Package 'CPGLIB'

May 11, 2021

Type Package

Title Competing Proximal Gradients Library

Version 1.0.1

Date 2021-05-10

Author Anthony Christidis <anthony.christidis@stat.ubc.ca>,
Stefan Van Aelst <stefan.vanaelst@kuleuven.be>,
Ruben Zamar <ruben@stat.ubc.ca>

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

Description Functions to generate ensembles of generalized linear models using competing proximal gradients. The optimal sparsity and diversity tuning parameters are selected via an alternating grid search.

License GPL (>= 2)

Biarch true

Imports Rcpp (>= 1.0.3)

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.0.2

Suggests testthat, vctrs, glmnet, mvnfast

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-05-11 00:02:11 UTC

R topics documented:

`coef.CPGLIB` *Coefficients for CPGLIB Object*

Description

`coef.CPGLIB` returns the coefficients for a CPGLIB object.

Usage

```
## S3 method for class 'CPGLIB'
coef(object, groups = NULL, ensemble_average = FALSE, ...)
```

Arguments

`object`	An object of class CPGLIB.
`groups`	The groups in the ensemble for the coefficients. Default is all of the groups in the ensemble.
`ensemble_average`	Option to return the average of the coefficients over all the groups in the ensemble. Default is FALSE.
`...`	Additional arguments for compatibility.

Value

The coefficients for the CPGLIB object.

Author(s)

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

See Also

`cpg`

Examples

```
# Data simulation
set.seed(1)
n <- 50
N <- 2000
p <- 300
beta.active <- c(abs(runif(p, 0, 1/2))*(-1)^rbinom(p, 1, 0.3))
# Parameters
p.active <- 150
beta <- c(beta.active[1:p.active], rep(0, p-p.active))
Sigma <- matrix(0, p, p)
Sigma[1:p.active, 1:p.active] <- 0.5
diag(Sigma) <- 1

# Train data
x.train <- mvnfast::rmvn(n, mu = rep(0, p), sigma = Sigma)
prob.train <- exp(x.train %*% beta)/
              (1+exp(x.train %*% beta))
y.train <- rbinom(n, 1, prob.train)
# Test data
x.test <- mvnfast::rmvn(N, mu = rep(0, p), sigma = Sigma)
prob.test <- exp(x.test %*% beta)/
             (1+exp(x.test %*% beta))
y.test <- rbinom(N, 1, prob.test)

# CPGLIB - Multiple Groups
cpg.out <- cpg(x.train, y.train,
               glm_type="Logistic",
               G=5, include_intercept=TRUE,
               alpha_s=3/4, alpha_d=1,
               lambda_sparsity=0.01, lambda_diversity=1,
               balanced_cycling=TRUE,
               tolerance=1e-5, max_iter=1e5)

# Coefficients for each group
cpg.coef <- coef(cpg.out, ensemble_average = FALSE)
```

`coef.cv.CPGLIB` *Coefficients for cv.CPGLIB Object*

Description

`coef.cv.CPGLIB` returns the coefficients for a cv.CPGLIB object.

Usage

```
## S3 method for class 'cv.CPGLIB'
coef(object, groups = NULL, ensemble_average = FALSE, ...)
```

Arguments

object	An object of class cv.CPGLIB.
groups	The groups in the ensemble for the coefficients. Default is all of the groups in the ensemble.
ensemble_average	Option to return the average of the coefficients over all the groups in the ensemble. Default is FALSE.
...	Additional arguments for compatibility.

Value

The coefficients for the cv.CPGLIB object. Default is FALSE.

Author(s)

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

See Also

cv.cpg

Examples

```
# Data simulation
set.seed(1)
n <- 50
N <- 2000
p <- 300
beta.active <- c(abs(runif(p, 0, 1/2))*(-1)^rbinom(p, 1, 0.3))
# Parameters
p.active <- 150
beta <- c(beta.active[1:p.active], rep(0, p-p.active))
Sigma <- matrix(0, p, p)
Sigma[1:p.active, 1:p.active] <- 0.5
diag(Sigma) <- 1

# Train data
x.train <- mvnfast::rmvn(n, mu = rep(0, p), sigma = Sigma)
prob.train <- exp(x.train %*% beta)/
              (1+exp(x.train %*% beta))
y.train <- rbinom(n, 1, prob.train)
# Test data
x.test <- mvnfast::rmvn(N, mu = rep(0, p), sigma = Sigma)
prob.test <- exp(x.test %*% beta)/
             (1+exp(x.test %*% beta))
y.test <- rbinom(N, 1, prob.test)
mean(y.test)

# CV CPGLIB - Multiple Groups
```

```
cpg.out <- cv.cpg(x.train, y.train,
                  glm_type = "Logistic",
                  G = 5, include_intercept = TRUE,
                  alpha_s = 3/4, alpha_d = 1,
                  n_lambda_sparsity = 100, n_lambda_diversity = 100,
                  balanced_cycling = TRUE,
                  tolerance = 1e-5, max_iter = 1e5)
cpg.coef <- coef(cpg.out)

# Coefficients for each group
cpg.coef <- coef(cpg.out, ensemble_average = FALSE)
```

