

Package ‘stacking’

July 23, 2025

Type Package

Title Building Predictive Models with Stacking

Version 0.2.1

Description Building predictive models with stacking which is a type of ensemble learning. Learners can be specified from those implemented in 'caret'. For more information of the package, see Nukui and Onogi (2023) <[doi:10.1101/2023.06.06.543970](https://doi.org/10.1101/2023.06.06.543970)>.

License MIT + file LICENSE

NeedsCompilation no

Author Taichi Nukui [aut, cph],
Tomohiro Ishibashi [aut, cph],
Akio Onogi [aut, cre, cph]

Maintainer Akio Onogi <onogiakio@gmail.com>

Depends caret, parallel

Repository CRAN

Date/Publication 2025-01-16 08:10:02 UTC

Contents

stacking_predict	2
stacking_train	4
train_basemodel	9
train_basemodel_core	13
train_metamodel	14
Index	19

stacking_predict	<i>Predict for new data</i>
------------------	-----------------------------

Description

Return predicted values for newX based on training results of stacking.

Usage

```
stacking_predict(newX, stacking_train_result)
```

Arguments

newX	An N x P matrix of explanatory variables of new data where N is the number of samples and P is the number of explanatory variables. Note that the order of explanatory variables should be the same as those for training. Column names of newX are ignored.
stacking_train_result	A list output by stacking_train. When train_basemodel and train_metamodel are directly used, a list combining each output should be created and given as stacking_train_result. See examples for this operation.

Details

Prediction processes of this package are as follows. First, newX is given to all base models. As a result, each base learner returns Nfold/num_sample predicted values where Nfold/num_sample is an argument of stacking_train. Then the predicted values are averaged for each learner. Giving these averaged values as the explanatory variables of the meta model, final predicted values are output.

Value

result	Vector of predicted values. When TrainEachFold of stacking_train or train_metamodel is TRUE (i.e., stacking_train_result\$meta\$TrainEachFold is TRUE), the values are the averages of the values predicted from the meta models trained for each cross-validation fold, and for random sampling, the values are the averages of the values predicted from the meta models trained for each random sampling iteration. In the case of classification, the probabilities of each category are returned.
--------	--

Author(s)

Taichi Nukui, Tomohiro Ishibashi, Akio Onogi

Examples

```

#Create a toy example
##Number of training samples
N1 <- 100

##Number of explanatory variables
P <- 200

##Create X of training data
X1 <- matrix(rnorm(N1 * P), nrow = N1, ncol = P)
colnames(X1) <- 1:P#column names are required by caret

##Assume that the first 10 variables have effects on Y
##Then add noise with rnorm
Y1 <- rowSums(X1[, 1:10]) + rnorm(N1)

##Test data
N2 <- 100
X2 <- matrix(rnorm(N2 * P), nrow = N2, ncol = P)
colnames(X2) <- 1:P#Ignored (not required)
Y2 <- rowSums(X2[, 1:10])

#Specify base learners
Method <- list(glmnet = data.frame(alpha = c(0.5, 0.8), lambda = c(0.1, 1)),
               pls = data.frame(ncomp = 5))
#=>This specifies 5 base learners.
##1. glmnet with alpha = 0.5 and lambda = 0.1
##2. glmnet with alpha = 0.5 and lambda = 1
##3. glmnet with alpha = 0.8 and lambda = 0.1
##4. glmnet with alpha = 0.8 and lambda = 1
##5. pls with ncomp = 5

#The followings are the training and prediction processes
#If glmnet and pls are not installed, please install them in advance.
#Please remove #s before execution

#Training
#stacking_train_result <- stacking_train(X = X1,
#                                         Y = Y1,
#                                         Method = Method,
#                                         Metamodel = "lm",
#                                         core = 2,
#                                         cross_validation = TRUE,
#                                         use_X = FALSE,
#                                         TrainEachFold = TRUE,
#                                         Nfold = 5)

#For random sampling, set cross_validation = FALSE and
#specify the number of samples and the sampling proportion
#using num_sample and proportion, respectively.
#To include the original features X when training the meta-model, set use_X = TRUE.
#When use_X is TRUE, simple linear regressions cannot be used

