Package 'HMC'

May 2, 2025

Title High-Dimensional Mean Comparison with Projection and Cross-Fitting

Version 1.2

Date 2025-05-02

Description Provides interpretable high-dimensional mean comparison methods (HMC). For example, users can apply these methods to assess the difference in gene expression between two treatment groups. It is not a gene-by-gene comparison. Instead, the methods focus on the interplay between features and identify those that are predictive of the group label. The tests are valid frequentist procedures and yield sparse estimates indicating which features contribute to the group differences.

License GPL-2

Encoding UTF-8

RoxygenNote 7.3.2

Imports glmnet, irlba, PMA, MASS, stats, grpreg

URL https://github.com/terrytianyuzhang/HMC/tree/main/HMC_package

NeedsCompilation no

Author Tianyu Zhang [aut, cre, cph]

Maintainer Tianyu Zhang <tianyuz3@andrew.cmu.edu>

Repository CRAN

Date/Publication 2025-05-02 17:00:02 UTC

Contents

anchored_lasso_testing 2
check_data_for_folds 4
check_non_null_and_identical_colnames 4
collect_active_features_proj 5
combine_folds_mean_diff 6
compute_predictive_contributions 6
debiased_pc_testing 7

estimate_leading_pc 9
estimate_nuisance_parameter_lasso 9
estimate_nuisance_pc 11
evaluate_influence_function_multi_factor 12
evaluate_pca_lasso_plug_in 13
evaluate_pca_plug_in 14
extract_lasso_coef 15
extract_pc 15
fit_lasso 16
index_spliter 17
mean_comparison_anchor 17
normalize_and_split 19
process_fold_mean_diff 20
simple_pc_testing 21
summarize_feature_name 22
summarize_pc_name 23
validate_and_convert_data 24

Index **25**

`anchored_lasso_testing` *Anchored test for two-sample mean comparison.*

Description

Anchored test for two-sample mean comparison.

Usage

```
anchored_lasso_testing(
  sample_1,
  sample_2,
  pca_method = "sparse_pca",
  mean_method = "lasso",
  lasso_tuning_method = "min",
  num_latent_factor = 1,
  n_folds = 5,
  verbose = TRUE
)
```

Arguments

`sample_1`	Group 1 sample. Each row is a subject and each column corresponds to a feature.
`sample_2`	Group 2 sample. Each row is a subject and each column corresponds to a feature.
`pca_method`	Methods used to estimate principle component The default is "sparse_pca", using sparse PCA from package PMA. Other choices are "dense_pca"—the regular PCA; and "hard"— hard-thresholding PCA, which also induces sparsity.

`mean_method` Methods used to estimate the discriminant direction. Default is logistic Lasso "lasso". Can also take value "lasso_no_truncation"

`lasso_tuning_method`
Method for Lasso penalty hyperparameter tuning. Default is "min", the minimizer of cross-validation error; users can also use "1se" for more sparse solutions.

`num_latent_factor`
The principle component that lasso coefficient anchors at. The default is PC1 = 1.

`n_folds` Number of splits when performing cross-fitting. The default is 5, if computational time allows, you can try to set it to 10.

`verbose` Print information to the console. Default is TRUE.

Value

A list of test statistics.

`test_statistics`
Test statistics. Each entry corresponds to the test result of one principle component.

`standard_error` Estimated standard error of test_statistics_before_studentization.

`test_statistics_before_studentization`
Similar to test_statistics but does not have variance = 1.

`split_data` Intermediate quantities needed for further assessment and interpretation of the test results.

