Package ‘mvs’

August 15, 2023

Type Package

Title Methods for High-Dimensional Multi-View Learning

Version 1.0.2

Description Methods for high-dimensional multi-view learning based on the multi-view stacking (MVS) framework.
For technical details on the MVS and stacked penalized logistic regression (StaPLR) methods see Van Loon, Fokkema, Szabo, & De Rooij (2020) <doi:10.1016/j.inffus.2020.03.007> and Van Loon et al. (2022) <doi:10.3389/fnins.2022.830630>.

License GPL-2

Encoding UTF-8

RoxygenNote 7.2.3

Depends glmnet (>= 1.9-8)

Imports foreach (>= 1.4.4)

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Wouter van Loon [aut, cre],
Marjolein Fokkema [ctb]

Maintainer Wouter van Loon <w.s.van.loon@fsw.leidenuniv.nl>

Repository CRAN

Date/Publication 2023-08-15 11:00:05 UTC

R topics documented:

`mvs-package` *mvs: Methods for High-Dimensional Multi-View Learning.*

Description

Methods for high-dimensional multi-view learning based on the multi-view stacking (MVS) framework. For technical details on the MVS and StaPLR methods see <doi:10.1016/j.inffus.2020.03.007> and <doi:10.3389/fnins.2022.830630>.

Author(s)

Wouter van Loon [cre, aut] <`<w.s.van.loon@fsw.leidenuniv.nl>`>

Marjolein Fokkema [ctb]

`coef.MVS` *Extract coefficients from an "MVS" object.*

Description

Extract coefficients at each level from an "MVS" object at the CV-optimal values of the penalty parameters.

Usage

```
## S3 method for class 'MVS'
coef(object, cvlambda = "lambda.min", ...)
```

Arguments

`object`	An object of class "MVS".
`cvlambda`	By default, the coefficients are extracted at the CV-optimal values of the penalty parameters. Choosing "lambda.1se" will extract them at the largest values within one standard error of the minima.
`...`	Further arguments to be passed to `coef.cv.glmnet`.

Value

An object of S3 class "MVScoef".

Author(s)

Wouter van Loon <w.s.van.loon@fsw.leidenuniv.nl>

Examples

```
set.seed(012)
n <- 1000
X <- matrix(rnorm(8500), nrow=n, ncol=85)
top_level <- c(rep(1,45), rep(2,20), rep(3,20))
bottom_level <- c(rep(1:3, each=15), rep(4:5, each=10), rep(6:9, each=5))
views <- cbind(bottom_level, top_level)
beta <- c(rep(10, 55), rep(0, 30)) * ((rbinom(85, 1, 0.5)*2)-1)
eta <- X %*% beta
p <- 1 /(1 + exp(-eta))
y <- rbinom(n, 1, p)

fit <- MVS(x=X, y=y, views=views, type="StaPLR", levels=3, alphas=c(0,1,1), nnc=c(0,1,1))
coefficients <- coef(fit)

new_X <- matrix(rnorm(2*85), nrow=2)
predict(fit, new_X)
```

---

coef.StaPLR — *Extract coefficients from a "StaPLR" object.*

Description

Extract base- and meta-level coefficients from a "StaPLR" object at the CV-optimal values of the penalty parameters.

Usage

```
## S3 method for class 'StaPLR'
coef(object, cvlambda = "lambda.min", ...)
```

Arguments

`object`	Fitted "StaPLR" model object.
`cvlambda`	By default, the coefficients are extracted at the CV-optimal values of the penalty parameters. Choosing "lambda.1se" will extract them at the largest values within one standard error of the minima.
`...`	Further arguments to be passed to `coef.cv.glmnet`.

Value

An object with S3 class "StaPLRcoef".

Author(s)

Wouter van Loon <w.s.van.loon@fsw.leidenuniv.nl>

Examples

```
set.seed(012)
n <- 1000
cors <- seq(0.1,0.7,0.1)
X <- matrix(NA, nrow=n, ncol=length(cors)+1)
X[,1] <- rnorm(n)

for(i in 1:length(cors)){
  X[,i+1] <- X[,1]*cors[i] + rnorm(n, 0, sqrt(1-cors[i]^2))
}

beta <- c(1,0,0,0,0,0,0,0)
eta <- X %*% beta
p <- exp(eta)/(1+exp(eta))
y <- rbinom(n, 1, p)
view_index <- rep(1:(ncol(X)/2), each=2)

fit <- StaPLR(X, y, view_index)
coef(fit)$meta

new_X <- matrix(rnorm(16), nrow=2)
predict(fit, new_X)
```

MVS *Multi-View Stacking*

Description

Fit a multi-view stacking model with two or more levels.