```

```

#as the meta learner because of rank deficient.
#The following code reflects the changes made to the relevant arguments.
#stacking_train_result <- stacking_train(X = X1,
#                                     Y = Y1,
#                                     Method = Method,
#                                     Metamodel = "glmnet",
#                                     core = 2,
#                                     cross_validation = FALSE,
#                                     use_X = TRUE,
#                                     TrainEachFold = TRUE,
#                                     num_sample = 5,
#                                     proportion = 0.8)

#Prediction
#result <- stacking_predict(newX = X2, stacking_train_result)
#plot(Y2, result)

#Training using train_basemodel and train_metamodel
#base <- train_basemodel(X = X1,
#                        Y = Y1,
#                        Method = Method,
#                        core = 2,
#                        cross_validation = TRUE,
#                        Nfold = 5)
#meta <- train_metamodel(X,
#                        base,
#                        which_to_use = 1:5,
#                        Metamodel = "lm",
#                        use_X = FALSE,
#                        TrainEachFold = TRUE)
#stacking_train_result <- list(base = base, meta = meta)
#=>The list should have elements named as base and meta to be used in stacking_predict

#Prediction
#result <- stacking_predict(newX = X2, stacking_train_result)
#plot(Y2, result)

```

stacking_train

Training base and meta models

Description

Training base and meta learners of stacking (an ensemble learning approach). The base and meta learners can be chosen from supervised methods implemented in caret. This function internally calls `train_basemodel` and `train_metamodel`. Packages `caret`, `parallel`, `snow`, and packages for base and meta learners should be installed.

Usage

```
stacking_train(X, Y, Method, Metamodel, core = 1, cross_validation = TRUE,
               use_X = FALSE, TrainEachFold = FALSE, Nfold = 10,
               num_sample = 10, proportion = 0.8)
```

Arguments

X	An N x P matrix of explanatory variables where N is the number of samples and P is the number of variables. Column names are required by caret.
Y	A length N Vector of objective variables. Use a factor for classification.
Method	A list specifying base learners. Each element of the list is a data.frame that contains hyperparameter values of base learners. The names of the list elements specifies the base learners and are passed to caret functions. See details and examples
Metamodel	A strings specifying the meta learner. This strings is passed to caret.
core	Number of cores for parallel processing
cross_validation	A parameter to specify whether to perform cross-validation. Set to TRUE to enable cross-validation or to FALSE to perform random sampling.
use_X	A logical indicating whether the meta-learner uses the original features X along with the base model predictions. If TRUE, it uses both X and the predictions; if FALSE, it uses only the predictions.
TrainEachFold	A logical indicating whether the meta learner learns using the predicted values of the base models at each cross-validation fold/random sample or not. If TRUE, the meta learners learns Nfold/num_sample times using the values predicted by the base models at each fold/sample. If FALSE, the meta learner learns once by pooling the predicted values of the base models of all folds/samples.
Nfold	Number of folds for cross-validation. Required when cross_validation is TRUE.
num_sample	The number of samples of random sampling, applicable when cross_validation is set to FALSE.
proportion	A parameter specifying the proportion of samples to be sampled when cross_validation is set to FALSE.

Details

Stacking by this package consists of the following 2 steps.

(1) Each base learner is trained. The training method can be chosen using the cross_validation argument: If cross_validation is TRUE: The function performs Nfold cross-validation for each base learner. If cross_validation is FALSE: The function trains each base learner using random sampling. The number of samples (num_sample) or the proportion of the data (proportion) can be specified to control the sampling process. (2) Using the predicted values of each learner as the explanatory variables, the meta learner is trained. Steps (1) and (2) are conducted by train_basemodel and train_metamodel, respectively. Another function stacking_train conducts both steps at once by calling these functions (train_basemodel and train_metamodel).

