Package ‘Sieve’

June 22, 2022

Type Package

Title Nonparametric Estimation by the Method of Sieves

Version 1.0

Date 2022-05-27

Author Tianyu Zhang

Maintainer Tianyu Zhang <zty@uw.edu>

Description

Performs multivariate nonparametric regression/classification by the method of sieves (or using orthogonal series). The method is suitable for continuous/binary problems with multivariate or moderate high-dimensional features (dimension < 100). The main estimator in this package, penalized sieve estimator, is adaptive to the feature dimension with provable theoretical guarantees. Moreover, such a method is computationally tractable in the sense it typically has a polynomial dependence (rather than an exponential one) on the feature dimension and an almost linear dependence on the sample size. Details of the methods and model assumptions can be found in: Tianyu Zhang, and Noah Simon (2022) <arXiv:2206.02994>.

License GPL-2

Imports Rcpp (>= 1.0.7), combinat, glmnet, methods

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.1.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2022-06-22 19:20:08 UTC

R topics documented:

`Sieve-package` *Nonparametric Estimation by the Method of Sieves*

Description

Performs multivariate nonparametric regression/classification by the method of sieves (or using orthogonal series). The method is suitable for continuous/binary problems with multivariate or moderate high-dimensional features (dimension < 100). The main estimator in this package, penalized sieve estimator, is adaptive to the feature dimension with provable theoretical guarantees. Moreover, such a method is computationally tractable in the sense it typically has a polynomial dependence (rather than an exponential one) on the feature dimension and an almost linear dependence on the sample size. Details of the methods and model assumptions can be found in: Tianyu Zhang, and Noah Simon (2022) <arXiv:2206.02994>.

Details

The DESCRIPTION file:

Package:	Sieve
Type:	Package
Title:	Nonparametric Estimation by the Method of Sieves
Version:	1.0
Date:	2022-05-27
Author:	Tianyu Zhang
Maintainer:	Tianyu Zhang <zty@uw.edu>
Description:	Performs multivariate nonparametric regression/classification by the method of sieves (or using orthogonal s
License:	GPL-2
Imports:	Rcpp (>= 1.0.7), combinat, glmnet, methods
LinkingTo:	Rcpp, RcppArmadillo
RoxygenNote:	7.1.1

Index of help topics:

```
GenSamples              Generate some simulation/testing samples with
                        nonlinear truth.
Sieve-package           Nonparametric Estimation by the Method of
                        Sieves
sieve_predict           Predict the outcome of interest for new samples
sieve_preprocess        Preprocess the original data for sieve
                        estimation.
sieve_solver            Calculate the coefficients for the basis
                        functions
```

~~ An overview of how to use the ~~ ~~ package, including the most ~~ ~~ important functions ~~

Author(s)

Tianyu Zhang

Maintainer: Tianyu Zhang <zty@uw.edu>

References

Tianyu Zhang and Noah Simon (2022) <arXiv:2206.02994>

Examples

```
xdim <- 5
basisN <- 1000
type <- 'cosine'

#non-linear additive truth. Half of the features are truly associated with the outcome
TrainData <- GenSamples(s.size = 300, xdim = xdim,
            frho = 'additive', frho.para = xdim/2)

#noise-free testing samples
TestData <- GenSamples(s.size = 1e3, xdim = xdim, noise.para = 0,
            frho = 'additive', frho.para = xdim/2)

sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
            basisN = basisN, type = type, interaction_order = 2)

sieve.model <- sieve_solver(sieve.model, TrainData$Y, l1 = TRUE)

sieve_model_prediction <- sieve_predict(testX = TestData[,2:(xdim+1)],
                testY = TestData$Y, sieve.model)
```

`GenSamples` *Generate some simulation/testing samples with nonlinear truth.*

Description

This function is used in several examples in the package.

Usage

```
GenSamples(
  s.size,
  xdim = 1,
  x.dis = "uniform",
  x.para = NULL,
  frho = "linear",
  frho.para = 100,
  y.type = "continuous",
  noise.dis = "normal",
  noise.para = 0.5
)
```

Arguments

`s.size`	a number. Sample size.
`xdim`	a number. Dimension of the feature vectors X.
`x.dis`	a string. It specifies the distribution of feature X. The default is uniform distribution over `xdim`-dimensional unit cube.
`x.para`	extra parameter to specify the feature distribution.
`frho`	a string. It specifies the true regression/log odds functions used to generate the data set. The default is a linear function.
`frho.para`	extra parameter to specify the true underlying regression/log odds function.
`y.type`	a string. Default is `y.type = 'continuous'`, meaning the outcome is numerical and the problem is regression. Set it to `y.type = 'binary'` for binary outcome.
`noise.dis`	a string. For the distribution of the noise variable (under regression probelm settings). Default is Gaussian distribution.
`noise.para`	a number. It specifies the magnitude of the noise in regression settings.

