, `TRexScreeningSelector`, `TRexSelector`, `TRexTikhonovSelector`, `trex_biobank_control()`, `trex_build_group_centering_k()`, `trex_control()`, `trex_da_control()`, `trex_default_lambda2_grid()`, `trex_dummy_distribution()`, `trex_fdp()`, `trex_fdp_dense()`, `trex_gamma_to_k()`, `trex_get_num_threads()`, `trex_gvs_control()`, `trex_gvs_lambda2_power_rule()`, `trex_normalize_design()`, `trex_precision()`, `trex_recall()`, `trex_screen_control()`, `trex_spca_control()`, `trex_tikhonov_control()`, `trex_tpp()`, `trex_tpp_dense()`

Examples

```
data(Gauss_data)
X <- Gauss_data$X
y <- Gauss_data$y
ctrl <- trex_tikhonov_control(tikhonov_K = diag(ncol(X)))
prof <- trex_tikhonov_lambda2_power_rule(X, y, tFDR = 0.1, tik_control = ctrl,
                                         grid = c(0, 1, 10), seed = 1)
prof$lambda2
prof$counts
```

`trex_tpp` *Compute True Positive Proportion (TPP) using index sets*

Description

Compute True Positive Proportion (TPP) using index sets

Usage

```
trex_tpp(selected_indices, true_support)
```

Arguments

`selected_indices`	Integer vector of selected indices.
`true_support`	Integer vector of true active indices.

Value

Numeric TPP value.

See Also

Other T-Rex selectors and controls: `Gauss_data`, `TRexBiobankScreeningSelector`, `TRexDASelector`, `TRexGVSSelector`, `TRexSPCASelector`, `TRexScreeningSelector`, `TRexSelector`, `TRexTikhonovSelector`, `trex_biobank_control()`, `trex_build_group_centering_k()`, `trex_control()`, `trex_da_control()`, `trex_default_lambda2_grid()`, `trex_dummy_distribution()`, `trex_fdp()`, `trex_fdp_dense()`, `trex_gamma_to_k()`, `trex_get_num_threads()`, `trex_gvs_control()`, `trex_gvs_lambda2_power_rule()`, `trex_normalize_design()`, `trex_precision()`, `trex_recall()`, `trex_screen_control()`, `trex_spca_control()`, `trex_tikhonov_control()`, `trex_tikhonov_lambda2_power_rule()`, `trex_tpp_dense()`

Examples

```
trex_tpp(c(1L, 3L, 5L), c(1L, 2L, 3L))
```

trex_tpp_dense *Compute True Positive Proportion (TPP) using dense vectors*

Description

Compute True Positive Proportion (TPP) using dense vectors

Usage

```
trex_tpp_dense(beta_hat, beta, eps = 1e-15)
```

Arguments

beta_hat	Numeric vector of estimated coefficients.
beta	Numeric vector of true coefficients.
eps	Numeric threshold for counting non-zeros (default: 1e-15).

Value

Numeric TPP value.

See Also

Other T-Rex selectors and controls: Gauss_data, TRexBiobankScreeningSelector, TRexDASelector, TRexGVSSelector, TRexSPCASelector, TRexScreeningSelector, TRexSelector, TRexTikhonovSelector, trex_biobank_control(), trex_build_group_centering_k(), trex_control(), trex_da_control(), trex_default_lambda2_grid(), trex_dummy_distribution(), trex_fdp(), trex_fdp_dense(), trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_control(), trex_gvs_lambda2_power_rule(), trex_normalize_design(), trex_precision(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp()

Examples

```
trex_tpp_dense(c(1.5, 0, 0.8, 0, 0), c(2.0, 0, 1.0, 0, 0))
```

TRexBiobankScreeningSelector *Biobank Screening with the T-Rex Selector*

Description

Per phenotype the fast screening (T-Rex+Screening), and the T-Rex selector as fallback when no FDR estimate of the screening lies in the accepted range.

Decision rule per phenotype

The screening gives two FDR estimates, the one of its ordinary variant and the one of its bootstrap variant. A variant is accepted when its estimate lies in `[lower_bound_FDR, upper_bound_FDR]` of `trex_biobank_control()`. Of two accepted variants the one with the larger estimate is taken, and the bootstrap variant on a tie. With no accepted variant the T-Rex selector runs at `target_FDR_trex`.

Both estimates come from one set of K experiments. The screening runs once, and the bootstrap variant is derived from its coefficients.

Methods

Public methods:

- `TRexBiobankScreeningSelector$new()`
- `TRexBiobankScreeningSelector$select()`
- `TRexBiobankScreeningSelector$clone()`

`TRexBiobankScreeningSelector$new()`: Create a new TRexBiobankScreeningSelector object.

Usage:

```
TRexBiobankScreeningSelector$new(
  X,
  Y,
  tFDR = NULL,
  seed = -1,
  verbose = FALSE,
  biobank_control = trex_biobank_control(),
  screen_control = trex_screen_control(),
  control = trex_control()
)
```

Arguments:

`X` Feature matrix (n x p) in memory, or an `mmap_matrix`, the handle of a memory-mapped file. Both are bound without a copy. The inner selectors standardize a memory-mapped design in place per call and restore it, which needs `"readwrite"` mode. Under `scaling_mode = "NONE"` they write nothing and `"readonly"` mode suffices.

`Y` Response vector (1D) or matrix (2D). The matrix is in memory or an `mmap_matrix`, the handle of a memory-mapped file, and is bound without a copy. It is only read, so `"readonly"` mode suffices for it.

`tFDR` Target FDR level of the fallback T-Rex selector. `NULL` (default) takes `target_FDR_trex` of `biobank_control`. A supplied value replaces it, as in TRexSelector 2.0.1.

`seed` Random seed. A negative value (default: -1) gives a run that is not reproducible. Phenotype `k` (the `k`-th column of `Y`) runs with `seed + k - 1`, so the phenotypes draw independent dummies and a single phenotype is reproduced by a single-phenotype run with that seed.

`verbose` Whether to print progress (default: FALSE).

`biobank_control` A biobank control list from `trex_biobank_control()` (default: `trex_biobank_control()`).

`screen_control` A screen control list from `trex_screen_control()` (default: `trex_screen_control()`). Its `use_bootstrap_CI` has no effect here: the biobank screening computes the ordinary and the bootstrap variant for every phenotype.

`control` A control list from `trex_control()` (default: `trex_control()`). The screening takes the values `"STANDARD"` (default), `"SEEDED"`, `"PERMUTATION"` and `"PERMUTATION_SEEDED"` of `lloop_strategy` and refuses `"HCONCAT"` and `"SKIPL"`, as `TRexScreeningSelector` does.

`TRexBiobankScreeningSelector$select()`**:** Run the Biobank Screening algorithm.

Usage:

```
TRexBiobankScreeningSelector$select()
```

Returns: For a vector `Y`, a single per-phenotype record: a named list with `phenotype_index`, `selected_indices`, `estimated_FDR`, `method_used` (`"Screen-TRex (ordinary)"`, `"Screen-TRex (bootstrap-CI)"` or `"T-Rex (fallback)"`), `estimated_FDR_screen_ordinary`, `estimated_FDR_screen_bootstrap`, `selected_indices_screen_ordinary`, `selected_indices_screen_bootstrap`, and `used_fallback_trex` (indices are 1-based). `estimated_FDR` is the estimate of the accepted screening variant, or, for a fallback result, the target level `target_FDR_trex`, at which the T-Rex selector controls the FDR. An estimate can equal the target, so `used_fallback_trex` tells the two cases apart. The two screening estimates and the two screening index vectors are stored for every phenotype, a fallback result included. For a matrix `Y`, a plain list of such records, one per column, so `res[[i]]` has the same shape as the vector case. This mirrors the Python binding, which returns one result object (vector) or a list of them (matrix). A tidy summary table is one line away, e.g. `do.call(rbind, lapply(res, function(r) as.data.frame(r[c("phenotype_index", "estimated_FDR", "method_used", "used_fallback_trex")])))`.

`TRexBiobankScreeningSelector$clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TRexBiobankScreeningSelector$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

J. Machkour, M. Muma, and D. P. Palomar, "False Discovery Rate Control for Fast Screening of Large-Scale Genomics Biobanks," in Proc. 22nd IEEE Statist. Signal Process. Workshop (SSP), 2023, pp. 666-670.

F. Scheidt, J. Machkour, and M. Muma, "Solving FDR-Controlled Sparse Regression Problems with Five Million Variables on a Laptop," in Proc. IEEE 9th Int. Workshop Comput. Adv. Multi-Sensor Adapt. Process. (CAMSAP), 2023, pp. 116-120.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-Rex selectors and controls: `Gauss_data`, `TRexDASelector`, `TRexGVSSelector`, `TRexSPCASelector`, `TRexScreeningSelector`, `TRexSelector`, `TRexTikhonovSelector`, `trex_biobank_control()`, `trex_build_group_centering_k()`, `trex_control()`, `trex_da_control()`, `trex_default_lambda2_grid()`, `trex_dummy_distribution()`, `trex_fdp()`, `trex_fdp_dense()`, `trex_gamma_to_k()`, `trex_get_num_threads()`,

