Package: pye (via r-universe)

June 4, 2026

Type Package

Title Penalized Youden Index Estimator

Version 0.1.0

Date 2026-05-31

Description Implements the Penalized Youden Index Estimator (PYE) and the Covariate-Adjusted Youden Index Estimator (covYI), providing a novel framework for feature and covariate selection and combination in high-dimensional binary classification problems. Methodologies are based on Salaroli and Pardo (2023) <doi:10.1016/j.chemolab.2023.104786> and an unpublished manuscript by Salaroli and Pardo (2026) under review.

License GPL (>= 2)

Encoding UTF-8

Depends R (>= 4.0.0)

RoxygenNote 7.3.3

Suggests knitr, rmarkdown, roxygen2, spelling, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Language en-US

Imports evmix, ggplot2, glmnet, MASS, Matrix, methods, ncvreg, OptimalCutpoints, pROC, penalizedSVM, plyr, ROCnReg, Rmpfr, sparseSVM, stats, survival

NeedsCompilation yes

Author Claudio J. Salaroli [aut, cre], Maria del Carmen Pardo [aut]

Maintainer Claudio J. Salaroli <clasalar@ucm.es>

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

Date/Publication 2026-06-04 14:12:19 UTC

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

RemoteRef HEAD

RemoteSha 846f3ebafc1203b25ff5233dd768e2656286359a

Contents

`AucPR_compute_cv` *Cross-Validation for Optimal AucPR Regularization Parameter Selection*

Description

This function performs k-fold cross-validation to determine the optimal values for the penalization parameters ‘lambda‘ and ‘tau‘ (if applicable) for the ‘AucPR‘ model. The primary goal is to select the hyperparameters that yield the best model performance based on a variety of accuracy measures. The function first divides the dataset into stratified folds to maintain the class distribution of the target variable. It then iterates through the specified ‘lambda‘ and ‘tau‘ values, training and evaluating

the model on each fold. The model's performance is assessed using metrics such as AUC, aAUC, aYI, Youden's Index, sensitivity, specificity, and others. The function returns the results of this process, including the performance metrics for each fold and the hyperparameter values that produced the best overall test performance for each measure.

Usage

```
AucPR_compute_cv(
  n_folds,
  df,
  X = NULL,
  y = "y",
  C = NULL,
  alpha = 0.5,
  lambda,
  tau = 0,
  w_g = 0.5,
  c_to_use = "Youden",
  regressors_betas = NULL,
  trace = 1,
  seed = 1,
  used_cores = 1,
  scaling = FALSE,
  c_function_of_covariates = FALSE,
  simultaneous = FALSE,
  measure_to_select_lambda = "ccr",
  alpha_g = 0.5,
  penalty_g = "L1",
  kernel_g = "gaussian",
  a1_g = 3.7,
  a2_g = 3,
  trend_g = "monotone",
  gamma_start_input = NULL,
  gamma_start_default = "zeros",
  regressors_gammas = NULL,
  max_iter_g = 10000,
  delta_g = 1e-05,
  max_alpha_g = 10000,
  stepsizeShrink_g = 0.8,
  min_alpha_g = 1e-12,
  convergence_error_g = 1e-07,
  run_aauc = FALSE,
  log_file = "log_AUC_models.txt"
)
```

Arguments

n_folds	Integer. Number of folds for cross-validation.
df	Data frame. Input dataset containing predictors and target.

X	Character vector or data frame. Names of predictor variables. Defaults to all columns not in ‘y‘ or ‘C‘.
y	Character or data frame. Name of the binary target variable (0/1). Defaults to ‘"y"‘.
C	Character vector or data frame. Names of covariate variables for ‘covYI‘. Default is ‘NULL‘.
alpha	Numeric in (0, 1]. Elastic-net mixing parameter for AucPR. ‘1‘ = Lasso, ‘(0, 1)‘ = Elastic-Net.
lambda	Numeric vector. Penalization parameter(s) for predictors ‘X‘.
tau	Numeric vector. Penalization parameter(s) for covariates ‘C‘ in ‘covYI‘. Ignored if ‘c_function_of_covariates = FALSE‘.
w_g	A ‘numeric‘ value between 0 and 1 specifying the weight for the Weighted Youden Index in covYI. Sensitivity is weighted by ‘w_g‘ and specificity by ‘1 - w_g‘. Default is 0.5 (which corresponds to the standard Youden Index).
c_to_use	character or numeric value specifying the classification cut-point used to convert continuous predictions into binary outcomes. If set to ‘"Youden"‘, the function computes the optimal cut-point using Youden’s Index. If a numeric value is provided, it is used directly as the threshold. Default is "Youden".
regressors_betas	numeric vector. An optional vector containing the "true" beta coefficients, if known. This is used to calculate additional performance measures related to variable selection accuracy. Default is ‘NULL‘.
trace	Integer (0, 1, 2). Level of printed output.
seed	Integer. Random seed for reproducibility.
used_cores	Integer. Number of CPU cores for parallelization.
scaling	Logical. If ‘TRUE‘, scale predictors and covariates.
c_function_of_covariates	Logical. If ‘TRUE‘, estimate cut-off as a function of covariates via ‘covYI‘.
simultaneous	Logical. If ‘TRUE‘, estimate betas and gammas jointly.
measure_to_select_lambda	Character. Metric to select ‘lambda‘ when ‘simultaneous = FALSE‘. One of ‘"auc"‘, ‘"aauc"‘, ‘"aYI"‘, ‘"yi"‘, ‘"sen"‘, ‘"spc"‘, ‘"gm"‘, ‘"fdr"‘, ‘"mcc"‘, ‘"ccr"‘.
alpha_g	Numeric. Elastic-net mixing parameter for ‘covYI‘.
penalty_g	Character. Penalty type for ‘covYI‘: ‘"L12"‘, ‘"L1"‘, ‘"EN"‘, ‘"SCAD"‘, ‘"MCP"‘.
kernel_g	Character. Kernel type for density estimation in ‘covYI‘.
a1_g, a2_g	Numeric. Parameters for SCAD/MCP penalties in ‘covYI‘.
trend_g	Character. ‘"monotone"‘ uses mmAPG, ‘"nonmonotone"‘ uses mnmAPG.
gamma_start_input	Numeric vector. Starting point for gammas.
gamma_start_default	Character. ‘"zeros"‘ or ‘"corr"‘.

`regressors_gammas`
: Numeric vector. True gamma coefficients (optional).

`max_iter_g`
: Integer. Max iterations for 'covYI' optimization.

`delta_g`, `max_alpha_g`, `stepsizeShrink_g`, `min_alpha_g`, `convergence_error_g`
: Numeric. Optimization parameters for 'covYI'.

`run_aauc`
: Logical. If 'FALSE', skip aAUC/aYI computation.

`log_file`
: Character. Path to a file for logging output from parallel workers. If 'NULL', output goes to the console. Default is '"log_AUC_models.txt"'.

Value

A list containing the results of the cross-validation process. The list includes:

`model_type`
: The type of model used (e.g., "AucPR_EN").

`alpha`
: The alpha value used for the AucPR model.

`penalty_g`
: The penalty type used for the 'covYI' model.

`cv_time`
: The total time taken for the cross-validation, in minutes

.

`auc_first_step`, `aauc_first_step`, `aYI_first_step`, `youden_index_first_step`, `sensitivity_first_step`, `specificity_first_step`, `geometric_mean_first_step`, `fdr_first_step`, `mcc_first_step`, `corrclass_first_step`
: A series of nested lists, each containing performance matrices for the respective measure of the first-step estimated method.

`auc`, `aauc`, `aYI`, `youden_index`, `sensitivity`, `specificity`, `geometric_mean`, `fdr`, `mcc`, `corrclass`
: A series of nested lists, each containing performance matrices for the respective measure of the final estimated method.

`n_betas`
: A matrix showing the number of non-zero beta coefficients for each fold and lambda value.

`n_gammas`
: A list of matrices showing the number of non-zero gamma coefficients for each fold, lambda, and tau value.

`betas`
: A list of the estimated beta coefficients for each fold.

`betas_star`
: A list of the omptimal estimated beta coefficients used for the estimation of the gammas, for each fold.

`gammas`
: A list of the estimated gamma coefficients for each fold.

`lambda_hat_*`
: A vector containing the optimal lambda value selected by each performance measure.

`tau_hat_*`
: A vector containing the optimal tau value selected by each performance measure (only if 'c_function_of_covariates' is TRUE).

Examples

```
# 1. Simulate a small dataset
set.seed(123)
sim_data <- create_sample_with_covariates(
  rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- sim_data$train_df_scaled
X <- sim_data$X
y <- sim_data$y
C <- sim_data$C
regressors_betas <- sim_data$nregressors
regressors_gammas <- sim_data$ncovariates

# 2. Define a small grid for hyperparameters
# In a real scenario, these would be calibrated (e.g., using calibrate_lambda_max)
# and the sequence would be longer.
lambda_seq <- c(0.1, 0.05)
tau_seq <- c(0.1, 0.05)

# 3. Run cross-validation
cv_results <- AucPR_compute_cv(alpha = 0.5,
                               trace = 1, n_folds = 2,
                               df = df, X = X, y = y, C = C,
                               lambda = lambda_seq, tau = tau_seq,
                               regressors_betas = regressors_betas,
                               regressors_gammas = regressors_gammas,
                               c_function_of_covariates = TRUE,
                               simultaneous = TRUE,
                               measure_to_select_lambda = "ccr",
                               penalty_g = "L1", max_iter_g = 5)

# 4. Inspect the results
cat("Cross-validation finished in:", round(cv_results$cv_time, 2), "minutes\n")
cat("Optimal lambda based on AUC:", cv_results$lambda_hat_auc, "\n")
cat("Optimal tau based on CCR:", cv_results$tau_hat_ccr, "\n")
```

`AucPR_estimation` *AucPR Estimation for Coefficient and Feature Selection*

Description

function to estimate the optimal value of betas using the AucPR method.

This function estimates the optimal regression coefficients (betas) for a binary classification problem by maximizing the penalized Area Under the Precision-Recall Curve (AUC-PR). The penalization is either L1 (Lasso) or Elastic-Net, which helps in feature selection and regularization. The function returns the estimated coefficients along with various performance metrics.

Usage

```
AucPR_estimation(
  df,
  X = NULL,
  y = "y",
  lambda,
  alpha = 1,
  c_to_use = "Youden",
  fold = NULL,
  regressors_betas = NULL,
  trace = 1,
  max.print = 10,
  ...
)
```

Arguments

df	A data frame containing the complete dataset.
X	A character vector of column names from 'df' to be used as regressor variables. Alternatively, a data frame containing only the regressors. If not specified, all columns not in 'y' are used.
y	A character string specifying the column name of the target variable in 'df'. It must be a binomial variable with values 0 or 1. Alternatively, a data frame containing only the target variable. Default is "y".
lambda	A single numeric value for the regularization parameter. A larger value results in stronger penalization.
alpha	The elastic-net mixing parameter. A value of 1 corresponds to L1 penalization (Lasso), while a value between 0 and 1 corresponds to Elastic-Net. Default is 1.
c_to_use	character or numeric value specifying the classification cut-point used to convert continuous predictions into binary outcomes. If set to '"Youden"', the function computes the optimal cut-point using Youden's Index. If a numeric value is provided, it is used directly as the threshold. Default is "Youden".
fold	numeric. An optional fold number, used when the function is called within a cross-validation loop. This is primarily for tracking and reporting purposes. Default is 'NULL'.
regressors_betas	numeric vector. An optional vector containing the "true" beta coefficients, if known. This is used to calculate additional performance measures related to variable selection accuracy. Default is 'NULL'.
trace	An integer controlling the amount of output. '0' for no output, '1' for a summary of the results, and '2' for a detailed breakdown of all steps. Default is 1.
max.print	The number of elements to show when printing results. Default is 10.
...	Additional arguments passed to methods

Value

A list containing the optimal betas and a collection of performance metrics.

`model_type`	A character string indicating the final model type used, either "AucPR_L1" or "AucPR_EN".
`betas_hat`	The estimated vector of optimal beta coefficients.
`X_model`	A character vector containing the names of all regressor variables (`X`) used in the model estimation. This vector defines the canonical order of the regressors used to align the coefficients in `betas_hat`.
`youden_index`	The Youden Index value at the optimal cut-point, which is a measure of the overall effectiveness of a diagnostic test.
`sensitivity`	The sensitivity (True Positive Rate) at the optimal cut-point.
`specificity`	The specificity (True Negative Rate) at the optimal cut-point.
`geometric_mean`	The geometric mean of sensitivity and specificity.
`fdr`	The False Discovery Rate at the optimal cut-point.
`mcc`	The Matthews Correlation Coefficient, a measure of the quality of binary classification.
`corrclass`	The overall correct classification rate.
`auc`	The Area Under the ROC Curve (AUC).
`lambda`	The penalization parameter used in the estimation.
`alpha`	The alpha parameter used in the estimation.
`c_hat`	The optimal cut-point for the ‘z_hat‘ score.
`z_hat`	A data frame with the estimated linear combination of regressors for each observation.
`y_hat`	A data frame with the predicted binary outcomes (0 or 1) for each observation.
`n_betas`	The number of non-zero beta coefficients estimated.
`n_total_var`	The total number of regressors considered.
`n_predicted_zeros`	The number of estimated betas that are zero.
`n_predicted_non_zeros`	The number of estimated betas that are non-zero.
`n_caught_betas`	The number of true non-zero betas correctly identified as non-zero (if ‘regressors_betas‘ is provided).
`n_non_caught_betas`	The number of true non-zero betas incorrectly identified as zero.
`n_caught_zero`	The number of true zero betas correctly identified as zero.
`n_zero_not_caught`	The number of true zero betas incorrectly identified as non-zero.
`estimation_time`	The time taken for the estimation, in minutes.

Examples

```
library(pye)
sim_data <- create_sample_with_covariates(
    rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- sim_data$train_df_scaled
X <- sim_data$X
y <- sim_data$y
regressors_betas <- sim_data$nregressors
alpha <- 1
lambda <- 0.1

AucPR_result <- AucPR_estimation(df = df, X = X, y = y, lambda = lambda, alpha = alpha,
        regressors_betas = regressors_betas, trace = 1)

print(AucPR_result)
```

AucPR_predict *Prediction and Performance Evaluation for Penalized AucPR Models*

Description

This function applies a previously trained penalized AUC-PR model to new data to generate predictions and evaluate performance metrics. The model to be used is the result of the 'AucPR_estimation' function.

Usage

```
AucPR_predict(
  df,
  y = "y",
  model_to_use,
  fold = NULL,
  regressors_betas = NULL,
  trace = 1,
  max.print = 10,
  c_function_of_covariates = FALSE,
  ...
)
```

Arguments

df	A data frame containing the new dataset for prediction.
y	A character string specifying the column name of the target variable in 'df'. It must be a binomial variable with values 0 or 1. Default is "y".
model_to_use	A list containing the parameters from a model trained by the 'AucPR_estimation' function, specifically the 'betas_hat' and 'c_hat' components.

fold	numeric. An optional fold number, used when the function is called within a cross-validation loop. This is primarily for tracking and reporting purposes. Default is 'NULL'.
regressors_betas	numeric vector. An optional vector containing the "true" beta coefficients, if known. This is used to calculate additional performance measures related to variable selection accuracy. Default is 'NULL'.
trace	An integer controlling the amount of output. '0' for no output, '1' for a summary of the results, and '2' for a detailed breakdown of all steps. Default is 1.
max.print	The number of elements to show when printing results. Default is 10.
c_function_of_covariates	A logical flag to suppress printing of results. If 'TRUE', the function will not print the summary to the console, as the output is expected to be handled by an external function (e.g., 'covYI'). Default is 'FALSE'.
...	Additional arguments passed to methods

Value

A list containing the predictions and a collection of performance metrics, including:

model_type	The model type ("AucPR_L1" or "AucPR_EN").
betas_hat	The estimated vector of optimal beta coefficients.
youden_index	The Youden Index value at the optimal cut-point.
sensitivity	The sensitivity at the optimal cut-point.
specificity	The specificity at the optimal cut-point.
geometric_mean	The geometric mean of sensitivity and specificity.
fdr	The False Discovery Rate at the optimal cut-point.
mcc	The Matthews Correlation Coefficient.
corrclass	The overall correct classification rate.
auc	The Area Under the ROC Curve (AUC).
lambda	The penalization parameter used in the estimation.
alpha	The alpha parameter used in the estimation.
c_hat	The optimal cut-point for the 'z_hat' score.
z_hat	A data frame with the estimated linear combination of regressors for each observation.
y_hat	A data frame with the predicted binary outcomes (0 or 1).
n_total_var	The total number of regressors considered.
n_predicted_zeros	The number of estimated betas that are zero.
n_predicted_non_zeros	The number of estimated betas that are non-zero.
n_caught_betas	The number of true non-zero betas correctly identified as non-zero (if 'regressors_betas' is provided).

`n_non_caught_betas`
: The number of true non-zero betas incorrectly identified as zero.

`n_caught_zero`
: The number of true zero betas correctly identified as zero.

`n_zero_not_caught`
: The number of true zero betas incorrectly identified as non-zero.

`TP`, `TN`, `FP`, `FN`
: The counts from the confusion matrix.

`estimation_time`
: The time taken for the prediction.

Examples

```
library(pye)

sim_data <- create_sample_with_covariates(
    rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
train_df <- sim_data$train_df_scaled
test_df <- sim_data$test_df_scaled
X <- sim_data$X
y <- sim_data$y
regressors_betas <- sim_data$nregressors
alpha <- 0.5
lambda <- 0.1
c_function_of_covariates <- FALSE

# 1. Train the model using AucPR_estimation
model <- AucPR_estimation(df = train_df, y = y, X = X, alpha = alpha,
   lambda = lambda, regressors_betas = regressors_betas, trace = 1)

# 2. Apply the trained model to new data using AucPR_predict
predictions <- AucPR_predict(df = test_df, model_to_use = model,
   trace = 1, regressors_betas = regressors_betas,
   c_function_of_covariates = c_function_of_covariates)

print(predictions)
```

`calibrate_lambda_max` *Calibrate Maximum Value for the Penalty Parameter*

Description

Function to calibrate the starting value of lambda to make cross-validation more effective. In particular, this function aim to solve the problem that every model has its own "best" range of lambdas at which the parameters are different from 0 and performs better. The function stops where the number of elements of var_to_check is greater then 0.

Usage

```
calibrate_lambda_max(
  function_to_run,
  var_to_check = "betas_hat",
  lambda_start = 5,
  lambda_inf = 1e-20,
  n_min_var = 0,
  factor = 2
)
```

Arguments

function_to_run	Function that needs to be cross-validated. It needs to have a have just one parameter to change (theoretically called lambda), changing which the number of selected variables might change.
var_to_check	Name of the variable (a vector) to check if, reducing lambda, the number of its elements increases.
lambda_start	Starting value of lambda. If left big might take longer (default is 5)
lambda_inf	floor value for lambda (default is 1e-20)
n_min_var	Minimum number of variables at which the process stops
factor	Number to multiply or divide the considered lambda in the step. The higher, the slower to find the optimal value. Default is 2.

Value

The first encountered value of the parameter (lambda inside the function) at which the number of elements of var_to_check is greater then 0.

Examples

```
library(pye)
simMicroarrayData_cov02_dim50_covariates <- create_sample_with_covariates(
    rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- simMicroarrayData_cov02_dim50_covariates$train_df_scaled
#create the wrapper
penalty <- "L1"
wrapper <- function(lambda) {
   return(pye_KS_estimation(df = df, penalty = penalty, trace = 2, lambda = lambda,
   c_zero_fixed = FALSE, max_iter=5))
}
#find the lambda max
var_to_check <- paste0("betas_hat_", penalty)
lambda_max <- calibrate_lambda_max (function_to_run = wrapper,
               var_to_check=var_to_check, lambda_start = 5)
print(lambda_max)
```

`calibrate_lambda_min` *Calibrate Minimum Value for the Penalty Parameter*

Description

Function to calibrate the last used value of lambda, to make cross-validation more effective. In particular, this function aim to solve the problem that every model has its own "best" range of lambdas at which the parameters are different from 0 and performs better. The function stops where the number of elements of var_to_check is greater then 0.

Usage

```
calibrate_lambda_min(
  function_to_run,
  var_to_check = "betas_hat",
  lambda_start = 1e-04,
  lambda_inf = 1e-20,
  max_var = 1000,
  factor = 2
)
```

Arguments

`function_to_run`	Function that needs to be cross-validated. It needs to have a have just one parameter to change (theoretically called lambda), changing which the number of selected variables might change.
`var_to_check`	Name of the variable (a vector) to check if, reducing lambda, the number of its elements increases.
`lambda_start`	Starting value of lambda. If left too small it might take longer (default is 0.0001)
`lambda_inf`	floor value for lambda (default is 1e-20)
`max_var`	Maximum number of variable to accept the result of the function (default is 100.000)
`factor`	Number to multiply or divide the considered lambda in the step. The higher, the slower to find the optimal value. Default is 2.

Value

The first encountered value of the parameter (lambda inside the function) at which the number of elements of var_to_check is lower then max_var.

Examples

```
library(pye)
simMicroarrayData_cov02_dim50_covariates <- create_sample_with_covariates(
    rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
```

```
df <- simMicroarrayData_cov02_dim50_covariates$train_df_scaled
#create the wrapper
penalty <- "L1"
wrapper <- function(lambda) {
   return(pye_KS_estimation(df = df, penalty = penalty, trace = 1, lambda = lambda,
   c_zero_fixed = FALSE, max_iter = 5))
}
#find the lambda min
var_to_check <- paste0("betas_hat_", penalty)
#accept maximum 100 betas
max_var <- 20
lambda_min <- calibrate_lambda_min (function_to_run = wrapper,
  var_to_check = var_to_check, lambda_start = 0.2, max_var = max_var)
print(lambda_min)
```

`covYI_KS` *Covariate-Adjusted Youden Index (covYI) method with Kernel Smoothing Estimation of the Density Functions*

Description

This function implements the Covariate-Adjusted Youden Index with Kernel Smoothing (covYI_KS). It calculates the Youden Index, incorporating covariate information through a kernel smooth density estimator, and can apply various penalization types (L1 / 2, L1, Elastic-Net (EN), SCAD, and MCP) to the covariate coefficients. The function returns the penalized Youden Index values, along with key classification accuracy measures, and the gradient of the Youden Index with respect to the covariate coefficients.

