# Package ‘mvs’

April 15, 2025

**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) meth-
ods see Van Loon, Fokkema, Sz-
abo, & 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** 7.3.2

**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 *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

| | |
|---|---|
| `fit` | an object of class `MVS`. |
| `constant` | the value at which to keep the predictions of the other views constant. The recommended value is the mean of the outcome variable. |
| `level` | the level at which to calculate the MRM. In a 3-level MVS model, `level` = 2 (the default) is generally the level for which one would want to calculate the MRM. Note that calculating the MRM for `level` = 1 (the feature level) is possible, but generally not sensible except under specific conditions. |
| `a` | the start of the interval over which to calculate the MRM. Defaults to 0. |
| `b` | the end of the interval over which to calculate the MRM. Defaults to 1. |
| `cvlambda` | 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

| | |
|---|---|
| `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` | a character vector of length 1 or length `levels`, 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. |
| `levels` | (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 `views` argument. |
| `alphas` | a numeric vector of length `levels` 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 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` *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)
```

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

| | |
|---|---|
| `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. |
| `skip.meta` | whether to skip training the metalearner. |
| `skip.cv` | whether to skip generating the cross-validated predictions. |
| `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 "pass". |
| `na.arguments` | (optional) a named list of arguments to pass to the imputation function (e.g. to `mice` or `missForest`). |
| `progress` | whether to show a progress bar (only supported when parallel = FALSE). |
| `...` | Additional arguments to be passed to function `randomForest`. |

## 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

| | |
|---|---|
| `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.meta) 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) |
| `relax` | logical, whether relaxed lasso should be used at base and meta level. |
| `nfolds` | number of folds to use for all cross-validation. |
| `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 "pass". |
| `na.arguments` | (optional) a named list of arguments to pass to the imputation function (e.g. to `mice` or `missForest`). |
| `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.meta` | (optional) either a vector of length nviews containing different penalty factors for the meta-learner, or "adaptive" to calculate the weights from the data. The default value NULL implies an equal penalty for each view. The penalty factor is set to 0 for `correct.for` features. |
| `penalty.weights.base` | (optional) either a list of length nviews, 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 NULL implies an equal penalty for each view. Note that using adaptive weights at the base level is generally only sensible if `alpha1` > 0. |
| `gamma.seq` | a sequence of gamma values over which to optimize the adaptive weights. Only used when `penalty.weights.meta="adaptive"` or `penalty.weights.base="adaptive"`. |
| `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). |
| `relax.base` | logical indicating whether relaxed lasso should be employed for fitting the base learners. If `TRUE`, 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 gamma as well as lambda. |

## 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