`coef.cv.ProxGrad` *Coefficients for cv.ProxGrad Object*

Description

`coef.cv.ProxGrad` returns the coefficients for a cv.ProxGrad object.

Usage

```
## S3 method for class 'cv.ProxGrad'
coef(object, ...)
```

Arguments

`object`	An object of class cv.ProxGrad.
`...`	Additional arguments for compatibility.

Value

The coefficients for the cv.ProxGrad object.

Author(s)

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

See Also

`cv.ProxGrad`

Examples

```
# Data simulation
set.seed(1)
n <- 50
N <- 2000
p <- 1000
beta.active <- c(abs(runif(p, 0, 1/2))*(-1)^rbinom(p, 1, 0.3))
# Parameters
p.active <- 100
beta <- c(beta.active[1:p.active], rep(0, p-p.active))
Sigma <- matrix(0, p, p)
Sigma[1:p.active, 1:p.active] <- 0.5
diag(Sigma) <- 1

# Train data
x.train <- mvnfast::rmvn(n, mu = rep(0, p), sigma = Sigma)
prob.train <- exp(x.train %*% beta)/
              (1+exp(x.train %*% beta))
y.train <- rbinom(n, 1, prob.train)
# Test data
x.test <- mvnfast::rmvn(N, mu = rep(0, p), sigma = Sigma)
prob.test <- exp(x.test %*% beta)/
             (1+exp(x.test %*% beta))
y.test <- rbinom(N, 1, prob.test)

# CV ProxGrad - Single Group
proxgrad.out <- cv.ProxGrad(x.train, y.train,
                            glm_type = "Logistic",
                            include_intercept = TRUE,
                            alpha_s = 3/4,
                            n_lambda_sparsity = 100,
                            acceleration = TRUE,
                            tolerance = 1e-5, max_iter = 1e5)

# Coefficients
coef(proxgrad.out)
```

`coef.ProxGrad` *Coefficients for ProxGrad Object*

Description

`coef.ProxGrad` returns the coefficients for a ProxGrad object.

Usage

```
## S3 method for class 'ProxGrad'
coef(object, ...)
```

Arguments

object	An object of class ProxGrad.
...	Additional arguments for compatibility.

Value

The coefficients for the ProxGrad object.

Author(s)

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

See Also

ProxGrad

Examples

```
# Data simulation
set.seed(1)
n <- 50
N <- 2000
p <- 1000
beta.active <- c(abs(runif(p, 0, 1/2))*(-1)^rbinom(p, 1, 0.3))
# Parameters
p.active <- 100
beta <- c(beta.active[1:p.active], rep(0, p-p.active))
Sigma <- matrix(0, p, p)
Sigma[1:p.active, 1:p.active] <- 0.5
diag(Sigma) <- 1

# Train data
x.train <- mvnfast::rmvn(n, mu = rep(0, p), sigma = Sigma)
prob.train <- exp(x.train %*% beta)/
              (1+exp(x.train %*% beta))
y.train <- rbinom(n, 1, prob.train)
# Test data
x.test <- mvnfast::rmvn(N, mu = rep(0, p), sigma = Sigma)
prob.test <- exp(x.test %*% beta)/
             (1+exp(x.test %*% beta))
y.test <- rbinom(N, 1, prob.test)

# ProxGrad - Single Group
proxgrad.out <- ProxGrad(x.train, y.train,
                         glm_type = "Logistic",
```

```
                        include_intercept = TRUE,
                        alpha_s = 3/4,
                        lambda_sparsity = 0.01,
                        acceleration = TRUE,
                        tolerance = 1e-5, max_iter = 1e5)

# Coefficients
coef(proxgrad.out)
```

---

cpg *Competing Proximal Gradients Library for Ensembles of Generalized Linear Models*

Description

cpg computes the coefficients for ensembles of generalized linear models via competing proximal gradients.