Usage

```
covYI_KS(
  df,
  z = "z_hat",
  y = "y",
  C,
  gammas,
  tau,
  w = 0.5,
  kernel = "gaussian",
  alpha = 0.5,
  a1 = 3.7,
  a2 = 3,
  penalty = "L1",
  h_exponent = 0.2,
  prediction = FALSE,
  run_aauc = FALSE
)
```

Arguments

df	A data frame containing the input data, including the latent variable/score (‘z‘), the binary target variable (‘y‘), and covariates (‘C‘).
z	Character string. The column name in ‘df‘ representing the latent variable or score (e.g., from a primary prediction model). Default is "z_hat".
y	Character string. The column name in ‘df‘ representing the binary target variable (0 or 1). Default is "y".
C	Character vector. Column names from ‘df‘ to be used as covariate variables for the cut-point ‘c‘. This vector must include "const" if an intercept is desired for the covariate combination.
gammas	Numeric vector. The coefficients of the covariates (‘C‘) used to evaluate the cut-point ‘c‘. The first element of this vector is assumed to correspond to the constant (intercept) term if "const" is in ‘C‘.
tau	Numeric. The penalization parameter for the covariates ‘C‘ in the ‘covYI‘ calculation. Default is 0, indicating no penalization. Must be a single non-negative value.
w	A ‘numeric‘ value between 0 and 1 specifying the weight for the Weighted Youden Index. Sensitivity is weighted by ‘w‘ and specificity by ‘1 - w‘. Default is 0.5 (which corresponds to the standard Youden Index).
kernel	Character string. The kernel type to use for the estimation of the density function in the Kernel Smooth method. Supported options include "gaussian", "normal", "uniform", "rectangular", "triangular", "epanechnikov", "biweight", "triweight", "tricube", "parzen", "cosine", and "optcosine". Default is "gaussian".
alpha	Numeric. A value between 0 and 1, representing the elastic-net mixing parameter for the EN penalization term. Default is 0.5.
a1	Numeric. The ‘a‘ parameter for the SCAD penalization term. Default is 3.7.
a2	Numeric. The ‘a‘ parameter for the MCP penalization term. Default is 3.0.
penalty	Character string. The type of penalization to apply. Must be one of "L12", "L1", "EN", "SCAD", or "MCP". Default is "L1".
h_exponent	Numeric. The exponent used in the bandwidth calculation for the Kernel Smooth density estimation. Default is 0.2.
prediction	Logical. If ‘TRUE‘, the Youden Index (YI) returned is the empirical maximum Youden Index derived from ‘OptimalCutpoints‘ for the given ‘z‘ and ‘y‘. If ‘FALSE‘, it’s the ‘yi‘ calculated from ‘f0 - f1‘. Default is ‘FALSE‘.
run_aauc	Logical. If ‘FALSE‘, the covariate-adjusted AUC (aAUC) and adjusted Youden Index (aYI) are not computed, which can save computation time if these measures are not required. Default is ‘FALSE‘.

Value

A list containing the results of the ‘covYI_KS‘ calculation and related performance measures.

covYI_KS_L12	Numeric. The Penalized Youden Index (pye) using the L1 / 2 penalty.
covYI_KS_L1	Numeric. The Penalized Youden Index (pye) using the L1 penalty.

covYI_KS_EN	Numeric. The Penalized Youden Index (pye) using the Elastic-Net penalty.
covYI_KS_SCAD	Numeric. The Penalized Youden Index (pye) using the SCAD penalty.
covYI_KS_MCP	Numeric. The Penalized Youden Index (pye) using the MCP penalty.
gr_yi	Numeric vector. The gradient of the Youden Index ('yi') with respect to the 'gammas' coefficients (excluding the constant term).
youden_index	Numeric. The unpenalized Youden Index calculated using the Kernel Smooth density estimator (or the empirical Youden Index if prediction = TRUE). It is weighted if w is not 0.5.
sensitivity	Numeric. The sensitivity (True Positive Rate) of the classification.
specificity	Numeric. The specificity (True Negative Rate) of the classification.
geometric_mean	Numeric. The geometric mean of sensitivity and specificity.
fdr	Numeric. The False Discovery Rate.
mcc	Numeric. The Matthews Correlation Coefficient.
auc	Numeric. The Area Under the ROC Curve (AUC) from 'OptimalCutpoints'.
aauc	Numeric. The covariate-adjusted Area Under the ROC Curve (aAUC). This is 0 if 'run_aauc' is 'FALSE' or if computation fails.
aYI	Numeric. The adjusted Youden Index (aYI). This is 0 if 'run_aauc' is 'FALSE' or if computation fails.
corrclass	Numeric. The overall correct classification rate.
z_hat	Data frame. A data frame with 'ID' and the 'z' values used in the calculation.
c_hat	Data frame. A data frame with 'ID' and the estimated cut-point values ('c_hat') for each observation.
y_hat	Data frame. A data frame with 'ID' and the predicted binary outcomes ('y_hat').
TP	Numeric. Number of True Positives.
TN	Numeric. Number of True Negatives.
FP	Numeric. Number of False Positives.
FN	Numeric. Number of False Negatives.
input_data	List. A list containing the input parameters used for this function call: 'gammas', 'tau', 'w', 'alpha', 'a1', 'a2', and 'kernel'.

Examples

```
library(pye)

# Simulate a sample dataset with covariates
sim_data <- create_sample_with_covariates(
    rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)

# Extract necessary components from the simulated data
df <- sim_data$train_df_scaled
y <- sim_data$y
C <- sim_data$C
```

```
# The combination of biomarkers (regressors) is known when we apply covYI
z <- sim_data$z

#Merge 'df' and 'z' by row names, adding 'z' as a column and preserving row names.
df$ID <- rownames(df)
df1 <- merge(x = df, y = z, by = "ID", all.x = TRUE)
rownames(df1) <- df1$ID
df1$ID <- NULL

# Input some variables
penalty <- "L12"
tau <- 0.1
gammas <- rep(1, length(C))
c <- 0

# Run covYI_KS to evaluate pye for the given gammas and penalty
covYI_result <- covYI_KS(df = df1, z = "z", y = y, C = C, gammas = gammas, tau = tau,
  alpha = 0.5, a1 = 3.7, a2 = 3, penalty = penalty)
print(covYI_result)

# Example with a different penalty
penalty <- "SCAD"

# Run covYI_KS to evaluate pye for the given gammas and penalty
covYI_result <- covYI_KS(df = df1, z = "z", y = y, C = C, gammas = gammas, tau = tau,
  alpha = 0.5, a1 = 3.7, a2 = 3, penalty = penalty)
print(covYI_result)
```

`covYI_KS_estimation` *Estimation of Optimal Covariate Coefficients through Penalized Covariate-Adjusted Youden Index*

Description

This function estimates the optimal values of the 'gammas' coefficients by maximizing the Penalized Youden Index (pye) function. To achieve this, it uses either the monotonic (mmAPG) or non-monotonic (mnmAPG) Accelerated Proximal Gradient algorithms for nonconvex programming.

Usage

```
covYI_KS_estimation(
  df,
  z = "z_hat",
  y = "y",
  C,
  tau,
  w = 0.5,
  penalty = "L1",
```

```
  gamma_start_input = NULL,
  gamma_start_default = "zeros",
  alpha = 0.5,
  a1 = 3.7,
  a2 = 3,
  regressors_gammas = NULL,
  fold = NULL,
  max_iter = 10000,
  max.print = 10,
  trend = "monotone",
  delta = 1e-05,
  max_alpha = 10000,
  stepsizeShrink = 0.8,
  min_alpha = 1e-10,
  convergence_error = 1e-07,
  trace = 1,
  seed = 1,
  kernel = "gaussian",
  run_aauc = FALSE
)
```

Arguments

df	The input dataset as a data frame.
z	A single regressor or a known combination of regressors. It should be a character string specifying the column name in ‘df‘. Default is "z_hat".
y	The target variable, which must be a binomial (0 or 1) variable. It should be a character string specifying the column name in ‘df‘. Default is "y".
C	A character vector of column names from ‘df‘ to be used as covariate variables. It is mandatory and cannot be ‘NULL‘ or empty.
tau	The penalization parameter applied to the covariates. Default is 0, which corresponds to no penalization.
w	A ‘numeric‘ value between 0 and 1 specifying the weight for the Weighted Youden Index. Sensitivity is weighted by ‘w‘ and specificity by ‘1 - w‘. Default is 0.5.
penalty	The type of penalization to apply. Must be one of "L12", "L1", "EN", "SCAD", or "MCP". Default is "L1".
gamma_start_input	A numeric vector of specific starting points for the ‘gammas‘ coefficients. Default is ‘NULL‘.
gamma_start_default	A character string to set the default starting point for ‘gammas‘. If "zeros", it starts with all zero values. If "corr", it starts with the correlation of each regressor with the target variable. Default is "zeros".
alpha	The elastic-net mixing parameter, for use with the "EN" penalty. A value between 0 and 1. Default is 0.5.

a1 The ‘a‘ parameter for the SCAD penalization term. Default is 3.7.

a2 The ‘a‘ parameter for the MCP penalization term. Default is 3.0.

regressors_gammas
A vector containing the true gamma coefficients (if known). Used for diagnostic purposes. Default is ‘NULL‘.

fold numeric. An optional fold number, used when the function is called within a cross-validation loop. This is primarily for tracking and reporting purposes. Default is ‘NULL‘.

max_iter The maximum number of iterations for the optimization algorithms. Default is 10000.

max.print The number of elements to show when printing results. Default is 10.

trend A character string specifying the optimization algorithm to use. If "monotone", the mmAPG algorithm is used. If "nonmonotone", mnmAPG is used. Default is "monotone".

delta The parameter for the convergence condition of the optimization algorithm. Default is 1e-5.

max_alpha The maximum value of the step-size parameter alpha. Default is 10000.

stepsizeShrink The parameter to adjust the step-size in the backtracking line search. Taking values between 0 and 1, the closer to 1, the more accurate the estimation will be. Default is 0.8.

min_alpha The minimum value of the step-size parameter alpha. Default is 1e-10.

convergence_error
The error tolerance for considering the algorithm to have converged. Default is 1e-7.

trace An integer controlling the level of output. 2: visualize all steps, 1: visualize just the final result, 0: visualize nothing. Default is 1.

seed An integer to fix the random seed. Default is 1.

kernel The kernel type to use for the estimation of the density function. Currently only "gaussian" is fully supported. Default is "gaussian".

run_aauc A logical value. If ‘FALSE‘, the aAUC and aYI metrics are not computed to save estimation time. Default is ‘FALSE‘.

Value

A list containing the optimal value of gammas and other key metrics.

gammas_hat_penalty
The vector of optimal gamma coefficients found by the algorithm.

covYI_KS_penalty
The final maximum penalized Youden Index (pye) value.

gr_yi The gradient of the Youden Index with respect to the gammas at the optimal solution.

tau The input penalization parameter.

penalty The type of penalty used.

`gammas_start` The starting point for the gamma coefficients.

`kernel` The kernel type used.

`c_hat` A data frame with the estimated cut-point for each observation.

`z_hat` A data frame with the regressor values used in the calculation.

`y_hat` A data frame with the predicted binary outcomes.

`youden_index` The Youden Index value at the optimal solution.

`sensitivity` The sensitivity (True Positive Rate) at the optimal cut-point.

`specificity` The specificity (True Negative Rate) at the optimal cut-point.

`geometric_mean` The geometric mean of sensitivity and specificity.

`fdr` The False Discovery Rate.

`mcc` The Matthews Correlation Coefficient.

`auc` The Area Under the ROC Curve (AUC).

`aauc` The covariate-adjusted AUC (aAUC). Only computed if 'run_aauc' is 'TRUE'.

`aYI` The adjusted Youden Index (aYI). Only computed if 'run_aauc' is 'TRUE'.

`corrclass` The overall correct classification rate.

`TP` Number of True Positives.

`TN` Number of True Negatives.

`FP` Number of False Positives.

`FN` Number of False Negatives.

`n_gammas` The number of non-zero gamma coefficients in the final estimation.

`n_total_var_gammas`
: The total number of gamma coefficients estimated.

`n_predicted_zeros_gammas`
: The number of estimated gammas that are zero.

`n_predicted_non_zeros_gammas`
: The number of estimated gammas that are non-zero.

`n_caught_gammas`
: Number of true non-zero gammas correctly identified as non-zero (if 'regressors_gammas' is provided).

`n_non_caught_gammas`
: Number of true non-zero gammas incorrectly identified as zero.

`n_caught_zero_gammas`
: Number of true zero gammas correctly identified as zero.

`n_zero_not_caught_gammas`
: Number of true zero gammas incorrectly identified as non-zero.

`input_parameters`
: `character vector`. A list containing the input parameters.

`estimation_time`
: The time taken for the estimation, in minutes.

`niter` The number of iterations performed by the optimization algorithm.

Examples

```
library(pye)
sim_data <- create_sample_with_covariates(
    rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- sim_data$train_df_scaled
y <- sim_data$y
C <- sim_data$C
regressors_gammas <- sim_data$ncovariates

# The combination of biomarkers (regressors) is known when we apply covYI
z <- sim_data$z

#Merge 'df' and 'z' by row names, adding 'z' as a column and preserving row names.
df$ID <- rownames(df)
df1 <- merge(x = df, y = z, by = "ID", all.x = TRUE)
rownames(df1) <- df1$ID
df1$ID <- NULL

tau <- 0.04

covYI_estimation_result <- covYI_KS_estimation(df = df1, z = "z", y = y,
  C = C, tau = tau, penalty = "SCAD", trace = 2,
  gamma_start_default = "zeros", regressors_gammas = regressors_gammas,
  max_iter = 5, run_aauc = TRUE)

print(covYI_estimation_result)
```

`create_data_all` *Generate Correlated Regressors and Covariates*

Description

This function generates synthetic regressors (biomarkers) and covariates using a joint multivariate normal distribution. The correlation structure is constructed via a latent factor model, guaranteeing that the output covariance matrix is positive semi-definite. All generated variables are subsequently transformed to standard normal scores (mean 0, variance 1).

Usage

```
create_data_all(
  rows,
  cols_reg,
  cols_cov,
  max_corr = 0.5,
  mu_reg = rep(0, cols_reg),
  mu_cov = rep(0, cols_cov),
  n_latent_factors = 2,
  seed = 1
)
```

Arguments

rows	integer. Number of observations (rows) to generate.
cols_reg	integer. Number of regressor variables. Must be a multiple of 4 (required by subsequent sampling functions).
cols_cov	integer. Number of covariate variables. Must be a multiple of 4.
max_corr	numeric. Maximum absolute correlation allowed among all generated variables. Actual correlations are randomly drawn from a normal distribution centered at 0, with a standard deviation scaled by max_corr. Must be strictly between 0 and 1.
mu_reg	numeric vector. Mean values for regressor variables. Its length must match cols_reg. Defaults to a zero vector.
mu_cov	numeric vector. Mean values for covariate variables. Its length must match cols_cov. Defaults to a zero vector.
n_latent_factors	integer. Number of latent factors used in the covariance matrix construction. Default is 2. Higher values can lead to more complex correlation structures.
seed	integer. Seed for reproducibility. Default is 1.

Details

The correlation structure for all cols_reg + cols_cov variables is defined using a latent factor model. Variables' loadings onto n_latent_factors latent components are randomly drawn, then clipped to ensure correlations are within [-max_corr, max_corr]. The covariance matrix Σ is derived from these loadings ($L \times L^T$), ensuring it is positive semi-definite. Finally, the diagonal elements of Σ are explicitly set to 1 to enforce unit variance across all generated variables.

The generated multivariate normal variables undergo a rank-preserving transformation to standard normal scores (using pnorm and qnorm) to ensure stable distribution properties.

Value

A list containing two data frames:

regressors	Data frame of standardized normal regressors (mean 0, variance 1).
covariates	Data frame of standardized normal covariates (mean 0, variance 1).

Examples

```
# Simulate 100 observations with 8 regressors and 8 covariates
sim <- create_data_all(
  rows = 100,
  cols_reg = 8,
  cols_cov = 8,
  max_corr = 0.6,
  seed = 123
)

# Inspect the structure of simulated outputs
```

```
str(sim$regressors)
str(sim$covariates)

# Visualize the correlation distribution
cor_mat <- cor(cbind(sim$regressors, sim$covariates))
hist(cor_mat[lower.tri(cor_mat)],
     main = "Histogram of Pairwise Correlations",
     xlab = "Correlation", col = "steelblue")
```

`create_lambda` *Generate a Sequence of Penalty Parameters (λ)*

Description

Creates a sequence of logarithmically or linearly spaced values for the regularization penalty parameter, λ, typically used in penalized regression models like Lasso or Elastic Net. The sequence runs from a specified maximum value (`lmax`) down to a minimum value (`lmin`).

Usage

```
create_lambda(n = 100, lmax = 10, lmin = 1)
```

Arguments

`n`	`integer`. The number of λ values to generate in the sequence. Must be ≥ 2.
`lmax`	`numeric`. The maximum (starting) value for λ. This is usually a value near where all coefficients are zero. Must be $> `lmin`$.
`lmin`	`numeric`. The minimum (ending) value for λ. This is usually a small positive number to ensure model sparsity. Must be ≥ 0.

Details

The function's primary purpose is to generate a sequence on a logarithmic scale, which is standard practice for tuning regularization parameters, as it allows for a finer grid search near zero while still covering large values.

Specifically:

- If $\log(\texttt{lmin}/\texttt{lmax}) \neq 0$, the function generates a geometric progression (logarithmic scale).
- If $\log(\texttt{lmin}/\texttt{lmax}) = 0$ (i.e., $\texttt{lmax} = \texttt{lmin}$), it generates a sequence using an arithmetic progression (linear scale), though typically `lmax` and `lmin` should be different for cross-validation.

Value

A `numeric` vector of length `n` containing the chosen λ values, sorted in descending order (from `lmax` to `lmin`). The sequence is logarithmically spaced unless `lmax` and `lmin` are equal, in which case it behaves like a linear sequence.

Examples

```
lambdas <- create_lambda(n = 100, lmax = 10, lmin = 1)
print(lambdas)
```

`create_sample` *Create Synthetic High-Dimensional Sample with Binary Target*

Description

Creates a synthetic dataset with a binary target variable (`y`) and a set of correlated normal regressors (`X`). This function is useful for testing models in high-dimensional settings, particularly those that do not incorporate covariates.

Usage

```
create_sample(
  rows_train = 50,
  cols = 2000,
  cov = 0.5,
  mu = rep(0, cols),
  rows_test = 1000,
  seed = 1,
  varsN = NULL,
  varsB = NULL,
  varsE = NULL,
  varsP = NULL
)
```

Arguments

`rows_train`	`numeric`. Number of rows for the training sample. Default is 50.
`cols`	`numeric`. Total number of regressor variables in both training and test samples. Must be a multiple of 4. Default is 2000.
`cov`	`numeric`. The correlation coefficient used to define the internal correlation within each of the four block-diagonal structures for generating regressors. Higher values result in more correlated features. Must be between -1 and 1. Default is 0.5.
`mu`	`numeric` vector. Mean vector for the multivariate normal distribution used to generate the regressors. Its length must match `cols`. Default is a vector of zeros of length `cols`.
`rows_test`	`numeric`. Number of rows for the test sample. Default is 1000. It is recommended to create a test sample larger than the training sample.
`seed`	`numeric`. An integer to fix the random seed for reproducibility. Default is 1.

varsN	numeric vector or NULL. A vector of two distinct integers specifying the column indices of the "real regressors" (used in the latent variable z) from the first cols/4 block. If NULL, two are randomly selected. Indices are relative to the full set of cols variables.
varsB	numeric vector or NULL. Indices of "real regressors" from the second cols/4 block.
varsE	numeric vector or NULL. Indices of "real regressors" from the third cols/4 block.
varsP	numeric vector or NULL. Indices of "real regressors" from the fourth cols/4 block.

Details

The function generates cols regressors, all following a standard normal distribution, structured into four equal blocks with internal correlation defined by cov using a block-diagonal covariance matrix.

A latent variable z is created as a linear combination of exactly eight of these regressors (two from each block). The binary target variable y is then generated by comparing z to its 0.7 quantile: $y = 1$ if $z >$ quantile$(z, 0.7)$ and $y = 0$ otherwise. Data is finally split into training and test sets.

Value

A list containing the following elements:

df	A data frame with the generated target variable (y) and all regressors, before scaling.
df_scaled	A list containing the full dataset, with regressors scaled (mean 0, variance 1) and the target y unscaled.
train_df_scaled	The data frame for the training set, with scaled regressors and unscaled target.
test_df_scaled	The data frame for the test set, with scaled regressors and unscaled target.
nregressors	A numeric vector indicating the cols column indices of the 8 selected "real regressors".
regressors	A character vector of the names of the 8 selected "real regressors".
coefficients	A numeric vector of the fixed coefficients used for the "real regressors" in the linear combination to generate z (fixed as c(-16, -4, 12, 8, -8, -12, 4, 16)).
linearformula	A character string representing the linear formula used to generate the latent variable z.
z	A numeric vector of the latent linear predictor values for all observations.
cutoff	A numeric value, the $\mathbf{0.7}$ quantile of z, used as the threshold to binarize y.

Examples

```
library(pye)
# Create a small sample with 20 regressors and specified 'real' variables
df <- create_sample(rows_train = 200, cols = 20, rows_test = 20,
  varsN = c(2, 4), varsB = c(6, 8), varsE = c(12, 14), varsP = c(16, 18))
head(df$df)
```

`create_sample_with_covariates` *Create a Synthetic Data Set with Covariates and Binary Target*

Description

Creates a synthetic data set featuring a binary target variable, a large set of correlated regressors (biomarkers), and a set of correlated covariates. This synthetic environment is designed for testing methods that account for covariates in binary classification, such as the Penalized Youden Index (pye). Regressors and covariates are generated with controlled correlation structures based on a latent factor model.