Value

a `data.frame`. The variable `Y` is the outcome (either continuous or binary). Each of the rest of the variables corresponds to one dimension of the feature vector.

Examples

```
xdim <- 1 #1 dimensional feature
#generate 1000 training samples
TrainData <- GenSamples(s.size = 1000, xdim = xdim)
#generate some noise-free testing samples
TestData <- GenSamples(s.size = 1000, xdim = xdim, noise.para = 0)
```

`sieve_predict` *Predict the outcome of interest for new samples*

Description

Use the fitted sieve regression model from sieve_solver. It also returns the testing mean-squared errors.

Usage

```
sieve_predict(model, testX, testY = NULL)
```

Arguments

`model`	a list. Use the fitted model from sieve_solver.
`testX`	a data frame. Dimension equals to test sample size x feature diemnsion. Should be of a similar format as the training feature provided to sieve_preprocess.
`testY`	a vector. The outcome of testing samples (if known). Default is NULL. For regression problems, the algorithm also returns the testing mean-squared errors.

Value

a list.

`predictY`	a matrix. Dimension is test sample size (# of rows) x number of penalty hyperparameter lambda (# of columns). For regression problem, that is, when family = "gaussian", each entry is the estimated conditional mean (or predictor of outcome Y). For classification problems (family = "binomial"), each entry is the predicted probability of having Y = 1 (which class is defined as "class 1" depends on the training data labeling).
`MSE`	For regression problem, when testY is provided, the algorithm also calculates the mean-sqaured errors using testing data. Each entry of `MSE` correponds to one value of penalization hyperparameter `lambda`

Examples

```
xdim <- 1 #1 dimensional feature
#generate 1000 training samples
TrainData <- GenSamples(s.size = 1000, xdim = xdim)
#use 50 cosine basis functions
type <- 'cosine'
basisN <- 50
sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
                                basisN = basisN, type = type)
sieve.fit<- sieve_solver(model = sieve.model, Y = TrainData$Y)
#generate 1000 testing samples
TestData <- GenSamples(s.size = 1000, xdim = xdim)
sieve.prediction <- sieve_predict(model = sieve.fit,
                                  testX = TestData[,2:(xdim+1)],
                                  testY = TestData$Y)
###if the outcome is binary,
###need to solve a nonparametric logistic regression problem
xdim <- 1
TrainData <- GenSamples(s.size = 1e3, xdim = xdim, y.type = 'binary', frho = 'nonlinear_binary')
sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
                                basisN = basisN, type = type)
sieve.fit<- sieve_solver(model = sieve.model, Y = TrainData$Y,
                         family = 'binomial')

###the predicted value is conditional probability (of taking class 1).
TrainData <- GenSamples(s.size = 1e3, xdim = xdim, y.type = 'binary', frho = 'nonlinear_binary')
sieve.prediction <- sieve_predict(model = sieve.fit,
                                  testX = TestData[,2:(xdim+1)])
```

sieve_preprocess *Preprocess the original data for sieve estimation.*

Description

Generate the design matrix for the downstream lasso-type penalized model fitting.

Usage

```
sieve_preprocess(
  X,
  basisN = NULL,
  maxj = NULL,
  type = "cosine",
  interaction_order = 3,
  index_matrix = NULL,
  norm_feature = TRUE,
  norm_para = NULL
)
```

Arguments

`X`	a data frame containing original features. The (i,j)-th element is the j-th dimension of the i-th sample's feature vector. So the number of rows equals to the sample size and the number of columns equals to the feature dimension.
`basisN`	number of sieve basis function. It is in general larger than the dimension of the original feature. Default is 50*dimension of original feature. A larger value has a smaller approximation error but it is harder to estimate. The computational time/memory requirement should scale linearly to `basisN`.
`maxj`	a number. the maximum index product of the basis function. A larger value means more basisN. If basisN is already specified, do not need to provide value for this argument.
`type`	a string. It specifies what kind of basis functions are used. The default is (aperiodic) cosine basis functions, which is suitable for most purpose.
`interaction_order`	a number. It also controls the model complexity. 1 means fitting an additive model, 2 means fitting a model allows, 3 means interaction terms between 3 dimensions of the feature, etc. The default is 3. For large sample size, lower dimension problems, try a larger value (but need to be smaller than the dimension of original features); for smaller sample size and higher dimensional problems, try set it to a smaller value (1 or 2).
`index_matrix`	a matrix. provide a pre-generated index matrix. The default is NULL, meaning sieve_preprocess will generate one for the user.
`norm_feature`	a logical variable. Default is TRUE. It means sieve_preprocess will rescale the each dimension of features to 0 and 1. Only set to FALSE when user already manually rescale them between 0 and 1.

`norm_para`	a matrix. It specifies how the features are normalized. For training data, use the default value NULL.