trex_gvs_control(), trex_gvs_lambda2_power_rule(), trex_normalize_design(), trex_precision(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
data(Gauss_data)
X <- Gauss_data$X
# A vector is one phenotype and gives one record, a matrix gives a list of records
y <- drop(Gauss_data$y)
ctrl <- trex_biobank_control(target_FDR_trex = 0.1)
sel <- TRexBiobankScreeningSelector$new(X, y, biobank_control = ctrl)
res <- sel$select()
res$selected_indices
```

TRexDASelector *Dependency-Aware T-Rex Selector (T-Rex+DA)*

Description

FDR-controlled variable selection for dependent variables.

The penalty of the last step

The dependency-aware relative occurrences of variable j at step t of the T-loop are

$$\Phi_t^{DA}(j) = \Psi_T(j)\,\Phi_t(j), \qquad \Psi_T(j) = \frac{1}{2 - \min_{j'} |\Phi_T(j) - \Phi_T(j')|}, \qquad t = 1, \ldots, T,$$

where T is the last step of the evaluated path and j' runs over the variables that j is compared with (its group, its window or all other variables). A variable without a group gets $\Psi_T(j) = 1/2$.

This is a deliberate deviation from the papers below. Read literally, they recompute the penalty at every step, $\Psi_t(j)$ from Φ_t. The selector takes the penalty of the last step for all steps, with all five methods. The reason is the FDP estimator. It sums over the increments $\Phi_t^{DA}(j) - \Phi_{t-1}^{DA}(j)$ and counts expected false discoveries only while no increment is negative. One penalty per variable keeps the increments non-negative. A penalty per step does not, because the penalty of a variable changes from step to step, and the FDP estimate then becomes too small.

A Monte Carlo comparison of the two constructions (1,000 runs per setting on block, AR(1) and independent designs) supports the choice. With the method `"NN"` the per-step penalty exceeded a target FDR of 0.1, with 0.15 on an independent design and 0.13 on a block design, where the last-step penalty gave 0.00 and 0.03. No setting had the last-step penalty above the target and the per-step penalty below it. The per-step penalty selected more variables only where its estimate was invalid, and on AR(1) designs the last-step penalty had the higher true positive rate.

The methods `"AR1"` and `"EQUI"` used the per-step penalty up to TRexSelector 2.0.1, so their selections can differ slightly from earlier versions. `"BT"`, `"NN"` and prior groups have always used the penalty of the last step.

The reference threshold

The methods "BT", "NN" and prior groups evaluate every step on the whole grid of correlation thresholds (rho_grid). The L-loop and the T-loop read the FDP estimate at one of them, the reference threshold, which sits at three quarters of the grid: element 0.75 times its length, a half rounded up (element 15 of the default 20). The 1.x versions of the package round a half to the even number and take one element lower for the grid lengths 6, 14, 22, 30 and so on. The selection then searches all cells of stopping time, voting level and threshold, and rho_thresh is the threshold of the cell it takes. The inherited fields fdp_hat_mat, phi_mat and phi_prime therefore hold the values at the reference threshold, while r_mat, v_thresh and the selection belong to rho_thresh. The values at every threshold are in FDP_hat_array_BT, Phi_array_BT and R_array_BT.

FDR limitation of BT, NN and prior groups

At a very high correlation within the groups (about 0.9 and above) and with no independent ("white") predictors, the equiangular solver "TLARS" can exceed the target FDR, because the true signal and its collinear shadows become statistically exchangeable. This is inherent to the method, and it also appears in the 1.x versions of the package. Mitigate with the default bt_selection_mode = "FeasibleOnly" and by preferring a greedy solver ("TOMP") in designs with high correlation.

Greedy solvers and the exchangeable-tie band

A greedy solver ("TOMP", "TOOLS", "TGP", "TACGP", "TMP", "TNCGMP", "TAFS") picks the top candidate of a correlated cluster deterministically, which removes the spread of occurrences within the cluster that the penalty relies on. The selector therefore runs a greedy solver with the exchangeable-tie band at 0.25 when trex_control(exch_tie_alpha =) is left at its default of -1. A value of 0 switches the band off, and a positive value sets its width. The path solvers ignore the band.

Evidence: a Monte Carlo study on AR(1) designs (1,000 runs per setting, target FDR 0.2, method "AR1", November 2026). At a correlation of 0.9 each of the seven greedy solvers had an FDR of 0.21 to 0.27 without the band and of 0.13 to 0.17 with it, at a true positive rate lower by 0.07 to 0.13. At correlations up to 0.8 the target held without the band as well. The study shows the control up to a correlation of 0.9. At 0.95 the target no longer held (500 runs per setting), with an FDR of up to 0.23 for the path solver "TLARS" and of up to 0.30 for "TOMP" with the band.

Super class

TRexSelector -> TRexDASelector

Active bindings

rho_thresh Correlation threshold of the selected cell of the calibration grid (BT, NN and prior groups). -2 for AR1 and EQUI and when no cell meets the target.

rho_grid Grid of correlation thresholds of BT, NN and prior groups: hc_grid_length ascending values. BT and prior groups take them from the merge heights of the dendrogram (1 - height) and end at 1, NN spaces them equally on [0, 1]. Empty for AR1 and EQUI.

da_method_used The dependency-aware method of the last select(): "BT", "NN", "AR1", "EQUI", or "PRIOR_GROUPS" when prior groups were supplied.

`FDP_hat_array_BT` FDP estimates of the methods BT, NN and prior groups: an array with one row per stopping time (`T_stop`), one column per voting level (`voting_grid`) and one slice per correlation threshold (`rho_grid`). Empty for AR1 and EQUI.

`Phi_array_BT` Dependency-aware relative occurrences of the methods BT, NN and prior groups: an array with one row per stopping time (`T_stop`), one column per variable and one slice per correlation threshold (`rho_grid`). Empty for AR1 and EQUI.

`R_array_BT` Numbers of selected variables of the methods BT, NN and prior groups: an array with one row per stopping time that can be selected (`T_stop - 1`), one column per voting level (`voting_grid`) and one slice per correlation threshold (`rho_grid`). Empty for AR1 and EQUI.

`cor_coef` Correlation coefficient of the methods AR1 and EQUI: the supplied value, or the estimate from the data when -2 requested it. The other methods do not use it and return the value of the control.

Methods

Public methods:

- `TRexDASelector$new()`
- `TRexDASelector$select()`
- `TRexDASelector$clone()`

`TRexDASelector$new()`**:** Create a new TRexDASelector object.

Usage:

```
TRexDASelector$new(
  X,
  y,
  tFDR = 0.1,
  seed = -1,
  verbose = TRUE,
  da_control = trex_da_control(),
  control = trex_control()
)
```

Arguments:

`X` Feature matrix in memory, or an `mmap_matrix`, the handle of a memory-mapped file. Both are bound without a copy. The selector standardizes a memory-mapped design in place and restores it, which needs `"readwrite"` mode. Under `scaling_mode = "NONE"` it writes nothing and `"readonly"` mode suffices.

`y` Response vector.

`tFDR` Target FDR level (default: 0.1).

`seed` Random seed. A negative value (default: -1) gives a run that is not reproducible.

`verbose` Whether to print progress (default: TRUE).

`da_control` A DA control list from `trex_da_control()` (default: `trex_da_control()`).

`control` A control list from `trex_control()` (default: `trex_control()`).

`TRexDASelector$select()`**:** Run the Dependency-Aware TRexSelector algorithm. A selector runs once: a second call is an error, see `TRexSelector`.

Usage:

```
TRexDASelector$select()
```

`TRexDASelector$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TRexDASelector$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

J. Machkour, M. Muma, and D. P. Palomar, "High-Dimensional False Discovery Rate Control for Dependent Variables," Signal Process., vol. 234, Art. no. 109990, 2025, doi: 10.1016/j.sigpro.2025.109990.

J. Machkour, D. P. Palomar, and M. Muma, "FDR-Controlled Portfolio Optimization for Sparse Financial Index Tracking," in Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP), 2025, pp. 1-5, doi: 10.1109/ICASSP49660.2025.10890629.

J. Machkour, M. Muma, and D. P. Palomar, "FDR-Controlled Sparse Index Tracking with Autoregressive Stock Dependency Models," in Proc. Eur. Signal Process. Conf. (EUSIPCO), 2024, pp. 2662-2666.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-Rex selectors and controls: `Gauss_data`, `TRexBiobankScreeningSelector`, `TRexGVSSelector`, `TRexSPCASelector`, `TRexScreeningSelector`, `TRexSelector`, `TRexTikhonovSelector`, `trex_biobank_control()`, `trex_build_group_centering_k()`, `trex_control()`, `trex_da_control()`, `trex_default_lambda2_grid()`, `trex_dummy_distribution()`, `trex_fdp()`, `trex_fdp_dense()`, `trex_gamma_to_k()`, `trex_get_num_threads()`, `trex_gvs_control()`, `trex_gvs_lambda2_power_rule()`, `trex_normalize_design()`, `trex_precision()`, `trex_recall()`, `trex_screen_control()`, `trex_spca_control()`, `trex_tikhonov_control()`, `trex_tikhonov_lambda2_power_rule()`, `trex_tpp()`, `trex_tpp_dense()`

Examples

```
# Dependent variables: an AR(1) process across the columns, five of them active
set.seed(321)
n <- 100
p <- 100
rho <- 0.7
X <- matrix(0, n, p)
X[, 1] <- rnorm(n)
for (j in 2:p) X[, j] <- rho * X[, j - 1] + sqrt(1 - rho^2) * rnorm(n)
true_support <- c(10, 30, 50, 70, 90)
beta <- numeric(p)
beta[true_support] <- 2
y <- as.numeric(X %*% beta + rnorm(n))
```

```
sel <- TRexDASelector$new(X, y, tFDR = 0.1, seed = 7, verbose = FALSE,
                          da_control = trex_da_control(da_method = "BT"))
sel$select()
sel$selected_indices
```

`TRexGVSSelector` *T-Rex Selector for Grouped Variable Selection (T-Rex+GVS)*

Description

FDR-controlled variable selection for groups of highly correlated variables.

L-loop strategies

The selector supports the strategies `"STANDARD"`, `"HCONCAT"`, `"SKIPL"` and `"SEEDED"` of `trex_control()`, and `"SEEDED"` equals `"STANDARD"` here. `"PERMUTATION"` and `"PERMUTATION_SEEDED"` are refused, because a row permutation destroys the covariance of the dummies within a cluster.

Super class

`TRexSelector` -> `TRexGVSSelector`

Active bindings

`lambda2_used` `lambda_2` of the last `select()`: the supplied value, or the pick of the cross-validation rule `lambda2_method`.

`lambda2_cv_diagnostics` Health record of the lambda_2 tuner of the last `select()`: a list with `ran`, `seconds`, `n_profile_at_edge` (lambda_2 values whose profiled lambda_1 optimum sat on a truncation edge; 0 is healthy), `support_cap` (-1 = disabled), `dfmax_ratio` (IEN tuner: the `cv_dfmax_ratio` it was called with; 0 when no IEN-geometry tuner ran), `low_dimensional`, and the boundary-pick guard fields: `grid_index` / `grid_size` (0-based position of the returned lambda_2 on the tuner's grid), `at_grid_edge` (the pick is the first or last grid point, so the grid boundary rather than a CV optimum), `profile_at_edge` (IEN tuner: the profiled lambda_1 optimum of the chosen lambda_2 sat on the truncation edge), `profile_at_support_cap` (that edge is the support cap; raise `cv_dfmax_ratio` to move it) and `boundary_pick` (`at_grid_edge` or `profile_at_edge`; the selector warned once and returned the pick unchanged).

`gvs_type` Penalty of the grouped selection that was applied, `"EN"` or `"IEN"`.

`max_clusters` Number of groups M, supplied or found by the clustering.

`groups` Integer vector of length p giving the group of each variable, numbered from 1: the supplied groups, or the assignment of the clustering.

`group_labels` Optional human-readable cluster names (length `max_clusters`), or an empty character vector when none were supplied.

`hc_method_used` Origin of the groups: `"prior"` when they were supplied, else the linkage of the clustering, `"single"`, `"complete"`, `"average"` or `"wpgma"`.

Methods

Public methods:

- TRexGVSSelector$new()
- TRexGVSSelector$select()
- TRexGVSSelector$clone()

`TRexGVSSelector$new()`: Create a new TRexGVSSelector object.

Usage:

```
TRexGVSSelector$new(
  X,
  y,
  tFDR = 0.1,
  seed = -1,
  verbose = TRUE,
  gvs_control = trex_gvs_control(),
  control = NULL
)
```

Arguments:

`X` Feature matrix in memory, or an `mmap_matrix`, the handle of a memory-mapped file. Both are bound without a copy. The selector standardizes a memory-mapped design in place and restores it, which needs `"readwrite"` mode. Under `scaling_mode = "NONE"` it writes nothing and `"readonly"` mode suffices.

`y` Response vector.

`tFDR` Target FDR level (default: 0.1).

`seed` Random seed. A negative value (default: -1) gives a run that is not reproducible.

`verbose` Whether to print progress (default: TRUE).

`gvs_control` A GVS control list from `trex_gvs_control()` (default: `trex_gvs_control()`). For `gvs_type = "EN"` its `en_solver` chooses the solver, "TENET" by default. "TCDENET" and "TCDIENET" are deprecated and will be removed in TRexSelector 2.1.0: the selector raises a deprecation warning for them.

`control` A control list from `trex_control()`, or `NULL` (default). With `NULL` the base control of the C++ class applies: the defaults of `trex_control()` with the solver "TENET". A control that is passed is taken as it is, so it has to name a solver of the track. For `gvs_type = "EN"` its `solver` must be "TENET" or the solver of `en_solver`. For `gvs_type = "IEN"` it must be "TIENET" (native pathwise) or "TIENET_AUG" (row-augmented), so the IEN track needs a base control: `trex_control(solver = "TIENET", ...)`. Every other solver is an error, also "TLARS", the default of `trex_control()`.

`TRexGVSSelector$select()`: Run the GVS algorithm. A selector runs once: a second call is an error, see `TRexSelector`.

Usage:

```
TRexGVSSelector$select()
```

`TRexGVSSelector$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TRexGVSSelector$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

J. Machkour, M. Muma, and D. P. Palomar, "False Discovery Rate Control for Grouped Variable Selection in High-Dimensional Linear Models Using the T-Knock Filter," in Proc. 30th Eur. Signal Process. Conf. (EUSIPCO), 2022, pp. 892-896.

J. Machkour, M. Muma, and D. P. Palomar, "The Informed Elastic Net for Fast Grouped Variable Selection and FDR Control in Genomics Research," in Proc. IEEE 9th Int. Workshop Comput. Adv. Multi-Sensor Adapt. Process. (CAMSAP), 2023, pp. 466-470.

H. Zou and T. Hastie, "Regularization and Variable Selection via the Elastic Net," J. R. Stat. Soc. Ser. B Stat. Methodol., vol. 67, no. 2, pp. 301-320, 2005.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-Rex selectors and controls: Gauss_data, TRexBiobankScreeningSelector, TRexDASelector, TRexSPCASelector, TRexScreeningSelector, TRexSelector, TRexTikhonovSelector, trex_biobank_control(), trex_build_group_centering_k(), trex_control(), trex_da_control(), trex_default_lambda2_grid(), trex_dummy_distribution(), trex_fdp(), trex_fdp_dense(), trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_control(), trex_gvs_lambda2_power_rule(), trex_normalize_design(), trex_precision(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
data(Gauss_data)
X <- Gauss_data$X
y <- Gauss_data$y
sel <- TRexGVSSelector$new(X, y, tFDR = 0.1, verbose = FALSE,
                           gvs_control = trex_gvs_control(gvs_type = "EN"))
sel$select()
sel$selected_indices
```

TRexScreeningSelector *T-Rex Screening Selector (T-Rex+Screening)*

Description

Fast FDR-controlled variable screening in one pass of the random experiments, until one dummy is included (T = 1), with as many dummies as variables (L = p).

Fields inherited from TRexSelector

The screening has no target FDR and no calibration, so the fields of `TRexSelector` hold fixed values after a run: `T_stop` is 1, `L` is 1, `num_dummies` is p, `v_thresh` is 0.5 and `tFDR` is 0. `r_mat` is a 1 x 1 matrix with the number of selected variables, `phi_prime` equals the relative occurrences, and `fdp_hat_mat` is a 1 x 1 matrix with 0, because no FDP is estimated per voting level. The FDR estimate of the run is `estimated_FDR`.

A response without signal

The screening stops at the first dummy, so about one null variable is expected to enter before it. On a response of pure noise it selected at least one variable in a quarter to a half of the runs in simulations, with fewer than one selected variable on average. A small selection is therefore no evidence of a signal. The estimate, one over the number of selected variables, rests on that expected number of false discoveries and is no bound on the FDP of a run: with two or more variables selected on noise it is 0.5 or less, while every one of them is false. `TRexBiobankScreeningSelector` accepts a screening result only when the estimate lies in its window and runs the T-Rex selector otherwise, which selected nothing on noise in about 99 % of the runs.

L-loop strategies

The screening takes the strategies `"STANDARD"` (default), `"SEEDED"`, `"PERMUTATION"` and `"PERMUTATION_SEEDED"` of `trex_control()`. `"SEEDED"` gives the results of `"STANDARD"` and keeps one dummy matrix in memory at a time. `"PERMUTATION_SEEDED"` gives the results of `"PERMUTATION"` without storing the base dummy matrix. `"HCONCAT"` and `"SKIPL"` are refused, because they describe the growth of the dummy matrix over the L-loop, and the screening is one pass at L = p.

Super class

`TRexSelector` -> `TRexScreeningSelector`

Active bindings

`estimated_FDR` The estimated FDR, one over the number of selected variables. An empty selection reports 1. It rests on the expected number of false discoveries, about one, and is no bound on the FDP of a run (see the section "A response without signal").

`used_bootstrap` Whether the confidence-based screening (bootstrap) made the selection.

`confidence_level` Confidence level that the grid search of the bootstrap picked. 0 without the bootstrap.

`estimated_correlation` Correlation coefficient of the dependency-aware variants: the supplied value, or the estimate from the data. 0 for "TREX".

`beta_mat` Coefficients of the variables, averaged over the K experiments (one per variable, a p x 1 matrix).

`dummy_betas` Non-zero coefficients of the dummies, collected over the K experiments. They are the sample of the bootstrap.

Methods

Public methods:

- TRexScreeningSelector$new()
- TRexScreeningSelector$select()
- TRexScreeningSelector$clone()

`TRexScreeningSelector$new()`: Create a new TRexScreeningSelector object.

Usage:

```
TRexScreeningSelector$new(
  X,
  y,
  seed = -1,
  verbose = TRUE,
  screen_control = trex_screen_control(),
  control = trex_control()
)
```

Arguments:

`X` Feature matrix in memory, or an `mmap_matrix`, the handle of a memory-mapped file. Both are bound without a copy. The selector standardizes a memory-mapped design in place and restores it, which needs `"readwrite"` mode. Under `scaling_mode = "NONE"` it writes nothing and `"readonly"` mode suffices.

`y` Response vector.

`seed` Random seed. A negative value (default: -1) gives a run that is not reproducible.

`verbose` Whether to print progress (default: TRUE).

`screen_control` A screen control list from `trex_screen_control()` (default: `trex_screen_control()`).

`control` A control list from `trex_control()` (default: `trex_control()`). See the section on the L-loop strategies for the values of `lloop_strategy` that the screening takes.

`TRexScreeningSelector$select()`: Run the Screening algorithm. A selector runs once: a second call is an error, see TRexSelector.

Usage:

```
TRexScreeningSelector$select()
```

`TRexScreeningSelector$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TRexScreeningSelector$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