Usage

```
create_sample_with_covariates(
  rows_train = 50,
  cols = 2000,
  cols_cov = 20,
  max_rho = 0.3,
  mu = rep(0, cols),
  mu_cov = rep(0, cols_cov),
  rows_test = 1000,
  seed = 1,
  varsN = NULL,
  varsB = NULL,
  varsE = NULL,
  varsP = NULL,
  varN_cov = NULL,
  varB_cov = NULL,
  varE_cov = NULL,
  varP_cov = NULL,
  n_latent_factors = 2
)
```

Arguments

`rows_train`	`integer`. Number of observations for the training sample. Default is 50.
`cols`	`integer`. Total number of regressor variables (biomarkers). Must be a multiple of 4. Default is 2000.
`cols_cov`	`integer`. Total number of covariate variables. Must be a multiple of 4. Default is 20. Increase this for high- dimensional covariate settings.
`max_rho`	`numeric`. The maximum correlation coefficient used in the latent factor model for generating variables. Higher values lead to more correlated features. Must be between 0 and 1. Default is 0.3.

`mu` `numeric` vector. Mean vector for the multivariate normal distribution used to generate the regressors. Its length must match `cols`. Default is a vector of zeros.

`mu_cov` `numeric` vector. Mean vector for the multivariate normal distribution used to generate the covariates. Its length must match `cols_cov`. Default is a vector of zeros.

`rows_test` `integer`. Number of observations for the test sample. Default is 1000.

`seed` `integer`. An integer to fix the random seed for reproducibility. Default is 1.

`varsN` `numeric` vector or `NULL`. Indices of the two "real regressors" chosen from the first `cols`/4 block of variables. If `NULL`, two are randomly selected.

`varsB` `numeric` vector or `NULL`. Indices of the two "real regressors" chosen from the second `cols`/4 block of variables. If `NULL`, two are randomly selected.

`varsE` `numeric` vector or `NULL`. Indices of the two "real regressors" chosen from the third `cols`/4 block of variables. If `NULL`, two are randomly selected.

`varsP` `numeric` vector or `NULL`. Indices of the two "real regressors" chosen from the fourth `cols`/4 block of variables. If `NULL`, two are randomly selected.

`varN_cov` `numeric` or `NULL`. Index of the single "real covariate" chosen from the first `cols_cov`/4 block. If `NULL`, one is randomly selected.

`varB_cov` `numeric` or `NULL`. Index of the single "real covariate" chosen from the second `cols_cov`/4 block. If `NULL`, one is randomly selected.

`varE_cov` `numeric` or `NULL`. Index of the single "real covariate" chosen from the third `cols_cov`/4 block. If `NULL`, one is randomly selected.

`varP_cov` `numeric` or `NULL`. Index of the single "real covariate" chosen from the fourth `cols_cov`/4 block. If `NULL`, one is randomly selected.

`n_latent_factors` `integer`. Number of latent factors used in the covariance matrix construction. Default is 2. Higher values create more complex correlation structures.

Details

The binary target variable `y` is generated using a Probit-like model: $y = 1$ if $z \geq$ cutoff, and $y = 0$ otherwise.

The latent linear predictor $\mathbf{z}$ is a linear combination of the few designated "real regressors":

$$\mathbf{z} = \mathbf{X}_{real}\beta_{real}$$

The dynamic cutoff for classification is a linear combination of the few designated "real covariates":

$$textcutoff = \mathbf{C}_{real}\gamma_{real}$$

All generated regressors and covariates are standardized (mean 0, variance 1) and generated with a controlled correlation structure via a latent factor model. The function includes input validation to enforce the block-based structure (e.g., `cols` and `cols_cov` must be multiples of 4).

Value

A `list` containing the following data elements:

`df`	A data frame of the full sample (`rows = rows_train + rows_test`) containing the generated target variable (`y`), all standardized regressors (`X`), and all standardized covariates (`C`).
`train_df_scaled`	The training subset of `df`.
`test_df_scaled`	The test subset of `df`.
`nregressors`	`numeric` vector of the column indices of the selected "real regressors" from the full `cols` set.
`regressors`	`character` vector of the names of the selected "real regressors."
`coefficients_regressors`	`numeric` vector of the coefficients used for the "real regressors" in the latent variable `z`.
`ncovariates`	`numeric` vector of the column indices of the selected "real covariates" from the full `cols_cov` set.
`covariates`	`character` vector of the names of the selected "real covariates."
`coefficients_covariates`	`numeric` vector of the coefficients used for the "real covariates" in the dynamic `cutoff`.
`target`	The name of the target variable (always "`y`").
`linearformula_z`	Character string of the formula used to generate the latent predictor `z`.
`linearformula_cutoff`	Character string of the formula used to generate the dynamic cutoff.
`X`	`character` vector of all regressor variable names.
`y`	`character` string, the name of the target variable.
`C`	`character` vector of all covariate variable names.
`z`	`numeric` vector of the latent linear predictor values for all observations.
`cutoff`	`numeric` vector of the dynamic cutoff values for all observations.

Examples

```
library(pye)
sample_data <- create_sample_with_covariates(cols = 20, cols_cov = 20,
  rows_test = 10,
  varsN = c(2, 4), varsB = c(6, 8), varsE = c(12, 14), varsP = c(16, 18),
  varN_cov = 2, varB_cov = 6, varE_cov = 15, varP_cov = 19)
names(sample_data)
head(sample_data$df)
```

`MCP_function` *Minimax Concave Penalty (MCP) Function Value*

Description

Calculates the value of the Minimax Concave Penalty (MCP) function, a regularization penalty often used in sparse regression models.

Usage

```
MCP_function(betas, lambda, a)
```

Arguments

`betas`	`numeric` vector. The vector of coefficients (e.g., β) on which to apply the MCP penalization.
`lambda`	`numeric`. The penalty parameter ($\lambda > 0$).
`a`	`numeric`. The hyperparameter of the MCP penalization ($a > 1$). This controls the concavity and asymptote of the penalty.

Details

The MCP penalty aims to apply an unbiased penalty to large coefficients while still performing continuous shrinkage, similar to SCAD. The penalty for a single coefficient $|\beta|$ is defined as:

- $\lambda|\beta| - \frac{\beta^2}{2a}$ if $|\beta| \leq a\lambda$
- $\frac{a\lambda^2}{2}$ if $|\beta| > a\lambda$

This function computes the sum of this penalty over all elements in `betas` using an assumed internal function, `single_MCP_function`.

Value

A single `numeric` value representing the sum of the MCP penalties applied to each element in the `betas` vector.

Examples

```
library(pye)

betas <- seq(0, 2, by = 0.2)
lambda <- 0.5
a <- 3.0
MCP_function(betas = betas, lambda = lambda, a = a)
```

mmAPG *Monotone Accelerated Proximal Gradient (APG) method*

Description

Implements the Monotone Accelerated Proximal Gradient (APG) method, inspired by Li and Lin (2015), for the optimization problem within the Penalized Youden index Estimator (pye) framework.

This variant is tailored for pye with key modifications: 1. Initialization: Adjusted for sparse starting points. 2. Selection Rule: Implements a non-reversible variable selection. Once a parameter is set to zero (after a warm-up phase), it is permanently excluded from the active set to enforce sparsity.

Usage

```
mmAPG(
  x0,
  c_pos = NULL,
  delta_fx,
  proxx,
  Fx,
  lambda = NULL,
  penalty = NULL,
  fold = NULL,
  stepsizeShrink = 0.8,
  max_alpha = 10000,
  min_alpha = 1e-10,
  delta = 1e-05,
  trace = 2,
  seed = 1,
  max_iter = 10000,
  convergence_error = 1e-07,
  zeros_stay_zeros_from_iteration = 20,
  max.print = 10
)
```

Arguments

x0	A named numeric vector representing the starting point for the optimization. It is highly recommended to use the zero vector to encourage a sparse solution.
c_pos	The index position of the constant term (intercept) in the $x0$ vector. Use $NULL$ if no constant is included. Default is $NULL$.
delta_fx	A function that computes the gradient of the smooth (loss) component, $f(x)$, of the objective function.
proxx	A function that computes the proximal operator related to the non-smooth (penalty) component, $g(x)$.
Fx	A function that computes the full objective function, $F(x) = f(x) + g(x)$.

`lambda` The penalization parameter (λ) related to $g(x)$. Used primarily for tracing and reporting. Default is $NULL$.

`penalty` The type of penalty (e.g., "L1", "SCAD"). Used for tracing and reporting. Default is $NULL$.

`fold` An optional numeric fold number, typically used when the function is called within a cross-validation loop. Default is $NULL$.

`stepsizeShrink` The shrinking factor for the step-size α in the backtracking line search. Must be in $(0, 1)$. A value closer to 1 increases accuracy but slows convergence. Default is 0.8.

`max_alpha` The maximum value allowed for the step-size α. Default is 10000.

`min_alpha` The minimum value allowed for the step-size α. If $\alpha_x + \alpha_y$ falls below this threshold, the algorithm is considered converged. Default is $1e-10$.

`delta` The convergence criterion parameter used in the line-search condition: $F(z_k) \leq F(y_k) - \delta\|z_k - y_k\|_2^2$. Default is $1e-5$.

`trace` An integer to control output verbosity: 2 = print details for every step; 1 = print only final result and convergence message; 0 = no output. Default is 2.

`seed` Numeric seed for reproducibility. Default is 1.

`max_iter` The maximum number of iterations. Default is 10000.

`convergence_error`
The absolute tolerance used for the 'minimum change' convergence criterion: convergence is assumed if the objective function changes by less than this value for 5 consecutive iterations. Default is $1e-7$.

`zeros_stay_zeros_from_iteration`
The iteration number after which any coefficient that becomes zero is permanently fixed to zero. This enforces sparsity. Default is 20.

`max.print` The number of non-zero elements to display in the trace output. Default is 10.

Value

A $list$ containing the following elements:

`x1` The final estimated vector of parameters, including the constant.

`tot_iters` The total number of main algorithm iterations performed.

`backtrack_iters`
The total number of backtracking steps executed.

`estimation_time`
The total time taken for the estimation (in minutes).

`reason_for_exit`
The convergence reason ("max_iter", "min_change", or "min_alpha").

References

Li, C., & Lin, Z. (2015). Accelerated Proximal Gradient Methods for Nonconvex Programming. *Advances in Neural Information Processing Systems*, *28*.

Examples

```
library(pye)
sim_data <- create_sample_with_covariates(
    rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- sim_data$train_df_scaled
X <- sim_data$X
y <- sim_data$y
ID <- rownames(df)
df1 <- cbind(ID, df[, c(y, X)], drop = FALSE)
penalty <- "L12"
c_zero_fixed <- TRUE
lambda <- 0.1
kernel <- "gaussian"
if (penalty == "SCAD") {a=3.7} else {a=3.0}
prox_penalty <- get(paste0("proximal_operator_", penalty))

#wrappers
delta_fx <- function(x) {
  if (c_zero_fixed == TRUE) {
    x[names(x) == "c"] <- 0
  }

  result <- -pye_KS(df = df1[, names(df1) != "ID", drop = FALSE],
    X = X[X %in% names(x)], y = y, betas = x[!(names(x) == "c")],
    lambda = lambda, c = x[(names(x) == "c")], kernel = kernel,
    alpha = 0.5, a1 = 3.7, a2 = 3, penalty = penalty)$gr_yi
  if (c_zero_fixed == TRUE) {
    result[names(result) == "c"] <- 0
  }
  return(result)
}

proxx <- function(x, eta) {
  if (c_zero_fixed == TRUE) {
    x[names(x) == "c"] <- 0
  }
  result <- c(prox_penalty(betas = x[!(names(x) == "c")], lambda = eta * lambda,
    alpha = 0.5, a = a), x[(names(x) == "c")])
  return(result)
}

Fx <- function(x) {
  if (c_zero_fixed == TRUE) {
    x[(names(x) == "c")] <- 0
  }
  result <- -getElement(pye_KS(df = df1[, names(df1) != "ID", drop = FALSE],
    X = X[X %in% names(x)], y = y, betas = x[!(names(x) == "c")],
    lambda = lambda, c = x[(names(x) == "c")], kernel = kernel,
    alpha = 0.5, a1 = 3.7, a2 = 3, penalty = penalty), paste0("pye_", penalty))
  return(result)
}
```

```
#starting point:
x0 <- c(rep(0, length(X)), 0) #the first zero is the constant term
names(x0) <- c(X, "c")
position_c <- which(names(x0) == "c")

estim <- mmAPG(x0 = x0, c_pos = position_c, delta_fx = delta_fx, proxx = proxx, Fx = Fx,
     lambda = lambda, penalty = penalty, max_iter = 8, trace = 2)
print(estim)
```

---

mnmAPG *Non-Monotone Accelerated Proximal Gradient (APG) method*

Description

Implements the Non-Monotone Accelerated Proximal Gradient (APG) method, inspired by Li and Lin (2015), for the optimization problem within the Penalized Youden index Estimator (pye) framework.

This variant is tailored for pye with key modifications: 1. Non-Monotone Condition: Uses the non-monotone line search condition, which generally allows for faster convergence by accepting temporary increases in the objective function. 2. Non-Reversible Selection: Implements a non-reversible variable selection rule. Once a parameter is set to zero (after a warm-up phase), it is permanently excluded from the active set to enforce sparsity.

Usage

```
mnmAPG(
  x0,
  c_pos = NULL,
  delta_fx,
  proxx,
  Fx,
  lambda = NULL,
  penalty = NULL,
  fold = NULL,
  stepsizeShrink = 0.8,
  eta = 0.8,
  max_alpha = 10000,
  min_alpha = 1e-10,
  delta = 1e-05,
  trace = 2,
  seed = 1,
  max_iter = 10000,
  convergence_error = 1e-07,
  zeros_stay_zeros_from_iteration = 20,
  max.print = 10
)
```

Arguments

x0	A named numeric vector representing the starting point for the optimization. It is highly recommended to use the zero vector to encourage a sparse solution.
c_pos	The index position of the constant term (intercept) in the $x0$ vector. Use $NULL$ if no constant is included. Default is $NULL$.
delta_fx	A function that computes the gradient of the smooth (loss) component, $f(x)$, of the objective function.
proxx	A function that computes the proximal operator related to the non-smooth (penalty) component, $g(x)$.
Fx	A function that computes the full objective function, $F(x) = f(x) + g(x)$.
lambda	The penalization parameter (λ) related to $g(x)$. Used primarily for tracing and reporting. Default is $NULL$.
penalty	The type of penalty (e.g., "L1", "SCAD"). Used for tracing and reporting. Default is $NULL$.
fold	An optional numeric fold number, typically used when the function is called within a cross-validation loop. Default is $NULL$.
stepsizeShrink	The shrinking factor for the step-size α in the backtracking line search. Must be in $(0, 1)$. A value closer to 1 increases accuracy but slows convergence. Default is 0.8.
eta	The non-monotonicity control parameter $\eta \in [0, 1]$. It dictates how much the current objective function value can deviate from the best previous value. Li and Lin (2015) suggest 0.8. Default is 0.8.
max_alpha	The maximum value allowed for the step-size α. Default is 10000.
min_alpha	The minimum value allowed for the step-size α. If $\alpha_x + \alpha_y$ falls below this threshold, the algorithm is considered converged. Default is $1e - 10$.
delta	The convergence criterion parameter used in the line-search condition (see $mmAPG$). Default is $1e - 5$.
trace	An integer to control output verbosity: 2 = print details for every step; 1 = print only final result and convergence message; 0 = no output. Default is 2.
seed	Numeric seed for reproducibility. Default is 1.
max_iter	The maximum number of iterations. Default is 10000.
convergence_error	The absolute tolerance used for the 'minimum change' convergence criterion: convergence is assumed if the objective function changes by less than this value for 5 consecutive iterations. Default is $1e - 7$.
zeros_stay_zeros_from_iteration	The iteration number after which any coefficient that becomes zero is permanently fixed to zero. This enforces sparsity. Default is 5.
max.print	The number of non-zero elements to display in the trace output. Default is 10.

Value

A $list$ containing the following elements:

x1	The final estimated vector of parameters, including the constant.
tot_iters	The total number of main algorithm iterations performed.
backtrack_iters	The total number of backtracking steps executed.
estimation_time	The total time taken for the estimation (in minutes).
reason_for_exit	The convergence reason ("max_iter", "min_change", or "min_alpha").

References

Li, C., & Lin, Z. (2015). Accelerated Proximal Gradient Methods for Nonconvex Programming. *Advances in Neural Information Processing Systems*, *28*.

Examples

```
library(pye)
sim_data <- create_sample_with_covariates(
    rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- sim_data$train_df_scaled
X <- sim_data$X
y <- sim_data$y
ID <- rownames(df)
df1 <- cbind(ID, df[, c(y, X)], drop = FALSE)
penalty <- "L12"
c_zero_fixed <- TRUE
lambda <- 0.1
kernel <- "gaussian"
if (penalty == "SCAD") {a=3.7} else {a=3.0}
prox_penalty <- get(paste0("proximal_operator_", penalty))

#wrappers
delta_fx <- function(x) {
 if (c_zero_fixed == TRUE) {
   x[names(x) == "c"] <- 0
  }
  result <- -pye_KS(df = df1[, names(df1) != "ID", drop = FALSE],
    X = X[X %in% names(x)], y = y, betas = x[!(names(x) == "c")],
     lambda = lambda, c = x[(names(x) == "c")], kernel = kernel,
     alpha = 0.5, a1 = 3.7, a2 = 3, penalty = penalty)$gr_yi
  if (c_zero_fixed == TRUE) {
    result[names(result) == "c"] <- 0
  }
  return(result)
}

proxx <- function(x, eta) {
```

```
 if (c_zero_fixed == TRUE) {
    x[names(x) == "c"] <- 0
  }
  result <- c(prox_penalty(betas = x[!(names(x) == "c")], lambda = eta * lambda,
    alpha = 0.5, a = a), x[(names(x) == "c")])
  return(result)
}

Fx <- function(x) {
  if (c_zero_fixed == TRUE) {
    x[(names(x) == "c")] <- 0
  }
  result <- -getElement(pye_KS(df = df1[, names(df1) != "ID", drop = FALSE],
    X = X[X %in% names(x)], y = y, betas = x[!(names(x) == "c")],
    lambda = lambda, c = x[(names(x) == "c")], kernel = kernel,
    alpha = 0.5, a1 = 3.7, a2 = 3, penalty = penalty), paste0("pye_", penalty))
  return(result)
}

#starting point:
x0 <- c(rep(0, length(X)), 0) #the first zero is the constant term
names(x0) <- c(X, "c")
position_c <- which(names(x0) == "c")

estim <- mnmAPG(x0 = x0, c_pos = position_c, delta_fx = delta_fx, proxx = proxx, Fx = Fx,
     lambda = lambda, penalty = penalty, max_iter = 10, trace = 2)
print(estim)
```

`model_simulation_study` *Simulation Study for Penalized Models on Real Data*

Description

This function performs a simulation study by repeatedly splitting a dataset into training and testing sets, estimating a competitor method (e.g., penalized logistic regression, penalized SVM, or penalized AUC-based method) on the training data, and evaluating its classification performance on the test set. Optionally, it can incorporate covariate information (covYI) for the cut-off point.

Usage

```
model_simulation_study(
  n = 1000,
  df,
  X = NULL,
  y = "y",
  C = NULL,
  model_estimation_function,
```

```
  model_prediction_function,
  model_type,
  lambda,
  tau = 0,
  w_g = 0.5,
  train_data_proportion = 0.7,
  trace = 1,
  alpha = 0.5,
  regressors_betas = NULL,
  used_cores = 1,
  scaling = FALSE,
  c_function_of_covariates = FALSE,
  alpha_g = 0.5,
  penalty_g = "L1",
  kernel_g = "gaussian",
  a1_g = 3.7,
  a2_g = 3,
  trend_g = "monotone",
  gamma_start_input = NULL,
  gamma_start_default = "zeros",
  regressors_gammas = NULL,
  max_iter_g = 10000,
  delta_g = 1e-05,
  max_alpha_g = 10000,
  stepsizeShrink_g = 0.8,
  min_alpha_g = 1e-12,
  convergence_error_g = 1e-07,
  run_aauc = FALSE,
  log_file = "log_sim_Other_real.txt"
)
```

Arguments

n	Integer. The number of simulation experiments to run. Default is 1000.
df	A data frame containing the complete dataset, including the target variable, regressors, and optional covariates.
X	Character vector. Column names from 'df' to be used as regressors in the model estimation. It can be a data frame (whose column names will be extracted) or a character vector. Defaults to all columns in 'df' not specified as 'y' or 'C'.
y	Character string. The column name in 'df' representing the binary target variable (0 or 1). It can be a data frame (whose first column name will be extracted) or a character string. Default is "y".
C	Character vector or 'NULL'. Column names from 'df' to be used as covariate variables in covYI. It can be a data frame or a character vector. Default is 'NULL'.
model_estimation_function	Function. The function to use for model estimation. Expected values are "plr_estimation", "psvm_estimation", "AucPR_estimation".

`model_prediction_function`
: Function. The function to use for model prediction. Expected values are "plr_predict", "psvm_predict", "AucPR_predict".

`model_type`
: Character string. The specific model to use in the estimation. Examples include "logLasso", "logElasticNet", "logSCAD", "logMCP", "SCADSVM", "ElasticSCADSVM", "l1SVM", "enSVM", "AucPR_L1", "AucPR_EN".

`lambda`
: Numeric. The penalization parameter for the regressors ‘X‘. Must be a single non-negative value.

`tau`
: Numeric. The penalization parameter for the covariates ‘C‘ in covYI. Default is 0, indicating no penalization. Must be a single non-negative value.

`w_g`
: Numeric. A value between 0 and 1 specifying the weight for the Weighted Youden Index in covYI. Sensitivity is weighted by ‘w_g‘ and specificity by ‘1 - w_g‘. Default is 0.5 (which corresponds to the standard Youden Index).

`train_data_proportion`
: Numeric. A numeric value between 0 and 1. The proportion of the total observations in ‘df‘ to be used for the training set in each simulation split. The split is performed using stratified sampling on ‘y‘ to maintain class balance. Default is 0.7 (70% training, 30% testing).

`trace`
: Integer (0, 1, or 2). Controls the verbosity of the output. 2 for all steps, 1 for main results, 0 for no output. Default is 1.