Usage

```
MVS(
  x,
  y,
  views,
  type = "StaPLR",
  levels = 2,
  alphas = c(0, 1),
  nnc = c(0, 1),
  parallel = FALSE,
  seeds = NULL,
  progress = TRUE,
  ...
)

mvs(
```

```
  x,
  y,
  views,
  type = "StaPLR",
  levels = 2,
  alphas = c(0, 1),
  nnc = c(0, 1),
  parallel = FALSE,
  seeds = NULL,
  progress = TRUE,
  ...
)
```

Arguments

`x`	input matrix of dimension nobs x nvars.
`y`	outcome vector of length nobs.
`views`	a matrix of dimension nvars x (levels - 1), where each entry is an integer describing to which view each feature corresponds.
`type`	the type of MVS model to be fitted. Currently only type "StaPLR" is supported.
`levels`	an integer >= 2, specifying the number of levels in the MVS procedure.
`alphas`	a vector specifying the value of the alpha parameter to use at each level.
`nnc`	a binary vector specifying whether to apply nonnegativity constraints or not (1/0) at each level.
`parallel`	whether to use foreach to fit the learners and obtain the cross-validated predictions at each level in parallel. Executes sequentially unless a parallel back-end is registered beforehand.
`seeds`	(optional) a vector specifying the seed to use at each level.
`progress`	whether to show a progress bar (only supported when parallel = FALSE).
`...`	additional arguments to pass to the learning algorithm. See e.g. ?StaPLR. Note that these arguments are passed to the the learner at every level of the MVS procedure.

Value

An object of S3 class "MVS".

Author(s)

Wouter van Loon <w.s.van.loon@fsw.leidenuniv.nl>

Examples

```
set.seed(012)
n <- 1000
X <- matrix(rnorm(8500), nrow=n, ncol=85)
```

```
top_level <- c(rep(1,45), rep(2,20), rep(3,20))
bottom_level <- c(rep(1:3, each=15), rep(4:5, each=10), rep(6:9, each=5))
views <- cbind(bottom_level, top_level)
beta <- c(rep(10, 55), rep(0, 30)) * ((rbinom(85, 1, 0.5)*2)-1)
eta <- X %*% beta
p <- 1 /(1 + exp(-eta))
y <- rbinom(n, 1, p)

fit <- MVS(x=X, y=y, views=views, type="StaPLR", levels=3, alphas=c(0,1,1), nnc=c(0,1,1))
coefficients <- coef(fit)

new_X <- matrix(rnorm(2*85), nrow=2)
predict(fit, new_X)
```

---

predict.MVS *Make predictions from an "MVS" object.*

Description

Make predictions from a "MVS" object.

Usage

```
## S3 method for class 'MVS'
predict(object, newx, predtype = "response", cvlambda = "lambda.min", ...)
```

Arguments

object	An object of class "MVS".
newx	Matrix of new values for x at which predictions are to be made. Must be a matrix.
predtype	The type of prediction returned by the meta-learner. Supported are types "response", "class" and "link".
cvlambda	Values of the penalty parameters at which predictions are to be made. Defaults to the values giving minimum cross-validation error.
...	Further arguments to be passed to `predict.cv.glmnet`.

Value

A matrix of predictions.

Author(s)

Wouter van Loon <w.s.van.loon@fsw.leidenuniv.nl>

Examples

```
set.seed(012)
n <- 1000
X <- matrix(rnorm(8500), nrow=n, ncol=85)
top_level <- c(rep(1,45), rep(2,20), rep(3,20))
bottom_level <- c(rep(1:3, each=15), rep(4:5, each=10), rep(6:9, each=5))
views <- cbind(bottom_level, top_level)
beta <- c(rep(10, 55), rep(0, 30)) * ((rbinom(85, 1, 0.5)*2)-1)
eta <- X %*% beta
p <- 1 /(1 + exp(-eta))
y <- rbinom(n, 1, p)

fit <- MVS(x=X, y=y, views=views, type="StaPLR", levels=3, alphas=c(0,1,1), nnc=c(0,1,1))
coefficients <- coef(fit)

new_X <- matrix(rnorm(2*85), nrow=2)
predict(fit, new_X)
```

`predict.StaPLR` *Make predictions from a "StaPLR" object.*

Description

Make predictions from a "StaPLR" object.

Usage

```
## S3 method for class 'StaPLR'
predict(
  object,
  newx,
  newcf = NULL,
  predtype = "response",
  cvlambda = "lambda.min",
  ...
)
```

Arguments

`object`	Fitted "StaPLR" model object.
`newx`	Matrix of new values for x at which predictions are to be made. Must be a matrix.
`newcf`	Matrix of new values of correction features, if correct.for was specified during model fitting.
`predtype`	The type of prediction returned by the meta-learner.