J. Machkour, M. Muma, and D. P. Palomar, "False Discovery Rate Control for Fast Screening of Large-Scale Genomics Biobanks," in Proc. 22nd IEEE Statist. Signal Process. Workshop (SSP), 2023, pp. 666-670.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-Rex selectors and controls: Gauss_data, TRexBiobankScreeningSelector, TRexDASelector, TRexGVSSelector, TRexSPCASelector, TRexSelector, TRexTikhonovSelector, trex_biobank_control(), trex_build_group_centering_k(), trex_control(), trex_da_control(), trex_default_lambda2_grid(), trex_dummy_distribution(), trex_fdp(), trex_fdp_dense(), trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_control(), trex_gvs_lambda2_power_rule(), trex_normalize_design(), trex_precision(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
data(Gauss_data)
X <- Gauss_data$X
y <- Gauss_data$y
sel <- TRexScreeningSelector$new(X, y, verbose = FALSE)
sel$select()
sel$selected_indices
```

TRexSelector *T-Rex Selector*

Description

FDR-controlled variable selection algorithm.

Active bindings

`.ptr` Get internal pointer.

`.ptr_X` internal proxy for data size

`selected_var` Logical vector indicating which variables were selected.

`selected_indices` Integer vector of selected 1-based indices.

`v_thresh` The chosen voting threshold.

`T_stop` Stopping time: the number of included dummies at which the T-loop stopped, so the random experiments ran until `T_stop` dummies were included. For `T_stop > 1` it is not the stopping time of the selection: the selection is taken from the stopping times 1 to `T_stop - 1`, mostly from `T_stop - 1` and sometimes from an earlier one (see `select()`).

`L` Calibrated dummy multiplier L (the selector uses L * p dummies).

`voting_grid` Voting levels 0.5, 0.5 + 1/K, ..., 1 - eps that the selection searches.

`fdp_hat_mat` Estimated false discovery proportion per T step (rows) and voting level (columns).

`phi_mat` Relative occurrences of the original variables over the K random experiments after each T step (rows).

`r_mat` Number of selected variables per stopping time (rows) and voting level (columns). It has `T_stop - 1` rows, the stopping time `T_stop` is not part of it, and one row for `T_stop = 1`.

`phi_prime` Calibrated relative occurrences used by the FDP estimate (length p), at most 1 and possibly negative.

`x_means` Column means of `X` that the selector subtracted (zeros under `scaling_mode = "NONE"`).

`x_l2_norms` Scale divisors of the columns of `X`: the L2 norms of the centered columns under `scaling_mode = "L2"`, the sample standard deviations under `"ZSCORE"`, ones under `"NONE"`.

`y_mean` Mean of `y` that the selector subtracted (0 under `scaling_mode = "NONE"`).

`tFDR` The target FDR level.

`K` The number of random experiments.

`max_num_dummies` The cap of the dummy multiplier, `max_dummy_multiplier` of `trex_control()`: the L-loop adds dummies up to this many times the number of variables. The field holds a multiplier, not a number of dummies.

`max_T_stop` Whether the bound on the stopping time is on, `use_max_T_stop` of `trex_control()`: the T-loop then ends at `ceiling(n / 2)` at the latest. The field holds a logical, not a stopping time.

`stagnation_check` Whether the T-loop stops on stagnation, `tloop_stagnation_stop` of `trex_control()` after the default by the solver family is resolved.

`num_selected` The number of selected variables.

`num_dummies` The number of dummies used by the last select() (the L-loop's calibration).

Methods

Public methods:

- `TRexSelector$new()`
- `TRexSelector$restore()`
- `TRexSelector$select()`
- `TRexSelector$clone()`

`TRexSelector$new()`**:** Create a new TRexSelector object.

Usage:

```
TRexSelector$new(
  X,
  y,
  tFDR = 0.1,
  seed = -1,
  verbose = TRUE,
  control = trex_control()
)
```

Arguments:

`X` Feature matrix in memory, or an `mmap_matrix`, the handle of a memory-mapped file. A numeric matrix of type double and a memory-mapped file are bound without a copy. Any other matrix or a data frame is converted first, which is a copy. The selector standardizes a memory-mapped design in place and restores it, which needs `"readwrite"` mode. Under `scaling_mode = "NONE"` it writes nothing and `"readonly"` mode suffices.

`y` Response vector. The selector works on a copy of it.

`tFDR` Target FDR level in `[0, 1]` (default: 0.1).

`seed` Random seed (default: -1). A seed of 0 or more makes the run deterministic and reproducible: the same seed draws the same dummies. A negative seed draws them from the hardware entropy source, and the run is not reproducible bit by bit. A Monte Carlo study gives every trial a seed of its own, or uses -1.

`verbose` Whether to print progress (default: TRUE).

`control` A control list from `trex_control()` (default: `trex_control()`).

`TRexSelector$restore()`**:** Give `X` back before `select()`: undo the in-place normalization applied at construction and release the buffer claim (idempotent; a no-op after `select()`, which restores `X` itself). A restored selector cannot run: `select()` then errors, construct a new one. Without it, an unselected selector gives `X` back only when it is garbage-collected.

Usage:

```
TRexSelector$restore()
```

Returns: Invisibly returns `self`.

`TRexSelector$select()`**:** Run the TRexSelector algorithm. A selector runs once: `select()` gives `X` back when it ends, so a second call is an error instead of a run on a design that is no longer standardized. Construct a new selector to run again. The fields keep the results of the run. A `select()` that ends with an error gives `X` back at once (original scale) and retires the selector as well.

The T-loop raises the stopping time until the FDP estimate at the highest voting level exceeds `tFDR`, or until its bound or the stagnation stop ends it, and `T_stop` is where it stopped. The selection is the cell with the largest number of selected variables among the stopping times 1 to `T_stop - 1` (the stopping time 1 for `T_stop = 1`) and the voting levels whose FDP estimate does not exceed `tFDR`. Among equal counts the highest voting level wins, then the latest stopping time. The stopping time of the selection is therefore below `T_stop`, mostly by one and sometimes by more.

Usage:

```
TRexSelector$select()
```

Returns: invisible(self)

`TRexSelector$clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TRexSelector$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-Rex selectors and controls: Gauss_data, TRexBiobankScreeningSelector, TRexDASelector, TRexGVSSelector, TRexSPCASelector, TRexScreeningSelector, TRexTikhonovSelector, trex_biobank_control(), trex_build_group_centering_k(), trex_control(), trex_da_control(), trex_default_lambda2_grid(), trex_dummy_distribution(), trex_fdp(), trex_fdp_dense(), trex_gamma_to_k(), trex_get_num_threads(), trex_gvs_control(), trex_gvs_lambda2_power_rule(), trex_normalize_design(), trex_precision(), trex_recall(), trex_screen_control(), trex_spca_control(), trex_tikhonov_control(), trex_tikhonov_lambda2_power_rule(), trex_tpp(), trex_tpp_dense()

Examples

```
data(Gauss_data)
X <- Gauss_data$X
y <- Gauss_data$y
sel <- TRexSelector$new(X, y, tFDR = 0.1, verbose = FALSE)
sel$select()
sel$selected_indices
```

TRexSPCASelector *T-Rex Sparse PCA Selector (T-Rex+SPCA)*

Description

Sparse principal components whose non-zero loadings are selected with FDR control. Each ordinary principal component is the response of a T-Rex+GVS run with the elastic net.

Details

Steps:

1. X is centered and its first M ordinary principal components are computed (a PCA of the covariance, the columns are not scaled).
2. For each component, a T-Rex+GVS run with the elastic net and the component as response selects its variables at the target FDR.
3. The sparse loading vector is assembled from the selected set (see mode of trex_spca_control()), and the scores of the sparse component are the centered X times the loading vector.
4. The explained variance is adjusted for components that are not orthogonal, by the QR decomposition of the scores.

A component without a selected variable has a zero column in V and Z and the explained variance 0.

Active bindings

`Z` Scores of the sparse components (n x M): column m is the centered `X` times column m of `V`.

`V` Sparse loadings (p x M), each column of unit L2 norm. In `"ActiveSet"` mode the non-zero entries of column m are the variables of `active_sets[[m]]`. In `"Thresholded"` mode they are the largest ordinary PCA loadings, as many as `active_sets[[m]]` has entries, which can be other variables.

`active_sets` Selected variables per component, a list of M integer vectors (1-based), the result of the T-Rex+GVS runs. The FDR control holds for these sets. In `"Thresholded"` mode they can differ from the non-zero entries of `V`.

`adjusted_ev` Adjusted explained variance per component (length M): `r_mm^2 / (n - 1)`, with `r_mm` the m-th diagonal element of R in the QR decomposition `Z = QR`. The adjustment accounts for sparse components that are not orthogonal.

`cumulative_ev` Cumulative adjusted explained variance as a fraction of the total variance of the centered `X` (length M), between 0 and 1. It is not the PEV of Definition 1 of the paper, whose denominator (signal EV plus mixed EV) needs the true support.

Methods

Public methods:

- `TRexSPCASelector$new()`
- `TRexSPCASelector$select()`
- `TRexSPCASelector$clone()`

`TRexSPCASelector$new()`: Create a new TRexSPCASelector.

Usage:

```
TRexSPCASelector$new(
  X,
  M,
  tFDR,
  control = trex_spca_control(),
  seed = -1L,
  verbose = FALSE
)
```

Arguments:

`X` Design matrix (n x p) in memory, or an `mmap_matrix` opened in `"readwrite"` mode. Both are bound without a copy. `X` is centered in place during `select()` and restored on return.

`M` Number of sparse principal components, 1 to `min(n, p)`.

`tFDR` Target FDR level of each component, in `[0, 1]`.

`control` A control list from `trex_spca_control()` (default: `trex_spca_control()`).

`seed` Random seed. A negative value (default: -1) gives a run that is not reproducible. Each component runs with its own seed derived from it.

`verbose` Whether to print one line per component (default: `FALSE`). The T-Rex+GVS runs stay quiet.

`TRexSPCASelector$select()`: Run the T-Rex sparse PCA selection.

Usage:

```
TRexSPCASelector$select()
```

Returns: Invisible `self` for method chaining. The results are in the fields `Z`, `V`, `active_sets`, `adjusted_ev` and `cumulative_ev`.

`TRexSPCASelector$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TRexSPCASelector$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

J. Machkour, A. Breloy, M. Muma, D. P. Palomar, and F. Pascal, "Sparse PCA with False Discovery Rate Controlled Variable Selection," in Proc. IEEE 49th Int. Conf. Acoust., Speech, Signal Process. (ICASSP), 2024, pp. 9716-9720.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-Rex selectors and controls: `Gauss_data`, `TRexBiobankScreeningSelector`, `TRexDASelector`, `TRexGVSSelector`, `TRexScreeningSelector`, `TRexSelector`, `TRexTikhonovSelector`, `trex_biobank_control()`, `trex_build_group_centering_k()`, `trex_control()`, `trex_da_control()`, `trex_default_lambda2_grid()`, `trex_dummy_distribution()`, `trex_fdp()`, `trex_fdp_dense()`, `trex_gamma_to_k()`, `trex_get_num_threads()`, `trex_gvs_control()`, `trex_gvs_lambda2_power_rule()`, `trex_normalize_design()`, `trex_precision()`, `trex_recall()`, `trex_screen_control()`, `trex_spca_control()`, `trex_tikhonov_control()`, `trex_tikhonov_lambda2_power_rule()`, `trex_tpp()`, `trex_tpp_dense()`

Examples

