

Package ‘mvs’

May 9, 2026

Type Package

Title Methods for High-Dimensional Multi-View Learning

Version 2.1.0

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](https://doi.org/10.1016/j.inffus.2020.03.007)> and Van Loon et al. (2022) <[doi:10.3389/fnins.2022.830630](https://doi.org/10.3389/fnins.2022.830630)>.

License GPL-2

Encoding UTF-8

RoxygenNote 7.3.2

Depends glmnet (>= 1.9-8), randomForest

Imports foreach (>= 1.4.4)

Suggests testthat (>= 3.0.0), mice (>= 3.16.0), missForest (>= 1.5),
knitr, rmarkdown, bookdown

Config/testthat/edition 3

VignetteBuilder knitr

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 2025-04-15 15:00:02 UTC

Contents

mvs-package	2
coef.MVS	2
coef.StaPLR	3
importance.MVS	4

MRM	5
MVS	6
predict.MVS	9
predict.StaPLR	10
predict.StaPLRcoef	11
RF	13
StaPLR	14

Index 20

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>.

Details

Details

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)
```

importance.MVS

Calculate feature importance from an "MVS" object.

Description

Calculate feature importance at each level from an "MVS" object based on random forests.

Usage

```
## S3 method for class 'MVS'
importance(x, ...)
```

Arguments

x An object of class "MVS".

... Further arguments to be passed to [importance](#).

Value

An object of S3 class "MVSimportance".

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)
views <- c(rep(1,45), rep(2,20), rep(3,20))
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)

## 2-level MVS with random forest
fit <- MVS(x=X, y=y, views=views, type = "RF")
importance(fit)
```

MRM

*Minority Report Measure***Description**

Calculate the Minority Report Measure (MRM) for each view in a (hierarchical) multi-view stacking model.

Usage

```
MRM(fit, constant, level = 2, a = 0, b = 1, cvlambda = "lambda.min")
```

```
mrm(fit, constant, level = 2, a = 0, b = 1, cvlambda = "lambda.min")
```

Arguments

<code>fit</code>	an object of class MVS .
<code>constant</code>	the value at which to keep the predictions of the other views constant. The recommended value is the mean of the outcome variable.
<code>level</code>	the level at which to calculate the MRM. In a 3-level MVS model, <code>level = 2</code> (the default) is generally the level for which one would want to calculate the MRM. Note that calculating the MRM for <code>level = 1</code> (the feature level) is possible, but generally not sensible except under specific conditions.
<code>a</code>	the start of the interval over which to calculate the MRM. Defaults to 0.
<code>b</code>	the end of the interval over which to calculate the MRM. Defaults to 1.
<code>cvlambda</code>	denotes which values of the penalty parameters to use for calculating predictions. This corresponds to the defaults used during model fitting.

Details

The Minority Report Measure (MRM) considers the view-specific sub-models at a given level of the hierarchy as members of a committee making predictions of the outcome variable. For each view, the MRM quantifies how much the final prediction of the stacked model changes if the prediction of the corresponding sub-model changes from a to b, while keeping the predictions corresponding to the other views constant at constant. For more information about the MRM see <doi:10.3389/fnins.2022.830630>.

Value

A numeric vector of a length equal to the number of views at the specified level, containing the values of the MRM for each view.

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)
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)

## 3-level MVS
bottom_level <- c(rep(1:3, each=15), rep(4:5, each=10), rep(6:9, each=5))
top_level <- c(rep(1,45), rep(2,20), rep(3,20))
views <- cbind(bottom_level, top_level)
fit <- MVS(x=X, y=y, views=views, levels=3, alphas=c(0,1,1), nnc=c(0,1,1))
MRM(fit, constant=mean(y))
```

MVS

Multi-View Stacking

Description

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

Usage