`cvlambda` Values of the penalty parameters at which predictions are to be made. Defaults to the values giving minimum cross-validation error.

`...` Further arguments to be passed to `predict.cv.glmnet`.

Value

A matrix of predictions.

Author(s)

Wouter van Loon <w.s.van.loon@fsw.leidenuniv.nl>

Examples

```
set.seed(012)
n <- 1000
cors <- seq(0.1,0.7,0.1)
X <- matrix(NA, nrow=n, ncol=length(cors)+1)
X[,1] <- rnorm(n)

for(i in 1:length(cors)){
  X[,i+1] <- X[,1]*cors[i] + rnorm(n, 0, sqrt(1-cors[i]^2))
}

beta <- c(1,0,0,0,0,0,0,0)
eta <- X %*% beta
p <- exp(eta)/(1+exp(eta))
y <- rbinom(n, 1, p)
view_index <- rep(1:(ncol(X)/2), each=2)

fit <- StaPLR(X, y, view_index)
coef(fit)$meta

new_X <- matrix(rnorm(16), nrow=2)
predict(fit, new_X)
```

`predict.StaPLRcoef` *Make predictions from a "StaPLRcoef" object.*

Description

Predict using a "StaPLRcoef" object. A "StaPLRcoef" object can be considerably smaller than a full "StaPLR" object for large data sets.

Usage

```
## S3 method for class 'StaPLRcoef'
predict(object, newx, view, newcf = NULL, predtype = "response", ...)
```

Arguments

`object`	Extracted StaPLR coefficients as a "StaPLRcoef" object.
`newx`	Matrix of new values for x at which predictions are to be made. Must be a matrix.
`view`	a vector of length nvars, where each entry is an integer describing to which view each feature corresponds.
`newcf`	Matrix of new values of correction features, if correct.for was specified during model fitting.
`predtype`	The type of prediction returned by the meta-learner. Allowed values are "response", "link", and "class".
`...`	Not currently used.

Value

A matrix of predictions.

Author(s)

Wouter van Loon <w.s.van.loon@fsw.leidenuniv.nl>

Examples

```
set.seed(012)
n <- 1000
cors <- seq(0.1,0.7,0.1)
X <- matrix(NA, nrow=n, ncol=length(cors)+1)
X[,1] <- rnorm(n)

for(i in 1:length(cors)){
  X[,i+1] <- X[,1]*cors[i] + rnorm(n, 0, sqrt(1-cors[i]^2))
}

beta <- c(1,0,0,0,0,0,0,0)
eta <- X %*% beta
p <- exp(eta)/(1+exp(eta))
y <- rbinom(n, 1, p)
view_index <- rep(1:(ncol(X)/2), each=2)

fit <- StaPLR(X, y, view_index)
coefficients <- coef(fit)

new_X <- matrix(rnorm(16), nrow=2)
predict(coefficients, new_X, view_index)
```

StaPLR *Stacked Penalized Logistic Regression*

Description

Fit a two-level stacked penalized (logistic) regression model with a single base-learner and a single meta-learner. Stacked penalized regression models with a Gaussian or Poisson outcome can be fitted using the family argument.

Usage

```
StaPLR(
  x,
  y,
  view,
  view.names = NULL,
  family = "binomial",
  correct.for = NULL,
  alpha1 = 0,
  alpha2 = 1,
  nfolds = 10,
  seed = NULL,
  std.base = FALSE,
  std.meta = FALSE,
  ll1 = -Inf,
  ul1 = Inf,
  ll2 = 0,
  ul2 = Inf,
  cvloss = "deviance",
  metadat = "response",
  cvlambda = "lambda.min",
  cvparallel = FALSE,
  lambda.ratio = 1e-04,
  fdev = 0,
  penalty.weights = NULL,
  parallel = FALSE,
  skip.version = TRUE,
  skip.meta = FALSE,
  skip.cv = FALSE,
  progress = TRUE
)

staplr(
  x,
  y,
  view,
  view.names = NULL,
```

```
  family = "binomial",
  correct.for = NULL,
  alpha1 = 0,
  alpha2 = 1,
  nfolds = 10,
  seed = NULL,
  std.base = FALSE,
  std.meta = FALSE,
  ll1 = -Inf,
  ul1 = Inf,
  ll2 = 0,
  ul2 = Inf,
  cvloss = "deviance",
  metadat = "response",
  cvlambda = "lambda.min",
  cvparallel = FALSE,
  lambda.ratio = 1e-04,
  fdev = 0,
  penalty.weights = NULL,
  parallel = FALSE,
  skip.version = TRUE,
  skip.meta = FALSE,
  skip.cv = FALSE,
  progress = TRUE
)
```