Examples

```
sample_size_1 <- sample_size_2 <- 300
true_mean_1 <- matrix(c(rep(1, 10), rep(0, 90)), ncol = 1)
true_mean_2 <- matrix(c(rep(1.5, 10), rep(0, 90)), ncol = 1)

sample_1 <- data.frame(MASS::mvrnorm(sample_size_1,
                               mu = true_mean_1,
                               Sigma = diag(1, 100)))
 sample_2 <- data.frame(MASS::mvrnorm(sample_size_2,
                               mu = true_mean_2,
                               Sigma = diag(1, 100)))
 result <- anchored_lasso_testing(sample_1, sample_2)
 result$test_statistics
 ##the test statistic. It should follow normal(0,1) when there is no difference between the groups.
 summarize_feature_name(result)
 #summarize which features contribute to discriminant vectors (i.e. logistic lasso)
 extract_pc(result) # extract the estimated discriminant coefficients
```

check_data_for_folds *Check that data has enough rows for cross-validation folds*

Description

Validates that the input data has at least as many rows as the number of desired folds.

Usage

```
check_data_for_folds(data, n_folds)
```

Arguments

data	A data frame or matrix.
n_folds	Integer. The number of folds to check for.

Value

NULL (called for its side effect). Throws an error if the number of rows is too small.

Examples

```
check_data_for_folds(matrix(1:20, nrow = 5), n_folds = 5)
## Not run:
check_data_for_folds(matrix(1:4, nrow = 2), n_folds = 5)  # This will throw an error

## End(Not run)
```

check_non_null_and_identical_colnames *Check non-null and consistent column names across datasets*

Description

Ensures all input datasets have non-null, non-empty, and identical column names.

Usage

```
check_non_null_and_identical_colnames(data_list)
```

Arguments

data_list	A list of matrices or data frames to be checked.

Value

NULL (called for side-effect). Throws an error if validation fails.

Examples

```
d1 <- data.frame(a = 1:2, b = 3:4)
d2 <- data.frame(a = 5:6, b = 7:8)
check_non_null_and_identical_colnames(list(d1, d2))
```

`collect_active_features_proj` *Collect active features and groups based on projection directions*

Description

Identifies consistently non-zero features across cross-validation folds using a voting scheme and returns active groups if a grouping vector is provided.

Usage

```
collect_active_features_proj(
  test_result,
  voting_method = c("majority_voting"),
  group = NULL,
  group_threshold = 1
)
```

Arguments

`test_result`	A result object from `mean_comparison_anchor()` containing `fold_data`.
`voting_method`	Character. Method to determine active features. Only `"majority_voting"` is currently supported.
`group`	Optional grouping vector with feature names. Must match the feature dimension of `classifier_coef`.
`group_threshold`	Integer. Minimum number of active features required to declare a group active. Default is 1.

Value

If `group` is provided, returns a list with:

active_features Character vector of consistently non-zero features.

active_groups Character vector of active groups.

If `group` is NULL, returns a character vector of active features only.

`combine_folds_mean_diff` *Combine fold-level test statistics from cross-validation*

Description

Aggregates fold-level test statistics and variances to compute an overall test statistic and p-value.

Usage

```
combine_folds_mean_diff(fold_data, verbose = FALSE)
```

Arguments

`fold_data`	A list of results from `process_fold_mean_diff()`, one for each fold.
`verbose`	Logical. Whether to print diagnostic messages. Default is FALSE.

Value

A list containing:

p_value Two-sided p-value for the overall test statistic.

test_statistic Standardized test statistic.

fold_data Original input list, for reference or diagnostics.

`compute_predictive_contributions` *Compute predictive contributions of feature groups*

Description

Analyzes the relative contribution of grouped features to the overall discriminant signal, based on averaged Lasso coefficients across cross-validation folds.

Usage

```
compute_predictive_contributions(result, group, group_threshold = 5)
```

Arguments

`result`	A result object returned by `mean_comparison_anchor()`, containing `fold_data` with classifier coefficients.
`group`	A grouping vector indicating group membership of features. Must be the same length as the number of features.
`group_threshold`	Integer. Minimum number of active features required in a group for it to be considered active. Default is 5.