Training of the meta learner can be modified by two arguments, `TrainEachFold` and `use_X`. `TrainEachFold` specifies whether the meta learner is trained for each fold or random sample individually, or once by pooling or combining all predicted values. `use_X` specifies whether the meta learner is trained with both the original features `X` and the base model predictions, or with only the base model predictions.

Base learners are specified by `Method`. For example,
`Method = list(glmnet = data.frame(alpha = 0, lambda = 5), pls = data.frame(ncomp = 10))`
 indicating that the first base learner is `glmnet` and the second is `pls` with the corresponding hyperparameters.

When the data.frames have multiple rows as
`Method = list(glmnet = data.frame(alpha = c(0, 1), lambda = c(5, 10)))`
 All combinations of hyperparameter values are automatically created as
`[alpha, lambda] = [0, 5], [0, 10], [1, 5], [1, 10]`
 Thus, in total 5 base learners (4 `glmnet` and 1 `pls`) are created.

When the number of candidate values differ among hyperparameters, use `NA` as
`Method = list(glmnet = data.frame(alpha = c(0, 0.5, 1), lambda = c(5, 10, NA)))`
 resulting in 6 combinations of
`[alpha, lambda] = [0, 5], [0, 10], [0.5, 5], [0.5, 10], [1, 5], [1, 10]`

When a hyperparameter includes only `NA` as
`Method = list(glmnet = data.frame(alpha = c(0, 0.5, 1), lambda = c(NA, NA, NA)), pls = data.frame(ncomp = NA))`
`lambda` of `glmnet` and `ncomp` of `pls` are automatically tuned by `caret`. However, it is notable that tuning is conducted assuming that all hyperparameters are unknown, and thus, the tuned `lambda` in the above example is not the value tuned under the given `alpha` values (0, 0.5, or 1).

Hyperparameters of meta learners are automatically tuned by `caret`.

The base and meta learners can be chosen from the methods implemented in `caret`. The choosable methods can be seen at <https://topepo.github.io/caret/available-models.html> or using `names(getModelInfo())` after loading `caret`.

Value

A list containing the following elements is output.

<code>base</code>	A list output by <code>train_basemodel</code> . See value of <code>train_basemodel</code> for the details
<code>meta</code>	A list output by <code>train_metamodel</code> . See value of <code>train_metamodel</code> for the details

Author(s)

Taichi Nukui, Tomohiro Ishibashi, Akio Onogi

See Also

`train_basemodel`, `train_metamodel`

Examples

```

#Create a toy example
##Number of training samples
N1 <- 100

##Number of explanatory variables
P <- 200

##Create X of training data
X1 <- matrix(rnorm(N1 * P), nrow = N1, ncol = P)
colnames(X1) <- 1:P#column names are required by caret

##Assume that the first 10 variables have effects on Y
##Then add noise with rnorm
Y1 <- rowSums(X1[, 1:10]) + rnorm(N1)

##Test data
N2 <- 100
X2 <- matrix(rnorm(N2 * P), nrow = N2, ncol = P)
colnames(X2) <- 1:P#Ignored (not required)
Y2 <- rowSums(X2[, 1:10])

#Specify base learners
Method <- list(glmnet = data.frame(alpha = c(0.5, 0.8), lambda = c(0.1, 1)),
               pls = data.frame(ncomp = 5))
#=>This specifies five base learners.
##1. glmnet with alpha = 0.5 and lambda = 0.1
##2. glmnet with alpha = 0.5 and lambda = 1
##3. glmnet with alpha = 0.8 and lambda = 0.1
##4. glmnet with alpha = 0.8 and lambda = 1
##5. pls with ncomp = 5

#The followings are the training and prediction processes
#If glmnet and pls are not installed, please install them in advance.
#Please remove #s before execution

#stacking_train_result <- stacking_train(X = X1,
#                                       Y = Y1,
#                                       Method = Method,
#                                       Metamodel = "lm",
#                                       core = 2,
#                                       cross_validation = TRUE,
#                                       use_X = FALSE,
#                                       TrainEachFold = TRUE,
#                                       Nfold = 5)

#For random sampling, set cross_validation = FALSE and
#specify the number of samples and the sampling proportion
#using num_sample and proportion, respectively.
#To include the original features X when training the meta-model, set use_X = TRUE.
#When use_X is TRUE, simple linear regressions cannot be used
#as the meta learner because of rank deficient.