Value

A list containing the necessary information for next step model fitting. Typically, the list is used as the main input of Sieve::sieve_solver.

`Phi`	a matrix. This is the design matrix directly used by the next step model fitting. The (i,j)-th element of this matrix is the evaluation of i-th sample's feature at the j-th basis function. The dimension of this matrix is sample size x basisN.
`X`	a matrix. This is the rescaled original feature/predictor matrix.
`type`	a string. The type of basis funtion.
`index_matrix`	a matrix. It specifies what are the product basis functions used when constructing the design matrix Phi. It has a dimension basisN x dimension of original features. There are at most interaction_order many non-1 elements in each row.
`basisN`	a number. Number of sieve basis functions.
`norm_para`	a matrix. It records how each dimension of the feature/predictor is rescaled, which is useful when rescaling the testing sample's predictors.

Examples

```
xdim <- 1 #1 dimensional feature
#generate 1000 training samples
TrainData <- GenSamples(s.size = 1000, xdim = xdim)
#use 50 cosine basis functions
type <- 'cosine'
basisN <- 50
sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
                                basisN = basisN, type = type)
#sieve.model$Phi #Phi is the design matrix

xdim <- 5 #1 dimensional feature
#generate 1000 training samples
#only the first two dimensions are truly associated with the outcome
TrainData <- GenSamples(s.size = 1000, xdim = xdim,
                              frho = 'additive', frho.para = 2)

#use 1000 basis functions
#each of them is a product of univariate cosine functions.
type <- 'cosine'
basisN <- 1000
sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
                                basisN = basisN, type = type)
#sieve.model$Phi #Phi is the design matrix

#fit a nonaprametric additive model by setting interaction_order = 1
sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
                                basisN = basisN, type = type,
                                interaction_order = 1)
```

```
#sieve.model$index_matrix #for each row, there is at most one entry >= 2.
#this means there are no basis functions varying in more than 2-dimensions
#that is, we are fitting additive models without interaction between features.
```

`sieve_solver` *Calculate the coefficients for the basis functions*

Description

This is the main function that performs sieve estimation. It calculate the coefficients by solving a penalized lasso type problem.

Usage

```
sieve_solver(
  model,
  Y,
  l1 = TRUE,
  family = "gaussian",
  lambda = NULL,
  nlambda = 100
)
```

Arguments

`model`	a list. Typically, it is the output of Sieve::sieve_preprocess.
`Y`	a vector. The outcome variable. The length of Y equals to the training sample size, which should also match the row number of X in model.
`l1`	a logical variable. TRUE means calculating the coefficients by sovling a l1-penalized empirical risk minimization problem. FALSE means solving a least-square problem. Default is TRUE.
`family`	a string. 'gaussian', mean-squared-error regression problem.
`lambda`	same as the lambda of glmnet::glmnet.
`nlambda`	a number. Number of penalization hyperparameter used when solving the lasso-type problem. Default is 100.

Value

a list. In addition to the preprocessing information, it also has the fitted value.

`Phi`	a matrix. This is the design matrix directly used by the next step model fitting. The (i,j)-th element of this matrix is the evaluation of i-th sample's feature at the j-th basis function. The dimension of this matrix is sample size x basisN.
`X`	a matrix. This is the rescaled original feature/predictor matrix.
`beta_hat`	a matrix. Dimension is basisN x nlambda. The j-th column corresponds to the fitted regression coeffcients using the j-th hyperparameter in lambda.

`type`	a string. The type of basis funtion.
`index_matrix`	a matrix. It specifies what are the product basis functions used when constructing the design matrix Phi. It has a dimension basisN x dimension of original features. There are at most interaction_order many non-1 elements in each row.
`basisN`	a number. Number of sieve basis functions.
`norm_para`	a matrix. It records how each dimension of the feature/predictor is rescaled, which is useful when rescaling the testing sample's predictors.
`lambda`	a vector. It records the penalization hyperparameter used when solving the lasso problems. Default has a length of 100, meaning the algorithm tried 100 different penalization hyperparameters.
`family`	a string. 'gaussian', continuous numerical outcome, regression probelm; 'binomial', binary outcome, classification problem.

Examples

```
xdim <- 1 #1 dimensional feature
#generate 1000 training samples
TrainData <- GenSamples(s.size = 1000, xdim = xdim)
#use 50 cosine basis functions
type <- 'cosine'
basisN <- 50
sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
                                basisN = basisN, type = type)
sieve.fit<- sieve_solver(model = sieve.model, Y = TrainData$Y)

###if the outcome is binary,
###need to solve a nonparametric logistic regression problem
xdim <- 1
TrainData <- GenSamples(s.size = 1e3, xdim = xdim, y.type = 'binary', frho = 'nonlinear_binary')
sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
                                basisN = basisN, type = type)
sieve.fit<- sieve_solver(model = sieve.model, Y = TrainData$Y,
                         family = 'binomial')
```

Index