# Package: srlars (via r-universe)

November 3, 2024

**Type** Package

**Title** Split Robust Least Angle Regression

**Version** 1.0.1

**Date** 2023-07-16

**Maintainer** Anthony Christidis <anthony.christidis@stat.ubc.ca>

**Description** Functions to perform split robust least angle regression. The approach first uses the least angle regression algorithm to split the variables into the models of an ensemble and robust estimates of the correlation between predictors. An elastic net estimator is then applied to the selected predictors in each model using the imputed data from the detect deviating cell (DDC) method.

**License** GPL (>= 2)

**Encoding** UTF-8

**Biarch** true

**Imports** cellWise, glmnet

**Suggests** testthat, mvnfast

**RoxygenNote** 7.2.3

**Repository** https://anthonychristidis.r-universe.dev

**RemoteUrl** https://github.com/anthonychristidis/srlars

**RemoteRef** HEAD

**RemoteSha** ed6c0e6191979eae3c2a0fc00dfab98155d6cb45

## Contents

coef.srlars *Coefficients for srlars Object*

## Description

coef.srlars returns the coefficients for a srlars object.

## Usage

```
## S3 method for class 'srlars'
coef(object, group_index = NULL, ...)
```

## Arguments

| | |
|---|---|
| object | An object of class srlars |
| group_index | Groups included in the ensemble. Default setting includes all the groups. |
| ... | Additional arguments for compatibility. |

## Value

The coefficients for the srlars object.

## Author(s)

Anthony-Alexander Christidis, <anthony.christidis@stat.ubc.ca>

## See Also

srlars

## Examples

```
# Required library
library(mvnfast)

# Simulation parameters
n <- 50
p <- 500
rho.within <- 0.8
rho.between <- 0.2
p.active <- 100
group.size <- 25
snr <- 3
contamination.prop <- 0.2

# Setting the seed
set.seed(0)

# Block correlation structure
```

```
sigma.mat <- matrix(0, p, p)
sigma.mat[1:p.active, 1:p.active] <- rho.between
for(group in 0:(p.active/group.size - 1))
  sigma.mat[(group*group.size+1):(group*group.size+group.size),
  (group*group.size+1):(group*group.size+group.size)] <- rho.within
diag(sigma.mat) <- 1

# Simulation of beta vector
true.beta <- c(runif(p.active, 0, 5)*(-1)^rbinom(p.active, 1, 0.7), rep(0, p - p.active))

# Setting the SD of the variance
sigma <- as.numeric(sqrt(t(true.beta) %*% sigma.mat %*% true.beta)/sqrt(snr))

# Simulation of uncontaminated data
x <- mvnfast::rmvn(n, mu = rep(0, p), sigma = sigma.mat)
y <- x %*% true.beta + rnorm(n, 0, sigma)

# Contamination of data
contamination_indices <- 1:floor(n*contamination.prop)
k_lev <- 2
k_slo <- 100
x_train <- x
y_train <- y
beta_cont <- true.beta
beta_cont[true.beta!=0] <- beta_cont[true.beta!=0]*(1 + k_slo)
beta_cont[true.beta==0] <- k_slo*max(abs(true.beta))
for(cont_id in contamination_indices){

  a <- runif(p, min = -1, max = 1)
  a <- a - as.numeric((1/p)*t(a) %*% rep(1, p))
  x_train[cont_id,] <- mvnfast::rmvn(1, rep(0, p), 0.1^2*diag(p)) +
    k_lev * a / as.numeric(sqrt(t(a) %*% solve(sigma.mat) %*% a))
  y_train[cont_id] <- t(x_train[cont_id,]) %*% beta_cont
}

# Ensemble models
ensemble_fit <- srlars(x_train, y_train,
                       n_models = 5,
                       model_saturation = c("fixed", "p-value")[1],
                       alpha = 0.05, model_size = n - 1,
                       robust = TRUE,
                       compute_coef = TRUE,
                       en_alpha = 1/4)

# Ensemble coefficients
ensemble_coefs <- coef(ensemble_fit, group_index = 1:ensemble_fit$n_models)
sens_ensemble <- sum(which((ensemble_coefs[-1]!=0)) <= p.active)/p.active
spec_ensemble <- sum(which((ensemble_coefs[-1]!=0)) <= p.active)/sum(ensemble_coefs[-1]!=0)

# Simulation of test data
m <- 2e3
x_test <- mvnfast::rmvn(m, mu = rep(0, p), sigma = sigma.mat)
y_test <- x_test %*% true.beta + rnorm(m, 0, sigma)
```

```
# Prediction of test samples
ensemble_preds <- predict(ensemble_fit, newx = x_test,
                          group_index = 1:ensemble_fit$n_models,
                          dynamic = FALSE)
mspe_ensemble <- mean((y_test - ensemble_preds)^2)/sigma^2
```

---

`predict.srlars` *Predictions for srlars Object*

---

## Description

`predict.srlars` returns the predictions for a srlars object.