Details

The function identifies active groups based on cross-validated non-zero coefficients, then decomposes the total L2 norm of the average coefficient vector across groups.

Value

A data frame with two columns:

group Group name or label.

score Proportion of total predictive signal attributable to that group.

See Also

collect_active_features_proj

`debiased_pc_testing` *Debiased one-step test for two-sample mean comparison. A small p-value tells us not only there is difference in the mean vectors, but can also indicates which principle component the difference aligns with.*

Description

Debiased one-step test for two-sample mean comparison. A small p-value tells us not only there is difference in the mean vectors, but can also indicates which principle component the difference aligns with.

Usage

```
debiased_pc_testing(
  sample_1,
  sample_2 = NULL,
  pca_method = "sparse_pca",
  mean_method = "naive",
  num_latent_factor = 1,
  n_folds = 5,
  verbose = TRUE
)
```

Arguments

`sample_1`	Group 1 sample. Each row is a subject and each column corresponds to a feature.
`sample_2`	Group 2 sample. Each row is a subject and each column corresponds to a feature.
`pca_method`	Methods used to estimate principle component The default is "sparse_pca", using sparse PCA from package PMA. Other choices are "dense_pca"—the regular PCA; and "hard"— hard-thresholding PCA, which also induces sparsity.

`mean_method` Methods used to estimate the mean vector. Default is sample mean "naive". There is also a hard-thresholding sparse estiamtor "hard".

`num_latent_factor` Number of principle to be estimated/tested. Default is 1.

`n_folds` Number of splits when performing cross-fitting. The default is 5, if computational time allows, you can try to set it to 10.

`verbose` Print information to the console. Default is TRUE.

Value

A list of test statistics.

`test_statistics` Test statistics. Each entry corresponds to the test result of one principle component.

`standard_error` Estimated standard error of test_statistics_before_studentization.

`test_statistics_before_studentization` Similar to test_statistics but does not have variance = 1.

`split_data` Intermediate quantities needed for further assessment and interpretation of the test results.

Examples

```
sample_size_1 <- sample_size_2 <- 300

true_mean_1 <- matrix(c(rep(1, 10), rep(0, 90)), ncol = 1)
true_mean_2 <- matrix(c(rep(1.5, 10), rep(0, 90)), ncol = 1)
pc1 <- c(rep(1, 10), rep(0, 90))
pc1 <- pc1/norm(pc1, type = '2')

simulation_covariance <- 10 * pc1 %*% t(pc1)
simulation_covariance <- simulation_covariance + diag(1, 100)

sample_1 <- data.frame(MASS::mvrnorm(sample_size_1,
                               mu = true_mean_1,
                               Sigma = simulation_covariance))
 sample_2 <- data.frame(MASS::mvrnorm(sample_size_2,
                               mu = true_mean_2,
                               Sigma = simulation_covariance))
 result <- debiased_pc_testing(sample_1, sample_2)
 result$test_statistics
 ##these are test statistics. Each one of them corresponds to one PC.
 summarize_pc_name(result, latent_fator_index = 1) #shows which features contribute to PC1
 extract_pc(result) # extract the estimated leading PCs.
```

estimate_leading_pc *Estimate the leading principal component*

Description

Estimates the leading principal component of the input matrix using dense or sparse PCA.

Usage

```
estimate_leading_pc(control, pca_method = c("dense_pca", "sparse_pca"))
```

Arguments

control	A matrix or data frame. Each row is a sample, and each column is a feature.
pca_method	Character. PCA method to use. Options are "dense_pca" (default) or "sparse_pca".

Details

For low-dimensional settings (≤ 30 features), the method automatically switches to dense PCA. For sparse PCA, the function uses the PMA::SPC.cv cross-validation method.

Value

A normalized numeric vector representing the leading principal component direction.