```

```

#The following code reflects the changes made to the relevant arguments.
#stacking_train_result <- stacking_train(X = X1,
#                                     Y = Y1,
#                                     Method = Method,
#                                     Metamodel = "glmnet",
#                                     core = 2,
#                                     cross_validation = FALSE,
#                                     use_X = TRUE,
#                                     TrainEachFold = TRUE,
#                                     num_sample = 5,
#                                     proportion = 0.8)

#Prediction
#result <- stacking_predict(newX = X2, stacking_train_result)
#plot(Y2, result)

#Training using train_basemodel and train_metamodel
#base <- train_basemodel(X = X1,
#                        Y = Y1,
#                        Method = Method,
#                        core = 2,
#                        cross_validation = TRUE,
#                        Nfold = 5)
#meta <- train_metamodel(X,
#                        base,
#                        which_to_use = 1:5,
#                        Metamodel = "lm",
#                        use_X = FALSE,
#                        TrainEachFold = TRUE)
#stacking_train_result <- list(base = base, meta = meta)
#=>The list should have elements named as base and meta to be used in stacking_predict

#Prediction
#result <- stacking_predict(newX = X2, stacking_train_result)
#plot(Y2, result)

#In the simulations of the reference paper (Nukui and Onogi 2023),
#we use 48 base learners as
Method <- list(ranger = data.frame(mtry = c(10, 100, 200),
                                   splitrule = c("extratrees", NA, NA),
                                   min.node.size = c(1, 5, 10)),
              xgbTree = data.frame(colsample_bytree = c(0.6, 0.8),
                                   subsample = c(0.5, 1),
                                   nrounds = c(50, 150),
                                   max_depth = c(6, NA),
                                   eta = c(0.3, NA),
                                   gamma = c(0, NA),
                                   min_child_weight = c(1, NA)),
              gbm = data.frame(interaction.depth = c(1, 3, 5),
                               n.trees = c(50, 100, 150),
                               shrinkage = c(0.1, NA, NA),
                               n.minobsinnode = c(10, NA, NA)),
              svmPoly = data.frame(C = c(0.25, 0.5, 1),

```



```

        scale = c(0.001, 0.01, 0.1),
        degree = c(1, NA, NA)),
glmnet = data.frame(alpha = c(1, 0.8, 0.6, 0.4, 0.2, 0),
                    lambda = rep(NA, 6)),
pls = data.frame(ncomp = seq(2, 70, 10))
)
#mtry of ranger and ncomp of pls should be arranged according to data size.

#In the classification example of the reference paper, for RNA features, we used
Method <- list(ranger = data.frame(mtry = c(10, 100, 500),
                                splitrule = c("extratrees", NA, NA),
                                min.node.size = c(1, 5, 10)),
              xgbTree = data.frame(colsample_bytree = c(0.6, 0.8),
                                subsample = c(0.5, 1),
                                nrounds = c(50, 150),
                                max_depth = c(6, NA),
                                eta = c(0.3, NA),
                                gamma = c(0, NA),
                                min_child_weight = c(1, NA)),
              gbm = data.frame(interaction.depth = c(1, 3, 5),
                                n.trees = c(50, 100, 150),
                                shrinkage = c(0.1, NA, NA),
                                n.minobsinnode = c(10, NA, NA)),
              svmPoly = data.frame(C = c(0.25, 0.5, 1),
                                scale = c(0.001, 0.01, 0.1),
                                degree = c(1, NA, NA)),
              glmnet = data.frame(alpha = c(1, 0.8, 0.6, 0.4, 0.2, 0),
                                lambda = rep(NA, 6)),
              pls = data.frame(ncomp = seq(2, 70, 10))
)
#svmRadial was replaced by svmPoly
#These base learners may be a good starting point.

