

# Package ‘stacking’

September 1, 2023

**Type** Package

**Title** Building Predictive Models with Stacking

**Version** 0.1.2

**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],  
Akio Onogi [aut, cre, cph]

**Maintainer** Akio Onogi <[onogiakio@gmail.com](mailto:onogiakio@gmail.com)>

**Depends** caret, parallel

**Repository** CRAN

**Date/Publication** 2023-09-01 08:20:02 UTC

## R topics documented:

|                                |    |
|--------------------------------|----|
| stacking_predict . . . . .     | 1  |
| stacking_train . . . . .       | 3  |
| train_basemodel . . . . .      | 7  |
| train_basemodel_core . . . . . | 10 |
| train_metamodel . . . . .      | 11 |

|              |           |
|--------------|-----------|
| <b>Index</b> | <b>15</b> |
|--------------|-----------|

---

|                  |                             |
|------------------|-----------------------------|
| 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  $N \times P$  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  $N_{\text{fold}}$  predicted values where  $N_{\text{fold}}$  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. In the case of classification, the probabilities of each category are returned.

**Author(s)**

Taichi Nukui, 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,
#                                       Nfold = 5,
#                                       Method = Method,
#                                       Metamodel = "lm",
#                                       core = 2)

#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, Nfold = 5, Method = Method, core = 3)
#meta <- train_metamodel(base, which_to_use = 1:5, Metamodel = "lm")
#stacking_train_result <- list(base = base, meta = meta)
#=>this list should have elements named as base and meta

#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.

**Usage**

```
stacking_train(X, Y, Nfold, Method, Metamodel, TrainEachFold = FALSE, core = 1)
```

**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.   |
| Nfold         | Number of folds for cross-validation. This cross-validation is required for training.  |
| 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.  |
| TrainEachFold | A logical indicating whether the meta learner learns using the predicted values of the base models at each cross-validation fold or not. If TRUE, the meta learners learns Nfold times using the values predicted by the base models at each fold. If FALSE, the meta learner learns once by pooling the predicted values of the base models of all folds. |
| core          | Number of cores for parallel processing  |

**Details**

Stacking by this function consists of the following 2 steps. (1) Nfold cross-validation is conducted with each base learner.(2) Using the predicted values of each learner as the explanatory variables, the meta learner is trained. This function conducts steps (1) and (2) at once by calling `train_basemodel` and `train_metamodel`, respectively. But users can conduct these steps separately by directly using these functions.

In the step (2), there are two options. One is to train the meta learner Nfold times using the predicted values returned by the base models for each fold. The other is to train the meta learner once pooling the predicted values by the base models across folds. `TrainEachModel` swiches these options.

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, 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,
#                                       Nfold = 5,
#                                       Method = Method,
#                                       Metamodel = "lm",
#                                       core = 3)

#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, Nfold = 5, Method = Method, core = 3)
#meta <- train_metamodel(base, which_to_use = 1:5, Metamodel = "lm")
#stacking_train_result <- list(base = base, meta = meta)
#=>this list should have elements named as base and meta

#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 | <i>Training base models</i> |
|-----------------|-----------------------------|

---

## Description

Training base models of stacking

## Usage

```
train_basemodel(X, Y, Nfold, Method, core = 1)
```

## 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.  |

|        |  |
|--------|--|
| Nfold  | Number of folds for cross-validation. This cross-validation is required for training.  |
| 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  |

### Details

Stacking by this package consists of the following 2 steps. (1) Nfold cross-validation is conducted with each base learner.(2) Using the predicted values of each learner as the explanatory variables, the meta learner is trained. This function conducts step (1). Step (2) is conducted by train\_metamodel. 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.

|              |   |
|--------------|---|
| train_result | A list containing the training results of the base models. The length of this list is the same as Nfold, 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 train function of caret, but the element "trainingData" is removed to save memory. |
|--------------|---|

|              |   |
|--------------|---|
| no_base      | Number of base models.  |
| valpr        | Predicted values of base models obtained in cross-validation. Used as explanatory variables for the meta learner. |
| Y.randomised | Y ans X are randomized when cross-validation. Randomized Y is output to enable evaluation of prediction accuracy  |
| Order        | Order in randomization.   |
| Type         | Type of task (regression or classification).  |
| Nfold        | Number of cross-validation folds  |

**Author(s)**

Taichi Nukui, 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

##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 4 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

#Training of base learners
#base <- train_basemodel(X = X1, Y = Y1, Nfold = 5, Method = Method, core = 2)

#Training of a meta learner
#meta <- train_metamodel(base, which_to_use = 1:4, Metamodel = "lm")

#Combine both results
#stacking_train_result <- list(base = base, meta = meta)
#=>this list should have elements named as base and meta

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

#Training using stacking_train
#stacking_train_result <- stacking_train(X = X1,
#                                       Y = Y1,
#                                       Nfold = 5,
#                                       Method = Method,
#                                       Metamodel = "lm",
#                                       core = 2)

#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, detailed 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(basemodel_train_result, which_to_use, Metamodel, TrainEachFold = FALSE)
```

**Arguments**

|                                     |  |
|-------------------------------------|--|
| <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>TrainEachFold</code>          | A logical indicating whether the meta learner learns using the predicted values of the base models at each cross-validation fold or not. If TRUE, the meta learners learns Nfold times using the values predicted by the base models at each fold. If FALSE, the meta learner learns once by pooling the predicted values of the base models of all folds. |

**Details**

Stacking by this package consists of the following 2 steps. (1) Nfold cross-validation is conducted with each base learner.(2) Using the predicted values of each learner as the explanatory variables, the meta learner is trained. This function conducts step (2). Step (1) is conducted by `train_basemodel`. Another function `stacking_train` conducts both steps at once by calling these functions (`train_basemodel` and `train_metamodel`).

In the step (2), there are two options. One is to train the meta learner Nfold times using the predicted values returned by the base models for each fold. The other is to train the meta learner once pooling the predicted values by the base models across folds. `TrainEachModel` switches these options.

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 TRUE, the length of list is Nfold because the meta learner is trained Nfold times. |
| <code>which_to_use</code>  | <code>which_to_use</code> given as the argument   |
| <code>TrainEachFold</code> | <code>TrainEachFold</code>  |

**Author(s)**

Taichi Nukui, 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

#Training of base learners
#base <- train_basemodel(X = X1, Y = Y1, Nfold = 5, Method = Method, core = 2)

#Training of a meta learner
#meta <- train_metamodel(base, which_to_use = 1:4, Metamodel = "lm")

#Combine both results
#stacking_train_result <- list(base = base, meta = meta)
#=>this list should have elements named as base and meta

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

#Training using stacking_train
#stacking_train_result <- stacking_train(X = X1,
#                                       Y = Y1,

```

```
#           Nfold = 5,  
#           Method = Method,  
#           Metamodel = "lm",  
#           core = 2)  
  
#Prediction  
#result <- stacking_predict(newX = X2, stacking_train_result)  
#plot(Y2, result)
```

# Index

stacking\_predict, 1

stacking\_train, 3

train\_basemodel, 7

train\_basemodel\_core, 10

train\_metamodel, 11