## Usage

```
## S3 method for class 'srlars'
predict(object, newx, group_index = NULL, dynamic = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `object` | An object of class srlars |
| `newx` | New data for predictions. |
| `group_index` | Groups included in the ensemble. Default setting includes all the groups. |
| `dynamic` | Argument to determine whether dynamic predictions are used based on deviating cells. Default is FALSE. |
| `...` | Additional arguments for compatibility. |

## Value

The predictions for the srlars object.

## Author(s)

Anthony-Alexander Christidis, <anthony.christidis@stat.ubc.ca>

## See Also

`srlars`

## Examples

```
# Required library
library(mvnfast)

# Simulation parameters
n <- 50
p <- 500
rho.within <- 0.8
rho.between <- 0.2
p.active <- 100
group.size <- 25
snr <- 3
contamination.prop <- 0.2

# Setting the seed
set.seed(0)

# Block correlation structure
sigma.mat <- matrix(0, p, p)
sigma.mat[1:p.active, 1:p.active] <- rho.between
for(group in 0:(p.active/group.size - 1))
  sigma.mat[(group*group.size+1):(group*group.size+group.size),
  (group*group.size+1):(group*group.size+group.size)] <- rho.within
diag(sigma.mat) <- 1

# Simulation of beta vector
true.beta <- c(runif(p.active, 0, 5)*(-1)^rbinom(p.active, 1, 0.7), rep(0, p - p.active))

# Setting the SD of the variance
sigma <- as.numeric(sqrt(t(true.beta) %*% sigma.mat %*% true.beta)/sqrt(snr))

# Simulation of uncontaminated data
x <- mvnfast::rmvn(n, mu = rep(0, p), sigma = sigma.mat)
y <- x %*% true.beta + rnorm(n, 0, sigma)

# Contamination of data
contamination_indices <- 1:floor(n*contamination.prop)
k_lev <- 2
k_slo <- 100
x_train <- x
y_train <- y
beta_cont <- true.beta
beta_cont[true.beta!=0] <- beta_cont[true.beta!=0]*(1 + k_slo)
beta_cont[true.beta==0] <- k_slo*max(abs(true.beta))
for(cont_id in contamination_indices){

  a <- runif(p, min = -1, max = 1)
  a <- a - as.numeric((1/p)*t(a) %*% rep(1, p))
  x_train[cont_id,] <- mvnfast::rmvn(1, rep(0, p), 0.1^2*diag(p)) +
    k_lev * a / as.numeric(sqrt(t(a) %*% solve(sigma.mat) %*% a))
  y_train[cont_id] <- t(x_train[cont_id,]) %*% beta_cont
}
```

```
# Ensemble models
ensemble_fit <- srlars(x_train, y_train,
                       n_models = 5,
                       model_saturation = c("fixed", "p-value")[1],
                       alpha = 0.05, model_size = n - 1,
                       robust = TRUE,
                       compute_coef = TRUE,
                       en_alpha = 1/4)

# Ensemble coefficients
ensemble_coefs <- coef(ensemble_fit, group_index = 1:ensemble_fit$n_models)
sens_ensemble <- sum(which((ensemble_coefs[-1]!=0)) <= p.active)/p.active
spec_ensemble <- sum(which((ensemble_coefs[-1]!=0)) <= p.active)/sum(ensemble_coefs[-1]!=0)

# Simulation of test data
m <- 2e3
x_test <- mvnfast::rmvn(m, mu = rep(0, p), sigma = sigma.mat)
y_test <- x_test %*% true.beta + rnorm(m, 0, sigma)

# Prediction of test samples
ensemble_preds <- predict(ensemble_fit, newx = x_test,
                          group_index = 1:ensemble_fit$n_models,
                          dynamic = FALSE)
mspe_ensemble <- mean((y_test - ensemble_preds)^2)/sigma^2
```

---

`srlars` *Robust Split Least Angle Regression*

---

## Description

`srlars` performs split robust least angle regression.

## Usage