`alpha`
: Numeric. A value between 0 and 1, the elastic-net mixing parameter for the primary model’s penalty. Default is 0.5.

`regressors_betas`
: numeric vector. An optional vector containing the "true" beta coefficients, if known. This is used to calculate additional performance measures related to variable selection accuracy. Default is ‘NULL‘.

`used_cores`
: Integer. Number of CPU cores to use for parallel processing. If 0, it automatically uses 70% of available physical cores. If 1, no parallelization is adopted. Default is 1.

`scaling`
: Logical. If ‘TRUE‘, the dataset ‘df‘ is scaled (mean 0, variance 1) before processing. Default is ‘FALSE‘.

`c_function_of_covariates`
: Logical. If ‘TRUE‘, ‘covYI‘ is used to estimate the cut-off point as a function of the covariate information ‘C‘. If ‘FALSE‘, the covariate information is ignored for the cutoff. Default is ‘FALSE‘.

`alpha_g`
: Numeric. A value between 0 and 1, the elastic-net mixing parameter for the ‘covYI‘ penalty. Default is 0.5.

`penalty_g`
: Character string. The penalty type for ‘covYI‘. To be chosen among "L12", "L1", "EN", "SCAD", "MCP". Default is "L1".

`kernel_g`
: Character string. The kernel type to use for the estimation of the density function in ‘covYI‘. (Note: tested primarily for "gaussian"). Default is "gaussian".

`a1_g`
: Numeric. The ‘a‘ parameter for SCAD and MCP penalties in ‘covYI‘. Default is 3.7.

`a2_g`
: Numeric. The ‘b‘ parameter for the MCP penalty in ‘covYI‘. Default is 3.0.

`trend_g` Character string. For ‘covYI‘ optimization. If "monotone", mmAPG is used; if "nonmonotone", mnmAPG is used. Default is "monotone".

`gamma_start_input`
Numeric vector or ‘NULL‘. A specific starting point for gamma coefficients in ‘covYI‘. Default is ‘NULL‘, meaning no custom starting point is provided.

`gamma_start_default`
Character string. Sets the default starting point of gamma coefficients if ‘gamma_start_input‘ is ‘NULL‘. If "zeros", it starts with all zero values; if "corr", it starts with the correlation of every covariate with the target variable. Default is "zeros".

`regressors_gammas`
Numeric vector or ‘NULL‘. A vector containing the indices of the true non-zero gamma coefficients (if known), used for performance evaluation of variable selection. Default is ‘NULL‘.

`max_iter_g` Integer. Maximum number of iterations for the ‘covYI‘ optimization algorithm (mmAPG and mnmAPG). Default is 10000.

`delta_g` Numeric. Parameter for the convergence condition of the ‘covYI‘ optimization algorithm. Default is 1e-5.

`max_alpha_g` Numeric. Maximum value of the step-size parameter alpha in ‘covYI‘’s backtracking line-search. Default is 10000.

`stepsizeShrink_g`
Numeric. Parameter to adjust the step-size in the backtracking line-search for ‘covYI‘ optimization. Takes values between 0 and 1. Closer to 1 implies more accurate estimation but longer computation time. Default is 0.8.

`min_alpha_g` Numeric. Minimum value of the step-size parameter alpha in ‘covYI‘’s backtracking line-search. Default is 1e-12.

`convergence_error_g`
Numeric. The error threshold to accept for considering the ‘covYI‘ algorithm converged. Default is 1e-7.

`run_aauc` Logical. If ‘FALSE‘, the aAUC and aYI measures are not computed, which can save estimation time if not requested. Default is ‘FALSE‘.

`log_file` Character. Path to a file for logging output from parallel workers. If ‘NULL‘, output goes to the console. Default is "log_sim_Other_real.txt".

Value

A list containing the aggregated classification measures and other results related to every simulation experiment.

`model_type` Character string. The type of model used for estimation.

`simulation_time`
Numeric. Total time taken for the entire simulation study in minutes.

`estimation_time_original_method`
Numeric. Mean estimation time for the primary model across all simulations.

`estimation_time_covYI`
Numeric. Mean estimation time for the covYI method across all simulations (0 if ‘c_function_of_covariates‘ is ‘FALSE‘).

`used_cores` Integer. The number of CPU cores used for parallelization.

`n` Integer. The number of simulation experiments performed.

`lambda` Numeric. The penalization parameter for the primary model.

`tau` Numeric. The penalization parameter for the covYI method.

`auc` Matrix (n x 2). AUC values for training and testing sets across all simulations for the primary model.

`youden_index` Matrix (n x 2). Youden's Index values for training and testing sets across all simulations for the primary model.

`sensitivity` Matrix (n x 2). Sensitivity values for training and testing sets across all simulations for the primary model.

`specificity` Matrix (n x 2). Specificity values for training and testing sets across all simulations for the primary model.

`geometric_mean` Matrix (n x 2). Geometric Mean values for training and testing sets across all simulations for the primary model.

`fdr` Matrix (n x 2). False Discovery Rate values for training and testing sets across all simulations for the primary model.

`mcc` Matrix (n x 2). Matthews Correlation Coefficient values for training and testing sets across all simulations for the primary model.

`corrclass` Matrix (n x 2). Correct Classification Rate values for training and testing sets across all simulations for the primary model.

`auc_covYI`, `aauc_covYI`, `aYI_covYI`, `youden_index_covYI`, `sensitivity_covYI`, `specificity_covYI`, `geometric_mean_covYI`, `fdr_covYI`, `mcc_covYI`, `corrclass_covYI`
Matrices (n x 2). Classification measures (AUC, aAUC, aYI, Youden's Index, Sensitivity, Specificity, Geometric Mean, FDR, MCC, Correct Classification Rate) for training and testing sets across all simulations for the covYI method (only if 'c_function_of_covariates' is 'TRUE').

`n_total_var_betas`
Matrix (n x 1). Total number of regressors for betas in each simulation.

`n_predicted_zeros_betas`
Matrix (n x 1). Number of predicted zero beta coefficients in each simulation.

`n_predicted_non_zeros_betas`
Matrix (n x 1). Number of predicted non-zero beta coefficients in each simulation.

`n_caught_betas` Matrix (n x 1). Number of true non-zero beta coefficients correctly identified as non-zero in each simulation.

`n_non_caught_betas`
Matrix (n x 1). Number of true non-zero beta coefficients incorrectly identified as zero in each simulation.

`n_caught_zero_betas`
Matrix (n x 1). Number of true zero beta coefficients correctly identified as zero in each simulation.

`n_zero_not_caught_betas`
Matrix (n x 1). Number of true zero beta coefficients incorrectly identified as non-zero in each simulation.

`n_total_var_gammas`
: Matrix (n x 1). Total number of covariates for gammas in each simulation (only if ‘c_function_of_covariates‘ is ‘TRUE‘).

`n_predicted_zeros_gammas`
: Matrix (n x 1). Number of predicted zero gamma coefficients in each simulation (only if ‘c_function_of_covariates‘ is ‘TRUE‘).

`n_predicted_non_zeros_gammas`
: Matrix (n x 1). Number of predicted non-zero gamma coefficients in each simulation (only if ‘c_function_of_covariates‘ is ‘TRUE‘).

`n_caught_gammas`
: Matrix (n x 1). Number of true non-zero gamma coefficients correctly identified as non-zero in each simulation (only if ‘c_function_of_covariates‘ is ‘TRUE‘).

`n_non_caught_gammas`
: Matrix (n x 1). Number of true non-zero gamma coefficients incorrectly identified as zero in each simulation (only if ‘c_function_of_covariates‘ is ‘TRUE‘).

`n_caught_zero_gammas`
: Matrix (n x 1). Number of true zero gamma coefficients correctly identified as zero in each simulation (only if ‘c_function_of_covariates‘ is ‘TRUE‘).

`n_zero_not_caught_gammas`
: Matrix (n x 1). Number of true zero gamma coefficients incorrectly identified as non-zero in each simulation (only if ‘c_function_of_covariates‘ is ‘TRUE‘).

`betas_times_selected`
: Matrix (p x 1). For each regressor, the number of times its coefficient was estimated as non-zero across all simulations. ‘p‘ is the number of regressors in ‘X‘.

`gammas_times_selected`
: Matrix (q x 1). For each covariate, the number of times its coefficient was estimated as non-zero across all simulations (only if ‘c_function_of_covariates‘ is ‘TRUE‘). ‘q‘ is the number of covariates in ‘C‘.

`roc_spec_points`
: Numeric vector. The fixed specificity points used for extracting ROC curve sensitivity values.

`roc_sens_points_train`
: List of numeric vectors. Sensitivity points for ROC curves on training data for each simulation, corresponding to ‘roc_spec_points‘.

`roc_sens_points_test`
: List of numeric vectors. Sensitivity points for ROC curves on testing data for each simulation, corresponding to ‘roc_spec_points‘.

`regressors_betas`
: Numeric vector or ‘NULL‘. The input ‘regressors_betas‘ parameter, indicating true non-zero beta indices.

`regressors_gammas`
: Numeric vector or ‘NULL‘. The input ‘regressors_gammas‘ parameter, indicating true non-zero gamma indices.

`input_parameters`
: `character vector`. A list containing the input parameters.

c_function_of_covariates
: Logical. The input ‘c_function_of_covariates‘ parameter.

run_aauc
: Logical. The input ‘run_aauc‘ parameter.

betas
: List of numeric vectors. The estimated beta coefficients from each simulation run. Each element of the list is a named numeric vector of coefficients for that simulation.

gammas
: List of numeric vectors. The estimated gamma coefficients from each simulation run (only if ‘c_function_of_covariates‘ is ‘TRUE‘). Each element of the list is a named numeric vector of coefficients for that simulation, including the ’const’ term.

Examples

```
library(pye)
sim_data <- create_sample_with_covariates(
    rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- sim_data$train_df_scaled
X <- sim_data$X
y <- sim_data$y
C <- sim_data$C
regressors_betas <- sim_data$nregressors
regressors_gammas <- sim_data$ncovariates
n <- 4 #number of simulations
c_function_of_covariates <- TRUE

#Possible choices for "method":
#"logLasso", "logElasticNet", "logSCAD", "logMCP",  "SCADSVM", "ElasticSCADSVM",
#"l1SVM", "enSVM", "AucPR_L1", "AucPR_EN"
method <- "logLasso"
model_estimation_function <- plr_estimation
model_prediction_function <- plr_predict

# Run a small simulation study
results_study <- model_simulation_study(
  n = n, # Small number of simulations for example
  df = df,
  X = X,
  y = y,
  C = C,
  lambda = 0.1,
  tau = 0.05,
  model_estimation_function = model_estimation_function,
  model_prediction_function = model_prediction_function,
  model_type = method,
  trace = 1,
  gamma_start_default = "zeros",
  alpha_g = 0.5,
  penalty_g = "L12", # Options: "L12", "L1", "EN", "SCAD", "MCP"
  used_cores = 1,
  c_function_of_covariates = c_function_of_covariates,
  regressors_betas = regressors_betas,
```

```
  regressors_gammas = regressors_gammas,
  max_iter_g = 8 #<---- Reduced to speed up estimation
)

# Print some of the results
cat("Simulation Time: ", format(results_study$simulation_time, digits = 4))
cat("\nCCR first-step method:\n")
print(results_study$corrclass)
cat("Betas Times Selected:\n")
print(results_study$betas_times_selected[results_study$betas_times_selected > 1])
if (results_study$input_parameters$c_function_of_covariates) {
  cat("CCR covYI:\n")
  print(results_study$corrclass_covYI)
  cat("\nGammas Times Selected:\n")
  print(results_study$gammas_times_selected[results_study$gammas_times_selected > 1])
}
```

`plr_compute_cv` *Cross-Validation for Optimal glmnet Regularization Parameter Selection*

Description

Performs k-fold cross-validation to select optimal penalization parameters for penalized logistic regression models. This function supports models fitted via the ‘glmnet‘ and ‘ncvreg‘ packages (Lasso, Elastic-Net, SCAD, MCP).

The primary goal is to identify the optimal ‘lambda‘ for the main regressors (‘X‘). Optionally, if ‘c_function_of_covariates‘ is ‘TRUE‘, the function can also perform a search for the optimal ‘tau‘ parameter for a set of covariates (‘C‘) used to model a subject-specific cut-off point via the ‘covYI‘ method.

The hyperparameter tuning can be done in two ways: 1. Simultaneous: A grid search over all combinations of ‘lambda‘ and ‘tau‘ is performed. 2. Sequential: A two-step process where the best ‘lambda‘ is first selected (based on ‘measure_to_select_lambda‘), and then, using that optimal ‘lambda‘, the best ‘tau‘ is determined.

Usage

```
plr_compute_cv(
  n_folds,
  df,
  X = NULL,
  y = "y",
  C = NULL,
  lambda,
  tau = 0,
  w_g = 0.5,
  c_to_use = "Youden",
```

```
  alpha = 0.5,
  regressors_betas = NULL,
  trace = 1,
  model_type = c("logLasso", "logElasticNet", "logSCAD", "logMCP"),
  seed = 1,
  used_cores = 1,
  scaling = FALSE,
  c_function_of_covariates = FALSE,
  simultaneous = FALSE,
  measure_to_select_lambda = "ccr",
  alpha_g = 0.5,
  penalty_g = "L1",
  kernel_g = "gaussian",
  a1_g = 3.7,
  a2_g = 3,
  trend_g = "monotone",
  gamma_start_input = NULL,
  gamma_start_default = "zeros",
  regressors_gammas = NULL,
  max_iter_g = 10000,
  delta_g = 1e-05,
  max_alpha_g = 10000,
  stepsizeShrink_g = 0.8,
  min_alpha_g = 1e-12,
  convergence_error_g = 1e-07,
  run_aauc = FALSE,
  log_file = "log_logistic_models.txt"
)
```

Arguments

n_folds	Integer. The number of folds for cross-validation. Must be at least 2.
df	A ‘data.frame‘ containing the complete dataset, including predictors, the target variable, and any covariates.
X	A character vector of column names from ‘df‘ to be used as regressors (predictors). Defaults to all columns in ‘df‘ that are not ‘y‘ or ‘C‘.
y	A character string specifying the column name of the binary target variable (must be coded as 0 and 1). Default is ‘"y"‘.
C	A character vector of column names from ‘df‘ to be used as covariates for ‘covYI‘. Default is ‘NULL‘.
lambda	A numeric vector of one or more non-negative penalization parameters for the regressors ‘X‘.
tau	A numeric vector of one or more non-negative penalization parameters for the covariates ‘C‘. Used only if ‘c_function_of_covariates‘ is ‘TRUE‘. Default is 0.
w_g	A ‘numeric‘ value between 0 and 1 specifying the weight for the Weighted Youden Index in covYI. Sensitivity is weighted by ‘w_g‘ and specificity by ‘1 - w_g‘. Default is 0.5 (which corresponds to the standard Youden Index).

`c_to_use` character or numeric value specifying the classification cut-point used to convert continuous predictions into binary outcomes. If set to ‘"Youden"‘, the function computes the optimal cut-point using Youden’s Index. If a numeric value is provided, it is used directly as the threshold. Default is "Youden".

`alpha` A numeric value between 0 and 1 for the Elastic-Net mixing parameter in ‘glmnet‘. ‘alpha = 1‘ is the lasso (default), and ‘alpha = 0‘ is ridge regression.

`regressors_betas` numeric vector. An optional vector containing the "true" beta coefficients, if known. This is used to calculate additional performance measures related to variable selection accuracy. Default is ‘NULL‘.

`trace` Integer (0, 1, or 2). Controls the verbosity of the output. ‘0‘ = no output, ‘1‘ = summary results (default), ‘2‘ = detailed step-by-step output.

`model_type` A character string specifying the penalized logistic regression model to use. Must be one of: ‘"logLasso"‘, ‘"logElasticNet"‘, ‘"logSCAD"‘, or ‘"logMCP"‘.

`seed` Integer. A random seed for ensuring reproducibility of the fold assignments. Default is 1.

`used_cores` Integer. The number of CPU cores to use for parallel processing of folds. If ‘1‘ (default), execution is sequential.

`scaling` Logical. If ‘TRUE‘, the predictor and covariate columns of ‘df‘ are standardized before model fitting. Default is ‘FALSE‘.

`c_function_of_covariates` Logical. If ‘TRUE‘, the cut-off point is estimated as a function of covariates ‘C‘ using the ‘covYI‘ method. Default is ‘FALSE‘.

`simultaneous` Logical. If ‘c_function_of_covariates‘ is ‘TRUE‘, this determines the tuning strategy. If ‘TRUE‘, a grid search over ‘lambda‘ and ‘tau‘ is performed. If ‘FALSE‘ (default), a sequential two-step search is performed.

`measure_to_select_lambda` Character. The performance metric used to select the optimal ‘lambda‘ in the first step of a sequential search (when ‘simultaneous = FALSE‘). Must be one of ‘"auc"‘, ‘"aauc"‘, ‘"aYI"‘, ‘"yi"‘, ‘"sen"‘, ‘"spc"‘, ‘"gm"‘, ‘"fdr"‘, ‘"mcc"‘, or ‘"ccr"‘. Default is ‘"ccr"‘ (correct classification rate).

`alpha_g` Numeric. The Elastic-Net mixing parameter for the ‘covYI‘ model. Default is 0.5.

`penalty_g` Character. The penalty type for ‘covYI‘. Must be one of ‘"L12"‘, ‘"L1"‘, ‘"EN"‘, ‘"SCAD"‘, ‘"MCP"‘. Default is ‘"L1"‘.

`kernel_g` Character. The kernel for density estimation in ‘covYI‘. Default is ‘"gaussian"‘.

`a1_g, a2_g` Numeric. Regularization parameters for SCAD and MCP penalties in ‘covYI‘. Defaults are 3.7 and 3.0, respectively.

`trend_g` Character. Optimization algorithm for ‘covYI‘: ‘"monotone"‘ (mmAPG) or ‘"nonmonotone"‘ (mnmAPG). Default is ‘"monotone"‘.

`gamma_start_input` An optional numeric vector for the starting coefficients (‘gammas‘) in ‘covYI‘.

`gamma_start_default` Character. Default starting point for ‘gammas‘ if ‘gamma_start_input‘ is ‘NULL‘. ‘"zeros"‘ (default) or ‘"corr"‘.

`regressors_gammas`
: An optional numeric vector of true gamma coefficients for simulation/evaluation. Default is 'NULL'.

`max_iter_g`
: Integer. Maximum iterations for 'covYI' optimization. Default is 10000.

`delta_g`, `max_alpha_g`, `stepsizeShrink_g`, `min_alpha_g`, `convergence_error_g`
: Numeric. Advanced optimization parameters for 'covYI'. See 'covYI_KS_estimation' for details.

`run_aauc`
: Logical. If 'FALSE' (default), the adjusted-AUC (aAUC) and adjusted-YI (aYI) are not computed to save time.

`log_file`
: Character. Path to a file for logging output from parallel workers. If 'NULL', output goes to the console. Default is '"log_logistic_models.txt"'.

Value

A list containing the aggregated results of the cross-validation.

`model_type`
: The primary model type used (e.g., '"logLasso"').

`penalty_g`
: The penalty type used for 'covYI' (if applicable).

`cv_time`
: Total time for the cross-validation process.

`auc_first_step`, `aauc_first_step`, `aYI_first_step`, `youden_index_first_step`, `sensitivity_first_step`, `specificity_first_step`, `geometric_mean_first_step`, `fdr_first_step`, `mcc_first_step`, `corrclass_first_step`
: A series of nested lists, each containing performance matrices for the respective measure of the first-step estimated method.

`auc`, `aauc`, `aYI`, `youden_index`, `sensitivity`, `specificity`, `geometric_mean`, `fdr`, `mcc`, `corrclass`
: A series of nested lists, each containing performance matrices for the respective measure of the final estimated method.

`lambda_hat_'[measure]'`
: The optimal 'lambda' value selected based on maximizing each performance measure (e.g., 'lambda_hat_auc', 'lambda_hat_ccr').

`tau_hat_'[measure]'`
: The optimal 'tau' value for each measure, if 'c_function_of_covariates' was 'TRUE'.

`c_function_of_covariates`, `simultaneous`, `measure_to_select_lambda`
: Input parameters defining the CV strategy.

`lambda_star`
: The optimal 'lambda' chosen in the first step of a sequential search. 'NA' otherwise.

`n_betas`
: A matrix ('n_folds' x 'length(lambda)') showing the number of non-zero beta coefficients.

`n_betas_star`
: Number of non-zero betas for 'lambda_star' in a sequential search.

`n_gammas`
: A list of matrices showing the number of non-zero gamma coefficients for each 'lambda' and 'tau' combination.

`betas`
: A list of the estimated beta coefficients for each fold and 'lambda'.

betas_star	A list of estimated beta coefficients for ‘lambda_star‘ in a sequential search.
gammas	A list of the estimated gamma coefficients for each fold, ‘lambda‘, and ‘tau‘ combination.

Examples

```
library(pye)

# 1. Simulate a small dataset
set.seed(123)
sim_data <- create_sample_with_covariates(
  rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- sim_data$train_df_scaled
X <- sim_data$X
y <- sim_data$y
C <- sim_data$C

# 2. Define a small grid for hyperparameters
# In a real scenario, these would be calibrated (e.g., using calibrate_lambda_max)
# and the sequence would be longer.
lambda_seq <- c(0.1, 0.05)
tau_seq <- c(0.2, 0.1)

# 3. Run cross-validation
# This example uses covariates in a sequential search.
# For a quick run, we use a small number of folds and iterations.
cv_results <- plr_compute_cv(
  n_folds = 2,
  df = df,
  X = X,
  y = y,
  C = C,
  model_type = "logLasso",
  lambda = lambda_seq,
  tau = tau_seq,
  trace = 0, # Use trace = 1 for more details
  used_cores = 1,
  c_function_of_covariates = TRUE,
  simultaneous = FALSE,
  measure_to_select_lambda = "auc", #"auc","aauc","aYI","ccr","yi","gm","pye")
  penalty_g = "L1",
  max_iter_g = 9 # Low for example speed
)

# 4. Inspect the results
cat("Cross-validation finished in:", round(cv_results$cv_time, 2), "minutes\n")
cat("Optimal lambda based on AUC:", cv_results$lambda_hat_auc, "\n")
cat("Optimal tau based on AUC:", cv_results$tau_hat_ccr, "\n")

# Show mean test AUC for each lambda/tau combination
# The structure is nested: results$auc$`lambda_...`$test
mean_test_auc <- sapply(cv_results$auc, function(res) colMeans(res$test))
```

```
print("Mean Test AUC matrix (rows = lambda, cols = tau):")
print(mean_test_auc)
```

`plr_estimation` *glmnet Estimation for Coefficient and Feature Selection*

Description

function to estimate the optimal regression coefficients (betas) using penalized logistic regression methods such as Lasso, Elastic-Net, SCAD, and MCP. This function fits a penalized logistic regression model for a binary outcome. The user chooses the type of penalization (L1, Elastic-Net, SCAD, or MCP) and provides the fixed penalization parameter, `lambda`. It returns the estimated coefficients and a set of comprehensive classification performance metrics based on the optimal cut-point derived from Youden's Index.