```
MVS(
  x,
  y,
  views,
```

```

    type = "StaPLR",
    levels = NULL,
    alphas = c(0, 1),
    nnc = c(0, 1),
    parallel = FALSE,
    seeds = NULL,
    progress = TRUE,
    relax = FALSE,
    adaptive = FALSE,
    na.action = "fail",
    na.arguments = NULL,
    ...
)

mvs(
  x,
  y,
  views,
  type = "StaPLR",
  levels = NULL,
  alphas = c(0, 1),
  nnc = c(0, 1),
  parallel = FALSE,
  seeds = NULL,
  progress = TRUE,
  relax = FALSE,
  adaptive = FALSE,
  na.action = "fail",
  na.arguments = NULL,
  ...
)

```

Arguments

<code>x</code>	input matrix of dimension <code>nobs</code> x <code>nvars</code> .
<code>y</code>	outcome vector of length <code>nobs</code> .
<code>views</code>	a matrix of dimension <code>nvars</code> x (<code>levels</code> - 1), where each entry is an integer describing to which view each feature corresponds.
<code>type</code>	a character vector of length 1 or length <code>levels</code> , specifying the type(s) of learner to be used at each level of MVS. Use type "StaPLR" when the desired learner(s) is/are penalized GLM(s); see StaPLR for supported families. Use type "RF" for random forests.
<code>levels</code>	(optional) an integer ≥ 2 , specifying the number of levels in the MVS procedure. The default is to infer the number of levels from the supplied <code>views</code> argument.
<code>alphas</code>	a numeric vector of length <code>levels</code> 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).
relax	either a logical vector of length levels specifying whether model relaxation (e.g. the relaxed lasso) should be employed at each level, or a single TRUE or FALSE to enable or disable relaxing across all levels. Defaults to FALSE.
adaptive	either a logical vector of length levels specifying whether adaptive weights (e.g. the adaptive lasso) should be employed at each level, or a single TRUE or FALSE to enable or disable adaptive weights across all levels. Note that using adaptive weights is generally only sensible if alpha > 0. Defaults to FALSE.
na.action	character specifying what to do with missing values (NA). Options are "pass", "fail", "mean", "mice", and "missForest". Options "mice" and "missForest" requires the respective R package to be installed. Defaults to "fail".
na.arguments	(optional) a named list of arguments to pass to the imputation function (e.g. to mice or missForest).
...	additional arguments to pass to the learning algorithm. See e.g. StaPLR . Note that these arguments are passed to 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)
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)

## 2-level MVS with ridge for baselearners and lasso for meta learner
views <- c(rep(1,45), rep(2,20), rep(3,20))
fit <- MVS(x=X, y=y, views=views)

## 2-level MVS with random forest for base learners and lasso for meta learner
fit <- MVS(x=X, y=y, views=views, type = c("RF", "StaPLR"))
new_X <- matrix(rnorm(2*85), nrow=2)
```

```

predict(fit, new_X)

## 3-level MVS
bottom_level <- c(rep(1:3, each=15), rep(4:5, each=10), rep(6:9, each=5))
top_level <- c(rep(1,45), rep(2,20), rep(3,20))
views <- cbind(bottom_level, top_level)
fit <- MVS(x=X, y=y, views=views, levels=3, alphas=c(0,1,1), nnc=c(0,1,1))
coefficients <- coef(fit)
predict(fit, new_X)

```

predict.MVS	<i>Make predictions from an "MVS" object.</i>
-------------	---

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

<code>object</code>	Extracted StaPLR coefficients as a "StaPLRcoef" object.
<code>newx</code>	Matrix of new values for x at which predictions are to be made. Must be a matrix.
<code>view</code>	a vector of length <code>nvars</code> , where each entry is an integer describing to which view each feature corresponds.
<code>newcf</code>	Matrix of new values of correction features, if <code>correct.for</code> was specified during model fitting.
<code>predtype</code>	The type of prediction returned by the meta-learner. Allowed values are "response", "link", and "class".
<code>...</code>	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)

```

RF

*Function for fitting random forests with multi-view stacking***Description**

A wrapper function around `randomForest` from package of the same name that allows to use it in function `MVS`.

Usage