Usage

```
cpg(
  x,
  y,
  glm_type = c("Linear", "Logistic", "Gamma", "Poisson")[1],
  G = 5,
  include_intercept = TRUE,
  alpha_s = 3/4,
  alpha_d = 1,
  lambda_sparsity,
  lambda_diversity,
  balanced_cycling = TRUE,
  permutate_search = FALSE,
  acceleration = FALSE,
  tolerance = 1e-05,
  max_iter = 1e+05
)
```

Arguments

x	Design matrix.
y	Response vector.
glm_type	Description of the error distribution and link function to be used for the model. Must be one of "Linear", "Logistic", "Gamma" or "Poisson". Default is "Linear".

G	Number of groups in the ensemble.
include_intercept	Argument to determine whether there is an intercept. Default is TRUE.
alpha_s	Sparsity mixing parmeter. Default is 3/4.
alpha_d	Diversity mixing parameter. Default is 1.
lambda_sparsity	Sparsity tuning parameter value.
lambda_diversity	Diversity tuning parameter value.
balanced_cycling	Argument to determine the cycling strategy for the optimal solution search. Default is TRUE.
permutate_search	Argument to determine whether permutations are used to search for the optimal solution. Default is FALSE.
acceleration	Argument to determine whether a gradient acceleration method is used. Default is FALSE.
tolerance	Convergence criteria for the coefficients. Default is 1e-3.
max_iter	Maximum number of iterations in the algorithm. Default is 1e5.

Value

An object of class cpg

Author(s)

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

See Also

coef.CPGLIB, predict.CPGLIB

Examples

```
# Data simulation
set.seed(1)
n <- 50
N <- 2000
p <- 300
beta.active <- c(abs(runif(p, 0, 1/2))*(-1)^rbinom(p, 1, 0.3))
# Parameters
p.active <- 150
beta <- c(beta.active[1:p.active], rep(0, p-p.active))
Sigma <- matrix(0, p, p)
Sigma[1:p.active, 1:p.active] <- 0.5
diag(Sigma) <- 1
```

```
# Train data
x.train <- mvnfast::rmvn(n, mu = rep(0, p), sigma  =  Sigma)
prob.train <- exp(x.train %*% beta)/
              (1+exp(x.train %*% beta))
y.train <- rbinom(n, 1, prob.train)
# Test data
x.test <- mvnfast::rmvn(N, mu = rep(0, p), sigma  =  Sigma)
prob.test <- exp(x.test %*% beta)/
             (1+exp(x.test %*% beta))
y.test <- rbinom(N, 1, prob.test)

# CPGLIB - Multiple Groups
cpg.out <- cpg(x.train, y.train,
               glm_type = "Logistic",
               G = 5, include_intercept = TRUE,
               alpha_s = 3/4, alpha_d = 1,
               lambda_sparsity = 0.01, lambda_diversity = 1,
               balanced_cycling = TRUE,
               tolerance = 1e-5, max_iter = 1e5)

# Predictions
cpg.prob <- predict(cpg.out, newx = x.test, type = "prob",
                    groups = 1:cpg.out$G, ensemble_type = "Model-Avg")
cpg.class <- predict(cpg.out, newx = x.test, type = "prob",
                     groups = 1:cpg.out$G, ensemble_type = "Model-Avg")
plot(prob.test, cpg.prob, pch = 20)
abline(h = 0.5,v = 0.5)
mean((prob.test-cpg.prob)^2)
mean(abs(y.test-cpg.class))
```

`cv.cpg` *Competing Proximal Gradients Library for Ensembles of Generalized Linear Models - Cross-Validation*

Description

`cv.cpg` computes and cross-validates the coefficients for ensembles of generalized linear models via competing proximal gradients.

Usage

```
cv.cpg(
  x,
  y,
  glm_type = c("Linear", "Logistic", "Gamma", "Poisson")[1],
  G = 5,
  full_diversity = FALSE,
```

```
  include_intercept = TRUE,
  alpha_s = 3/4,
  alpha_d = 1,
  n_lambda_sparsity = 100,
  n_lambda_diversity = 100,
  balanced_cycling = TRUE,
  permutate_search = FALSE,
  acceleration = FALSE,
  tolerance = 1e-05,
  max_iter = 1e+05,
  n_folds = 10,
  n_threads = 1
)
```