```

train_basemodel

*Training base models***Description**

Training base models of stacking. This function internally calls train_basemodel_core.

Usage

```
train_basemodel(X, Y, Method, core = 1, cross_validation = TRUE,
               Nfold = 10, num_sample = 10, proportion = 0.8)
```

Arguments

X An N x P matrix of explanatory variables where N is the number of samples and P is the number of variables. Column names are required by caret.

Y	A length N Vector of objective variables. Use a factor for classification.
Method	A list specifying base learners. Each element of the list is a data.frame that contains hyperparameter values of base learners. The names of the list elements specifies the base learners and are passed to caret functions. See details and examples
core	Number of cores for parallel processing
cross_validation	A parameter to specify whether to perform cross-validation. Set to TRUE to enable cross-validation or to FALSE to perform random sampling.
Nfold	Number of folds for cross-validation. Required when cross_validation is TRUE.
num_sample	The number of samples for random sampling, applicable when cross_validation is set to FALSE.
proportion	A parameter specifying the proportion of samples to be sampled when cross_validation is set to FALSE.

Details

Stacking by this package consists of the following 2 steps. (1) Each base learner is trained. The training method can be chosen based on the cross_validation argument. If cross_validation is TRUE: The function performs Nfold cross-validation for each base learner. If cross_validation is FALSE: The function trains each base learner using random sampling. The number of samples (num_sample) or the proportion of the data (proportion) can be specified to control the sampling process. (2) Using the predicted values of each learner as the explanatory variables, the meta learner is trained. Steps (1) and (2) are conducted by train_basemodel and train_metamodel, respectively. Another function stacking_train conducts both steps at once by calling these functions (train_basemodel and train_metamodel).

Base learners are specified by Method. For example,
`Method = list(glmnet = data.frame(alpha = 0, lambda = 5), pls = data.frame(ncomp = 10))`
 indicating that the first base learner is glmnet and the second is pls with corresponding hyperparameters.

When the data.frames have multiple rows as
`Method = list(glmnet = data.frame(alpha = c(0, 1), lambda = c(5, 10)))`
 All combinations of hyperparameter values are automatically created as
`[alpha, lambda] = [0, 5], [0, 10], [1, 5], [1, 10]`
 Thus, in total 5 base learners (4 glmnet and 1 pls) are created.

When the number of candidate values differ among hyperparameters, use NA as
`Method = list(glmnet = data.frame(alpha = c(0, 0.5, 1), lambda = c(5, 10, NA)))`
 resulting in 6 combinations of
`[alpha, lambda] = [0, 5], [0, 10], [0.5, 5], [0.5, 10], [1, 5], [1, 10]`

When a hyperparameter includes only NA as
`Method = list(glmnet = data.frame(alpha = c(0, 0.5, 1), lambda = c(NA, NA, NA)), pls = data.frame(ncomp = NA))`
 lambda of glmnet and ncomp of pls are automatically tuned by caret. However, it is notable that tuning is conducted assuming that all hyperparameters are unknown, and thus, the tuned lambda in the above example is not the value tuned under the given alpha values (0, 0.5, or 1).

Hyperparameters of meta learners are automatically tuned by caret.

The base and meta learners can be chosen from the methods implemented in caret. The choosable methods can be seen at <https://topepo.github.io/caret/available-models.html> or using `names(getModelInfo())` after loading caret.

Value

A list containing the following elements is output.