Examples

```
## Not run:
X <- matrix(rnorm(100), nrow = 20)
estimate_leading_pc(X, pca_method = "dense_pca")

## End(Not run)
```

estimate_nuisance_parameter_lasso *The function for nuisance parameter estimation in anchored_lasso_testing().*

Description

The function for nuisance parameter estimation in anchored_lasso_testing().

Usage

```
estimate_nuisance_parameter_lasso(
  nuisance_sample_1,
  nuisance_sample_2,
  pca_method = "sparse_pca",
  mean_method = "lasso",
  lasso_tuning_method = "min",
  num_latent_factor = 1,
  local_environment = local_environment,
  verbose = TRUE
)
```

Arguments

nuisance_sample_1	Group 1 sample. Each row is a subject and each column corresponds to a feature.
nuisance_sample_2	Group 2 sample. Each row is a subject and each column corresponds to a feature.
pca_method	Methods used to estimate principle component The default is "sparse_pca", using sparse PCA from package PMA. Other choices are "dense_pca"—the regular PCA; and "hard"— hard-thresholding PCA, which also induces sparsity.
mean_method	Methods used to estimate the discriminant direction. Default is logistic Lasso "lasso". Can also take value "lasso_no_truncation"
lasso_tuning_method	Method for Lasso penalty hyperparameter tuning. Default is "min", the minimizer of cross-validation error; users can also use "1se" for more sparse solutions.
num_latent_factor	The principle component that lasso coefficient anchors at. The default is PC1 = 1.
local_environment	An environment for hyperparameters shared between folds.
verbose	Print information to the console. Default is TRUE.

Value

A list of estimated nuisance quantities.

estimate_leading_pc	Leading principle components
estimate_mean_1	Sample mean for group 1
estimate_mean_2	Sample mean for group 1
estimate_lasso_beta	Logistic Lasso regression coefficients.

estimate_projection_direction
: Anchored projection direction. It is similar to PC1 when signal is weak but similar to estimate_optimal_direction when the signal is moderately large.

estimate_optimal_direction
: Discriminant direction.

`estimate_nuisance_pc` *The function for nuisance parameter estimation in simple_pc_testing() and debiased_pc_testing().*

Description

The function for nuisance parameter estimation in simple_pc_testing() and debiased_pc_testing().

Usage

```
estimate_nuisance_pc(
  nuisance_sample_1,
  nuisance_sample_2 = NULL,
  pca_method = "sparse_pca",
  mean_method = "naive",
  num_latent_factor = 1,
  local_environment = NA
)
```

Arguments

nuisance_sample_1
: Group 1 sample. Each row is a subject and each column corresponds to a feature.

nuisance_sample_2
: Group 2 sample. Each row is a subject and each column corresponds to a feature.

pca_method
: Methods used to estimate principle component The default is "sparse_pca", using sparse PCA from package PMA. Other choices are "dense_pca"—the regular PCA; and "hard"— hard-thresholding PCA, which also induces sparsity.

mean_method
: Methods used to estimate the mean vector. Default is sample mean "naive". There is also a hard-thresholding sparse estiamtor "hard".

num_latent_factor
: Number of principle to be estimated/tested. Default is 1.

local_environment
: A environment for hyperparameters shared between folds.

Value

A list of estimated nuisance quantities.

`estimate_leading_pc`
: Leading principle components

`estimate_mean_1`
: Sample mean for group 1

`estimate_mean_2`
: Sample mean for group 1

`estimate_eigenvalue`
: Eigenvalue for each principle compoenent.

`estimate_noise_variance`
: Noise variance, I need this to construct block-diagonal estimates of the covariance matrix.

`evaluate_influence_function_multi_factor`
: *Calculate the test statistics on the left-out samples. Called in debiased_pc_testing().*

Description

Calculate the test statistics on the left-out samples. Called in debiased_pc_testing().

Usage

```
evaluate_influence_function_multi_factor(
  cross_fitting_sample_1,
  cross_fitting_sample_2 = NULL,
  nuisance_collection,
  num_latent_factor = 1
)
```

Arguments

`cross_fitting_sample_1`
: Group 1 sample. Each row is a subject and each column corresponds to a feature.