```
set.seed(1)
n <- 100; p <- 30
X <- matrix(rnorm(n * p), n, p)
sel <- TRexSPCASelector$new(X, M = 2, tFDR = 0.2,
                            control = trex_spca_control(K = 5))
sel$select()
sel$adjusted_ev
sel$active_sets
```

`TRexTikhonovSelector` *T-Rex Selector with a General Tikhonov Penalty (T-Rex+Tikhonov)*

Description

FDR-controlled variable selection with the penalty `(lambda_2 / 2) * t(beta) %*% K %*% beta`, whose positive semi-definite matrix K carries the structure among the variables.

The penalty

`K = t(Gamma) %*% Gamma` (p x p) for a regularization operator Gamma, see `trex_gamma_to_k`. Special cases:

- `K = I`: the ridge part of the elastic net.
- The group-mean matrix, with the block `J / p_m` for every group m of size `p_m` (J is the matrix of ones): the penalty of the informed elastic net of `TRexGVSSelector`.
- The centering matrix, with the block `I - J / p_m` for every group (`penalty_structure = "GROUP_CENTERING"`, `trex_build_group_centering_k`): the members of a group share a coefficient.
- A banded matrix or a graph Laplacian: neighbouring coefficients are smooth.

The solver

`"TIKNET"`, the homotopy solver, is the one solver of the selector. It is the solver of the base control that applies when `control` is left out, and a control that is passed has to name it. Every other solver is refused, also `"TCDTIKNET"`, the deprecated coordinate-descent solver, which the selector accepted up to TRexSelector 2.0.1.

The dummies

By default they are drawn per cluster of correlated variables, as in `TRexGVSSelector` (`cluster_dummies`): the clusters are `prior_groups`, or come from a hierarchical clustering of the correlations (`corr_max`, `hc_linkage`). The field `dummy_clusters` reports them. With an informed K, i.i.d. dummies would be less competitive than correlated null variables, which deforms the exchangeability the FDR calibration relies on. In the penalty every dummy takes the place of the variable it was drawn for (folded coupling, `fold_dummy_coupling`), so dummies and null variables stay exchangeable for every K.

L-loop strategies

With cluster dummies the strategies `"STANDARD"`, `"HCONCAT"`, `"SKIPL"` and `"SEEDED"` of `trex_control()` are supported, and `"SEEDED"` equals `"STANDARD"`. `"PERMUTATION"` and `"PERMUTATION_SEEDED"` are refused, because a row permutation destroys the covariance of the dummies within a cluster. With `cluster_dummies = FALSE` the two run as well.

lambda_2

A supplied value is used as it is. Otherwise `lambda2_method` of `trex_tikhonov_control()` chooses it before the L-loop, on the normalized data:

- `"PILOT_KNEE"` (default), the pilot rule. For every value of `pilot_grid`, `pilot_K` random experiments run with `L_0` = p dummies to the stopping time `pilot_T`, on one set of dummies, and the variables ahead of the dummies in more than half of the experiments are counted (`pilot_statistic`). The pick is the knee of this profile, the smallest lambda_2 whose count is within `pilot_knee_tolerance` of the largest. It costs about the length of the grid times `pilot_K` paths, a fraction of one selection per grid value. The pilot dummies come from a seed stream of their own, so the main run is a T-Rex run at a fixed lambda_2 whose dummies do not depend on the pick. The pick itself depends on y and lies outside the FDR guarantee of a fixed lambda_2. In simulations the pick cost nothing under the global null and at most 0.02 of true positive rate against the best fixed value under the alternatives. The record is the field `lambda2_pilot`, and `trex_tikhonov_lambda2_power_rule()` runs the same rule on full selections.
- `"CV_1SE_IEN_CCD"` and `"CV_MIN_IEN_CCD"`: cross-validation with the lambda_1 path profiled out per lambda_2 candidate, well-posed for every p and n.
- `"CV_1SE_TIK_SVD"` and `"CV_MIN_TIK_SVD"`: analytic ridge cross-validation (lambda_1 = 0), exact for n > p and refused when the unpenalized null space interpolates the training folds.

The scale. A supplied `lambda_2` refers to the scale of the standardized columns: under `scaling_mode` = `"ZSCORE"` the columns have the squared norm `n - 1`, so the same shrinkage needs the value times `n - 1`. `pilot_grid` is stated for columns of unit norm, and the pilot rule and the cross-validation rules apply the factor `n - 1` themselves. Every rule reports its pick on the scale of a supplied value (the field `lambda2_used`).

A selector runs once

`select()` and `run_pilot_only()` exclude each other, and neither runs a second time. Construct a new selector to run again.

Super class

`TRexSelector` -> `TRexTikhonovSelector`

Active bindings

`lambda2_used` The lambda_2 the last run used: the supplied value, or the pick of the pilot rule or of a cross-validation rule, on the scale of a supplied value. `-1` before a run.

`lambda2_cv_diagnostics` Health record of the rule that chose lambda_2 in the last run: a list with `ran` (a cross-validation rule or the pilot ran, `FALSE` when the value was supplied), `seconds`, `n_profile_at_edge` (lambda_2 values whose profiled lambda_1 optimum sat on a truncation edge; 0 is healthy), `support_cap` (-1 = disabled), `dfmax_ratio` (the `cv_dfmax_ratio` the IEN rules were called with; 0 otherwise), `low_dimensional`, and the fields of the boundary-pick guard, `grid_index` and `grid_size`, `at_grid_edge`, `profile_at_edge`, `profile_at_support_cap` and `boundary_pick` (see `TRexGVSSelector`). The cross-validation rules warn once on a boundary pick and return it unchanged. The pilot rule leaves the guard fields at their defaults.

`lambda2_pilot` Record of the lambda_2 pilot of the last `select()` or `run_pilot_only()`, a list:

- `ran` A pilot ran. `FALSE`: lambda_2 came from the caller or from a cross-validation rule.
- `grid` The lambda_2 values the pilot ran at, ascending and without duplicates: `pilot_grid` times `grid_scale`.
- `grid_scale` The factor between `pilot_grid` and `grid`: `n - 1` under `scaling_mode = "ZSCORE"`, 1 otherwise.
- `counts` The count the pick was read off, per grid value: the count of `pilot_statistic`.
- `curve_majority` The majority count per grid value (row) and pilot stopping time 1 to `T_pilot` (column).
- `curve_soft` The mean number of variables ahead of the t-th dummy, per grid value (row) and t (column). A diagnostic.
- `calibrated` The calibrated pilot selection count per grid value.
- `group_majority` The group-majority count per grid value: the members ahead, summed over the groups that reach `pilot_group_fraction`. Zeros without `prior_groups`.
- `groups_qualifying` The number of groups that reach `pilot_group_fraction`, per grid value. Zeros without `prior_groups`.
- `probe_seconds` Wall-clock seconds per grid value.
- `index` Index of the pick into `grid`, counted from 1.
- `lambda2` The picked lambda_2, `grid[index]`, on the working scale.
- `max_selected` The largest count on the grid.
- `n_ties` Grid values within the tolerance of the largest count.
- `at_lower_end` The pick sits on the first grid value.
- `at_upper_end` The pick sits on the last grid value: widen the grid before trusting it.
- `K_pilot` Pilot experiments per grid value that ran: `pilot_K`, capped at `K`.
- `T_pilot` Stopping time of the pilot paths that was used: `pilot_T`, capped at `min(L_0, ceiling(n / 2))`.
- `L_0` Number of pilot dummies, `pilot_L_multiplier` times p.
- `dummy_tag` Seed tag under which the pilot dummies were drawn.
- `seconds` Wall-clock seconds of the whole pilot.

`dummy_clusters` The clusters the dummies of the last run were drawn for, a list with the names of the fields of `TRexGVSSelector`: `groups` (the cluster of each variable, numbered from 1), `max_clusters` (their number M) and `hc_method_used` (`"prior"` when `prior_groups` was supplied, else the linkage of the clustering). Before a run and with `cluster_dummies = FALSE` the record is empty: no groups, 0 and `""`.

Methods

Public methods:

- `TRexTikhonovSelector$new()`
- `TRexTikhonovSelector$select()`
- `TRexTikhonovSelector$run_pilot_only()`
- `TRexTikhonovSelector$clone()`

`TRexTikhonovSelector$new()`: Create a new TRexTikhonovSelector object.

Usage:

```
TRexTikhonovSelector$new(
  X,
  y,
  tFDR = 0.1,
  seed = -1,
  verbose = TRUE,
  tik_control,
  control = NULL
)
```

Arguments:

`X` Feature matrix in memory, or an `mmap_matrix`, the handle of a memory-mapped file. Both are bound without a copy. The selector standardizes a memory-mapped design in place and restores it, which needs `"readwrite"` mode. Under `scaling_mode = "NONE"` it writes nothing and `"readonly"` mode suffices.

`y` Response vector.

`tFDR` Target FDR level between 0 and 1 (default: 0.1).

`seed` Random seed. A negative value (default: -1) gives a run that is not reproducible.

`verbose` Whether to print progress (default: TRUE).

`tik_control` A Tikhonov control list from `trex_tikhonov_control()`. Required, because it carries the penalty matrix `tikhonov_K` or the penalty structure.

`control` A control list from `trex_control()`, or `NULL` (default). With `NULL` the base control of the C++ class applies: the defaults of `trex_control()` with the solver `"TIKNET"`, the pathwise homotopy solver. A control that is passed is taken as it is, so its `solver` must be `"TIKNET"`: `trex_control(solver = "TIKNET", ...)`. Every other solver is an error, also `"TLARS"`, the default of `trex_control()`, and `"TCDTIKNET"`, the deprecated coordinate-descent solver, which the selector accepted up to TRexSelector 2.0.1.

`TRexTikhonovSelector$select()`: Run the T-Rex+Tikhonov selection. A selector runs once: a second call is an error, and so is a call after `run_pilot_only()`, see `TRexSelector`.

Usage:

```
TRexTikhonovSelector$select()
```

`TRexTikhonovSelector$run_pilot_only()`: Run only the lambda_2 pilot, without the selection. The pilot builds the cluster model, runs the pilot experiments and picks lambda_2, and `X` is given back. It is for a caller who wants the pick alone. It needs `lambda_2 < 0` and `lambda2_method = "PILOT_KNEE"` in `tik_control` and is an error otherwise. A selector runs once: after `run_pilot_only()` a call of `select()` or of `run_pilot_only()` is an error, and so is `run_pilot_only()` after `select()`.

Usage:

```
TRexTikhonovSelector$run_pilot_only()
```

Returns: The record of the pilot, as the field `lambda2_pilot` holds it afterwards.

`TRexTikhonovSelector$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TRexTikhonovSelector$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

A. N. Tikhonov, "Solution of Incorrectly Formulated Problems and the Regularization Method," Soviet Math. Dokl., vol. 4, pp. 1035-1038, 1963 (English translation of Dokl. Akad. Nauk SSSR 151 (1963), no. 3, 501-504).

Y. Park, L. Reichel, G. Rodriguez, and X. Yu, "Parameter Determination for Tikhonov Regularization Problems in General Form," J. Comput. Appl. Math., vol. 343, pp. 12-25, 2018, doi: 10.1016/j.cam.2018.04.049.

A. J. Smola and R. Kondor, "Kernels and Regularization on Graphs," in Learning Theory and Kernel Machines (COLT/Kernel 2003), Lecture Notes in Artificial Intelligence vol. 2777, 2003, pp. 144-158.

F. R. K. Chung, Spectral Graph Theory. CBMS Regional Conference Series in Mathematics, no. 92. American Mathematical Society, 1997.

A. E. Hoerl and R. W. Kennard, "Ridge Regression: Biased Estimation for Nonorthogonal Problems," Technometrics, vol. 42, no. 1, pp. 80-86, 2000 (Reprint of the 1970 article in the Special 40th Anniversary Issue).

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-Rex selectors and controls: `Gauss_data`, `TRexBiobankScreeningSelector`, `TRexDASelector`, `TRexGVSSelector`, `TRexSPCASelector`, `TRexScreeningSelector`, `TRexSelector`, `trex_biobank_control()`, `trex_build_group_centering_k()`, `trex_control()`, `trex_da_control()`, `trex_default_lambda2_grid()`, `trex_dummy_distribution()`, `trex_fdp()`, `trex_fdp_dense()`, `trex_gamma_to_k()`, `trex_get_num_threads()`, `trex_gvs_control()`, `trex_gvs_lambda2_power_rule()`, `trex_normalize_design()`, `trex_precision()`, `trex_recall()`, `trex_screen_control()`, `trex_spca_control()`, `trex_tikhonov_control()`, `trex_tikhonov_lambda2_power_rule()`, `trex_tpp()`, `trex_tpp_dense()`

Examples

```
data(Gauss_data)
X <- Gauss_data$X
y <- Gauss_data$y
ctrl <- trex_tikhonov_control(tikhonov_K = diag(ncol(X)), lambda_2 = 0.5)
sel <- TRexTikhonovSelector$new(X, y, tFDR = 0.1, verbose = FALSE,
                                tik_control = ctrl)
sel$select()
sel$selected_indices
```

`tsolver_options` *T-Solver Options*

Description

Every T-solver input except the data `X`, `D` and `y`, as one list (the C++ `SolverOptions`). Pass it to a solver's `$new(X, D, y, options = ...)`. Each solver reads the fields it needs and ignores the rest; the defaults are the solver-level defaults.

Usage

```
tsolver_options(
  normalize = TRUE,
  intercept = TRUE,
  precentered = FALSE,
  scaling_mode = c("L2", "ZSCORE"),
  verbose = FALSE,
  tol = .Machine$double.eps,
  tie_seed = 0,
  n_threads = 1,
  exch_tie_alpha = 0,
  exch_tie_floor = 0.5,
  lambda2 = 0.1,
  rho_afs = 0.3,
  ncgmp_variant = c("line_search", "fully_corrective"),
  tenet_aug_use_lars = FALSE,
  tikhonov_validated = FALSE,
  kappa_dummy = 1,
  fold_dummies = FALSE,
  group_penalty = c("MEAN", "CENTERING"),
  cd_lambda_rel_tol = 0.001,
  cd_tol_certify = 1e-10,
  cd_tol_probe = 1e-08,
  cd_gram_cap = 400,
  cd_max_sweeps = 2000,
  cd_certified_localization = TRUE
)
```

Arguments

`normalize`	Logical; scale each column of X and D to unit L2 norm (or unit SD, see `scaling_mode`). `FALSE` when the caller pre-scaled (see the in-place note below). Default `TRUE`.
`intercept`	Logical; center X, D and y (an intercept is fitted). `FALSE` when the caller pre-centered. Default `TRUE`.
`precentered`	Logical; the caller centered X, D and y (`intercept = FALSE` on centered data, as in the T-Rex selectors). The centered columns span at most n - 1 dimensions, so the solver counts the intercept as with `intercept = TRUE`: the active set is capped at n - 1 and the degrees of freedom and Cp include the intercept. The data are not touched. Redundant when `intercept` is `TRUE`. Default `FALSE`.
`scaling_mode`	`"L2"` (default) or `"ZSCORE"`: the column scaling applied when `normalize` is `TRUE`.

`verbose` Logical; print status and diagnostics. Default `FALSE`.

`tol` Numerical tolerance of the solver's internal comparisons. Default: machine epsilon. A selector sets its own tolerance on the solvers it runs, `tol` of `trex_control` (1e-6 by default).

`tie_seed` Seed of the solver's tie-break engine. Default 0: a fixed seed, so a solver run is reproducible also where exact ties occur; a negative value seeds from the system entropy source (nondeterministic).

`n_threads` Thread budget of the solver's own passes over the design (the correlation refreshes, the step-length search, the standardization). Default 1 (serial); a whole number of at least 1. A pass splits only above about a million entries of the design, and never inside a parallel region of the caller. The T-Rex selectors set it from `trex_control(max_inner_threads = )`.

`exch_tie_alpha` Exchangeable-tie band width of the greedy solvers (TOMP, TOOLS, TGP, TACGP, TMP, TNCGMP and TAFS), in units of the pairwise ranking-noise sd; 0 (default) turns it off. The path solvers ignore it.

`exch_tie_floor` Minimum absolute correlation for exchangeable-tie candidates, in [0, 1). Default 0.5.

`lambda2` L2 (ridge / group-ridge) penalty weight, >= 0. Read by TENET, TENETAug and TIENETAug. Default 0.1, as inside the selectors. With 0 there is no quadratic penalty and the elastic nets reduce to their lasso.

`rho_afs` TAFS shrinkage step in (0, 1]. Default 0.3, as inside the selectors; 1 is forward stepwise.

`ncgmp_variant` TNCGMP variant: `"line_search"` (default) or `"fully_corrective"`.

`tenet_aug_use_lars`
Logical; TENETAug and TIENETAug run the augmented system with a pure-LARS inner path that never drops variables (default `FALSE`: LASSO inner path).

`tikhonov_validated`
Logical; TIKNET and TCDTIKNET. `TRUE` says that the Tikhonov matrix was checked by the caller (finite entries, symmetric, positive semi-definite), and the solver skips its own check. For a caller that builds many solvers on one matrix, as the T-Rex selector does. Default `FALSE`.

`kappa_dummy` Ridge weight of the dummy block of TCDENET, and of TCDTIKNET with `fold_dummies = FALSE`. Default 1. Deprecated: it will be removed with the coordinate-descent family in TRexSelector 2.1.0.

`fold_dummies` Logical; TCDTIKNET dummy coupling. Default `FALSE`. Deprecated: it will be removed with the coordinate-descent family in TRexSelector 2.1.0.

`group_penalty` TCDIENET structured penalty, `"MEAN"` (default) or `"CENTERING"`. Deprecated: it will be removed with the coordinate-descent family in TRexSelector 2.1.0.

`cd_lambda_rel_tol`, `cd_tol_certify`, `cd_tol_probe`, `cd_gram_cap`, `cd_max_sweeps`, `cd_certified_localization`
Knobs of the coordinate-descent solver family; the defaults are the solver defaults (1e-3, 1e-10, 1e-8, 400, 2000, `TRUE`). A solve that stops at the sweep budget `cd_max_sweeps` far from a solution (KKT violation above 0.001 lambda_1) raises an error instead of continuing on an unconverged path. Deprecated: the six knobs will be removed with the coordinate-descent family in TRexSelector 2.1.0.

Value

A named list suitable for the `options` argument of every T-solver constructor.

In-place preprocessing (the writer/reader contract)

The T-solvers work on the caller's matrices without copying, in R as in C++ and Python. With `normalize = TRUE` or `intercept = TRUE` a solver is a *writer*: it centers / scales the columns of X and D in place (visible on the caller's matrix or memory-mapped file), keeps the statistics itself, and holds a claim on both until `$restore()` or garbage collection, both of which write X and D back; a second writer (T-solver or T-Rex selector) on a claimed matrix is an error, because a second in-place transformation moves the data by rounding. With `normalize = FALSE` and `intercept = FALSE` a solver is a *reader* that runs as constructed and never writes; a pipeline with several solvers on one matrix pre-scales once in front (MLLpNormScaler) and constructs readers. y is never written. See TSolver_Base.

See Also

Other T-solvers: TACGP_Solver, TAFS_Solver, TCDENET_Solver, TCDIENET_Solver, TCDLASSO_Solver, TCDTIKNET_Solver, TENETAug_Solver, TENET_Solver, TGP_Solver, TIENETAug_Solver, TIENET_Solver, TIKNET_Solver, TLARS_Solver, TLASSO_Solver, TMP_Solver, TNCGMP_Solver, TOMP_Solver, TOOLS_Solver, TSTAGEWISE_Solver, TSTEPWISE_Solver

Examples

```
tsolver_options()
tsolver_options(lambda2 = 0.5, normalize = FALSE)
```

TSTAGEWISE_Solver *T-Stagewise Solver*

Description

Terminating Stagewise Regression solver. Inherits all common methods from TSolver_Base.

Super class

TSolver_Base -> TSTAGEWISE_Solver

Methods

Public methods:

- TSTAGEWISE_Solver$new()
- TSTAGEWISE_Solver$get_num_removals()
- TSTAGEWISE_Solver$get_cycling_ratio()
- TSTAGEWISE_Solver$get_lambda()
- TSTAGEWISE_Solver$clone()

`TSTAGEWISE_Solver$new()`: Create a new `TSTAGEWISE_Solver`.

Usage:

```
TSTAGEWISE_Solver$new(X, D, y, options = tsolver_options())
```

Arguments:

`X` Numeric matrix of predictors in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`D` Numeric matrix of dummies in memory, or `mmap_matrix` (the handle of a memory-mapped file).

`y` Numeric response vector.

`options` Solver options from `tsolver_options` (default: `tsolver_options()`).

`TSTAGEWISE_Solver$get_num_removals()`: Get the number of coefficient removals.

Usage:

```
TSTAGEWISE_Solver$get_num_removals()
```

Returns: Integer count of removals.

`TSTAGEWISE_Solver$get_cycling_ratio()`: Get the cycling ratio (removals / total steps).

Usage:

```
TSTAGEWISE_Solver$get_cycling_ratio()
```

Returns: Scalar numeric cycling ratio.

`TSTAGEWISE_Solver$get_lambda()`: Get the regularization (lambda) path.

Usage:

```
TSTAGEWISE_Solver$get_lambda()
```

Returns: Numeric vector of lambda values, one per step.

`TSTAGEWISE_Solver$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TSTAGEWISE_Solver$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least Angle Regression," Ann. Statist., vol. 32, no. 2, pp. 407-499, 2004.

T. Hastie, J. Taylor, R. Tibshirani, and G. Walther, "Forward Stagewise Regression and the Monotone Lasso," Electron. J. Stat., vol. 1, pp. 1-29, 2007, doi: 10.1214/07-EJS004.

R. J. Tibshirani, "A General Framework for Fast Stagewise Algorithms," J. Mach. Learn. Res., vol. 16, no. 78, pp. 2543-2588, 2015.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-solvers: TACGP_Solver, TAFS_Solver, TCDENET_Solver, TCDIENET_Solver, TCDLASSO_Solver, TCDTIKNET_Solver, TENETAug_Solver, TENET_Solver, TGP_Solver, TIENETAug_Solver, TIENET_Solver, TIKNET_Solver, TLARS_Solver, TLASSO_Solver, TMP_Solver, TNCGMP_Solver, TOMP_Solver, TOOLS_Solver, TSTEPWISE_Solver, tsolver_options()

Examples

```
set.seed(1)
n <- 20; p <- 5
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TSTAGEWISE_Solver$new(X, D, y)
solver$execute_step(T_stop = 3)
solver$get_beta()
```

TSTEPWISE_Solver *T-Stepwise Solver*

Description

Terminating Forward Stepwise Regression solver. Inherits all common methods from TSolver_Base.

Super class

TSolver_Base -> TSTEPWISE_Solver

Methods

Public methods:

- TSTEPWISE_Solver$new()
- TSTEPWISE_Solver$get_lambda()
- TSTEPWISE_Solver$clone()

TSTEPWISE_Solver$new(): Create a new TSTEPWISE_Solver.

Usage:

```
TSTEPWISE_Solver$new(X, D, y, options = tsolver_options())
```

Arguments:

X Numeric matrix of predictors in memory, or mmap_matrix (the handle of a memory-mapped file).

D Numeric matrix of dummies in memory, or mmap_matrix (the handle of a memory-mapped file).

y Numeric response vector.

options Solver options from tsolver_options (default: tsolver_options()).

`TSTEPWISE_Solver$get_lambda()`: Get the regularization (lambda) path.

Usage:

```
TSTEPWISE_Solver$get_lambda()
```

Returns: Numeric vector of lambda values, one per step.

`TSTEPWISE_Solver$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TSTEPWISE_Solver$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least Angle Regression," Ann. Statist., vol. 32, no. 2, pp. 407-499, 2004.

J. Machkour, M. Muma, and D. P. Palomar, "The Terminating-Random Experiments Selector: Fast High-Dimensional Variable Selection with False Discovery Rate Control," Signal Process., vol. 231, Art. no. 109894, 2025, doi: 10.1016/j.sigpro.2025.109894.

See Also

Other T-solvers: TACGP_Solver, TAFS_Solver, TCDENET_Solver, TCDIENET_Solver, TCDLASSO_Solver, TCDTIKNET_Solver, TENETAug_Solver, TENET_Solver, TGP_Solver, TIENETAug_Solver, TIENET_Solver, TIKNET_Solver, TLARS_Solver, TLASSO_Solver, TMP_Solver, TNCGMP_Solver, TOMP_Solver, TOOLS_Solver, TSTAGEWISE_Solver, tsolver_options()

Examples