Arguments

x	Design matrix.
y	Response vector.
glm_type	Description of the error distribution and link function to be used for the model. Must be one of "Linear", "Logistic", "Gamma" or "Poisson". Default is "Linear".
G	Number of groups in the ensemble.
full_diversity	Argument to determine if the overlap between the models should be zero. Default is FALSE.
include_intercept	Argument to determine whether there is an intercept. Default is TRUE.
alpha_s	Sparsity mixing parmeter. Default is 3/4.
alpha_d	Diversity mixing parameter. Default is 1.
n_lambda_sparsity	Number of candidates for sparsity tuning parameter. Default is 100.
n_lambda_diversity	Number of candidates for diveristy tuning parameter. Default is 100.
balanced_cycling	Argument to determine the cycling strategy for the optimal solution search. Default is TRUE.
permutate_search	Argument to determine whether permutations are used to search for the optimal solution. Default is FALSE.
acceleration	Argument to determine whether a gradient acceleration method is used. Default is FALSE.
tolerance	Convergence criteria for the coefficients. Default is 1e-3.
max_iter	Maximum number of iterations in the algorithm. Default is 1e5.
n_folds	Number of cross-validation folds. Default is 10.
n_threads	Number of threads. Default is a single thread.

Value

An object of class cv.cpg

Author(s)

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

See Also

coef.cv.CPGLIB, predict.cv.CPGLIB

Examples

```
# Data simulation
set.seed(1)
n <- 50
N <- 2000
p <- 300
beta.active <- c(abs(runif(p, 0, 1/2))*(-1)^rbinom(p, 1, 0.3))
# Parameters
p.active <- 150
beta <- c(beta.active[1:p.active], rep(0, p-p.active))
Sigma <- matrix(0, p, p)
Sigma[1:p.active, 1:p.active] <- 0.5
diag(Sigma) <- 1

# Train data
x.train <- mvnfast::rmvn(n, mu = rep(0, p), sigma = Sigma)
prob.train <- exp(x.train %*% beta)/
              (1+exp(x.train %*% beta))
y.train <- rbinom(n, 1, prob.train)
# Test data
x.test <- mvnfast::rmvn(N, mu = rep(0, p), sigma = Sigma)
prob.test <- exp(x.test %*% beta)/
             (1+exp(x.test %*% beta))
y.test <- rbinom(N, 1, prob.test)

# CV CPGLIB - Multiple Groups
cpg.out <- cv.cpg(x.train, y.train,
                  glm_type = "Logistic",
                  G = 5, include_intercept = TRUE,
                  alpha_s = 3/4, alpha_d = 1,
                  n_lambda_sparsity = 100, n_lambda_diversity = 100,
                  balanced_cycling = TRUE,
                  tolerance = 1e-5, max_iter = 1e5)

# Predictions
cpg.prob <- predict(cpg.out, newx = x.test, type = "prob",
                    groups = 1:cpg.out$G, ensemble_type = "Model-Avg")
cpg.class <- predict(cpg.out, newx = x.test, type = "class",
                     groups = 1:cpg.out$G, ensemble_type = "Model-Avg")
```

```
plot(prob.test, cpg.prob, pch = 20)
abline(h = 0.5,v = 0.5)
mean((prob.test-cpg.prob)^2)
mean(abs(y.test-cpg.class))
```

`cv.ProxGrad` *Generalized Linear Models via Proximal Gradients - Cross-validation*

Description

`cv.ProxGrad` computes and cross-validates the coefficients for generalized linear models using accelerated proximal gradients.

Usage

```
cv.ProxGrad(
  x,
  y,
  glm_type = c("Linear", "Logistic", "Gamma", "Poisson")[1],
  include_intercept = TRUE,
  alpha_s = 3/4,
  n_lambda_sparsity = 100,
  acceleration = FALSE,
  tolerance = 0.001,
  max_iter = 1e+05,
  n_folds = 10,
  n_threads = 1
)
```

Arguments

`x`	Design matrix.
`y`	Response vector.
`glm_type`	Description of the error distribution and link function to be used for the model. Must be one of "Linear", "Logistic", "Gamma" or "Poisson". Default is "Linear".
`include_intercept`	Argument to determine whether there is an intercept. Default is TRUE.
`alpha_s`	Elastic net mixing parmeter. Default is 3/4.
`n_lambda_sparsity`	Sparsity tuning parameter value. Default is 100.
`acceleration`	Argument to determine whether a gradient acceleration method is used. Default is FALSE.

tolerance	Convergence criteria for the coefficients. Default is 1e-3.
max_iter	Maximum number of iterations in the algorithm. Default is 1e5.
n_folds	Number of cross-validation folds. Default is 10.
n_threads	Number of threads. Default is a single thread.