`cross_fitting_sample_2`
: Group 2 sample. Each row is a subject and each column corresponds to a feature.

`nuisance_collection`
: A collection of nuisance quantities estimated using "nuisance" samples. It is the output of estimate_nuisance_pc().

`num_latent_factor`
: Number of principle components to be considered.

Value

A list of test statistics.

`inner_product_1`
: Simple inner products for sample 1.

`inner_product_2`
: Simple inner products for sample 2.

`influence_eigenvector_each_subject_1`
: Debiased test statistics, sample 1.

`influence_eigenvector_each_subject_2`
: Debiased test statistics, sample 1.

`for_variance_subject_1`
: Statistics for variance calculation, sample 1.

`for_variance_subject_2`
: Statistics for variance calculation, sample 2.

---

evaluate_pca_lasso_plug_in

Calculate the test statistics on the left-out samples. Called in anchored_lasso_testing().

Description

Calculate the test statistics on the left-out samples. Called in anchored_lasso_testing().

Usage

```
evaluate_pca_lasso_plug_in(
  cross_fitting_sample_1,
  cross_fitting_sample_2,
  nuisance_collection,
  mean_method = "lasso"
)
```

Arguments

`cross_fitting_sample_1`
: Group 1 sample. Each row is a subject and each column corresponds to a feature.

`cross_fitting_sample_2`
: Group 2 sample. Each row is a subject and each column corresponds to a feature.

`nuisance_collection`
: A collection of nuisance quantities estimated using "nuisance" samples. It is the output of estimate_nuisance_pc().

`mean_method`
: Methods used to estimate the discriminant direction. Default is logistic Lasso "lasso". Can also take value "lasso_no_truncation"

Value

A list of test statistics.

`influence_each_subject_1`
: Test statistics for sample 1.

`influence_each_subject_1`
: Test statistics for sample 2.

`for_variance_each_subject_1`
: Statistics for variance calculation, sample 1.

`for_variance_each_subject_2`
: Statistics for variance calculation, sample 2.

`evaluate_pca_plug_in` *Calculate the test statistics on the left-out samples. Called in simple_pc_testing().*

Description

Calculate the test statistics on the left-out samples. Called in simple_pc_testing().

Usage

```
evaluate_pca_plug_in(
  cross_fitting_sample_1,
  cross_fitting_sample_2 = NULL,
  nuisance_collection
)
```

Arguments

`cross_fitting_sample_1`
: Group 1 sample. Each row is a subject and each column corresponds to a feature.

`cross_fitting_sample_2`
: Group 2 sample. Each row is a subject and each column corresponds to a feature.

`nuisance_collection`
: A collection of nuisance quantities estimated using "nuisance" samples. It is the output of estimate_nuisance_pc().

Value

A list of test statistics.

`influence_each_subject_1`
: Statistics for sample 1.

`influence_each_subject_2`
: Statistics for sample 2.

extract_lasso_coef *Extract the lasso estimate from the output of anchored_lasso_testing().*

Description

Extract the lasso estimate from the output of anchored_lasso_testing().

Usage

```
extract_lasso_coef(testing_result)
```

Arguments

testing_result	The output/test result list from anchored_lasso_testing().

Value

A list, whose elements are the estimated discriminant directions for each split—the length of the output list is the same as n_folds.

The discriminant vectors for each split.

extract_pc *Extract the principle components from the output of simple_pc_testing() and debiased_pc_testing().*

Description

Extract the principle components from the output of simple_pc_testing() and debiased_pc_testing().

Usage

```
extract_pc(testing_result)
```

Arguments

testing_result	The output/test result list from simple_pc_testing() or debiased_pc_testing().

Value

A list, whose elements are the estimated PC for each split—the length of the output list is the same as n_folds.