<code>train_result</code>	A list containing the training results of the base models. The length of this list is the same as <code>Nfold/num_sample</code> , and each element is a list of which length is the same as the number of base models. These elements are the lists output by <code>train</code> function of caret, but the element "trainingData" is removed to save memory.
<code>no_base</code>	Number of base models.
<code>valpr</code>	Predicted values of base models obtained in cross-validation/random sampling. Used as explanatory variables for the meta learner.
<code>Y.randomised</code>	Y of the test sets of cross-validation/random sampling. Used as the response variable for the meta learner.
<code>Order</code>	Indices of Y used in cross-validation/random sampling. The indices were those of Y without NA values.
<code>Type</code>	Type of task (regression or classification).
<code>Nfold</code>	Number of cross-validation folds.
<code>num_sample</code>	Number of samples in random sampling.
<code>which_valid</code>	Indices of Y without NA values.
<code>cross_validation</code>	Specifies which cross-validation (TRUE) or random sampling (FALSE) was used during training.

Author(s)

Taichi Nukui, Tomohiro Ishibashi, Akio Onogi

See Also

`stacking_train`, `train_metamodel`

Examples

```
#Create a toy example
##Number of training samples
N1 <- 100

##Number of explanatory variables
P <- 200

##Create X of training data
X1 <- matrix(rnorm(N1 * P), nrow = N1, ncol = P)
colnames(X1) <- 1:P#column names are required by caret
```

[illegible]

```

#                               use_X = FALSE,
#                               TrainEachFold = FALSE,
#                               Nfold = 5)

#For random sampling, set cross_validation = FALSE and
#specify the number of samples and the sampling proportion
#using num_sample and proportion, respectively.
#To include the original features X when training the meta-model, set use_X = TRUE.
#When use_X is TRUE, simple linear regressions cannot be used
#as the meta learner because of rank deficient.
#The following code reflects the changes made to the relevant arguments.
#stacking_train_result <- stacking_train(X = X1,
#                               Y = Y1,
#                               Method = Method,
#                               Metamodel = "glmnet",
#                               core = 2,
#                               cross_validation = FALSE,
#                               use_X = TRUE,
#                               TrainEachFold = FALSE,
#                               num_sample = 5,
#                               proportion = 0.8)

#Prediction
#result <- stacking_predict(newX = X2, stacking_train_result)
#plot(Y2, result)

```

train_basemodel_core *Internal function called by train_basemodel*

Description

Training base models of stacking. This function is called by train_basemodel and designed for the internal use of train_basemodel.

Usage

```
train_basemodel_core(repeat.parLapply, division, 1, core, x, y, exclude)
```

Arguments

repeat.parLapply	A scalar indicating the number of repeats of parallel computation. If the number of base models is 10 and 5 cores are used for computation, repeat.parLapply is 2.
division	A matrix of which the number of columns is equal to repeat.parLapply. The elements are integers indicating the base models. For example, division[, 1] indicates the base models trained in the first calculation round.

l	A nested list indicating the training method and hyperparameters. The length is the number of base models. Each element is a list consisting of two elements, method and hyp, which are strings indicating the training method and a data frame including hyperparameter values, respectively. The number of columns of the data frame is the number of hyperparameters of the method, and the hyperparameter names should be specified as the column names.
core	Number of cores for parallel processing
x	An N x P matrix of explanatory variables where N is the number of samples and P is the number of variables
y	A length N Vector of objective variables
exclude	A vector of integers indicating the samples excluded from training as testing data

Details

This function is designed for the internal use and not for direct use by users. Thus, detaled usages are not provided.

Value

A list containing the training results of base models.

Author(s)

Taichi Nukui, Akio Onogi

See Also

train_basemodel

train_metamodel	<i>Training a meta model based on base models</i>
-----------------	---

Description

Training a meta model of stacking

Usage

```
train_metamodel(X, basemodel_train_result, which_to_use, Metamodel,
                use_X = FALSE, TrainEachFold = FALSE)
```

Arguments

<code>X</code>	An $N \times P$ matrix of explanatory variables where N is the number of samples and P is the number of variables. Column names are required by caret.
<code>basemodel_train_result</code>	The list output by <code>train_basemodel</code>
<code>which_to_use</code>	A vector of integers between 1 and L where L is the number of base models. These integers specify the base models used for training the meta model.
<code>Metamodel</code>	A strings specifying the meta learner
<code>use_X</code>	A logical indicating whether the meta-learner uses the original features X along with the base model predictions. If <code>TRUE</code> , it uses both X and the predictions; if <code>FALSE</code> , it uses only the predictions.
<code>TrainEachFold</code>	A logical indicating whether the meta learner learns using the predicted values of the base models at each cross-validation fold/random sample or not. If <code>TRUE</code> , the meta learners learns N fold times using the values predicted by the base models at each fold/sample. If <code>FALSE</code> , the meta learner learns once by pooling the predicted values of the base models of all folds/samples.