Value

An object of class cv.ProxGrad

Author(s)

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

See Also

coef.cv.ProxGrad, predict.cv.ProxGrad

Examples

```
# Data simulation
set.seed(1)
n <- 50
N <- 2000
p <- 1000
beta.active <- c(abs(runif(p, 0, 1/2))*(-1)^rbinom(p, 1, 0.3))
# Parameters
p.active <- 100
beta <- c(beta.active[1:p.active], rep(0, p-p.active))
Sigma <- matrix(0, p, p)
Sigma[1:p.active, 1:p.active] <- 0.5
diag(Sigma) <- 1

# Train data
x.train <- mvnfast::rmvn(n, mu = rep(0, p), sigma = Sigma)
prob.train <- exp(x.train %*% beta)/
              (1+exp(x.train %*% beta))
y.train <- rbinom(n, 1, prob.train)
# Test data
x.test <- mvnfast::rmvn(N, mu = rep(0, p), sigma = Sigma)
prob.test <- exp(x.test %*% beta)/
             (1+exp(x.test %*% beta))
y.test <- rbinom(N, 1, prob.test)

# ProxGrad - Single Groups
proxgrad.out <- cv.ProxGrad(x.train, y.train,
                            glm_type = "Logistic",
                            include_intercept = TRUE,
                            alpha_s = 3/4,
                            n_lambda_sparsity = 100,
                            acceleration = TRUE,
```

```
                              tolerance = 1e-5, max_iter = 1e5)

# Predictions
proxgrad.prob <- predict(proxgrad.out, newx = x.test, type = "prob")
proxgrad.class <- predict(proxgrad.out, newx = x.test, type = "class")
plot(prob.test, proxgrad.prob, pch = 20)
abline(h = 0.5,v = 0.5)
mean((prob.test-proxgrad.prob)^2)
mean(abs(y.test-proxgrad.class))
```

`plot.cv.CPGLIB` *Plot of coefficients paths for cv.CPGLIB Object*

Description

`plot.cv.CPGLIB` returns the coefficients for a cv.CPGLIB object, or a cross-validation errors plot.

Usage

```
## S3 method for class 'cv.CPGLIB'
plot(
  x,
  group_index = NULL,
  plot_type = c("Coef", "CV-Error")[1],
  active_only = TRUE,
  path_type = c("Log-Lambda", "L1-Norm")[1],
  labels = TRUE,
  ...
)
```

Arguments

`x`	An object of class cv.CPGLIB.
`group_index`	The group for which to return the coefficients. Default is the ensemble coefficients.
`plot_type`	Plot of coefficients, "Coef", or cross-validated error or deviance, "CV-Error". Default is "Coef".
`active_only`	Only include the variables selected in final model. Default is TRUE.
`path_type`	Plot of coefficients paths as a function of either "Log-Lambda" or "L1-Norm". Default is "Log-Lambda".
`labels`	Include the labels of the variables. Default is FALSE.
`...`	Additional arguments for compatibility.

Value

The coefficients or the cross-validation errors plot for a cv.CPGLIB object.

Author(s)

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

See Also

cv.cpg

Examples

```
# Data simulation
set.seed(1)
n <- 50
N <- 2000
p <- 300
beta.active <- c(abs(runif(p, 0, 1/2))*(-1)^rbinom(p, 1, 0.3))
# Parameters
p.active <- 150
beta <- c(beta.active[1:p.active], rep(0, p-p.active))
Sigma <- matrix(0, p, p)
Sigma[1:p.active, 1:p.active] <- 0.5
diag(Sigma) <- 1

# Train data
x.train <- mvnfast::rmvn(n, mu = rep(0, p), sigma = Sigma)
prob.train <- exp(x.train %*% beta)/
              (1+exp(x.train %*% beta))
y.train <- rbinom(n, 1, prob.train)
# Test data
x.test <- mvnfast::rmvn(N, mu = rep(0, p), sigma = Sigma)
prob.test <- exp(x.test %*% beta)/
             (1+exp(x.test %*% beta))
y.test <- rbinom(N, 1, prob.test)

# SplitGLM - CV (Multiple group_index)
cpg.out <- cv.cpg(x.train, y.train,
                  glm_type = "Logistic",
                  G = 10, include_intercept = TRUE,
                  alpha_s = 3/4, alpha_d = 1,
                  n_lambda_sparsity = 50, n_lambda_diversity = 50,
                  tolerance = 1e-3, max_iter = 1e3,
                  n_folds = 5,
                  n_threads = 1)

# Plot of coefficients paths (function of Log-Lambda)
plot(cpg.out, plot_type = "Coef", path_type = "Log-Lambda", group_index = 1, labels = FALSE)

# Plot of coefficients paths (function of L1-Norm)
```

```
plot(cpg.out, plot_type = "Coef", path_type = "L1-Norm", group_index = 1, labels = FALSE)

# Plot of CV error
plot(cpg.out, plot_type = "CV-Error")
```

---

predict.CPGLIB *Predictions for CPGLIB Object*

Description

predict.CPGLIB returns the predictions for a CPGLIB object.