The PC vectors for each split.

`fit_lasso` *Fit a (group) Lasso logistic regression classifier*

Description

Performs Lasso or group Lasso logistic regression to distinguish between two groups of samples.

Usage

```
fit_lasso(
  control_train,
  treat_train,
  lambda_type = c("lambda.min", "lambda.1se"),
  classifier_method = c("lasso", "group_lasso"),
  group = NULL
)
```

Arguments

`control_train`	A matrix or data frame for the control group. Rows are samples, columns are features.
`treat_train`	A matrix or data frame for the treatment group. Rows are samples, columns are features.
`lambda_type`	Character. Type of lambda to use from cross-validation. Options are `"lambda.min"` (default) and `"lambda.1se"`.
`classifier_method`	Character. Choice of classifier. `"lasso"` (default) or `"group_lasso"`.
`group`	Optional grouping vector for `group_lasso`, same length as the number of columns in the input data.

Details

The function fits a logistic regression using either `glmnet` for Lasso or `grpreg` for group Lasso. Coefficients are soft-thresholded by the maximum coefficient times `n^(-1/3)` where `n` is the effective sample size.

Value

A numeric vector of estimated regression coefficients (excluding intercept), thresholded for small values.

Examples

```
## Not run:
X1 <- matrix(rnorm(100), nrow = 10)
X2 <- matrix(rnorm(100), nrow = 10)
fit_lasso(X1, X2, classifier_method = "lasso")
```

```
## End(Not run)
```

`index_spliter` *Split indices into folds*

Description

Randomly splits a given vector of indices into approximately equal-sized folds.

Usage

```
index_spliter(array, n_folds = 5)
```

Arguments

`array`	A vector of indices (e.g., `1:n`) to be split into folds.
`n_folds`	Integer. Number of folds. Default is 5.

Value

A list of length `n_folds`, each containing a subset of the shuffled indices.

Examples

```
index_spliter(1:10, n_folds = 3)
```

`mean_comparison_anchor` *High-dimensional two-sample mean comparison with anchored projection*

Description

Performs a cross-validated, projection-based mean comparison between two high-dimensional groups using sparse or dense PCA and (group) Lasso classifiers.

Usage

```
mean_comparison_anchor(
  control,
  treatment,
  pca_method = c("dense_pca", "sparse_pca"),
  classifier_method = c("lasso", "group_lasso"),
  lambda_type = "lambda.1se",
  n_folds = 10,
  group = NULL,
  standardize_feature = TRUE,
  verbose = TRUE
)
```

Arguments

control	A matrix or data frame for the control group. Rows are samples; columns are features.
treatment	A matrix or data frame for the treatment group. Rows are samples; columns are features.
pca_method	Character. Method for estimating the projection direction. Options are "dense_pca" or "sparse_pca". Default is "sparse_pca".
classifier_method	Character. Classifier to guide the projection. Options are "lasso" or "group_lasso". Default is "lasso".
lambda_type	Character. Regularization parameter choice in Lasso. Options are "lambda.min" or "lambda.1se". Default is "lambda.1se".
n_folds	Integer. Number of cross-validation folds. Default is 10.
group	Optional. A grouping vector (required for group_lasso), same length as the number of columns in control.
standardize_feature	Logical. Whether to standardize features using pooled mean and standard deviation. Default is TRUE.
verbose	Logical. Whether to print messages during execution. Default is TRUE.

Details

This function applies a projection-based method for high-dimensional mean testing. The projection direction is computed by anchoring the leading principal component with a regularized classifier (Lasso or group Lasso), and test statistics are aggregated across folds.

Value

A list with:

p_value Two-sided p-value for the overall test.

test_statistic Standardized test statistic.

fold_data Per-fold results, including projections and scores.