Arguments

`x`	input matrix of dimension nobs x nvars
`y`	outcome vector of length nobs
`view`	a vector of length nvars, where each entry is an integer describing to which view each feature corresponds.
`view.names`	(optional) a character vector of length nviews specifying a name for each view.
`family`	Either a character string representing one of the built-in families, or else a `glm()` family object. For more information, see `family` argument's documentation in `glmnet`. Note that "multinomial", "mgaussian", "cox", or 2-column responses with "binomial" family are not yet supported.
`correct.for`	(optional) a matrix with nrow = nobs, where each column is a feature which should be included directly into the meta.learner. By default these features are not penalized (see penalty.weights) and appear at the top of the coefficient list.
`alpha1`	(base) alpha parameter for glmnet: lasso(1) / ridge(0)
`alpha2`	(meta) alpha parameter for glmnet: lasso(1) / ridge(0)
`nfolds`	number of folds to use for all cross-validation.
`seed`	(optional) numeric value specifying the seed. Setting the seed this way ensures the results are reproducible even when the computations are performed in parallel.

`std.base`	should features be standardized at the base level?
`std.meta`	should cross-validated predictions be standardized at the meta level?
`ll1`	lower limit(s) for each coefficient at the base-level. Defaults to -Inf.
`ul1`	upper limit(s) for each coefficient at the base-level. Defaults to Inf.
`ll2`	lower limit(s) for each coefficient at the meta-level. Defaults to 0 (non-negativity constraints). Does not apply to correct.for features.
`ul2`	upper limit(s) for each coefficient at the meta-level. Defaults to Inf. Does not apply to correct.for features.
`cvloss`	loss to use for cross-validation.
`metadat`	which attribute of the base learners should be used as input for the meta learner? Allowed values are "response", "link", and "class".
`cvlambda`	value of lambda at which cross-validated predictions are made. Defaults to the value giving minimum internal cross-validation error.
`cvparallel`	whether to use 'foreach' to fit each CV fold (DO NOT USE, USE OPTION parallel INSTEAD).
`lambda.ratio`	the ratio between the largest and smallest lambda value.
`fdev`	sets the minimum fractional change in deviance for stopping the path to the specified value, ignoring the value of fdev set through glmnet.control. Setting fdev=NULL will use the value set through glmnet.control instead. It is strongly recommended to use the default value of zero.
`penalty.weights`	(optional) a vector of length nviews, containing different penalty factors for the meta-learner. Defaults to rep(1,nviews). The penalty factor is set to 0 for correct.for features.
`parallel`	whether to use foreach to fit the base-learners and obtain the cross-validated predictions in parallel. Executes sequentially unless a parallel backend is registered beforehand.
`skip.version`	whether to skip checking the version of the glmnet package.
`skip.meta`	whether to skip training the metalearner.
`skip.cv`	whether to skip generating the cross-validated predictions.
`progress`	whether to show a progress bar (only supported when parallel = FALSE).

Value

An object with S3 class "StaPLR".

Author(s)

Wouter van Loon <w.s.van.loon@fsw.leidenuniv.nl>

Examples

```
set.seed(012)
n <- 1000
cors <- seq(0.1,0.7,0.1)
X <- matrix(NA, nrow=n, ncol=length(cors)+1)
X[,1] <- rnorm(n)

for(i in 1:length(cors)){
  X[,i+1] <- X[,1]*cors[i] + rnorm(n, 0, sqrt(1-cors[i]^2))
}

beta <- c(1,0,0,0,0,0,0,0)
eta <- X %*% beta
p <- exp(eta)/(1+exp(eta))
y <- rbinom(n, 1, p) ## create binary response
view_index <- rep(1:(ncol(X)/2), each=2)

# Stacked penalized logistic regression
fit <- StaPLR(X, y, view_index)
coef(fit)$meta

new_X <- matrix(rnorm(16), nrow=2)
predict(fit, new_X)

# Stacked penalized linear regression
y <- eta + rnorm(100) ## create continuous response
fit <- StaPLR(X, y, view_index, family = "gaussian")
coef(fit)$meta
coef(fit)$base
new_X <- matrix(rnorm(16), nrow=2)
predict(fit, new_X)

# Stacked penalized Poisson regression
y <- ceiling(eta + 4) ## create count response
fit <- StaPLR(X, y, view_index, family = "poisson")
coef(fit)$meta
coef(fit)$base
new_X <- matrix(rnorm(16), nrow=2)
predict(fit, new_X)
```

Index