Usage

```
## S3 method for class 'CPGLIB'
predict(
  object,
  newx,
  groups = NULL,
 ensemble_type = c("Model-Avg", "Coef-Avg", "Weighted-Prob", "Majority-Vote")[1],
  class_type = c("prob", "class")[1],
  ...
)
```

Arguments

object	An object of class CPGLIB.
newx	New data for predictions.
groups	The groups in the ensemble for the predictions. Default is all of the groups in the ensemble.
ensemble_type	The type of ensembling function for the models. Options are "Model-Avg", "Coef-Avg" or "Weighted-Prob" for classifications predictions. Default is "Model-Avg".
class_type	The type of predictions for classification. Options are "prob" and "class". Default is "prob".
...	Additional arguments for compatibility.

Value

The predictions for the CPGLIB object.

Author(s)

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

See Also

cpg

Examples

```
# Data simulation
set.seed(1)
n <- 50
N <- 2000
p <- 300
beta.active <- c(abs(runif(p, 0, 1/2))*(-1)^rbinom(p, 1, 0.3))
# Parameters
p.active <- 150
beta <- c(beta.active[1:p.active], rep(0, p-p.active))
Sigma <- matrix(0, p, p)
Sigma[1:p.active, 1:p.active] <- 0.5
diag(Sigma) <- 1

# Train data
x.train <- mvnfast::rmvn(n, mu = rep(0, p), sigma = Sigma)
prob.train <- exp(x.train %*% beta)/
              (1+exp(x.train %*% beta))
y.train <- rbinom(n, 1, prob.train)
# Test data
x.test <- mvnfast::rmvn(N, mu = rep(0, p), sigma = Sigma)
prob.test <- exp(x.test %*% beta)/
             (1+exp(x.test %*% beta))
y.test <- rbinom(N, 1, prob.test)

# CPGLIB - Multiple Groups
cpg.out <- cpg(x.train, y.train,
               glm_type = "Logistic",
               G = 5, include_intercept = TRUE,
               alpha_s = 3/4, alpha_d = 1,
               lambda_sparsity = 0.01, lambda_diversity = 1,
               balanced_cycling = TRUE,
               tolerance = 1e-5, max_iter = 1e5)

# Predictions
cpg.prob <- predict(cpg.out, newx = x.test, type = "prob",
                    groups = 1:cpg.out$G, ensemble_type = "Model-Avg")
cpg.class <- predict(cpg.out, newx = x.test, type = "prob",
                     groups = 1:cpg.out$G, ensemble_type = "Model-Avg")
plot(prob.test, cpg.prob, pch=20)
abline(h=0.5,v=0.5)
mean((prob.test-cpg.prob)^2)
mean(abs(y.test-cpg.class))
```

predict.cv.CPGLIB *Predictions for cv.ProxGrad Object*

Description

predict.cv.CPGLIB returns the predictions for a ProxGrad object.

Usage

```
## S3 method for class 'cv.CPGLIB'
predict(
  object,
  newx,
  groups = NULL,
 ensemble_type = c("Model-Avg", "Coef-Avg", "Weighted-Prob", "Majority-Vote")[1],
  class_type = c("prob", "class")[1],
  ...
)
```

Arguments

object	An object of class cv.CPGLIB.
newx	New data for predictions.
groups	The groups in the ensemble for the predictions. Default is all of the groups in the ensemble.
ensemble_type	The type of ensembling function for the models. Options are "Model-Avg", "Coef-Avg" or "Weighted-Prob" for classifications predictions. Default is "Model-Avg".
class_type	The type of predictions for classification. Options are "prob" and "class". Default is "prob".
...	Additional arguments for compatibility.

Value

The predictions for the cv.CPGLIB object.