```
srlars(
  x,
  y,
  n_models = 1,
  model_saturation = c("fixed", "p-value")[1],
  alpha = 0.05,
  model_size = NULL,
  robust = TRUE,
  compute_coef = FALSE,
  en_alpha = 1/4
)
```

## Arguments

| | |
|---|---|
| x | Design matrix. |
| y | Response vector. |
| n_models | Number of models into which the variables are split. |
| model_saturation | Criterion to determine if a model is saturated. Must be one of "fixed" (default) or "p-value". |
| alpha | P-value used to determine when the model is saturated |
| model_size | Size of the models in the ensemble. |
| robust | Argument to determine if robust measures of location, scale and correlation are used. Default is TRUE. |
| compute_coef | Argument to determine if coefficients are computed (via the elastic net) for each model. Default is FALSE. |
| en_alpha | Elastic net mixing parmeter for parameters shrinkage. Default is 1/4. |

## Value

An object of class srlars

## Author(s)

Anthony-Alexander Christidis, <anthony.christidis@stat.ubc.ca>

## See Also

coef.srlars, predict.srlars

## Examples

```
# Required library
library(mvnfast)

# Simulation parameters
n <- 50
p <- 500
rho.within <- 0.8
rho.between <- 0.2
p.active <- 100
group.size <- 25
snr <- 3
contamination.prop <- 0.2

# Setting the seed
set.seed(0)

# Block correlation structure
sigma.mat <- matrix(0, p, p)
sigma.mat[1:p.active, 1:p.active] <- rho.between
```

```
for(group in 0:(p.active/group.size - 1))
  sigma.mat[(group*group.size+1):(group*group.size+group.size),
  (group*group.size+1):(group*group.size+group.size)] <- rho.within
diag(sigma.mat) <- 1

# Simulation of beta vector
true.beta <- c(runif(p.active, 0, 5)*(-1)^rbinom(p.active, 1, 0.7), rep(0, p - p.active))

# Setting the SD of the variance
sigma <- as.numeric(sqrt(t(true.beta) %*% sigma.mat %*% true.beta)/sqrt(snr))

# Simulation of uncontaminated data
x <- mvnfast::rmvn(n, mu = rep(0, p), sigma = sigma.mat)
y <- x %*% true.beta + rnorm(n, 0, sigma)

# Contamination of data
contamination_indices <- 1:floor(n*contamination.prop)
k_lev <- 2
k_slo <- 100
x_train <- x
y_train <- y
beta_cont <- true.beta
beta_cont[true.beta!=0] <- beta_cont[true.beta!=0]*(1 + k_slo)
beta_cont[true.beta==0] <- k_slo*max(abs(true.beta))
for(cont_id in contamination_indices){

  a <- runif(p, min = -1, max = 1)
  a <- a - as.numeric((1/p)*t(a) %*% rep(1, p))
  x_train[cont_id,] <- mvnfast::rmvn(1, rep(0, p), 0.1^2*diag(p)) +
    k_lev * a / as.numeric(sqrt(t(a) %*% solve(sigma.mat) %*% a))
  y_train[cont_id] <- t(x_train[cont_id,]) %*% beta_cont
}

# Ensemble models
ensemble_fit <- srlars(x_train, y_train,
                       n_models = 5,
                       model_saturation = c("fixed", "p-value")[1],
                       alpha = 0.05, model_size = n - 1,
                       robust = TRUE,
                       compute_coef = TRUE,
                       en_alpha = 1/4)

# Ensemble coefficients
ensemble_coefs <- coef(ensemble_fit, group_index = 1:ensemble_fit$n_models)
sens_ensemble <- sum(which((ensemble_coefs[-1]!=0)) <= p.active)/p.active
spec_ensemble <- sum(which((ensemble_coefs[-1]!=0)) <= p.active)/sum(ensemble_coefs[-1]!=0)

# Simulation of test data
m <- 2e3
x_test <- mvnfast::rmvn(m, mu = rep(0, p), sigma = sigma.mat)
y_test <- x_test %*% true.beta + rnorm(m, 0, sigma)

# Prediction of test samples
```

```
ensemble_preds <- predict(ensemble_fit, newx = x_test,
                          group_index = 1:ensemble_fit$n_models,
                          dynamic = FALSE)
mspe_ensemble <- mean((y_test - ensemble_preds)^2)/sigma^2
```

# Index