```
RF(
  x,
  y,
  view,
  view.names = NULL,
  skip.meta = FALSE,
  skip.cv = FALSE,
  na.action = "fail",
  na.arguments = NULL,
  progress = TRUE,
  ...
)
```

Arguments

<code>x</code>	input matrix of dimension <code>nobs</code> x <code>nvars</code>
<code>y</code>	outcome vector of length <code>nobs</code>
<code>view</code>	a vector of length <code>nvars</code> , where each entry is an integer describing to which view each feature corresponds.
<code>view.names</code>	(optional) a character vector of length <code>nviews</code> specifying a name for each view.
<code>skip.meta</code>	whether to skip training the metalearner.
<code>skip.cv</code>	whether to skip generating the cross-validated predictions.
<code>na.action</code>	character specifying what to do with missing values (NA). Options are "pass", "fail", "mean", "mice", and "missForest". Options "mice" and "missForest" requires the respective R package to be installed. Defaults to "pass".
<code>na.arguments</code>	(optional) a named list of arguments to pass to the imputation function (e.g. to <code>mice</code> or <code>missForest</code>).
<code>progress</code>	whether to show a progress bar (only supported when <code>parallel = FALSE</code>).
<code>...</code>	Additional arguments to be passed to function <code>randomForest</code> .

Value

An object with S3 class "RF".

Author(s)

Marjolein Fokkema <m.fokkema@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 random forest
fit <- RF(X, y, view_index, skip.meta = FALSE, skip.cv = FALSE)

# Stacked random forest
y <- eta + rnorm(100) ## create continuous response
fit <- RF(X, y, view_index, skip.meta = FALSE, skip.cv = FALSE)

```

 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,

```

```
    relax = FALSE,
    nfolds = 10,
    na.action = "fail",
    na.arguments = NULL,
    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.meta = NULL,
    penalty.weights.base = NULL,
    gamma.seq = c(0.5, 1, 2),
    parallel = FALSE,
    skip.version = TRUE,
    skip.meta = FALSE,
    skip.cv = FALSE,
    progress = TRUE,
    relax.base = FALSE,
    relax.meta = FALSE
  )