Author(s)

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

See Also

cv.cpg

Examples

```
# Data simulation
set.seed(1)
n <- 50
N <- 2000
p <- 300
beta.active <- c(abs(runif(p, 0, 1/2))*(-1)^rbinom(p, 1, 0.3))
# Parameters
p.active <- 150
beta <- c(beta.active[1:p.active], rep(0, p-p.active))
Sigma <- matrix(0, p, p)
Sigma[1:p.active, 1:p.active] <- 0.5
diag(Sigma) <- 1

# Train data
x.train <- mvnfast::rmvn(n, mu = rep(0, p), sigma = Sigma)
prob.train <- exp(x.train %*% beta)/
              (1+exp(x.train %*% beta))
y.train <- rbinom(n, 1, prob.train)
# Test data
x.test <- mvnfast::rmvn(N, mu = rep(0, p), sigma = Sigma)
prob.test <- exp(x.test %*% beta)/
             (1+exp(x.test %*% beta))
y.test <- rbinom(N, 1, prob.test)
mean(y.test)

# CV CPGLIB - Multiple Groups
cpg.out <- cv.cpg(x.train, y.train,
                  glm_type = "Logistic",
                  G = 5, include_intercept = TRUE,
                  alpha_s = 3/4, alpha_d = 1,
                  n_lambda_sparsity = 100, n_lambda_diversity = 100,
                  balanced_cycling = TRUE,
                  tolerance = 1e-5, max_iter = 1e5)

# Predictions
cpg.prob <- predict(cpg.out, newx = x.test, type = "prob",
                    groups = 1:cpg.out$G, ensemble_type = "Model-Avg")
cpg.class <- predict(cpg.out, newx = x.test, type = "class",
                     groups = 1:cpg.out$G, ensemble_type = "Model-Avg")
plot(prob.test, cpg.prob, pch = 20)
abline(h = 0.5,v = 0.5)
mean((prob.test-cpg.prob)^2)
mean(abs(y.test-cpg.class))
```

predict.cv.ProxGrad *Predictions for cv.ProxGrad Object*

Description

predict.cv.ProxGrad returns the predictions for a ProxGrad object.

Usage

```
## S3 method for class 'cv.ProxGrad'
predict(object, newx, type = c("prob", "class")[1], ...)
```

Arguments

object	An object of class cv.ProxGrad.
newx	New data for predictions.
type	The type of predictions for binary response. Options are "prob" (default) and "class".
...	Additional arguments for compatibility.

Value

The predictions for the cv.ProxGrad object.

Author(s)

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

See Also

cv.ProxGrad

Examples

```
# Data simulation
set.seed(1)
n <- 50
N <- 2000
p <- 1000
beta.active <- c(abs(runif(p, 0, 1/2))*(-1)^rbinom(p, 1, 0.3))
# Parameters
p.active <- 100
beta <- c(beta.active[1:p.active], rep(0, p-p.active))
Sigma <- matrix(0, p, p)
Sigma[1:p.active, 1:p.active] <- 0.5
diag(Sigma) <- 1
```

```
# Train data
x.train <- mvnfast::rmvn(n, mu = rep(0, p), sigma = Sigma)
prob.train <- exp(x.train %*% beta)/
              (1+exp(x.train %*% beta))
y.train <- rbinom(n, 1, prob.train)
# Test data
x.test <- mvnfast::rmvn(N, mu = rep(0, p), sigma = Sigma)
prob.test <- exp(x.test %*% beta)/
             (1+exp(x.test %*% beta))
y.test <- rbinom(N, 1, prob.test)

# CV ProxGrad - Single Group
proxgrad.out <- cv.ProxGrad(x.train, y.train,
                            glm_type = "Logistic",
                            include_intercept = TRUE,
                            alpha_s = 3/4,
                            n_lambda_sparsity = 100,
                            acceleration = TRUE,
                            tolerance = 1e-5, max_iter = 1e5)

# Predictions
proxgrad.prob <- predict(proxgrad.out, newx = x.test, type = "prob")
proxgrad.class <- predict(proxgrad.out, newx = x.test, type = "class")
plot(prob.test, proxgrad.prob, pch = 20)
abline(h = 0.5,v = 0.5)
mean((prob.test-proxgrad.prob)^2)
mean(abs(y.test-proxgrad.class))
```

`predict.ProxGrad` *Predictions for ProxGrad Object*

Description

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

Usage

```
## S3 method for class 'ProxGrad'
predict(object, newx, type = c("prob", "class")[1], ...)
```

Arguments

`object`	An object of class ProxGrad
`newx`	New data for predictions.
`type`	The type of predictions for binary response. Options are "prob" (default) and "class".
`...`	Additional arguments for compatibility.

Value

The predictions for the ProxGrad object.