See Also

process_fold_mean_diff, combine_folds_mean_diff, estimate_leading_pc, fit_lasso

Examples

```
## Not run:
X <- matrix(rnorm(200 * 100), nrow = 100)
Y <- matrix(rnorm(200 * 100), nrow = 100)
result <- mean_comparison_anchor(X, Y, pca_method = "dense_pca", classifier_method = "lasso")

## End(Not run)
```

---

normalize_and_split *Normalize and split two datasets using pooled mean and standard deviation*

Description

Combines two datasets, normalizes features using pooled mean and standard deviation, and returns the normalized datasets separately.

Usage

```
normalize_and_split(df1, df2)
```

Arguments

df1	A data frame or matrix. Typically group 1.
df2	A data frame or matrix. Typically group 2.

Value

A list with elements:

df1 Normalized version of df1.

df2 Normalized version of df2.

Examples

```
set.seed(123)
df1 <- matrix(rnorm(20), nrow = 5)
df2 <- matrix(rnorm(20), nrow = 5)
normalize_and_split(df1, df2)
```

process_fold_mean_diff *Process one cross-validation fold for mean difference testing*

Description

Computes the test statistic, variance, and projection direction for one fold in a cross-validated comparison of two groups.

Usage

```
process_fold_mean_diff(
  fold_index,
  control,
  treatment,
  control_split_index,
  tr_split_index,
  pca_method,
  classifier_method,
  lambda_type,
  group,
  verbose
)
```

Arguments

fold_index	Integer index of the current fold.
control	Matrix or data frame for the control group (rows = samples, columns = features).
treatment	Matrix or data frame for the treatment group (rows = samples, columns = features).
control_split_index	A list of row indices for each fold of the control group.
tr_split_index	A list of row indices for each fold of the treatment group.
pca_method	Character. PCA method to use. Options are "dense_pca" or "sparse_pca".
classifier_method	Character. Classifier method. Options are "lasso" or "group_lasso".
lambda_type	Character. Lambda selection method. Options are "lambda.min" or "lambda.1se".
group	Optional grouping vector for group lasso.
verbose	Logical. Whether to print progress messages.

Value

A list containing the test statistic, its variance, scores for each group, the projection direction, and intermediate quantities.

simple_pc_testing *Simple plug-in test for two-sample mean comparison.*

Description

Simple plug-in test for two-sample mean comparison.

Usage

```
simple_pc_testing(
  sample_1,
  sample_2 = NULL,
  pca_method = "sparse_pca",
  mean_method = "naive",
  num_latent_factor = 1,
  n_folds = 5,
  verbose = TRUE
)
```

Arguments

sample_1	Group 1 sample. Each row is a subject and each column corresponds to a feature.
sample_2	Group 2 sample. Each row is a subject and each column corresponds to a feature.
pca_method	Methods used to estimate principle component The default is "sparse_pca", using sparse PCA from package PMA. Other choices are "dense_pca"—the regular PCA; and "hard"— hard-thresholding PCA, which also induces sparsity.
mean_method	Methods used to estimate the mean vector. Default is sample mean "naive". There is also a hard-thresholding sparse estiamtor "hard".
num_latent_factor	Number of principle to be estimated/tested. Default is 1.
n_folds	Number of splits when performing cross-fitting. The default is 5, if computational time allows, you can try to set it to 10.
verbose	Print information to the console. Default is TRUE.

Value

A list of test statistics.

test_statistics	Test statistics. Each entry corresponds to the test result of one principle component.
standard_error	Estimated standard error of test_statistics_before_studentization.
test_statistics_before_studentization	Similar to test_statistics but does not have variance = 1.
split_data	Intermediate quantities needed for further assessment and interpretation of the test results.