Usage

```
plr_estimation(
  df,
  X = NULL,
  y = "y",
  alpha = 0.5,
  lambda,
  model_type = "logLasso",
  c_to_use = "Youden",
  regressors_betas = NULL,
  fold = NULL,
  trace = 1,
  max.print = 10
)
```

Arguments

`df`	A data frame containing the complete dataset.
`X`	A character vector of column names from `df` to be used as regressor variables. Alternatively, a data frame containing only the regressors. If not specified, all columns not in `y` are used.
`y`	A character string specifying the column name of the target variable in `df`. It must be a binomial variable with values 0 or 1. Alternatively, a data frame containing only the target variable. Default is "y".
`alpha`	The elastic-net mixing parameter. A value of 1 corresponds to L1 penalization (Lasso). A value between 0 and 1 is used for Elastic-Net. Note: This parameter is only used when `model_type = "logElasticNet"`; for `logLasso`, `alpha` is forced to 1, and for `logSCAD`/`logMCP`, it is ignored as they do not typically use an `alpha` mixing parameter. Default is 0.5.

lambda	A single non-negative numeric value for the regularization parameter. A larger value results in stronger penalization.
model_type	A character string specifying the penalized logistic model to use. Options are: "logLasso" (L1 penalty, via glmnet), "logElasticNet" (Elastic-Net penalty, via glmnet), "logSCAD" (SCAD penalty, via ncvreg_modified in pye), and "logMCP" (MCP penalty, via ncvreg_modified in pye). Default is "logLasso".
c_to_use	character or numeric value specifying the classification cut-point used to convert continuous predictions into binary outcomes. If set to '"Youden"', the function computes the optimal cut-point using Youden's Index. If a numeric value is provided, it is used directly as the threshold. Default is "Youden".
regressors_betas	numeric vector. An optional vector containing the "true" beta coefficients, if known. This is used to calculate additional performance measures related to variable selection accuracy. Default is 'NULL'.
fold	numeric. An optional fold number, used when the function is called within a cross-validation loop. This is primarily for tracking and reporting purposes. Default is 'NULL'.
trace	An integer controlling the amount of output printed to the console. 0 for no output, 1 for a result summary, and 2 for a detailed breakdown. Default is 1.
max.print	The number of elements to show when printing results. Default is 10.

Value

A list containing the model fit, estimated coefficients, and a collection of performance metrics.

model	The fitted model object (from glmnet or ncvreg_modified).
model_type	A character string indicating the model type used.
betas_hat	The estimated vector of optimal beta coefficients (excluding the intercept).
X_model	A character vector containing the names of all regressor variables (X) used in the model estimation. This vector defines the canonical order of the regressors used to align the coefficients in betas_hat.
youden_index	The Youden Index value at the optimal cut-point.
sensitivity	The sensitivity (True Positive Rate) at the optimal cut-point.
specificity	The specificity (True Negative Rate) at the optimal cut-point (spec).
geometric_mean	The geometric mean of sensitivity and specificity (gm).
fdr	The False Discovery Rate at the optimal cut-point.
mcc	The Matthews Correlation Coefficient.
corrclass	The overall correct classification rate (CCR).
auc	The Area Under the ROC Curve (AUC).
lambda	The penalization parameter used.
c_hat	The optimal cut-point for the z_hat score.
z_hat	A data frame with the estimated linear combination of regressors (or probability) for each observation.

`y_hat` A data frame with the predicted binary outcomes (0 or 1).

`n_betas` The number of non-zero beta coefficients estimated.

`n_total_var` The total number of regressors considered.

`n_predicted_zeros`
The number of estimated betas that are zero.

`n_predicted_non_zeros`
The number of estimated betas that are non-zero.

`n_caught_betas` The number of true non-zero betas correctly identified as non-zero (if `regressors_betas` is provided).

`n_non_caught_betas`
The number of true non-zero betas incorrectly identified as zero (False Negatives).

`n_caught_zero` The number of true zero betas correctly identified as zero (True Negatives).

`n_zero_not_caught`
The number of true zero betas incorrectly identified as non-zero (False Positives).

`TP`, `TN`, `FP`, `FN` The counts of True Positives, True Negatives, False Positives, and False Negatives at the optimal cut-point.

`estimation_time`
The time taken for the estimation, in minutes.

Examples

```
library(pye)
# --- 1. Simulate data ---
# Simulate a small dataset with 50 observations, 200 total features,
# and 20 covariates (not used here, but good for context).
set.seed(42) # for reproducibility
sim_data <- create_sample_with_covariates(
    rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- sim_data$train_df_scaled # scaled training data
X <- sim_data$X                  # names of regressors
y <- sim_data$y                 # name of target variable
regressors_betas <- sim_data$nregressors # true non-zero betas count

# --- 2. Estimate using logLasso ---
model_type_lasso <- "logLasso"
lambda_lasso <- 0.05

lasso_result <- plr_estimation(df = df, X = X, y = y,
  model_type = model_type_lasso, lambda = lambda_lasso, alpha = 1,
  regressors_betas = regressors_betas, trace = 1)

print(lasso_result$model_type)
print(lasso_result$corrclass)
print(lasso_result$n_betas)

# --- 3. Estimate using logElasticNet ---
en_result <- plr_estimation(df = df, X = X, y = y,
```

```
  model_type = "logElasticNet", lambda = 0.05, alpha = 0.5,
  regressors_betas = regressors_betas, trace=0)

print(en_result$model_type)
print(en_result$auc)
```

---

plr_predict *Prediction and Performance Evaluation for Penalized glmnet Models*

Description

This function applies a previously trained penalized logistic model to new data to generate predictions and evaluate performance metrics. The model to be used is the result of the 'plr_estimation' function.

Usage

```
plr_predict(
  df,
  y = "y",
  model_to_use,
  fold = NULL,
  regressors_betas = NULL,
  trace = 1,
  c_function_of_covariates = FALSE,
  max.print = 10
)
```

Arguments

df	A data frame containing the new dataset for prediction.
y	A character string specifying the column name of the target variable in 'df'. It must be a binomial variable with values 0 or 1. Default is "y".
model_to_use	A list containing the parameters from a model trained by the 'plr_estimation' function, specifically the 'betas_hat', 'c_hat', and 'model' components.
fold	numeric. An optional fold number, used when the function is called within a cross-validation loop. This is primarily for tracking and reporting purposes. Default is 'NULL'.
regressors_betas	numeric vector. An optional vector containing the "true" beta coefficients, if known. This is used to calculate additional performance measures related to variable selection accuracy. Default is 'NULL'.
trace	An integer controlling the amount of output. '0' for no output, '1' for a summary of the results, and '2' for a detailed breakdown of all steps. Default is 1.

`c_function_of_covariates`
: A logical flag used internally to suppress printing of results if the output is handled by an external function (e.g., 'covYI'). Default is 'FALSE'.

`max.print`
: The number of elements to show when printing results. Default is 10.

Value

A list containing the predictions and a collection of performance metrics, including:

`model`
: The trained model object (e.g., from 'glmnet' or 'ncvreg').

`model_type`
: The model type ("logLasso", "logMCP", etc.).

`betas_hat`
: The estimated vector of optimal beta coefficients.

`youden_index`
: The Youden Index value at the optimal cut-point.

`sensitivity`
: The sensitivity at the optimal cut-point.

`specificity`
: The specificity at the optimal cut-point.

`geometric_mean`
: The geometric mean of sensitivity and specificity.

`fdr`
: The False Discovery Rate at the optimal cut-point.

`mcc`
: The Matthews Correlation Coefficient.

`corrclass`
: The overall correct classification rate.

`auc`
: The Area Under the ROC Curve (AUC).

`lambda`
: The penalization parameter used in the estimation.

`c_hat`
: The optimal cut-point for the 'z_hat' score.

`z_hat`
: A data frame with the estimated linear combination of regressors for each observation.

`y_hat`
: A data frame with the predicted binary outcomes (0 or 1).

`n_total_var`
: The total number of regressors considered.

`n_predicted_non_zeros`
: The number of estimated betas that are non-zero.

`TP`, `TN`, `FP`, `FN`
: The counts from the confusion matrix.

`estimation_time`
: The time taken for the prediction.

Examples

```
library(pye)
# --- 1. Simulate data ---
# Simulate a small dataset with 50 observations, 200 total features,
# and 20 covariates (not used here, but good for context).
set.seed(42) # for reproducibility
sim_data <- create_sample_with_covariates(
    rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df_train <- sim_data$train_df_scaled # scaled training data
df_test <- sim_data$test_df_scaled # scaled training data
X <- sim_data$X                  # names of regressors
y <- sim_data$y                  # name of target variable
```

```
regressors_betas <- sim_data$nregressors # true non-zero betas count

# --- 2. Estimate using logLasso ---
model <- plr_estimation(df = df_train, X = X, y = y,
  model_type = "logMCP", lambda = 0.05,
  regressors_betas = regressors_betas, trace = 1)

predictions <- plr_predict(df = df_test, y = y, model_to_use = model,
   trace = 1, regressors_betas = regressors_betas)

print(predictions)
```

`proximal_operator_EN` *Proximal Operator for the Elastic-Net Penalty*

Description

Computes the proximal operator for the Elastic-Net penalty, which is a convex combination of L_1 (Lasso) and L_2 (Ridge). This operator serves as the backward step in Proximal Gradient optimization.

Usage

```
proximal_operator_EN(betas, lambda, alpha, ...)
```

Arguments

`betas`	`numeric` vector to which the proximal operator should be applied.
`lambda`	`numeric`. The overall non-negative penalization parameter λ.
`alpha`	`numeric`. The mixing parameter $\alpha \in [0, 1]$. $\alpha = 1$ is pure L_1 (Lasso); $\alpha = 0$ is pure L_2 (Ridge).
`...`	Additional arguments passed to methods

Details

The Elastic-Net penalty is a convex combination of the L_1 and L_2 penalties. The combined penalty $P(\beta)$ is:

$$P(\beta) = \lambda \left(\alpha|\beta| + (1-\alpha)\frac{1}{2}\beta^2 \right)$$

The proximal operator for the Elastic-Net penalty has a closed-form solution:

$$\text{prox}_P(\beta) = \frac{1}{1+\lambda(1-\alpha)} \cdot \text{sign}(\beta) \cdot \max(0, |\beta| - \lambda\alpha)$$

This function applies this operator element-wise to the input `betas` vector.

Value

A numeric vector of the same length as betas, containing the result of applying the Elastic-Net proximal operator.

Examples

```
library(pye)
betas <- seq(-2, 2, by = 0.2)
lambda <- 0.5
alpha <- 0.5
proximal_operator_EN(betas = betas, lambda = lambda, alpha = alpha)

# Equivalent to Lasso
prox_lasso <- proximal_operator_EN(betas = betas, lambda = 0.5, alpha = 1)

# Equivalent to a scaled Ridge
prox_ridge <- proximal_operator_EN(betas = betas, lambda = 0.5, alpha = 0)
```

---

proximal_operator_L1 *Proximal Operator for the Convex L_1 (Lasso) Penalty*

Description

Computes the proximal operator for the convex L_1 (Lasso) penalty. This operator is the soft-thresholding operator and serves as the backward step in Proximal Gradient optimization.

Usage

```
proximal_operator_L1(betas, lambda, ...)
```

Arguments

betas	numeric vector to which the proximal operator should be applied.
lambda	numeric. The non-negative penalization parameter λ.
...	Additional arguments passed to methods

Details

The L_1 penalty, also known as the Lasso penalty, is a convex penalty commonly used to enforce sparsity (set coefficients to zero). Its proximal operator has a closed-form solution known as the soft-thresholding operator:

$$\text{prox}_{\lambda L_1}(\beta) = \text{sign}(\beta) \cdot \max(0, |\beta| - \lambda)$$

This function applies this operator element-wise to the input betas vector.

Value

A numeric vector of the same length as betas, containing the result of applying the soft-thresholding operator.

Examples

```
# Example usage:
betas <- seq(-2, 2, by = 0.2)
lambda <- 0.5
proximal_operator_L1(betas = betas, lambda = lambda)

# Another example with different lambda
lambda2 <- 0.1
proximal_operator_L1(betas = betas, lambda = lambda2)
```

---

proximal_operator_L12 *Proximal Operator for the Non-Convex L_1/2 Penalty*

Description

Computes the proximal operator for the non-convex $L_{1/2}$ penalty. This operator serves as the backward step in the Forward-Backward Splitting method (or Proximal Gradient method) used to optimize the Penalized Youden Index (pye) or other objectives with this penalty. The solution is found using a fixed-point iterative method.

Usage

```
proximal_operator_L12(
  betas,
  lambda,
  q = 0.5,
  max_iter = 10000,
  tol = 1e-10,
  ...
)
```

Arguments

betas	numeric vector to which the proximal operator should be applied.
lambda	numeric. The non-negative penalization parameter λ.
q	numeric. The exponent for the $L_{1/2}$ penalty, typically $q = 0.5$. Must be $0 < q < 1$. Default is 0.5.
max_iter	integer. Maximum number of iterations for the fixed-point optimization loop for x_{bar}. Default is 10000.
tol	numeric. Tolerance for the convergence of the fixed-point optimization loop. Default is 1e-10.
...	Additional arguments passed to methods

Details

The $L_{1/2}$ penalty is non-convex and promotes sparser solutions than the Lasso (L_1). The proximal operator for a non-convex penalty often lacks a closed-form solution.

This implementation computes the solution by:

1. Identifying a threshold h_λ such that if $|\beta_j| \leq h_\lambda$, the solution is $\text{prox}_{\lambda L_{1/2}}(\beta_j) = 0$.
2. If $|\beta_j| > h_\lambda$, the non-zero solution $\text{prox}_{\lambda L_{1/2}}(\beta_j)$ is found using a fixed-point iteration method to solve for x_{bar}, where $|\beta_j| = x_{\text{bar}} + \lambda q x_{\text{bar}}^{q-1}$.

The final proximal value is $\text{sign}(\beta_j) \cdot x_{\text{bar}}$.

Value

A `numeric` vector of the same length as `betas`, containing the result of applying the proximal operator.

Examples

```
# Example usage:
betas <- seq(-2, 2, by = 0.2)
lambda <- 0.5
q <- 0.5

# Apply the proximal operator
prox_betas <- proximal_operator_L12(betas = betas, lambda = lambda, q = q)
print(prox_betas)

# Another example with different lambda
lambda2 <- 0.1
prox_betas2 <- proximal_operator_L12(betas = betas, lambda = lambda2)
print(prox_betas2)
```

`proximal_operator_MCP` *Proximal Operator for the Non-Convex MCP Penalty*

Description

Computes the proximal operator for the non-convex Minimax Concave Penalty (MCP). This operator serves as the backward step in Proximal Gradient optimization.

Usage

```
proximal_operator_MCP(betas, lambda, a, ...)
```

Arguments

`betas`	`numeric` vector to which the proximal operator should be applied.
`lambda`	`numeric`. The non-negative penalization parameter λ.
`a`	`numeric`. The hyperparameter $a > 1$ for the MCP penalty. It controls the threshold after which the penalty is zero. $a=3.0$ is a common choice.
`...`	Additional arguments passed to methods

Details

The MCP penalty is a non-convex penalty that smoothly transitions from the L_1 penalty (for small coefficients) to zero (for large coefficients), thereby reducing estimation bias. Its proximal operator is defined piecewise:

1. If $|\beta| \leq \lambda$:
$$\text{prox}_{\lambda P}(\beta) = 0$$
2. If $\lambda < |\beta| \leq a\lambda$:
$$\text{prox}_{\lambda P}(\beta) = \frac{\beta - \text{sign}(\beta)\lambda}{1 - 1/a}$$
3. If $|\beta| > a\lambda$:
$$\text{prox}_{\lambda P}(\beta) = \beta$$

This function applies this operator element-wise to the input `betas` vector.

Value

A `numeric` vector of the same length as `betas`, containing the result of applying the MCP proximal operator.

Examples

```
# Example usage:
betas <- seq(-2, 2, by = 0.2)
lambda <- 0.5
a <- 3.0 # Common choice for 'a'
prox_betas <- proximal_operator_MCP(betas = betas, lambda = lambda, a = a)
print(prox_betas)

# Example with a different 'a' value
prox_betas2 <- proximal_operator_MCP(betas = betas, lambda = 0.3, a = 2)
print(prox_betas2)
```

`proximal_operator_SCAD` *Proximal Operator for the Non-Convex SCAD Penalty*

Description

Computes the proximal operator for the non-convex Smoothly Clipped Absolute Deviation (SCAD) penalty. This operator serves as the backward step in Proximal Gradient optimization.

Usage

```
proximal_operator_SCAD(betas, lambda, a, ...)
```

Arguments

`betas`	`numeric` vector to which the proximal operator should be applied.
`lambda`	`numeric`. The non-negative penalization parameter λ.
`a`	`numeric`. The hyperparameter $a > 2$ for the SCAD penalty. It controls the concavity of the penalty function. $a = 3.7$ is a common choice.
`...`	Additional arguments passed to methods

Details

The SCAD penalty is a non-convex penalty designed for sparse and unbiased estimation. Its proximal operator is defined piecewise based on λ and a:

1. If $|\beta| \leq 2\lambda$:
$$\text{prox}_{\lambda P}(\beta) = \text{sign}(\beta) \cdot \max(0, |\beta| - \lambda)$$
2. If $2\lambda < |\beta| \leq a\lambda$:
$$\text{prox}_{\lambda P}(\beta) = \frac{(a-1)\beta - \text{sign}(\beta)a\lambda}{a-2}$$
3. If $|\beta| > a\lambda$:
$$\text{prox}_{\lambda P}(\beta) = \beta$$

This function applies this operator element-wise to the input `betas` vector.

Value

A `numeric` vector of the same length as `betas`, containing the result of applying the SCAD proximal operator.

Examples

```
# Example usage:
betas <- seq(-2, 2, by = 0.2)
lambda <- 0.5
a <- 3.7 # Common choice for 'a'
prox_betas <- proximal_operator_SCAD(betas = betas, lambda = lambda, a = a)
print(prox_betas)

# Example with a different 'a' value
prox_betas2 <- proximal_operator_SCAD(betas = betas, lambda = 0.3, a = 3)
print(prox_betas2)
```

---

psvm_compute_cv *Cross-Validation for Optimal Penalized svm Regularization Parameter Selection*

Description

Performs k-fold cross-validation for penalized SVM models, optionally incorporating covariate-adjusted cut-point optimization via the covYI method. This function estimates the optimal values of the regularization parameters lambda (for feature selection) and tau (for covariate adjustment of the classification threshold).

Usage

```
psvm_compute_cv(
  n_folds,
  df,
  X = NULL,
  y = "y",
  C = NULL,
  lambda,
  tau = 0,
  w_g = 0.5,
  c_to_use = NULL,
  regressors_betas = NULL,
  trace = 1,
  alpha = 0.5,
  model_type = c("SCADSVM", "ElasticSCADSVM", "l1SVM", "enSVM"),
  seed = 1,
  used_cores = 1,
  scaling = FALSE,
  c_function_of_covariates = FALSE,
  simultaneous = FALSE,
  measure_to_select_lambda = "ccr",
  alpha_g = 0.5,
```

```
  penalty_g = "L1",
  kernel_g = "gaussian",
  a1_g = 3.7,
  a2_g = 3,
  trend_g = "monotone",
  gamma_start_input = NULL,
  gamma_start_default = "zeros",
  regressors_gammas = NULL,
  max_iter_g = 10000,
  delta_g = 1e-05,
  max_alpha_g = 10000,
  stepsizeShrink_g = 0.8,
  min_alpha_g = 1e-12,
  convergence_error_g = 1e-07,
  run_aauc = FALSE,
  log_file = "log_SVM_models.txt"
)
```

Arguments

n_folds	Integer. The number of folds to use for cross-validation (must be >= 2).
df	Data frame. The input dataset containing the target variable, regressors, and covariates.
X	Character vector. Names of the regressor variables (features). Defaults to all columns in "df" that are not "y" or "C". Can also be a data frame containing only the regressors.
y	Character. Name of the binary target variable (outcome), coded as 0 and 1. Defaults to "y". Can also be a data frame containing only the target variable.
C	Character vector. Names of the covariate variables for covariate-adjusted cut-point estimation. Defaults to "NULL" (no covariates). Can also be a data frame containing only the covariates.
lambda	Numeric vector. The regularization parameter(s) for the regressors "X".
tau	Numeric vector. The regularization parameter(s) for the covariates "C" in "covYI". If "c_function_of_covariates = TRUE", this parameter cannot be "NULL" or contain only zeros. Default is 0 (no penalization).
w_g	A ‘numeric‘ value between 0 and 1 specifying the weight for the Weighted Youden Index in covYI. Sensitivity is weighted by ‘w_g‘ and specificity by ‘1 - w_g‘. Default is 0.5 (which corresponds to the standard Youden Index).
c_to_use	Character or numeric. Specifies the classification cut-point used to convert continuous predictions into binary outcomes. If set to "Youden", the function computes the optimal cut-point using Youden's Index. If a numeric value is provided, it is used directly as the threshold. If "NULL", a default is assigned based on "model_type": "SCADSVM" uses "0", "ElasticSCADSVM", "l1SVM" or "enSVM" use "Youden".

`regressors_betas`
: numeric vector. An optional vector containing the "true" beta coefficients, if known. This is used to calculate additional performance measures related to variable selection accuracy. Default is "NULL".