```
set.seed(1)
n <- 20; p <- 5
X <- matrix(rnorm(n * p), n, p)
D <- matrix(rnorm(n * p), n, p)
y <- X[, 1] * 2 + rnorm(n)
solver <- TSTEPWISE_Solver$new(X, D, y)
solver$execute_step(T_stop = 3)
solver$get_beta()
```

Index

∗ **ML methods**
ml_agglomerative_cluster, 9
ml_cut_tree, 10
ml_distance_metric, 11
ml_linkage_method, 12
MLElasticNet, 12
MLElasticNetCV, 16
MLLpNormScaler, 19
MLPCA, 23
MLRidge, 25
MLRidgeCV, 27
MLSVD, 30
MLZScoreScaler, 31
∗ **T-Rex selectors and controls**
Gauss_data, 6
trex_biobank_control, 65
trex_build_group_centering_k, 66
trex_control, 67
trex_da_control, 72
trex_default_lambda2_grid, 73
trex_dummy_distribution, 74
trex_fdp, 76
trex_fdp_dense, 77
trex_gamma_to_k, 77
trex_get_num_threads, 78
trex_gvs_control, 79
trex_gvs_lambda2_power_rule, 82
trex_normalize_design, 84
trex_precision, 85
trex_recall, 86
trex_screen_control, 87
trex_spca_control, 88
trex_tikhonov_control, 90
trex_tikhonov_lambda2_power_rule, 94
trex_tpp, 96
trex_tpp_dense, 97
TRexBiobankScreeningSelector, 97
TRexDASelector, 100
TRexGVSSelector, 104
TRexScreeningSelector, 106
TRexSelector, 109
TRexSPCASelector, 112
TRexTikhonovSelector, 115
∗ **T-solvers**
TACGP_Solver, 35
TAFS_Solver, 36
TCDENET_Solver, 38
TCDIENET_Solver, 39
TCDLASSO_Solver, 41
TCDTIKNET_Solver, 43
TENET_Solver, 45
TENETAug_Solver, 47
TGP_Solver, 48
TIENET_Solver, 50
TIENETAug_Solver, 52
TIKNET_Solver, 54
TLARS_Solver, 56
TLASSO_Solver, 58
TMP_Solver, 60
TNCGMP_Solver, 61
TOMP_Solver, 62
TOOLS_Solver, 64
tsolver_options, 119
TSTAGEWISE_Solver, 122
TSTEPWISE_Solver, 124
∗ **datasets**
Gauss_data, 6
∗ **memory-mapped matrices**
[.mmap_matrix, 3
as.matrix.mmap_matrix, 4
dim.mmap_matrix, 5
mem_matrix, 7
mem_matrix_from, 8
print.mmap_matrix, 34
[.mmap_matrix, 3
[.mmap_matrix(), *5*, *8*, *35*
[<-.mmap_matrix *(*[.mmap_matrix*)*, 3

as.matrix.mmap_matrix, 4
as.matrix.mmap_matrix(), *4, 5*, *8*, *35*

dim.mmap_matrix, 5
dim.mmap_matrix(), *4, 5*, *8*, *35*

Gauss_data, 6, *66, 67*, *71*, *73–79*, *81*, *83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, *103*, *106*, *109*, *112*, *114*, *119*

mem_matrix, 7, *19*, *28*, *31*
mem_matrix(), *4, 5*, *8*, *35*
mem_matrix_from, 8, *9*, *19*, *28*, *31*
mem_matrix_from(), *4, 5*, *8*, *35*
ml_agglomerative_cluster, 9
ml_agglomerative_cluster(), *11, 12*, *16*, *19*, *22*, *25, 26*, *29*, *31*, *34*
ml_cut_tree, 10
ml_cut_tree(), *10–12*, *16*, *19*, *22*, *25, 26*, *29*, *31*, *34*
ml_distance_metric, *10*, 11, *11, 12*, *16*, *19*, *22*, *25, 26*, *29*, *31*, *34*
ml_linkage_method, *10, 11*, 12, *16*, *19*, *22*, *25, 26*, *29*, *31*, *34*
MLElasticNet, *10, 11*, 12, *12*, *16*, *19*, *22*, *25, 26*, *29*, *31*, *34*
MLElasticNetCV, *10–12*, 16, *16*, *22*, *25, 26*, *29*, *31*, *34*
MLLpNormScaler, *10–12*, *16*, 19, *19*, *25, 26*, *29*, *31*, *34*, *122*
MLPCA, *10–12*, *16*, *19*, *22*, 23, *26*, *29*, *31*, *34*
MLRidge, *10–12*, *16*, *19*, *22*, 25, *25*, *29*, *31*, *34*
MLRidgeCV, *10–12*, *16*, *19*, *22*, *25, 26*, 27, *31*, *34*
MLSVD, *10–12*, *16*, *19*, *22*, *25, 26*, *29*, 30, *34*
MLZScoreScaler, *10–12*, *16*, *19*, *22*, *25, 26*, *29*, 31, *31*

print.mmap_matrix, 34
print.mmap_matrix(), *4, 5*, *8*

scale, *19*, *31*

TACGP_Solver, 35, *37*, *39*, *41, 42*, *44*, *46*, *48, 49*, *51*, *53*, *56, 57*, *59*, *61–63*, *65*, *122*, *124, 125*
TAFS_Solver, 36, *36*, *39*, *41, 42*, *44*, *46*, *48, 49*, *51*, *53*, *56, 57*, *59*, *61–63*, *65*, *122*, *124, 125*
TCDENET_Solver, *36, 37*, 38, *41, 42*, *44*, *46*, *48, 49*, *51*, *53*, *56, 57*, *59*, *61–63*, *65*, *122*, *124, 125*
TCDIENET_Solver, *36, 37*, 39, *39*, *42*, *44*, *46*, *48, 49*, *51*, *53*, *56, 57*, *59*, *61–63*, *65*, *122*, *124, 125*
TCDLASSO_Solver, *36, 37*, *39*, 41, *41*, *44*, *46*, *48, 49*, *51*, *53*, *56, 57*, *59*, *61–63*, *65*, *122*, *124, 125*
TCDSolver_Base, *38*, *40*, *42, 43*
TCDTIKNET_Solver, *36, 37*, *39*, *41, 42*, 43, *46*, *48, 49*, *51*, *53*, *56, 57*, *59*, *61–63*, *65*, *122*, *124, 125*
TENET_Solver, *36–39*, *41, 42*, *44*, 45, *48, 49*, *51*, *53*, *56, 57*, *59*, *61–63*, *65*, *122*, *124, 125*
TENETAug_Solver, *36, 37*, *39*, *41, 42*, *44*, *46*, 47, *49*, *51*, *53*, *56, 57*, *59*, *61–63*, *65*, *122*, *124, 125*
TGP_Solver, *36, 37*, *39*, *41, 42*, *44*, *46*, 48, *48*, *51*, *53*, *56, 57*, *59*, *61–63*, *65*, *122*, *124, 125*
TIENET_Solver, *36, 37*, *39–42*, *44*, *46*, *48, 49*, 50, *53*, *56, 57*, *59*, *61–63*, *65*, *122*, *124, 125*
TIENETAug_Solver, *36, 37*, *39*, *41, 42*, *44*, *46*, *48, 49*, *51*, 52, *56, 57*, *59*, *61–63*, *65*, *122*, *124, 125*
TIKNET_Solver, *36, 37*, *39*, *41–44*, *46*, *48, 49*, *51*, *53*, 54, *57*, *59*, *61–63*, *65*, *122*, *124, 125*
TLARS_Solver, *36, 37*, *39*, *41, 42*, *44*, *46*, *48, 49*, *51*, *53*, 56, *56*, *59*, *61–63*, *65*, *122*, *124, 125*
TLASSO_Solver, *36, 37*, *39*, *41, 42*, *44*, *46*, *48, 49*, *51*, *53*, *56, 57*, 58, *61–63*, *65*, *122*, *124, 125*
TMP_Solver, *36, 37*, *39*, *41, 42*, *44*, *46*, *48, 49*, *51*, *53*, *56, 57*, *59*, 60, *62, 63*, *65*, *122*, *124, 125*
TNCGMP_Solver, *36, 37*, *39*, *41, 42*, *44*, *46*, *48, 49*, *51*, *53*, *56, 57*, *59*, 61, *61*, *63*, *65*, *122*, *124, 125*
TOMP_Solver, *36, 37*, *39*, *41, 42*, *44*, *46*, *48, 49*, *51*, *53*, *56, 57*, *59*, *61*, 62, *62*, *65*, *122*, *124, 125*
TOOLS_Solver, *36, 37*, *39*, *41, 42*, *44*, *46*, *48, 49*, *51*, *53*, *56, 57*, *59*, *61–63*, 64,

122, *124, 125*
trex_biobank_control, 65, *98*
trex_biobank_control(), *6*, *67*, *71*, *73–79*, *81*, *83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, *103*, *106*, *109*, *112*, *114*, *119*
trex_build_group_centering_k, 66, *91*, *115*
trex_build_group_centering_k(), *6*, *66*, *71*, *73–79*, *81*, *83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, *103*, *106*, *109*, *112*, *114*, *119*
trex_control, 67, *74, 75*, *79*, *83*, *88*, *92*, *94*, *99*, *102*, *104, 105*, *107, 108*, *111*, *115*, *118*, *121*
trex_control(), *6*, *66, 67*, *73–79*, *81*, *83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, *103*, *106*, *109*, *112*, *114*, *119*
trex_da_control, 72, *102*
trex_da_control(), *6*, *66, 67*, *71*, *74–79*, *81*, *83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, *103*, *106*, *109*, *112*, *114*, *119*
trex_default_lambda2_grid, 73, *83*, *92*, *95*
trex_default_lambda2_grid(), *6*, *66, 67*, *71*, *73*, *75–79*, *81*, *83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, *103*, *106*, *109*, *112*, *114*, *119*
trex_dummy_distribution, *70*, 74
trex_dummy_distribution(), *6*, *66, 67*, *71*, *73, 74*, *76–79*, *81*, *83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, *103*, *106*, *109*, *112*, *114*, *119*
trex_fdp, 76
trex_fdp(), *6*, *66, 67*, *71*, *73–75*, *77–79*, *81*, *83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, *103*, *106*, *109*, *112*, *114*, *119*
trex_fdp_dense, 77
trex_fdp_dense(), *6*, *66, 67*, *71*, *73–76*, *78, 79*, *81*, *83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, *103*, *106*, *109*, *112*, *114*, *119*
trex_gamma_to_k, *54*, 77, *91*, *115*
trex_gamma_to_k(), *6*, *66, 67*, *71*, *73–77*, *79*, *82, 83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, *103*, *106*, *109*, *112*, *114*, *119*
trex_get_max_threads *(*trex_get_num_threads*)*, 78
trex_get_num_threads, *9*, *13*, *17*, *20*, *23*, *26, 27*, *30*, *32*, 78, *84*
trex_get_num_threads(), *6*, *66, 67*, *71*, *73–78*, *82, 83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, *103*, *106*, *109*, *112*, *114*, *119*
trex_gvs_control, 79, *83*, *88*, *105*
trex_gvs_control(), *6*, *66, 67*, *71*, *73–79*, *83*, *85, 86*, *88*, *90*, *93*, *95–97*, *100*, *103*, *106*, *109*, *112*, *114*, *119*
trex_gvs_lambda2_power_rule, *73*, 82, *94*
trex_gvs_lambda2_power_rule(), *6*, *66, 67*, *71*, *73–79*, *82*, *85, 86*, *88*, *90*, *93*, *95–97*, *100*, *103*, *106*, *109*, *112*, *114*, *119*
trex_normalize_design, *7*, *71*, *78*, 84
trex_normalize_design(), *6*, *66, 67*, *71*, *73–79*, *82, 83*, *85, 86*, *88*, *90*, *93*, *95–97*, *100*, *103*, *106*, *109*, *112*, *114*, *119*
trex_precision, 85
trex_precision(), *6*, *66, 67*, *71*, *73–79*, *82, 83*, *85, 86*, *88*, *90*, *93*, *95–97*, *100*, *103*, *106*, *109*, *112*, *114*, *119*
trex_recall, 86
trex_recall(), *6*, *66, 67*, *71*, *73–79*, *82, 83*, *85*, *88*, *90*, *93*, *95–97*, *100*, *103*, *106*, *109*, *112*, *114*, *119*
trex_screen_control, 87, *98*, *108*
trex_screen_control(), *6*, *66, 67*, *71*, *73–79*, *82, 83*, *85, 86*, *90*, *93*, *95–97*, *100*, *103*, *106*, *109*, *112*, *114*, *119*
trex_set_num_threads *(*trex_get_num_threads*)*, 78
trex_spca_control, 88, *112, 113*
trex_spca_control(), *6*, *66, 67*, *71*, *73–79*, *82, 83*, *85, 86*, *88*, *93*, *95–97*, *100*, *103*, *106*, *109*, *112*, *114*, *119*
trex_tikhonov_control, *66*, *73*, *77*, 90, *94*, *116*, *118*
trex_tikhonov_control(), *6*, *66, 67*, *71*, *73–79*, *82, 83*, *85, 86*, *88*, *90*, *95–97*, *100*, *103*, *106*, *109*, *112*, *114*, *119*
trex_tikhonov_lambda2_power_rule, *73*, *82, 83*, *91*, 94, *116*
trex_tikhonov_lambda2_power_rule(), *6*, *66, 67*, *71*, *73–79*, *82, 83*, *85, 86*, *88*, *90*, *93*, *96, 97*, *100*, *103*, *106*, *109*, *112*, *114*, *119*
trex_tpp, 96
trex_tpp(), *6*, *66, 67*, *71*, *73–79*, *82, 83*, *85*,

86, *88*, *90*, *93*, *95*, *97*, *100*, *103*, *106*, *109*, *112*, *114*, *119*
trex_tpp_dense, 97
trex_tpp_dense(), *6*, *66, 67*, *71*, *73–79*, *82, 83*, *85, 86*, *88*, *90*, *93*, *95, 96*, *100*, *103*, *106*, *109*, *112*, *114*, *119*
TRexBiobankScreeningSelector, *6*, *66, 67*, *71*, *73–79*, *81*, *83*, *85, 86*, *88*, *90*, *93*, *95, 96*, 97, *97*, *103*, *106, 107*, *109*, *112*, *114*, *119*
TRexDASelector, *6*, *66, 67*, *71*, *73–79*, *81*, *83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, 100, *106*, *109*, *112*, *114*, *119*
TRexGVSSelector, *6*, *66, 67*, *71*, *73–79*, *81–83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, *103*, 104, *109*, *112*, *114–117*, *119*
TRexScreeningSelector, *6*, *66, 67*, *71*, *73–79*, *81*, *83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, *103*, 106, *106*, *112*, *114*, *119*
TRexSelector, *6*, *66, 67*, *71*, *73–79*, *81*, *83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, *101–108*, 109, *109*, *114*, *116*, *118, 119*
TRexSPCASelector, *6*, *66, 67*, *71*, *73–79*, *81*, *83*, *85, 86*, *88*, *90*, *93*, *95–97*, *99*, *103*, *106*, *109*, 112, *112*, *119*
TRexTikhonovSelector, *6*, *66, 67*, *71*, *73–79*, *81*, *83*, *85, 86*, *88*, *90*, *93–97*, *99*, *103*, *106*, *109*, *112*, *114*, 115
TSolver_Base, *35–38*, *40–43*, *45*, *47, 48*, *50*, *52*, *54*, *56*, *58*, *60–62*, *64*, *122*, *124*
tsolver_options, *36–38*, *40*, *42*, *44, 45*, *47*, *49, 50*, *53*, *55*, *57, 58*, *60*, *62–64*, *69*, 119, *123, 124*
tsolver_options(), *36, 37*, *39*, *41, 42*, *44*, *46*, *48, 49*, *51*, *53*, *56, 57*, *59*, *61–63*, *65*, *124, 125*
TSTAGEWISE_Solver, *36, 37*, *39*, *41, 42*, *44*, *46*, *48, 49*, *51*, *53*, *56, 57*, *59*, *61–63*, *65*, 122, *122*, *125*
TSTEPWISE_Solver, *36, 37*, *39*, *41, 42*, *44*, *46*, *48, 49*, *51*, *53*, *56, 57*, *59*, *61–63*, *65*, *122*, 124, *124*