Examples

```
sample_size_1 <- sample_size_2 <- 300
true_mean_1 <- matrix(c(rep(1, 10), rep(0, 90)), ncol = 1)
true_mean_2 <- matrix(c(rep(1.5, 10), rep(0, 90)), ncol = 1)
pc1 <- c(rep(1, 10), rep(0, 90))
pc1 <- pc1/norm(pc1, type = '2')

simulation_covariance <- 10 * pc1 %*% t(pc1)
simulation_covariance <- simulation_covariance + diag(1, 100)

sample_1 <- data.frame(MASS::mvrnorm(sample_size_1,
                                     mu = true_mean_1,
                                     Sigma = simulation_covariance))
sample_2 <- data.frame(MASS::mvrnorm(sample_size_2,
                                     mu = true_mean_2,
                                     Sigma = simulation_covariance))
result <- simple_pc_testing(sample_1, sample_2)
result$test_statistics
##these are test statistics. Each one of them corresponds to one PC.
summarize_pc_name(result, latent_fator_index = 1) #shows which features contribute to PC1
extract_pc(result) # extract the estimated leading PCs.
```

`summarize_feature_name` *Summarize the features (e.g. genes) that contribute to the test result, i.e. those features consistently show up in Lasso vectors.*

Description

Summarize the features (e.g. genes) that contribute to the test result, i.e. those features consistently show up in Lasso vectors.

Usage

```
summarize_feature_name(testing_result, method = "majority voting")
```

Arguments

`testing_result`	The output/test result list from anchored_lasso_testing().
`method`	How to combine the feature list across different splits. Default is 'majority voting'—features that show up more than 50% of the splits are considered active/useful. It can be 'union'—all the features pooled together; or 'intersection'—only include features showing up in all splits.

Value

A list of names of features (your very original input data need to have column names!) that contribute to the test result. An empty list means there is barely any difference between the two groups.

Feature names that consistently showing up in the discriminant vectors.

`summarize_pc_name` *Summarize the features (e.g. genes) that contribute to the test result, i.e. those features consistently show up in the sparse principle components.*

Description

Summarize the features (e.g. genes) that contribute to the test result, i.e. those features consistently show up in the sparse principle components.

Usage

```
summarize_pc_name(
  testing_result,
  latent_fator_index = 1,
  method = "majority voting"
)
```

Arguments

`testing_result`	The output/test result list from simple_pc_testing() or debiased_pc_testing().
`latent_fator_index`	Which principle component should the algorithm summarize? Default is PC1.
`method`	How to combine the feature list across different splits. Default is 'majority voting'—features that show up more than 50% of the splits are considered active/useful. It can be 'union'—all the features pooled together; or 'intersection'—only include features showing up in all splits.

Value

A list of names of features (your very original input data need to have column names!) that contribute to the test result.

Feature names that consistently showing up in the estimated PC vectors.

validate_and_convert_data *Validate and convert input data*

Description

Checks whether the input is a matrix or data frame, and converts it to a matrix if valid.

Usage

```
validate_and_convert_data(data, name)
```

Arguments

data	A matrix or data frame.
name	A string used in error messages to identify the variable name.

Value

A numeric matrix.

Examples

```
validate_and_convert_data(data.frame(x = 1:3, y = 4:6), "example_data")
```

Index

anchored_lasso_testing, 2

check_data_for_folds, 4
check_non_null_and_identical_colnames, 4
collect_active_features_proj, 5, *7*
combine_folds_mean_diff, 6, *19*
compute_predictive_contributions, 6

debiased_pc_testing, 7

estimate_leading_pc, 9, *19*
estimate_nuisance_parameter_lasso, 9
estimate_nuisance_pc, 11
evaluate_influence_function_multi_factor, 12
evaluate_pca_lasso_plug_in, 13
evaluate_pca_plug_in, 14
extract_lasso_coef, 15
extract_pc, 15

fit_lasso, 16, *19*

index_spliter, 17

mean_comparison_anchor, 17

normalize_and_split, 19

process_fold_mean_diff, *19*, 20

simple_pc_testing, 21
summarize_feature_name, 22
summarize_pc_name, 23

validate_and_convert_data, 24