`trace`
: Integer. Level of verbosity: 0 = silent, 1 = brief summary, 2 = verbose. Default is 1.

`alpha`
: numeric in (0, 1]. Elastic-net mixing parameter for sparseSVM (alpha = 1 corresponds to L1). Default is 0.5.

`model_type`
: Character. The SVM model to use. Options: "SCADSVM", "ElasticSCADSVM", "l1SVM", "enSVM".

`seed`
: Integer. Random seed for reproducibility. Default is 1.

`used_cores`
: Integer. Number of cores to use for parallel processing. If 1, no parallelization is used. Default is 1.

`scaling`
: Logical. If "TRUE", the dataset is scaled. Default is "FALSE".

`c_function_of_covariates`
: Logical. If "TRUE", "covYI" is used to estimate the cut-off point as a function of the covariates. Default is "FALSE".

`simultaneous`
: Logical. If "TRUE" (and "c_function_of_covariates = TRUE"), gammas are estimated simultaneously with betas. If "FALSE", gammas are estimated as a second step. Default is "FALSE".

`measure_to_select_lambda`
: Character. The performance measure used to select the best lambda when "simultaneous = FALSE". Options: "auc", "aauc", "aYI", "ccr", "yi", "gm", "fdr", "mcc", "sen", "spc". Default is "ccr" (correct classification rate).

`alpha_g`
: Numeric. Elastic-Net mixing parameter for "covYI". Default is 0.5.

`penalty_g`
: Character. The penalty of "covYI". Options: "L12", "L1", "EN", "SCAD", "MCP". Default is "L1".

`kernel_g`
: Character. Kernel type for density estimation in "covYI". Default is "gaussian".

`a1_g`
: Numeric. Parameter for the SCAD and MCP penalties in "covYI". Default is 3.7.

`a2_g`
: Numeric. Parameter for the MCP penalty in "covYI". Default is 3.0.

`trend_g`
: Character. For "covYI", if "monotone", mmAPG is used; if "nonmonotone", mnmAPG is used. Default is "monotone".

`gamma_start_input`
: Numeric vector. A specific starting point for gammas. Default is "NULL".

`gamma_start_default`
: Character. Sets the default starting point of gamma. If "zeros", it starts with all zero values; if "corr", it starts with the value of the correlation of every regressor with the target variable.

`regressors_gammas`
: Numeric vector. Optional vector containing the true gammas (if known).

`max_iter_g`
: Integer. Maximum number of iterations in the algorithms mmAPG and mnmAPG in "covYI". Default is 10000.

`delta_g` Numeric. Parameter for the convergence condition of the optimization algorithm of "covYI". Default is 1e-5.

`max_alpha_g` Numeric. Maximum value of the step-parameter alpha in "covYI". Default is 100.

`stepsizeShrink_g` Numeric. Parameter to adjust the step-size in the backtracking line-search, in the optimization of "covYI". Taking values between 0 and 1, the closer to 1, the more accurate the estimation will be, the longer it will take and vice versa. Default is 0.8.

`min_alpha_g` Numeric. Minimum value of the step-parameter alpha in "covYI". Default is 1e-12.

`convergence_error_g` Numeric. Error to accept for considering the algorithm converged in "covYI". Default is 1e-7.

`run_aauc` Logical. If "FALSE", the aAUC and aYI are not computed, to save estimation time if not requested. Default is "FALSE".

`log_file` Character. Path to a file for logging output from parallel workers. If "NULL", output goes to the console. Defaults to "log_SVM_models.txt".

Value

A list containing the optimal values of lambda (and possibly tau) to estimate betas (and possibly gammas) and the value of the main accuracy measure for all the folds. The list includes:

`model_type` The SVM model type used.

`c_to_use` The classification cut-point used.

`penalty_g` The penalty type used for covYI.

`cv_time` The time taken for cross-validation.

`auc_first_step`, `aauc_first_step`, `aYI_first_step`, `youden_index_first_step`, `sensitivity_first_step`, `specificity_first_step`, `geometric_mean_first_step`, `fdr_first_step`, `mcc_first_step`, `corrclass_first_step` A series of nested lists, each containing performance matrices for the respective measure of the first-step estimated method.

`auc`, `aauc`, `aYI`, `youden_index`, `sensitivity`, `specificity`, `geometric_mean`, `fdr`, `mcc`, `corrclass` A series of nested lists, each containing performance matrices for the respective measure of the final estimated method.

`lambda_hat_yi`, `lambda_hat_auc`, `lambda_hat_aauc`, `lambda_hat_aYI`, `lambda_hat_ccr`, `lambda_hat_sen`, `lambda_hat_spc`, `lambda_hat_gm` The optimal lambda values selected based on different performance measures.

`tau_hat_yi`, `tau_hat_auc`, `tau_hat_aauc`, `tau_hat_aYI`, `tau_hat_ccr`, `tau_hat_sen`, `tau_hat_spc`, `tau_hat_gm` The optimal tau values selected based on different performance measures (if 'c_function_of_covariates = TRUE').

`c_function_of_covariates` Indicates whether the cut-off point was estimated as a function of covariates.

simultaneous	Indicates whether betas and gammas were estimated simultaneously.
measure_to_select_lambda	The measure used to select lambda.
lambda_star	The optimal lambda value when using a sequential search.
n_betas	The number of non-zero beta coefficients.
n_betas_star	The number of non-zero beta coefficients when using lambda_star.
n_gammas	The number of non-zero gamma coefficients.
betas_star	The estimated beta coefficients when using lambda_star.
betas	The estimated beta coefficients.
gammas	The estimated gamma coefficients.

Examples

```
library(pye)
# Generate a sample dataset
sim_data <- create_sample_with_covariates(
  rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- sim_data$train_df_scaled
X <- sim_data$X
y <- sim_data$y
C <- sim_data$C
regressors_betas <- sim_data$nregressors
regressors_gammas <- sim_data$ncovariates

# Define parameters
lambda <- c(0.1, 0.05)
c_function_of_covariates <- FALSE
model_type <- "SCADSVM"

# Run cross-validation
result <- psvm_compute_cv(n_folds = 2,
                          df = df,
                          X = X,
                          y = y,
                          C = C,
                          lambda = lambda,
                          regressors_betas = regressors_betas,
                          regressors_gammas = regressors_gammas,
                          c_function_of_covariates = c_function_of_covariates,
                          model_type = model_type
)

# Print the results
print(result$model_type)
print(result$lambda_hat_ccr)
```

psvm_estimation *Penalized svm Estimation for Coefficient and Feature Selection*

Description

Fit a support vector machine (SVM) classifier with optional feature selection / regularization using multiple penalty families and kernels. This function implements a consistent interface to: (i) SCAD penalized SVM estimators, (ii) SCAD+L2 (elastic-SCAD) estimators, and (iii) sparse/elastic sparse SVM via sparseSVM. The routine returns estimated coefficients, sample decision scores, an estimated optimal cutpoint (Youden criterion) and a comprehensive set of classification performance measures.

Usage

```
psvm_estimation(
  df,
  X = NULL,
  y = "y",
  lambda = NULL,
  lambda2 = 0.05,
  alpha = 0.5,
  model_type,
  c_to_use = NULL,
  regressors_betas = NULL,
  fold = NULL,
  trace = 1,
  seed = 1,
  max.print = 10
)
```

Arguments

df	data.frame containing the observations (rows) and variables (columns).
X	character vector with names of predictors in 'df'. If omitted the function will attempt to use all columns in 'df' except 'y'.
y	single character string with the name of the binary outcome column in 'df' (allowed codings: 0/1 or -1/1).
lambda	numeric. Primary regularization parameter used by penalized estimators (SCAD, ElasticSCAD, sparseSVM). If NULL, methods that require a penalty will return an error.
lambda2	numeric. Secondary L2 penalty used by ElasticSCADSVM.
alpha	numeric in (0, 1]. Elastic-net mixing parameter for sparseSVM (alpha = 1 corresponds to L1). Default is 0.5.
model_type	character scalar selecting the estimation method. Allowed values include: "SCADSVM", "ElasticSCADSVM", "l1SVM", "enSVM".

`c_to_use` character or numeric value specifying the classification cut-point used to convert continuous predictions into binary outcomes. If set to ‘"Youden"‘, the function computes the optimal cut-point using Youden’s Index. If a numeric value is provided, it is used directly as the threshold. If ‘NULL‘, a default is assigned based on ‘model_type‘: ‘"SCADSVM"‘ uses ‘0‘, ‘"ElasticSCADSVM"‘, ‘"l1SVM"‘ or ‘"enSVM"‘ use ‘"Youden"‘.

`regressors_betas` numeric vector. An optional vector containing the "true" beta coefficients, if known. This is used to calculate additional performance measures related to variable selection accuracy. Default is ‘NULL‘.

`fold` numeric. An optional fold number, used when the function is called within a cross-validation loop. This is primarily for tracking and reporting purposes. Default is ‘NULL‘.

`trace` integer verbosity level: 0 = silent, 1 = brief summary, 2 = verbose.

`seed` integer used to set RNG where supported by the back-end routines.

`max.print` The number of elements to show when printing results. Default is 10.

Details

The function accepts a user supplied design matrix specification ‘X‘ and a binary response ‘y‘ coded as 0/1 (or -1/1). Internally ‘y‘ is converted to the form required by the selected estimation routine. For standard linear SVMs the function will attempt to reconstruct linear predictor weights (when available) from the fitted model object. For penalized estimators the function extracts penalty-specific coefficients when present. The optimal cutpoint for mapping decision scores to class labels is obtained by applying the Youden index via OptimalCutpoints; when that routine fails the function falls back to a conservative default. Output is returned as a named list designed for subsequent evaluation or cross-validation.

Value

A named list with the principal elements:

`model` The fitted model object returned by the selected backend (e.g. penalizedSVM object, sparseSVM object).

`model_type` Character string with the selected model type.

`betas_hat` Numeric named vector with estimated predictor coefficients where recoverable (zeros if unavailable). Names follow ‘X‘.

`c_hat` Numeric cutpoint computed by Youden’s criterion on the training decision scores.

`z_hat` Data.frame with columns ‘ID‘ and ‘z_hat‘ (decision score).

`y_hat` Data.frame with columns ‘ID‘ and ‘y_hat‘ (predicted labels at ‘c_hat‘).

`youden_index`, `auc`, `sensitivity`, `specificity`, `geometric_mean`, `fdr`, `mcc`, `corrclass` Numeric scalar performance summaries computed on the training data.

`TP`, `TN`, `FP`, `FN` Confusion matrix cell counts at ‘c_hat‘.

`lambda`, `lambda2`, `alpha` Model and tuning parameters used.

`n_betas`, `n_total_var`, `n_predicted_zeros`, `n_predicted_non_zeros`
: Variable selection diagnostics.

`estimation_time`
: Elapsed wall clock time (minutes).

References

See references for penalizedSVM, sparseSVM and OptimalCutpoints packages for algorithmic details.

Examples

```
library(pye)
sim_data <- create_sample_with_covariates(
    rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- sim_data$train_df_scaled
X <- sim_data$X
y <- sim_data$y
regressors_betas<-sim_data$nregressors
model_type <- "SCADSVM"
lambda <- 0.1

res <- psvm_estimation(df = df, X = X, y = y, model_type = model_type, lambda = lambda,
        regressors_betas = regressors_betas, trace = 1)

print(res)
```

---

psvm_predict *Prediction and Performance Evaluation for Penalized svm Models*

Description

Apply a previously fitted SVM model (from 'psvm_estimation') to new data and generate predictions along with performance metrics.

Usage

```
psvm_predict(
  df,
  y = "y",
  model_to_use,
  fold = NULL,
  regressors_betas = NULL,
  trace = 1,
  max.print = 10,
  c_function_of_covariates = FALSE
)
```

Arguments

df	data.frame. The input dataset containing observations (rows) and variables (columns) for prediction.
y	character. The name of the target variable column in 'df'. This column should contain binary outcomes (0 and 1). Default is "y".
model_to_use	list. A list containing the fitted model and related information, as returned by the 'psvm_estimation' function. This list must include elements named 'betas_hat', 'c_hat', 'X_model', and 'model_type'.
fold	numeric. An optional fold number, used when the function is called within a cross-validation loop. This is primarily for tracking and reporting purposes. Default is 'NULL'.
regressors_betas	numeric vector. An optional vector containing the "true" beta coefficients, if known. This is used to calculate additional performance measures related to variable selection accuracy. Default is 'NULL'.
trace	integer. Controls the level of verbosity during the function's execution. - 0: No output is printed. - 1: A summary of the results is printed. - 2: Detailed information about the calculations is printed. Default is 1.
max.print	The number of elements to show when printing results. Default is 10.
c_function_of_covariates	logical. If 'TRUE', indicates that the cut-off point ('c_hat') should be estimated as a function of covariates (using a method like covYI). If 'FALSE', a single cut-off value is used for all predictions. This parameter primarily affects whether results are printed within this function or handled externally. Default is 'FALSE'.

Details

This function takes a data frame and a fitted SVM model (produced by 'psvm_estimation') to generate predictions on new data. It supports various SVM model types, including linear, polynomial, radial, sigmoid, SCADSVM, ElasticSCADSVM, l1SVM, and enSVM. The function calculates decision scores (z_hat) and predicted binary outcomes (y_hat) based on a cut-off value. It also computes and returns a comprehensive set of classification performance measures.

Value

A named list with the following elements:

model	The fitted model object used for prediction (same as in 'model_to_use').
model_type	Character string indicating the type of SVM model used (same as in 'model_to_use').
betas_hat	Numeric vector of estimated beta coefficients from the fitted model (same as in 'model_to_use').
youden_index	Numeric. Youden's J statistic, calculated on the predicted values.
sensitivity	Numeric. Sensitivity (true positive rate) of the predictions.
specificity	Numeric. Specificity (true negative rate) of the predictions.
geometric_mean	Numeric. Geometric mean of sensitivity and specificity.

fdr	Numeric. False discovery rate.
mcc	Numeric. Matthews correlation coefficient.
corrclass	Numeric. Correct classification rate (accuracy).
auc	Numeric. Area under the ROC curve.
lambda	Numeric. Value of lambda used in the model (same as in 'model_to_use').
lambda2	Numeric. Value of lambda2 used in the model (same as in 'model_to_use').
alpha	Numeric. Value of alpha used in the model (same as in 'model_to_use').
c_hat	Numeric. The cut-off value used to convert decision scores into binary predictions (same as in 'model_to_use').
z_hat	Data.frame with columns "ID" and "z_hat" (decision scores).
y_hat	Data.frame with columns "ID" and "y_hat" (predicted binary outcomes).
n_total_var	Integer. Total number of variables (regressors) in the model.
n_predicted_zeros	Integer. Number of regressors with estimated coefficients of zero.
n_predicted_non_zeros	Integer. Number of regressors with estimated coefficients different from zero.
n_caught_betas	Integer. If 'regressors_betas' is provided, the number of non-zero "true" betas that were also identified as non-zero in the model.
n_non_caught_betas	Integer. If 'regressors_betas' is provided, the number of non-zero "true" betas that were incorrectly estimated as zero in the model.
n_caught_zero	Integer. If 'regressors_betas' is provided, the number of zero "true" betas that were also estimated as zero in the model.
n_zero_not_caught	Integer. If 'regressors_betas' is provided, the number of zero "true" betas that were incorrectly estimated as non-zero in the model.
TP	Integer. Number of true positives.
TN	Integer. Number of true negatives.
FP	Integer. Number of false positives.
FN	Integer. Number of false negatives.
estimation_time	difftime. The time elapsed during the prediction process.

Examples

```
library(pye)
# Generate synthetic data
sim_data <- create_sample_with_covariates(
  rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
train_df <- sim_data$train_df_scaled
test_df <- sim_data$test_df_scaled
X <- sim_data$X
y <- sim_data$y
regressors_betas <- sim_data$nregressors
```

```
# Estimate an SVM model
model <- psvm_estimation(
  df = train_df, X = X, y = y, lambda = 0.1, model_type = "SCADSVM",
  regressors_betas = regressors_betas, trace = 0
)

# Apply the model to the test data
predictions <- psvm_predict(df = test_df, y = y, model_to_use = model,
  trace = 1, regressors_betas = regressors_betas, c_function_of_covariates = FALSE
)

# Print the predictions
print(predictions)
```

---

pye_KS *Penalized Youden Index (pye) function via Kernel Smoothing Density*

Description

The Penalized Youden Index (pye) function based on the Kernel Smooth density estimator. It not only performs value of pye but it returns all the necessary for the estimation process, like measure of fit and derivatives. It works for all the considered penalties (L12, L1, EN, SCAD and MCP)

Usage

```
pye_KS(
  df,
  X = NULL,
  y = "y",
  betas,
  lambda,
  c = 0,
  w = 0.5,
  kernel = "gaussian",
  alpha = 0.5,
  a1 = 3.7,
  a2 = 3,
  penalty = "L1",
  h_exponent = 0.2,
  use_opt_c = FALSE,
  prediction = FALSE,
  print.CDF.plot = FALSE
)
```

Arguments

df	A ‘data.frame‘ containing the input dataset. Must include the columns specified in ‘X‘ and ‘y‘.
X	A ‘character‘ vector specifying the names of the regressor variables (biomarkers) to consider for the pye evaluation. Alternatively, a ‘data.frame‘ containing only these regressors can be provided, in which case their column names will be used. Default is all columns in ‘df‘ not specified as ‘y‘.
y	A ‘character‘ string specifying the name of the target binary variable (outcome). This variable must contain only values 0 and 1. Alternatively, a single-column ‘data.frame‘ containing only the target variable can be provided, in which case its column name will be used. Default is "y".
betas	A ‘numeric‘ vector representing the coefficients of the biomarker combination (Z = X %*% betas) used for the evaluation of pye.
lambda	A ‘numeric‘ value specifying the penalization parameter for the regressors ‘X‘.
c	A ‘numeric‘ value representing the cut-off point used to classify observations based on the combined biomarker score Z. Can be a single value (applied to all observations) or a numeric vector of length ‘nrow(df)‘ for observation-specific cut-offs. Default is 0.
w	A ‘numeric‘ value between 0 and 1 specifying the weight for the Weighted Youden Index. Sensitivity is weighted by ‘w‘ and specificity by ‘1 - w‘. Default is 0.5 (which corresponds to the standard Youden Index).
kernel	A ‘character‘ string indicating the kernel type to use for the estimation of the density function. Currently, only "gaussian" is tested and supported. Default is "gaussian".
alpha	A ‘numeric‘ value between 0 and 1, used as the mixing parameter for the Elastic-Net penalization term. Only relevant if ‘penalty‘ is "EN". Default is 0.5.
a1	A ‘numeric‘ value specifying the regularization parameter ’a’ for the SCAD penalty (as defined in the original paper by Fan and Li). Only relevant if ‘penalty‘ is "SCAD". Default is 3.7.
a2	A ‘numeric‘ value specifying the regularization parameter ’gamma’ for the MCP penalty (as defined in the original paper by Zhang). Only relevant if ‘penalty‘ is "MCP". Default is 3.0.
penalty	A ‘character‘ string indicating the type of penalty considered for the pye calculation. Must be one of "L12", "L1", "EN" (Elastic-Net), "SCAD", or "MCP". Default is "L1".
h_exponent	A ‘numeric‘ value for the exponent of the bandwidth ‘h‘ in the Kernel Smooth density estimation.
use_opt_c	A ‘logical‘ value. If ‘TRUE‘, the function internally estimates and uses the optimal cut-off point ’c’ based on the given betas. If ‘FALSE‘ (default), the ‘c‘ parameter provided directly is used.
prediction	A ‘logical‘ value. If ‘TRUE‘, the empirical maximum Youden Index is returned as the Youden index performance measure. This is useful for evaluating predictive performance on new data.

print.CDF.plot A 'logical' value. If 'TRUE', a plot of the Cumulative Distribution Functions (CDFs) for cases and controls, based on the combined regressor scores Z, will be printed. Default is 'FALSE'.

Value

A list with the following components:

pye_L12 numeric. The Penalized Youden Index value using the L1 / 2 penalty.

pye_L1 numeric. The Penalized Youden Index value using the L1 penalty.

pye_EN numeric. The Penalized Youden Index value using the Elastic-Net penalty.

pye_SCAD numeric. The Penalized Youden Index value using the SCAD penalty.

pye_MCP numeric. The Penalized Youden Index value using the MCP penalty.

gr_yi numeric vector. The gradient of the Youden Index with respect to the betas and the cut-off point 'c'. The last element corresponds to the gradient with respect to 'c'.

youden_index numeric. The unpenalized Youden Index calculated using the Kernel Smooth density estimator (or the empirical Youden Index if prediction = TRUE). It is weighted if w is not 0.5.

sensitivity numeric. The sensitivity (True Positive Rate) for the given 'betas' and 'c'.

specificity numeric. The specificity (True Negative Rate) for the given 'betas' and 'c'.

geometric_mean numeric. The geometric mean of sensitivity and specificity.

fdr numeric. The False Discovery Rate (FP / (FP + TP)).

mcc numeric. The Matthews Correlation Coefficient.

auc numeric. The Area Under the ROC Curve (AUC) for the given 'z_hat' scores.

corrclass numeric. The overall correct classification rate ((TP + TN) / N).

empir_suggest_yi_c
numeric. The optimal cut-off point 'c' suggested by the 'Youden' method from OptimalCutpoints package based on the empirical Youden Index.

corrclass_with_YI_c
numeric. The correct classification rate using the empirically suggested optimal cut-off point (empir_suggest_yi_c).

yi1_yi_c numeric. The Youden Index obtained using the empirically suggested optimal cut-off point (empir_suggest_yi_c).

c_hat data.frame with columns ID and c_hat. The cut-off point(s) used for each observation (either a single value repeated or input 'c').

z_hat data.frame with columns ID and z_hat. The combined biomarker scores (Z = X * betas) for each observation.

y_hat data.frame with columns ID and y_hat. The predicted binary outcomes (0 or 1) based on 'z_hat' and 'c_hat'.

TP numeric. Number of True Positives.

TN numeric. Number of True Negatives.

FP numeric. Number of False Positives.