staplr(
  x,
  y,
  view,
  view.names = NULL,
  family = "binomial",
  correct.for = NULL,
  alpha1 = 0,
  alpha2 = 1,
  relax = FALSE,
  nfolds = 10,
  na.action = "fail",
  na.arguments = NULL,
  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.meta = NULL,
penalty.weights.base = NULL,
gamma.seq = c(0.5, 1, 2),
parallel = FALSE,
skip.version = TRUE,
skip.meta = FALSE,
skip.cv = FALSE,
progress = TRUE,
relax.base = FALSE,
relax.meta = FALSE
)

```

Arguments

<code>x</code>	input matrix of dimension <code>nobs</code> x <code>nvars</code>
<code>y</code>	outcome vector of length <code>nobs</code>
<code>view</code>	a vector of length <code>nvars</code> , where each entry is an integer describing to which view each feature corresponds.
<code>view.names</code>	(optional) a character vector of length <code>nviews</code> specifying a name for each view.
<code>family</code>	Either a character string representing one of the built-in families, or else a <code>glm()</code> family object. For more information, see <code>family</code> argument's documentation in glmnet . Note that "multinomial", "mgaussian", "cox", or 2-column responses with "binomial" family are not yet supported.
<code>correct.for</code>	(optional) a matrix with <code>nrow</code> = <code>nobs</code> , where each column is a feature which should be included directly into the <code>meta.learner</code> . By default these features are not penalized (see <code>penalty.weights.meta</code>) and appear at the top of the coefficient list.
<code>alpha1</code>	(base) alpha parameter for <code>glmnet</code> : <code>lasso(1) / ridge(0)</code>
<code>alpha2</code>	(meta) alpha parameter for <code>glmnet</code> : <code>lasso(1) / ridge(0)</code>
<code>relax</code>	logical, whether relaxed lasso should be used at base and meta level.
<code>nfolds</code>	number of folds to use for all cross-validation.
<code>na.action</code>	character specifying what to do with missing values (NA). Options are "pass", "fail", "mean", "mice", and "missForest". Options "mice" and "missForest" requires the respective R package to be installed. Defaults to "pass".
<code>na.arguments</code>	(optional) a named list of arguments to pass to the imputation function (e.g. to <code>mice</code> or <code>missForest</code>).
<code>seed</code>	(optional) numeric value specifying the seed. Setting the seed this way ensures the results are reproducible even when the computations are performed in parallel.

<code>std.base</code>	should features be standardized at the base level?
<code>std.meta</code>	should cross-validated predictions be standardized at the meta level?
<code>l11</code>	lower limit(s) for each coefficient at the base-level. Defaults to <code>-Inf</code> .
<code>u11</code>	upper limit(s) for each coefficient at the base-level. Defaults to <code>Inf</code> .
<code>l12</code>	lower limit(s) for each coefficient at the meta-level. Defaults to 0 (non-negativity constraints). Does not apply to <code>correct.for</code> features.
<code>u12</code>	upper limit(s) for each coefficient at the meta-level. Defaults to <code>Inf</code> . Does not apply to <code>correct.for</code> features.
<code>cvloss</code>	loss to use for cross-validation.
<code>metadat</code>	which attribute of the base learners should be used as input for the meta learner? Allowed values are "response", "link", and "class".
<code>cvlambda</code>	value of lambda at which cross-validated predictions are made. Defaults to the value giving minimum internal cross-validation error.
<code>cvparallel</code>	whether to use 'foreach' to fit each CV fold (DO NOT USE, USE OPTION <code>parallel</code> INSTEAD).
<code>lambda.ratio</code>	the ratio between the largest and smallest lambda value.
<code>fdev</code>	sets the minimum fractional change in deviance for stopping the path to the specified value, ignoring the value of <code>fdev</code> set through <code>glmnet.control</code> . Setting <code>fdev=NULL</code> will use the value set through <code>glmnet.control</code> instead. It is strongly recommended to use the default value of zero.
<code>penalty.weights.meta</code>	(optional) either a vector of length <code>nviews</code> containing different penalty factors for the meta-learner, or "adaptive" to calculate the weights from the data. The default value <code>NULL</code> implies an equal penalty for each view. The penalty factor is set to 0 for <code>correct.for</code> features.
<code>penalty.weights.base</code>	(optional) either a list of length <code>nviews</code> , where each entry is a vector containing different penalty factors for each feature in that view, or "adaptive" to calculate the weights from the data. The default value <code>NULL</code> implies an equal penalty for each view. Note that using adaptive weights at the base level is generally only sensible if <code>alpha1 > 0</code> .
<code>gamma.seq</code>	a sequence of gamma values over which to optimize the adaptive weights. Only used when <code>penalty.weights.meta="adaptive"</code> or <code>penalty.weights.base="adaptive"</code> .
<code>parallel</code>	whether to use <code>foreach</code> to fit the base-learners and obtain the cross-validated predictions in parallel. Executes sequentially unless a parallel backend is registered beforehand.
<code>skip.version</code>	whether to skip checking the version of the <code>glmnet</code> package.
<code>skip.meta</code>	whether to skip training the metalearner.
<code>skip.cv</code>	whether to skip generating the cross-validated predictions.
<code>progress</code>	whether to show a progress bar (only supported when <code>parallel = FALSE</code>).
<code>relax.base</code>	logical indicating whether relaxed lasso should be employed for fitting the base learners. If <code>TRUE</code> , then CV is done with respect to the mixing parameter gamma as well as lambda.

`relax.meta` logical indicating whether relaxed lasso should be employed for fitting the meta learner. If TRUE, then CV is done with respect to the mixing parameter γ as well as λ .

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

* TBA

- coef.MVS, [2](#)
- coef.StaPLR, [3](#)
- importance.MVS, [4](#)
- MRM, [5](#)
- MVS, [6](#)
- predict.MVS, [9](#)
- predict.StaPLR, [10](#)
- predict.StaPLRcoef, [11](#)
- RF, [13](#)
- StaPLR, [14](#)

- coef.cv.glmnet, [2](#), [3](#)
- coef.MVS, [2](#)
- coef.StaPLR, [3](#)

- glmnet, [16](#)

- importance, [4](#)
- importance.MVS, [4](#)

- MRM, [5](#)
- mrmm (MRM), [5](#)
- MVS, [5](#), [6](#)
- mvs (MVS), [6](#)
- mvs-package, [2](#)

- predict.cv.glmnet, [9](#), [10](#)
- predict.MVS, [9](#)
- predict.StaPLR, [10](#)
- predict.StaPLRcoef, [11](#)

- randomForest, [13](#)
- RF, [13](#)

- StaPLR, [7](#), [8](#), [14](#)
- staplR (StaPLR), [14](#)