Author(s)

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

See Also

ProxGrad

Examples

```
# Data simulation
set.seed(1)
n <- 50
N <- 2000
p <- 1000
beta.active <- c(abs(runif(p, 0, 1/2))*(-1)^rbinom(p, 1, 0.3))
# Parameters
p.active <- 100
beta <- c(beta.active[1:p.active], rep(0, p-p.active))
Sigma <- matrix(0, p, p)
Sigma[1:p.active, 1:p.active] <- 0.5
diag(Sigma) <- 1

# Train data
x.train <- mvnfast::rmvn(n, mu = rep(0, p), sigma = Sigma)
prob.train <- exp(x.train %*% beta)/
              (1+exp(x.train %*% beta))
y.train <- rbinom(n, 1, prob.train)
# Test data
x.test <- mvnfast::rmvn(N, mu = rep(0, p), sigma = Sigma)
prob.test <- exp(x.test %*% beta)/
             (1+exp(x.test %*% beta))
y.test <- rbinom(N, 1, prob.test)

# ProxGrad - Single Group
proxgrad.out <- ProxGrad(x.train, y.train,
                         glm_type = "Logistic",
                         include_intercept = TRUE,
                         alpha_s = 3/4,
                         lambda_sparsity = 0.01,
                         acceleration = TRUE,
                         tolerance = 1e-5, max_iter = 1e5)

# Predictions
proxgrad.prob <- predict(proxgrad.out, newx = x.test, type = "prob")
proxgrad.class <- predict(proxgrad.out, newx = x.test, type = "class")
plot(prob.test, proxgrad.prob, pch = 20)
abline(h = 0.5,v = 0.5)
```

```
mean((prob.test-proxgrad.prob)^2)
mean(abs(y.test-proxgrad.class))
```

ProxGrad *Generalized Linear Models via Proximal Gradients*

Description

ProxGrad computes the coefficients for generalized linear models using accelerated proximal gradients.

Usage

```
ProxGrad(
  x,
  y,
  glm_type = c("Linear", "Logistic", "Gamma", "Poisson")[1],
  include_intercept = TRUE,
  alpha_s = 3/4,
  lambda_sparsity,
  acceleration = FALSE,
  tolerance = 1e-05,
  max_iter = 1e+05
)
```

Arguments

x	Design matrix.
y	Response vector.
glm_type	Description of the error distribution and link function to be used for the model. Must be one of "Linear", "Logistic", "Gamma" or "Poisson". Default is "Linear".
include_intercept	Argument to determine whether there is an intercept. Default is TRUE.
alpha_s	Elastic net mixing parmeter. Default is 3/4.
lambda_sparsity	Sparsity tuning parameter value.
acceleration	Argument to determine whether a gradient acceleration method is used. Default is FALSE.
tolerance	Convergence criteria for the coefficients. Default is 1e-3.
max_iter	Maximum number of iterations in the algorithm. Default is 1e5.

Value

An object of class ProxGrad.

Author(s)

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

See Also

coef.ProxGrad, predict.ProxGrad

Examples

```
# Data simulation
set.seed(1)
n <- 50
N <- 2000
p <- 1000
beta.active <- c(abs(runif(p, 0, 1/2))*(-1)^rbinom(p, 1, 0.3))
# Parameters
p.active <- 100
beta <- c(beta.active[1:p.active], rep(0, p-p.active))
Sigma <- matrix(0, p, p)
Sigma[1:p.active, 1:p.active] <- 0.5
diag(Sigma) <- 1

# Train data
x.train <- mvnfast::rmvn(n, mu = rep(0, p), sigma = Sigma)
prob.train <- exp(x.train %*% beta)/
              (1+exp(x.train %*% beta))
y.train <- rbinom(n, 1, prob.train)
# Test data
x.test <- mvnfast::rmvn(N, mu = rep(0, p), sigma = Sigma)
prob.test <- exp(x.test %*% beta)/
             (1+exp(x.test %*% beta))
y.test <- rbinom(N, 1, prob.test)

# ProxGrad - Single Group
proxgrad.out <- ProxGrad(x.train, y.train,
                         glm_type = "Logistic",
                         include_intercept = TRUE,
                         alpha_s = 3/4,
                         lambda_sparsity = 0.01,
                         acceleration = TRUE,
                         tolerance = 1e-5, max_iter = 1e5)

# Predictions
proxgrad.prob <- predict(proxgrad.out, newx = x.test, type = "prob")
proxgrad.class <- predict(proxgrad.out, newx = x.test, type = "class")
plot(prob.test, proxgrad.prob, pch = 20)
abline(h = 0.5,v = 0.5)
```

```
mean((prob.test-proxgrad.prob)^2)
mean(abs(y.test-proxgrad.class))
```

Index