Package ‘stacking’

June 11, 2023

Type Package
Title Building Predictive Models with Stacking
Version 0.1.0
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>.
License MIT + file LICENSE
NeedsCompilation no
Author Taichi Nukui [aut, cph], Akio Onogi [aut, cre, cph]
Maintainer Akio Onogi <onogiakio@gmail.com>
Depends caret, parallel
Repository CRAN
Date/Publication 2023-06-11 11:20:02 UTC

R topics documented:

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

Index

stacking_predict  Predict for new data

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 predicted values where Nfold 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

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)
```
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, 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.

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.

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.

base A list output by train_basemodel. See value of train_basemodel for the details
meta A list output by train_metamodel. See value of train_metamodel for the details

Author(s)

Taichi Nukui, Akio Onogi

See Also

train_basemodel, train_metamodel

Examples

#Create another example
#packages 'glmnet' and 'pls' are required
##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(0.2, 0.5, 0.8), lambda = c(0.1, 1, NA)),
                pls = data.frame(ncomp = c(2, 5)))
=>This specifies 8 base learners.
#1. glmnet with alpha = 0.2 and lambda = 0.1
#2. glmnet with alpha = 0.2 and lambda = 1
#3. glmnet with alpha = 0.5 and lambda = 0.1
#4. glmnet with alpha = 0.5 and lambda = 1
#5. glmnet with alpha = 0.8 and lambda = 0.1
#6. glmnet with alpha = 0.8 and lambda = 1
#7. pls with ncomp = 2
#8. pls with ncomp = 5

#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 = 2)
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),
                                      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),
                                  ncomp = seq(2, 70, 10))
)
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),
                      lambda = rep(NA, 6)),
  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

###train_basemodel

**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,

```r
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

```r
Method = list(glmnet = data.frame(alpha = c(0, 1), lambda = c(5, 10)))
```

All combinations of hyperparameter values are automatically created as

\[\text{[alpha, lambda]} = [0, 5], [0, 10], [1, 5], [1, 10]\]

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

```r
Method = list(glmnet = data.frame(alpha = c(0, 0.5, 1), lambda = c(5, 10, NA)))
```

resulting in 6 combinations of

\[\text{[alpha, lambda]} = [0, 5], [0, 10], [0.5, 5], [0.5, 10], [1, 5], [1, 10]\]

When a hyperparameter includes only NA as

```r
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).
train_basemodel

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

#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**

```r
train_basemodel_core(repeat.parLapply, division, l, 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

Description
Training a meta model of stacking

Usage
train_metamodel(basemodel_train_result, which_to_use, Metamodel)

Arguments
basemodel_train_result
The list output by train_basemodel
which_to_use
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.
Metamodel
A strings specifying the meta learner

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).

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 meta model, which is the list output by the `train` function of caret.
- `which_to_use`: `which_to_use` given as the argument.

**Author(s)**

Taichi Nukui, Akio Onogi

**See Also**

`stacking_train`, `train_basemodel`

**Examples**

```r
# 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

# 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