FN numeric. Number of False Negatives.

input_data list. A list containing the input parameters used for the calculation:

- beta (numeric vector): The input 'betas' coefficients.
- w (numeric vector): The input 'w' weighting parameter.
- lambda (numeric): The input 'lambda' penalization parameter.
- alpha (numeric): The input 'alpha' parameter for Elastic-Net.
- a1 (numeric): The input 'a1' parameter for SCAD.
- a2 (numeric): The input 'a2' parameter for MCP.
- prediction (logical): The input 'prediction' flag.
- c_hat (numeric or vector): The cut-off point(s) that were ultimately used (could be the input 'c' or 'empir_suggest_yi_c' if use_opt_c = TRUE).
- kernel (character): The kernel type used for density estimation.

Examples

```
library(pye)
sim_data <- create_sample_with_covariates(
    rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- sim_data$train_df_scaled
X <- sim_data$X
y <- sim_data$y
C <- sim_data$C
penalty <- "L12"
lambda <- 0.1
betas <- rep(1, length(X))
c <- 0

pye_result <- pye_KS(df = df[, names(df) %in% c(X,y)], X = X, y = y, betas = betas,
  lambda = lambda, c = c, alpha = 0.5, a1 = 3.7, a2 = 3, penalty = penalty)
print(pye_result)
```

---

pye_KS_compute_cv *Cross-Validation for Optimal pye KS Regularization Parameter Selection*

Description

This function performs cross-validation to select the optimal lambda (and potentially tau) values for estimating biomarker coefficients (betas) and the cut-off point (c), using the Penalized Youden Index (pye) based on Kernel Smooth density estimation. It can also integrate covariate information via covYI.

Usage

```
pye_KS_compute_cv(
  n_folds,
  df,
  X = NULL,
  y = "y",
  C = NULL,
  lambda,
  w = 0.5,
  w_g = 0.5,
  trace = 1,
  alpha = 0.5,
  alpha_g = 0.5,
  tau = 0,
  a1 = 3.7,
  a2 = 3,
  penalty = "L1",
  regressors_betas = NULL,
  seed = 1,
  used_cores = 1,
  trend = "monotone",
  delta = 1e-05,
  max_alpha = 10000,
  kernel = "gaussian",
  beta_start_input = NULL,
  max_iter = 10000,
  min_alpha = 1e-10,
  convergence_error = 1e-07,
  stepsizeShrink = 0.8,
  beta_start_default = "zeros",
  scaling = FALSE,
  c_zero_fixed = FALSE,
  c_function_of_covariates = FALSE,
  simultaneous = FALSE,
  measure_to_select_lambda = "ccr",
  penalty_g = "L1",
  kernel_g = "gaussian",
  a1_g = 3.7,
  a2_g = 3,
  trend_g = "monotone",
  gamma_start_input = NULL,
  gamma_start_default = "zeros",
  regressors_gammas = NULL,
  max_iter_g = 10000,
  delta_g = 1e-05,
  max_alpha_g = 10000,
  stepsizeShrink_g = 0.8,
  min_alpha_g = 1e-12,
```

```
  convergence_error_g = 1e-07,
  run_aauc = FALSE,
  log_file = "log_pye_ks_models.txt"
)
```

Arguments

n_folds	A ‘numeric‘ value specifying the number of folds for the cross-validation.
df	A ‘data.frame‘ containing the input dataset. It must include the columns specified in ‘X‘ and ‘y‘, and optionally ‘C‘.
X	A ‘character‘ vector specifying the names of the regressor variables (biomarkers) to consider in the estimation. Alternatively, a ‘data.frame‘ containing only these regressors can be provided, in which case their column names will be used. Defaults to all columns in ‘df‘ not specified as ‘y‘ or ‘C‘.
y	A ‘character‘ string specifying the name of the target binary variable (outcome). This variable must contain only values 0 and 1. Alternatively, a single-column ‘data.frame‘ containing only the target variable can be provided, in which case its column name will be used. Default is "y".
C	A ‘character‘ vector specifying the names of covariate variables. Alternatively, a ‘data.frame‘ containing these covariates can be provided, in which case their column names will be used. Default is ‘NULL‘, indicating no covariates are used.
lambda	A ‘numeric‘ vector of penalization parameters for the regressors ‘X‘. The cross-validation will evaluate models across these lambda values.
w	A ‘numeric‘ value between 0 and 1 specifying the weight for the Weighted Youden Index in pye. Sensitivity is weighted by ‘w‘ and specificity by ‘1 - w‘. Default is 0.5 (which corresponds to the standard Youden Index).
w_g	A ‘numeric‘ value between 0 and 1 specifying the weight for the Weighted Youden Index in covYI. Sensitivity is weighted by ‘w_g‘ and specificity by ‘1 - w_g‘. Default is 0.5 (which corresponds to the standard Youden Index).
trace	A ‘numeric‘ value controlling the verbosity of the output. ‘0‘: no visualization; ‘1‘: visualize only the final result (default); ‘2‘: visualize all steps during optimization.
alpha	A ‘numeric‘ value between 0 and 1, used as the mixing parameter for the Elastic-Net penalization term applied to ‘X‘. Only relevant if ‘penalty‘ is "EN". Default is 0.5.
alpha_g	A ‘numeric‘ value between 0 and 1, used as the mixing parameter for the Elastic-Net penalization term applied to covariates ‘C‘ in covYI. Only relevant if ‘penalty_g‘ is "EN". Default is 0.5.
tau	A ‘numeric‘ vector of penalization parameters for the covariates ‘C‘ in covYI. If 0 (default), no penalization term is applied to covariates. If ‘c_function_of_covariates‘ is ‘TRUE‘, ‘tau‘ cannot be ‘NULL‘ or contain only zeros.
a1	A ‘numeric‘ value specifying the regularization parameter ’a’ for the SCAD penalty (as defined by Fan and Li). Only relevant if ‘penalty‘ is "SCAD". Default is 3.7.

a2 A ‘numeric‘ value specifying the regularization parameter ’gamma’ for the MCP penalty (as defined by Zhang). Only relevant if ‘penalty‘ is "MCP". Default is 3.0.

penalty A ‘character‘ string indicating the type of penalty considered for the pye calculation. Must be one of "L12", "L1", "EN" (Elastic-Net), "SCAD", or "MCP". Default is "L1".

regressors_betas
numeric vector. An optional vector containing the "true" beta coefficients, if known. This is used to calculate additional performance measures related to variable selection accuracy. Default is ‘NULL‘.

seed A ‘numeric‘ value to fix the random seed for reproducibility. Default is 1.

used_cores A ‘numeric‘ value indicating the number of cores to use for parallelization. If equal to 1 (default), no parallelization is adopted.

trend A ‘character‘ string indicating the optimization algorithm variant to use for pye estimation. If "monotone" (default), the Monotone Accelerated Proximal Gradient (mmAPG) algorithm is used. If "nonmonotone", the Non-Monotone Accelerated Proximal Gradient (mnmAPG) algorithm is used.

delta A ‘numeric‘ value specifying the parameter for the convergence condition of the optimization algorithm for pye. Default is 1e-5.

max_alpha A ‘numeric‘ value representing the maximum allowed value for the step-size parameter ‘alpha‘ in the backtracking line-search for pye optimization. Default is 10000.

kernel A ‘character‘ string indicating the kernel type to use for the estimation of the density function for pye. Currently, only "gaussian" is tested and supported. Default is "gaussian".

beta_start_input
A ‘numeric‘ vector of specific starting points for the beta coefficients. If ‘NULL‘ (default), the starting points are determined by ‘beta_start_default‘. If provided, its length must match the number of regressors in ‘X‘.

max_iter A ‘numeric‘ value specifying the maximum number of iterations for the optimization algorithms (mmAPG and mnmAPG) for pye. Default is 10000.

min_alpha A ‘numeric‘ value representing the minimum allowed value for the step-size parameter ‘alpha‘ for pye optimization. Default is 1e-10.

convergence_error
A ‘numeric‘ value specifying the error tolerance for considering the pye optimization algorithm converged. Default is 1e-7.

stepsizeShrink A ‘numeric‘ value between 0 and 1, used to adjust the step-size in the backtracking line-search within the pye optimization. Values closer to 1 lead to higher accuracy but longer estimation times. Default is 0.8.

beta_start_default
A ‘character‘ string defining the default starting points for betas if ‘beta_start_input‘ is ‘NULL‘. If "zeros" (default), betas start with a vector of all zeros. If "corr", betas start with the absolute value of the correlation of each regressor with the target variable ‘y‘.

`scaling` A ‘logical‘ value. If ‘TRUE‘, the input dataset ‘df‘ is scaled internally. If ‘FALSE‘ (default), no scaling is performed.

`c_zero_fixed` A ‘logical‘ value. If ‘TRUE‘, the estimation process considers the cut-off point ‘c‘ as fixed and equal to zero, which can reduce estimation complexity. If ‘FALSE‘ (default), ‘c‘ can vary and is estimated by the pye optimization.

`c_function_of_covariates`
A ‘logical‘ value. If ‘TRUE‘, covYI is used to estimate the cut-off point ‘c‘ as a function of the covariate information (‘C‘). If ‘FALSE‘ (default), covariate information is ignored for the cut-off estimation.

`simultaneous` A ‘logical‘ value. If ‘c_function_of_covariates‘ is ‘TRUE‘, this parameter defines if ‘gammas‘ (covariate coefficients) need to be estimated simultaneously with ‘betas‘ or as a second step. Default is ‘FALSE‘, meaning a sequential estimation.

`measure_to_select_lambda`
A ‘character‘ string specifying the measure used to select ‘lambda‘ if ‘simultaneous‘ is ‘FALSE‘ (i.e., when the cross-validation process selects ‘lambda‘ first and then ‘tau‘). Must be one of "auc", "aauc", "aYI", "yi", "sen", "spc", "gm", "fdr", "mcc", or "ccr". Default is "ccr" (correct classification rate).

`penalty_g` A ‘character‘ string indicating the type of penalty for covYI (when ‘c_function_of_covariates‘ is ‘TRUE‘). Must be one of "L12", "L1", "EN" (Elastic-Net), "SCAD", or "MCP". Default is "L1".

`kernel_g` A ‘character‘ string indicating the kernel type to use for the density estimation in covYI. Currently, only "gaussian" is tested and supported. Default is "gaussian".

`a1_g` A ‘numeric‘ value specifying the regularization parameter ’a’ for the SCAD penalty in covYI. Only relevant if ‘penalty_g‘ is "SCAD". Default is 3.7.

`a2_g` A ‘numeric‘ value specifying the regularization parameter ’gamma’ for the MCP penalty in covYI. Only relevant if ‘penalty_g‘ is "MCP". Default is 3.0.

`trend_g` A ‘character‘ string indicating the optimization algorithm variant to use for covYI. If "monotone" (default), mmAPG is used. If "nonmonotone", mnmAPG is used.

`gamma_start_input`
A ‘numeric‘ vector of specific starting points for the gamma coefficients in covYI. If ‘NULL‘ (default), starting points are determined by ‘gamma_start_default‘.

`gamma_start_default`
A ‘character‘ string defining the default starting point of gamma coefficients. If "zeros" (default), they start with all zero values. If "corr", they start with the absolute correlation of each covariate with the target variable.

`regressors_gammas`
A ‘numeric‘ vector containing the true gamma coefficients (if known, for simulation/testing purposes). Default is ‘NULL‘.

`max_iter_g` A ‘numeric‘ value specifying the maximum number of iterations for the optimization algorithms (mmAPG and mnmAPG) in covYI. Default is 10000.

`delta_g` A ‘numeric‘ value specifying the parameter for the convergence condition of the optimization algorithm of covYI. Default is 1e-5.

`max_alpha_g` A ‘numeric‘ value representing the maximum allowed value for the step-size parameter ‘alpha‘ in covYI. Default is 10000.

`stepsizeShrink_g`
: A 'numeric' value between 0 and 1, used to adjust the step-size in the backtracking line-search within the covYI optimization. Values closer to 1 lead to higher accuracy but longer estimation times. Default is 0.8.

`min_alpha_g`
: A 'numeric' value representing the minimum allowed value for the step-size parameter 'alpha' in covYI. Default is 1e-12.

`convergence_error_g`
: A 'numeric' value specifying the error tolerance for considering the covYI algorithm converged. Default is 1e-7.

`run_aauc`
: A 'logical' value. If 'FALSE' (default), the aAUC and aYI measures are not computed, saving estimation time if not requested.

`log_file`
: Character. Path to a file for logging output from parallel workers. If 'NULL', output goes to the console. Default is "log_pye_ks_models.txt".

Value

A 'list' with the following components:

`penalty`
: `character`. The type of penalty used for regressors X.

`penalty_g`
: `character`. The type of penalty used for covariates C (if applicable).

`kernel`
: `character`. The kernel type used for density estimation of pye.

`cv_time`
: `difftime`. The total time taken for the cross-validation process, in minutes.

`pye_KS_L12`, `pye_KS_L1`, `pye_KS_EN`, `pye_KS_SCAD`, `pye_KS_MCP`
: `list`. For each penalty type, a list containing 'train' and 'test' matrices. These matrices store the Penalized Youden Index values for each fold and each combination of 'lambda' and 'tau' (if applicable). Their dimensions are 'n_folds' rows by 'length(tau)' columns (or 1 if 'length(tau)' is 1).

`auc_first_step`, `aauc_first_step`, `aYI_first_step`, `youden_index_first_step`, `sensitivity_first_step`, `specificity_first_step`, `geometric_mean_first_step`, `fdr_first_step`, `mcc_first_step`, `corrclass_first_step`, `auc`, `aauc`, `aYI`, `youden_index`, `sensitivity`, `specificity`, `geometric_mean`, `fdr`, `mcc`, `corrclass`
: `list`. Similar to 'pye_KS_penalty' components, these lists contain 'train' and 'test' matrices for each respective performance measure (AUC, adaptive AUC, adaptive Youden Index, unpenalized Youden Index, sensitivity, specificity, geometric mean, false discovery rate, Matthews Correlation Coefficient, and correct classification rate) of the first-step method and of the final method (which coincide if c_function_of_covariates is FALSE), across all folds and 'lambda'/'tau' combinations.

`lambda_hat_yi`, `lambda_hat_auc`, `lambda_hat_aauc`, `lambda_hat_aYI`, `lambda_hat_ccr`, `lambda_hat_sen`, `lambda_hat_spc`, `lambda_hat_gm`, `lambda_hat_pye`
: `numeric`. The optimal 'lambda' value selected based on maximizing the corresponding performance measure (e.g., '_yi' for Youden Index, '_auc' for AUC). If no optimal lambda is found, it will be 'NA'.

`tau_hat_yi`, `tau_hat_auc`, `tau_hat_aauc`, `tau_hat_aYI`, `tau_hat_ccr`, `tau_hat_sen`, `tau_hat_spc`, `tau_hat_gm`, `tau_hat_pye`
: `numeric`. The optimal ‘tau‘ value selected based on maximizing the corresponding performance measure. This is only relevant and will be a value other than ‘NA‘ if ‘c_function_of_covariates‘ is ‘TRUE‘. Otherwise, it will be ‘NA‘.

`c_function_of_covariates`
: `logical`. Indicates whether the cut-off point was estimated as a function of covariates.

`simultaneous`
: `logical`. Indicates whether betas and gammas were estimated simultaneously (if ‘c_function_of_covariates‘ was ‘TRUE‘).

`measure_to_select_lambda`
: `character`. The measure used to select the optimal ‘lambda‘ (if ‘simultaneous‘ is ‘FALSE‘).

`lambda_star`
: `numeric`. The single optimal lambda value chosen (when ‘c_function_of_covariates‘ is ‘TRUE‘ and ‘simultaneous‘ is ‘FALSE‘), based on ‘measure_to_select_lambda‘. ‘NA‘ otherwise.

`n_betas`
: `matrix`. A matrix showing the number of non-zero (selected) betas for each fold and each ‘lambda‘ value. Its dimensions are ‘n_folds‘ rows by ‘length(lambda)‘ columns.

`n_betas_star`
: `matrix`. If a two-step optimization (sequential lambda then tau) is performed (‘c_function_of_covariates = TRUE‘ and ‘simultaneous = FALSE‘), this matrix shows the number of non-zero betas for each fold using ‘lambda_star‘. Dimensions are ‘n_folds‘ rows by 1 column.

`n_gammas`
: `list`. A list, where each element corresponds to a ‘lambda‘ value and contains a matrix showing the number of non-zero (selected) gammas for each fold and each ‘tau‘ value. Dimensions are ‘n_folds‘ rows by ‘length(tau)‘ columns.

`betas`
: `matrix` or `list`. The estimated beta coefficients. If ‘length(lambda)‘ is 1, a matrix with ‘n_folds‘ rows. Otherwise, a list of beta vectors for each fold and each lambda.

`c_pye`
: `list`. A list where each element corresponds to a `lambda` value and contains a matrix showing the estimated optimal cut-off point c for each fold and each `lambda` value. If `c_zero_fixed` is `TRUE`, all entries will be 0.

`betas_star`
: `matrix` or `list`. If a two-step optimization is performed, the estimated beta coefficients using ‘lambda_star‘. A matrix with ‘n_folds‘ rows if ‘length(lambda_star)‘ is 1, otherwise a list of beta vectors.

`gammas`
: `list`. A list, where each element corresponds to a ‘lambda‘ value (or ‘lambda_star‘ if sequential) and contains a matrix or vector of estimated gamma coefficients for each fold and ‘tau‘ value.

Examples

```
# Load the package
library(pye)

# 1. Simulate data for the example
```

```
sim_data <- create_sample_with_covariates(
  rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- sim_data$train_df_scaled
X <- sim_data$X
y <- sim_data$y
C <- sim_data$C
regressors_betas <- sim_data$nregressors # True betas for evaluation
regressors_gammas <- sim_data$ncovariates # True gammas for evaluation

# 2. Set cross-validation parameters
penalty <- "SCAD" # Penalty for betas in pye estimation
penalty_g <- "L12" # Penalty for gammas in covYI estimation
trend <- "monotone" # Trend for the KS estimation
alpha <- 0.5
c_function_of_covariates <- TRUE # Use covariates for 'c' estimation
used_cores <- 1 # For this example, no parallelization
max_iter <- 10 # Keep iterations low for a quick example run

# 3. Calibrate lambda_max and lambda_min for betas (pye estimation)
lambda_seq <- create_lambda(n = 3, lmax = 1.5, lmin = 0.05)
lambda_seq <- as.numeric(formatC(lambda_seq, format = "e", digits = 9))

# 4. Calibrate tau_max and tau_min for gammas (covYI estimation), if
tau_seq <- create_lambda(n = 3, lmax = 0.15, lmin = 0.005)
tau_seq <- as.numeric(formatC(tau_seq, format = "e", digits = 9))

# 5. Run the cross-validation
pye_cv_result <- pye_KS_compute_cv(
  n_folds = 3,
  df = df,
  X = X,
  y = y,
  C = C,
  lambda = lambda_seq,
  tau = tau_seq,
  trace = 1, # Show final results
  alpha = alpha,
  penalty = penalty,
  regressors_betas = regressors_betas,
  regressors_gammas = regressors_gammas,
  seed = 1,
  used_cores = used_cores,
  trend = trend,
  max_iter = max_iter,
  c_function_of_covariates = c_function_of_covariates,
  measure_to_select_lambda = "ccr",
  penalty_g = penalty_g,
  trend_g = trend,
  max_iter_g = max_iter
)

# 6. Print results and access optimal lambda/tau
cat("\nOptimal Lambda (based on CCR):", pye_cv_result$lambda_hat_ccr, "\n")
```

```
if (c_function_of_covariates == TRUE) {
  cat("Optimal Tau (based on CCR):", pye_cv_result$tau_hat_ccr, "\n")
}

# You can access other results like:
pye_cv_result$auc # AUC values
pye_cv_result$n_betas # Number of non-zero betas for each lambda
pye_cv_result$n_gammas # Number of non-zero gammas for each tau
```

---

pye_KS_estimation *pye KS Estimation for Coefficient and Feature Selection*

Description

function to estimate the optimal value of betas and c maximizing the pye function. To find the optimum the mmAPG and mnmAPG algorithms are used.

Usage

```
pye_KS_estimation(
  df,
  X = NULL,
  y = "y",
  lambda,
  penalty = "L1",
  w = 0.5,
  beta_start_input = NULL,
  beta_start_default = "zeros",
  max.print = 10,
  alpha = 0.5,
  a1 = 3.7,
  a2 = 3,
  regressors_betas = NULL,
  fold = NULL,
  max_iter = 10000,
  trend = "monotone",
  delta = 1e-05,
  max_alpha = 10000,
  stepsizeShrink = 0.8,
  min_alpha = 1e-10,
  convergence_error = 1e-07,
  trace = 1,
  seed = 1,
  c_zero_fixed = FALSE,
  kernel = "gaussian",
  zeros_stay_zeros_from_iteration = 20
)
```

Arguments

df	A ‘data.frame‘ containing the input dataset. Must include the columns specified in ‘X‘ and ‘y‘.
X	A ‘character‘ vector specifying the names of the regressor variables (biomarkers) to consider in the estimation. Alternatively, a ‘data.frame‘ containing only these regressors can be provided, in which case their column names will be used. Default is all columns in ‘df‘ not specified as ‘y‘.
y	A ‘character‘ string specifying the name of the target binary variable (outcome). This variable must contain only values 0 and 1. Alternatively, a single-column ‘data.frame‘ containing only the target variable can be provided, in which case its column name will be used. Default is "y".
lambda	A ‘numeric‘ value specifying the penalization parameter for the regressors ‘X‘. Must be a single non-negative value.
penalty	A ‘character‘ string indicating the type of penalty to apply. Must be one of "L12", "L1", "EN" (Elastic-Net), "SCAD", or "MCP". Default is "L1".
w	A ‘numeric‘ value between 0 and 1 specifying the weight for the Weighted Youden Index. Sensitivity is weighted by ‘w‘ and specificity by ‘1 - w‘. Default is 0.5.
beta_start_input	A ‘numeric‘ vector of specific starting points for the beta coefficients. If ‘NULL‘ (default), the starting points are determined by ‘beta_start_default‘. If provided, its length must match the number of regressors in ‘X‘.
beta_start_default	A ‘character‘ string defining the default starting points for betas if ‘beta_start_input‘ is ‘NULL‘. If "zeros" (default), betas start with a vector of all zeros. If "corr", betas start with the absolute value of the correlation of each regressor with the target variable ‘y‘.
max.print	The number of elements to show when printing results. Default is 10.
alpha	A ‘numeric‘ value between 0 and 1, used as the mixing parameter for the Elastic-Net penalization term. Only relevant if ‘penalty‘ is "EN". Default is 0.5.
a1	A ‘numeric‘ value specifying the regularization parameter ’a’ for the SCAD penalty (as defined in the original paper by Fan and Li). Only relevant if ‘penalty‘ is "SCAD". Default is 3.7.
a2	A ‘numeric‘ value specifying the regularization parameter ’gamma’ for the MCP penalty (as defined in the original paper by Zhang). Only relevant if ‘penalty‘ is "MCP". Default is 3.0.
regressors_betas	numeric vector. An optional vector containing the "true" beta coefficients, if known. This is used to calculate additional performance measures related to variable selection accuracy. Default is ‘NULL‘.
fold	numeric. An optional fold number, used when the function is called within a cross-validation loop. This is primarily for tracking and reporting purposes. Default is ‘NULL‘.
max_iter	A ‘numeric‘ value specifying the maximum number of iterations for the optimization algorithms (mmAPG and mnmAPG). Default is 10000.