Details

Stacking by this package consists of the following 2 steps.

(1) Each base learner is trained. The training method can be chosen based on the `cross_validation` argument: If `cross_validation` is `TRUE`: The function performs N fold cross-validation for each base learner. If `cross_validation` is `FALSE`: The function trains each base learner using random sampling. The number of samples (`num_sample`) or the proportion of the data (`proportion`) can be specified to control the selection process. (2) Using the predicted values of each learner as the explanatory variables, the meta learner is trained. Steps (1) and (2) are conducted by `train_basemodel` and `train_metamodel`, respectively. Another function `stacking_train` conducts both steps at once by calling these functions (`train_basemodel` and `train_metamodel`).

Training of the meta learner can be modified by two arguments, `TrainEachFold` and `use_X`. `TrainEachFold` specifies whether the meta learner is trained for each fold or random sample individually, or once by pooling or combining all predicted values. `use_X` specifies whether the meta learner is trained with both the original features X and the base model predictions, or with only the base model predictions.

Meta learners can be chosen from the methods implemented in caret. The choosable methods can be seen at <https://topepo.github.io/caret/available-models.html> or using `names(getModelInfo())` after loading caret.

Value

A list containing the following elements is output.

<code>train_result</code>	A list containing the training results of the meta model, which is the list output by <code>train</code> function of caret. When <code>TrainEachFold</code> is <code>TRUE</code> , the length of list is N fold/ <code>num_sample</code> because the meta learner is trained N fold/ <code>num_sample</code> times.
---------------------------	---

which_to_use which_to_use given as the argument
 cross_validation A logical to specify whether to perform cross-validation. Set to TRUE to enable cross-validation or to FALSE to perform random sampling.
 use_X use_X
 TrainEachFold TrainEachFold

Author(s)

Taichi Nukui, Tomohiro Ishibashi, Akio Onogi

See Also

stacking_train, train_basemodel

Examples

```
#Create a toy example
##Number of training samples
N1 <- 100

##Number of explanatory variables
P <- 200

##Create X of training data
X1 <- matrix(rnorm(N1 * P), nrow = N1, ncol = P)
colnames(X1) <- 1:P#column names are required by caret

##Assume that the first 10 variables have effects on Y
##Then add noise with rnorm
Y1 <- rowSums(X1[, 1:10]) + rnorm(N1)

##Test data
N2 <- 100
X2 <- matrix(rnorm(N2 * P), nrow = N2, ncol = P)
colnames(X2) <- 1:P#Ignored (not required)
Y2 <- rowSums(X2[, 1:10])

#Specify base learners
Method <- list(glmnet = data.frame(alpha = c(0, 0.5, 1), lambda = rep(NA, 3)),
               pls = data.frame(ncomp = 5))
#=>This specifies four base learners.
##1. glmnet with alpha = 0 and lambda tuned
##2. glmnet with alpha = 0.5 and lambda tuned
##3. glmnet with alpha = 1 and lambda tuned
##4. pls with ncomp = 5

#The followings are the training and prediction processes
#If glmnet and pls are not installed, please install them in advance.
#Please remove #s before execution
```



```
#Prediction  
#result <- stacking_predict(newX = X2, stacking_train_result)  
#plot(Y2, result)
```

Index

`stacking_predict`, [2](#)

`stacking_train`, [4](#)

`train_basemodel`, [9](#)

`train_basemodel_core`, [13](#)

`train_metamodel`, [14](#)