`trend` A ‘character‘ string indicating the optimization algorithm variant to use. If "monotone" (default), the Monotone Accelerated Proximal Gradient (mmAPG) algorithm is used. If "nonmonotone", the Non-Monotone Accelerated Proximal Gradient (mnmAPG) algorithm is used.

`delta` A ‘numeric‘ value specifying the parameter for the convergence condition of the optimization algorithm. Default is 1e-5.

`max_alpha` A ‘numeric‘ value representing the maximum allowed value for the step-size parameter ‘alpha‘ in the backtracking line-search. Default is 10000.

`stepsizeShrink` A ‘numeric‘ value between 0 and 1, used to adjust the step-size in the backtracking line-search within the optimization of pye. Values closer to 1 lead to higher accuracy but longer estimation times. Default is 0.8.

`min_alpha` A ‘numeric‘ value representing the minimum allowed value for the step-size parameter ‘alpha‘. Default is 1e-10.

`convergence_error`
A ‘numeric‘ value specifying the error tolerance for considering the optimization algorithm converged. Default is 1e-7.

`trace` A ‘numeric‘ value controlling the verbosity of the output. 0: no visualization, 1: visualize only the final result (default), 2: visualize all steps during optimization.

`seed` A ‘numeric‘ value to fix the random seed for reproducibility. Default is 1.

`c_zero_fixed` A ‘logical‘ value. If ‘TRUE‘, the estimation process considers the cut-off point ‘c‘ as fixed and equal to zero, which can reduce estimation complexity. If ‘FALSE‘ (default), ‘c‘ can vary and is estimated by the pye optimization.

`kernel` A ‘character‘ string indicating the kernel type to use for the estimation of the density function. Currently, only "gaussian" is tested and supported. Default is "gaussian".

`zeros_stay_zeros_from_iteration`
A ‘numeric‘ value specifying the iteration number from which parameters that have reached zero in the estimation cannot change (i.e., remain zero) anymore. This helps preserve the sparsity of the solution. Default is 20.

Value

A list with the following components:

`betas_hat_penalty`
`numeric vector`. The estimated optimal coefficients (betas) for the regressors X, with `penalty` replaced by the actual penalty used (e.g., `betas_hat_L1`). Non-zero values indicate selected features.

`pye_KS_penalty` `numeric`. The optimized Penalized Youden Index (pye) value achieved for the given `penalty` (e.g., `pye_KS_L1`).

`gr_yi` `numeric vector`. The gradient of the Youden Index at the estimated optimal point.

`lambda` `numeric`. The penalization parameter used in the estimation.

`penalty` `character`. The type of penalty used for regularization (e.g., "L1", "SCAD").

`betas_start` numeric vector. The initial starting point for the betas used in the optimization algorithm.

`kernel` character. The kernel type used for density estimation (e.g., "gaussian").

`c_hat` numeric. The estimated optimal cut-off point 'c'.

`z_hat` data.frame with columns `ID` and `z_hat`. The combined biomarker scores (`Z = X * betas`) for each observation based on the estimated `betas_hat`.

`y_hat` data.frame with columns `ID` and `y_hat`. The predicted binary outcomes (0 or 1) based on `z_hat` and `c_hat`.

`youden_index` numeric. The unpenalized Youden Index calculated using the Kernel Smooth density estimator at the estimated optimal point.

`sensitivity` numeric. The sensitivity (True Positive Rate) at the estimated optimal point.

`specificity` numeric. The specificity (True Negative Rate) at the estimated optimal point.

`geometric_mean` numeric. The geometric mean of sensitivity and specificity at the estimated optimal point.

`fdr` numeric. The False Discovery Rate (FP / (FP + TP)) at the estimated optimal point.

`mcc` numeric. The Matthews Correlation Coefficient at the estimated optimal point.

`auc` numeric. The Area Under the ROC Curve (AUC) for the `z_hat` scores at the estimated optimal point.

`corrclass` numeric. The overall correct classification rate ((TP + TN) / N) at the estimated optimal point.

`n_betas` numeric. The number of non-zero (selected) betas in the final estimation.

`n_total_var` numeric. The total number of regressors considered (length of X).

`n_predicted_zeros` numeric. The number of estimated betas that are exactly zero.

`n_predicted_non_zeros` numeric. The number of estimated betas that are non-zero.

`n_caught_betas` numeric or NA. If `regressors_betas` is provided, the number of true non-zero betas correctly estimated as non-zero. Otherwise NA.

`n_non_caught_betas` numeric or NA. If `regressors_betas` is provided, the number of true non-zero betas incorrectly estimated as zero. Otherwise NA.

`n_caught_zero` numeric or NA. If `regressors_betas` is provided, the number of true zero betas correctly estimated as zero. Otherwise NA.

`n_zero_not_caught` numeric or NA. If `regressors_betas` is provided, the number of true zero betas incorrectly estimated as non-zero. Otherwise NA.

`input_parameters` character vector. A list containing the input parameters.

`estimation_time` difftime. The total time taken for the estimation process, in minutes.

`niter` numeric. The total number of iterations performed by the optimization algorithm (mmAPG or mnmAPG).

Examples

```
# Load the package
library(pye)

# 1. Simulate data for the example
sim_data <- create_sample_with_covariates(
  rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- sim_data$train_df_scaled
X <- sim_data$X
y <- sim_data$y
regressors_betas <- sim_data$nregressors # True betas for evaluation

# 2. Set estimation parameters
penalty_type <- "SCAD"
lambda_val <- 0.1
trend_algo <- "monotone" # or "nonmonotone"
start_betas <- "zeros"
alpha_val <- 0.5
c_fixed <- FALSE

# 3. Run the pye estimation
pye_estimation_result <- pye_KS_estimation(
  df = df,
  X = X,
  y = y,
  penalty = penalty_type,
  trend = trend_algo,
  trace = 2, # Show full trace for example
  beta_start_default = start_betas,
  beta_start_input = NULL, # No specific starting point
  lambda = lambda_val,
  alpha = alpha_val,
  a1 = 3.7, # Default for SCAD
  a2 = 3.0, # Default for MCP (not used for SCAD here, but good to include if relevant)
  regressors_betas = regressors_betas,
  c_zero_fixed = c_fixed,
  max_iter = 5 # Keep iterations low for quick example run
)

# 4. Print results
print(pye_estimation_result)
```

---

pye_KS_simulation_study *Simulation Study for Penalized Youden Index (pye) on Real Data*

Description

This function performs a simulation study (repeated train/test splits) to estimate a penalized Youden Index (pye KS) model, and evaluates its performance on hold-out test data. It supports various penalties and optional cut-off point adjustment using covariates (covYI).

Usage

```
pye_KS_simulation_study(
  n = 1000,
  df,
  X = NULL,
  y = "y",
  C = NULL,
  lambda,
  tau = 0,
  w = 0.5,
  w_g = 0.5,
  train_data_proportion = 0.7,
  beta_start_input = NULL,
  beta_start_default = "zeros",
  trace = 1,
  alpha = 0.5,
  a1 = 3.7,
  a2 = 3,
  penalty = "L1",
  regressors_betas = NULL,
  used_cores = 1,
  scaling = FALSE,
  c_zero_fixed = FALSE,
  max_iter = 10000,
  trend = "monotone",
  delta = 1e-05,
  max_alpha = 10000,
  stepsizeShrink = 0.8,
  min_alpha = 1e-10,
  convergence_error = 1e-07,
  kernel = "gaussian",
  c_function_of_covariates = FALSE,
  alpha_g = 0.5,
  penalty_g = "L1",
  kernel_g = "gaussian",
  a1_g = 3.7,
  a2_g = 3,
  trend_g = "monotone",
  gamma_start_input = NULL,
  gamma_start_default = "zeros",
  regressors_gammas = NULL,
  max_iter_g = 10000,
```

```
  delta_g = 1e-05,
  max_alpha_g = 10000,
  stepsizeShrink_g = 0.8,
  min_alpha_g = 1e-12,
  convergence_error_g = 1e-07,
  run_aauc = FALSE,
  log_file = "log_sim_pye_real.txt"
)
```

Arguments

n	An integer, the number of simulation experiments to run. Default is 1000.
df	A data frame containing the complete dataset (y, X, C).
X	A character vector of column names from 'df' for regressors in the pye KS model. Defaults to all columns not 'y' or 'C'.
y	A character string, the column name for the binary target variable (0 or 1). Default is "y".
C	A character vector of column names from 'df' for covariates in the covYI model. Default is 'NULL'.
lambda	A numeric value, the penalization parameter λ for regressors X (pye KS model).
tau	A numeric value, the penalization parameter τ for covariates C (covYI model). Default is 0 (no penalization).
w	A numeric value between 0 and 1 specifying the weight for the Weighted Youden Index in pye. Sensitivity is weighted by 'w' and specificity by '1 - w'. Default is 0.5 (which corresponds to the standard Youden Index).
w_g	A numeric value between 0 and 1 specifying the weight for the Weighted Youden Index in covYI. Sensitivity is weighted by 'w_g' and specificity by '1 - w_g'. Default is 0.5 (which corresponds to the standard Youden Index).
train_data_proportion	A numeric value between 0 and 1. The proportion of the total observations in 'df' to be used for the training set in each simulation split. The split is performed using stratified sampling on 'y' to maintain class balance. Default is 0.7 (70% training, 30% testing).
beta_start_input	A numeric vector for custom starting β coefficients. Default is 'NULL'.
beta_start_default	A character string for default starting βs if 'beta_start_input' is 'NULL'. Options: "zeros", "corr". Default is "zeros".
trace	An integer (0, 1, or 2) controlling output verbosity. 0=none, 1=main, 2=all steps. Default is 1.
alpha	A numeric value [0, 1], the elastic-net mixing parameter for pye KS. Default is 0.5.
a1	A numeric value, the a parameter for SCAD/MCP in pye KS. Default is 3.7.
a2	A numeric value, the b parameter for MCP in pye KS. Default is 3.0.

penalty character. The penalty type for regressors (X) in pye. Options: "L12", "L1" (Lasso), "EN" (Elastic-Net), "SCAD", and "MCP". Default is "L1".

regressors_betas
numeric vector. The "true" β coefficients, if known, for variable selection accuracy. Default is 'NULL'.

used_cores An integer specifying CPU cores for parallelization. 0 uses 70% of available cores. Default is 1 (no parallel).

scaling A logical value. If 'TRUE', the dataset 'df' is scaled before processing. Default is 'FALSE'.

c_zero_fixed A logical value. If 'TRUE', the cut-off point c is fixed at zero. Default is 'FALSE'.

max_iter An integer, max iterations for pye KS optimization. Default is 10000.

trend A character string specifying the pye KS optimization algorithm trend. Options: "monotone" (mmAPG), "nonmonotone" (mnmAPG). Default is "monotone".

delta A numeric value, convergence tolerance for pye KS. Default is 1e-5.

max_alpha A numeric value, max step-size for pye KS backtracking. Default is 10000.

stepsizeShrink A numeric value [0, 1], shrinkage factor for pye KS backtracking step-size. Default is 0.8.

min_alpha A numeric value, min step-size for pye KS backtracking. Default is 1e-10.

convergence_error
A numeric value, the convergence threshold in pye KS. Default is 1e-7.

kernel A character string specifying the kernel type for pye KS density estimation (e.g., "gaussian"). Default is "gaussian".

c_function_of_covariates
A logical value. If 'TRUE', covYI is used to estimate cut-off c as a function of C. Default is 'FALSE'.

alpha_g A numeric value [0, 1], the elastic-net mixing parameter for covYI. Default is 0.5.

penalty_g A character string specifying the penalty type for covYI. Must be "L12", "L1", "EN", "SCAD", or "MCP". Default is "L1".

kernel_g A character string specifying the kernel type for covYI density estimation. Default is "gaussian".

a1_g A numeric value, the a parameter for SCAD/MCP in covYI. Default is 3.7.

a2_g A numeric value, the b parameter for MCP in covYI. Default is 3.0.

trend_g A character string specifying the covYI optimization algorithm trend. Default is "monotone".

gamma_start_input
A numeric vector for custom starting γ coefficients. Default is 'NULL'.

gamma_start_default
A character string for default starting γs if 'gamma_start_input' is 'NULL'. Options: "zeros", "corr". Default is "zeros".

regressors_gammas
A numeric vector representing the true γ coefficients (if known), for evaluation. Default is 'NULL'.

`max_iter_g` An integer, max iterations for covYI optimization. Default is 10000.

`delta_g` A numeric value, convergence tolerance for covYI. Default is 1e-5.

`max_alpha_g` A numeric value, max step-size for covYI backtracking. Default is 10000.

`stepsizeShrink_g`
A numeric value [0, 1], shrinkage factor for covYI backtracking step-size. Default is 0.8.

`min_alpha_g` A numeric value, min step-size for covYI backtracking. Default is 1e-12.

`convergence_error_g`
A numeric value, the convergence threshold in covYI. Default is 1e-7.

`run_aauc` A logical value. If 'FALSE', aAUC and aYI measures are not computed to save time. Default is 'FALSE'.

`log_file` Character. Path to a file for logging parallel output. Default is "log_sim_pye_real.txt".

Value

A `list` containing the aggregated results of the simulation study, including:

`simulation_time`
Total time taken for the entire simulation (time unit is in the output).

`estimation_time_original_method`
Mean estimation time for the pye KS model across all simulations.

`estimation_time_covYI`
Mean estimation time for the covYI model, if `c_function_of_covariates` is `TRUE`.

`used_cores` Number of CPU cores used for parallel processing.

`n` Number of simulation experiments performed.

`lambda`, `tau` The penalization parameters used for `X` and `C` respectively.

`pye_L12`, `pye_L1`, `pye_EN`, `pye_SCAD`, `pye_MCP`
Matrices (n x 2, for train/test) containing the pye KS objective function values, based on the selected `penalty` and the fitted betas.

`auc`, `youden_index`, `sensitivity`, `specificity`, `geometric_mean`, `fdr`, `mcc`, `corrclass`
Matrices (n x 2, for train/test) of classification performance measures for the pye KS model.

`auc_covYI`, `aauc_covYI`, `aYI_covYI`, `youden_index_covYI`, `sensitivity_covYI`, `specificity_covYI`, `geometric_mean_covYI`, `fdr_covYI`, `mcc_covYI`, `corrclass_covYI`
Matrices (n x 2, for train/test) of classification performance measures for the covYI model, if `c_function_of_covariates` is `TRUE`.

`n_total_var_betas`, `n_predicted_zeros_betas`, `n_predicted_non_zeros_betas`, `n_caught_betas`, `n_non_caught_betas`, `n_caught_zero_betas`, `n_zero_not_caught_betas`
Matrices related to variable selection performance (comparison with `regressors_betas`), where relevant.

n_total_var_gammas, n_predicted_zeros_gammas, n_predicted_non_zeros_gammas, n_caught_gammas, n_non_caught_gammas, n_caught_zero_gammas, n_zero_not_caught_gammas
: Matrices related to variable selection performance for gammas (comparison with regressors_gammas), if c_function_of_covariates is TRUE.

betas_times_selected
: A vector counting the number of times (out of n) each X regressor was estimated as non-zero.

gammas_times_selected
: A vector counting the number of times (out of n) each C covariate was estimated as non-zero, if c_function_of_covariates is TRUE.

betas_start
: The starting beta coefficients used.

c_zero_fixed
: A logical value indicating if the cut-off point c was fixed at zero.

roc_spec_points
: A vector of specificity points (e.g., seq(0, 1, by = 0.05)) used to compute the ROC curves.

roc_sens_points_train
: A list where each element contains the sensitivity points corresponding to roc_spec_points on the training data for each simulation.

roc_sens_points_test
: A list where each element contains the sensitivity points corresponding to roc_spec_points on the testing data for each simulation.

regressors_betas, regressors_gammas
: The true regressor vectors used for evaluation (if provided).

input_parameters
: character vector. A list containing the input parameters.

c_function_of_covariates, run_aauc
: The input parameters indicating if covYI was used and if advanced AUC metrics were run.

betas
: A list containing the vector of estimated beta coefficients from each simulation run.

gammas
: A list containing the vector of estimated gamma coefficients from each simulation run, if c_function_of_covariates is TRUE.

Examples

```
library(pye)

sim_data <- create_sample_with_covariates(
    rows_train = 100, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- sim_data$train_df_scaled
X <- sim_data$X
y <- sim_data$y
C <- sim_data$C
regressors_betas <- sim_data$nregressors
regressors_gammas <- sim_data$ncovariates
```

```
# Run a small simulation study
results_study <- pye_KS_simulation_study(
  n = 2, # Small number of simulations for example
  df = df,
  X = X,
  y = y,
  C = C,
  lambda = 1.3,
  tau = 0.8,
  beta_start_default = "zeros",
  gamma_start_default = "zeros",
  trace = 1,
  alpha = 0.5,
  alpha_g = 0.5,
  penalty = "L1",
  penalty_g = "L1",
  kernel = "gaussian",
  used_cores = 1, # Use 1 core for example
  c_function_of_covariates = TRUE,
  c_zero_fixed = FALSE,
  run_aauc = FALSE,
  max_iter = 5, # Reduced iterations for example
  max_iter_g = 5 # Reduced iterations for example
)

# Print some of the results
cat("Simulation Time: ", format(results_study$simulation_time, digits = 4))
cat("CCR:\n")
print(results_study$corrclass)
cat("Betas Times Selected:\n")
print(results_study$betas_times_selected[results_study$betas_times_selected > 0])
if (results_study$input_parameters$c_function_of_covariates) {
  cat("CCR with covYI:\n")
  print(results_study$corrclass_covYI)
  cat("\nGammas Times Selected:\n")
  print(results_study$gammas_times_selected[results_study$gammas_times_selected > 0])
}
```

`SCAD_function` *Smoothly Clipped Absolute Deviation (SCAD) Function Value*

Description

Calculates the value of the Smoothly Clipped Absolute Deviation (SCAD) penalty function, a popular type of non-convex regularization used for feature selection and estimation.

Usage

```
SCAD_function(betas, lambda, a)
```

Arguments

betas	numeric vector. The vector of coefficients (e.g., β) on which to apply the SCAD penalization.
lambda	numeric. The penalty parameter ($\lambda > 0$).
a	numeric. The hyperparameter of the SCAD penalization ($a > 2$). This parameter controls the clipping point and the rate at which the penalty tapers off. Common values include $a = 3.7$.

Details

The SCAD penalty applies hard-thresholding to large coefficients to reduce estimation bias while retaining the sparsity property of Lasso. The first derivative of the penalty function $p_\lambda(|\beta|)$ is:

- λ if $|\beta| \leq \lambda$
- $\frac{a\lambda - |\beta|}{a-1}$ if $\lambda < |\beta| \leq a\lambda$
- 0 if $|\beta| > a\lambda$

This function computes the sum of the integral of this derivative (the penalty function value itself) over all elements in betas using an assumed internal function, single_SCAD_function.

Value

A single numeric value representing the sum of the SCAD penalties applied to each element in the betas vector.

Examples

```
library(pye)

betas <- seq(0, 2, by = 0.2)
lambda <- 0.5
a <- 3.7
SCAD_function(betas = betas, lambda = lambda, a = a)
```

---

scaling_df_for_pye *Center and Scale Continuous Regressors for pye Modeling*

Description

Scales and centers specified numeric variables in a data frame (mean 0, SD 1). It automatically excludes the target variable (y) and variables identified as binary (dummy), ensuring only continuous regressors are standardized. This prepares data for models where standardization is necessary, such as the pye optimization.

Usage

```
scaling_df_for_pye(df, X, y)
```

Arguments

df	A data.frame or tibble containing all variables.
X	A character vector of column names considered as potential regressors. Only numeric variables from this list will be scaled.
y	A character string: the name of the target variable. This variable will not be scaled.

Details

The function first identifies potential regressors from the X argument. It then excludes variables that are the target (y), are not numeric, or are identified as binary (having exactly two unique values). Only the remaining continuous numeric variables are scaled. The scaling parameters are returned to allow for the consistent transformation of future datasets (e.g., test or new data).

Value

A list containing two elements:

df_scaled	A data.frame with the selected numeric regressors scaled (centered to mean 0 and scaled to SD 1).
scaling_params	A list containing: • original_mu: Means of the variables that were scaled. • original_stdev: Standard deviations of the variables. These parameters enable consistent scaling of new data.

Examples

```
library(pye)

# Simulate the dataframe
sim_data <- create_sample_with_covariates(
  rows_train = 100, rows_test = 50, cols = 40, cols_cov = 12, max_rho = 0.3, seed = 1)
df <- sim_data$df
X_cols_name <- sim_data$X
y_col_name <- sim_data$y
C_cols_name <- sim_data$C

# Now call the scaling function
scaled_output <- scaling_df_for_pye(
  df = df,
  X = c(X_cols_name, C_cols_name),
  y = y_col_name
)

print(head(scaled_output$df_scaled))
print(scaled_output$scaling_params$original_mu)
print(scaled_output$scaling_params$original_stdev)
```

Index