Package ‘npcs’

April 27, 2023

Type Package

Title Neyman-Pearson Classification via Cost-Sensitive Learning

Version 0.1.1

Description We connect the multi-class Neyman-Pearson classification (NP) problem to the cost-sensitive learning (CS) problem, and propose two algorithms (NPMC-CX and NPMC-ER) to solve the multi-class NP problem through cost-sensitive learning tools. Under certain conditions, the two algorithms are shown to satisfy multi-class NP properties. More details are available in the paper ``Neyman-Pearson Multi-class Classification via Cost-sensitive Learning" (Ye Tian and Yang Feng, 2021).

Imports dfoptim, magrittr, smotefamily, foreach, caret, formatR, dplyr, forcats, ggplot2, tidyr, nnet

License GPL-2

Depends R (>= 3.5.0)

Encoding UTF-8

RoxygenNote 7.2.2

Suggests knitr, rmarkdown, gbm

VignetteBuilder knitr

NeedsCompilation no

Author Ye Tian [aut],
Ching-Tsung Tsai [aut, cre],
Yang Feng [aut]

Maintainer Ching-Tsung Tsai <tctsung@nyu.edu>

Repository CRAN

Date/Publication 2023-04-27 08:20:02 UTC

R topics documented:

`cv.npcs` *Compare the performance of the NPMC-CX, NPMC-ER, and vanilla models through cross-validation or bootstrapping methods*

Description

Compare the performance of the NPMC-CX, NPMC-ER, and vanilla models through cross-validation or bootstrapping methods. The function will return a summary of evaluation which includes various evaluation metrics, and visualize the class-specific error rates.

Usage

```
cv.npcs(
  x,
  y,
  classifier,
  alpha,
  w,
  fold = 5,
  stratified = TRUE,
  partition_ratio = 0.7,
  resample = c("bootstrapping", "cv"),
  seed = 1,
  verbose = TRUE,
  plotit = TRUE,
  trControl = list(),
  tuneGrid = list()
)
```

Arguments

`x`	matrix; the predictor matrix of complete data
`y`	numeric/factor/string; the response vector of complete data.
`classifier`	string; Model to use for npcs function
`alpha`	the levels we want to control for error rates of each class. The length must be equal to the number of classes
`w`	the weights in objective function. Should be a vector of length K, where K is the number of classes.
`fold`	integer; number of folds in CV or number of bootstrapping iterations, default=5
`stratified`	logical; if TRUE, sample will be split into groups based on the proportion of response vector

partition_ratio
: numeric; the proportion of data to be used for model construction when parameter resample=="bootstrapping"

resample
: string; the resampling method
  - bootstrapping: bootstrapping, which iteration number is set by parameter "fold"
  - cv: cross validation, the number of folds is set by parameter "fold"

seed
: random seed

verbose
: logical; if TRUE, cv.npcs will print the progress. If FALSE, the model will remain silent

plotit
: logical; if TRUE, the output list will return a box plot summarizing the error rates of vanilla model and NPMC model

trControl
: list; resampling method within each fold

tuneGrid
: list; for hyperparameters tuning or setting

Examples

```
# data generation: case 1 in Tian, Y., & Feng, Y. (2021) with n = 1000
set.seed(123, kind = "L'Ecuyer-CMRG")
train.set <- generate_data(n = 1000, model.no = 1)
x <- train.set$x
y <- train.set$y
test.set <- generate_data(n = 2000, model.no = 1)
x.test <- test.set$x
y.test <- test.set$y
alpha <- c(0.05, NA, 0.01)
w <- c(0, 1, 0)
# contruct the multi-class NP problem

cv.npcs.knn <- cv.npcs(x, y, classifier = "knn", w = w, alpha = alpha)
# result summary and visualization
cv.npcs.knn$summaries
cv.npcs.knn$plot
```

---

error_rate — *Calculate the error rates for each class.*

Description

Calculate the error rate for each class given the predicted labels and true labels.

Usage

```
error_rate(y.pred, y, class.names = NULL)
```

Arguments

`y.pred`	the predicted labels.
`y`	the true labels.
`class.names`	the names of classes. Should be a string vector. Default = NULL, which will set the name as 1, ..., K, where K is the number of classes.

Value

A vector of the error rate for each class. The vector name is the same as `class.names`.

References

Tian, Y., & Feng, Y. (2021). Neyman-Pearson Multi-class Classification via Cost-sensitive Learning. Submitted. Available soon on arXiv.

See Also

`npcs`, `predict.npcs`, `generate_data`, `gamma_smote`.

Examples

```
# data generation
set.seed(123, kind = "L'Ecuyer-CMRG")
train.set <- generate_data(n = 1000, model.no = 1)
x <- train.set$x
y <- train.set$y

test.set <- generate_data(n = 1000, model.no = 1)
x.test <- test.set$x
y.test <- test.set$y

library(nnet)
fit.vanilla <- multinom(y~., data = data.frame(x = x, y = factor(y)), trace = FALSE)
y.pred.vanilla <- predict(fit.vanilla, newdata = data.frame(x = x.test))
error_rate(y.pred.vanilla, y.test)
```

`gamma_smote` *Gamma-synthetic minority over-sampling technique (gamma-SMOTE).*

Description

gamma-SMOTE with some gamma in [0,1], which is a variant of the original SMOTE proposed by Chawla, N. V. et. al (2002). This can be combined with the NPMC methods proposed in Tian, Y., & Feng, Y. (2021). See Section 5.2.3 in Tian, Y., & Feng, Y. (2021) for more details.

Usage

```
gamma_smote(x, y, dup_rate = 1, gamma = 0.5, k = 5)
```

Arguments

x	the predictor matrix, where each row and column represents an observation and predictor, respectively.
y	the response vector. Must be integers from 1 to K for some K >= 2. Can either be a numerical or factor vector.
dup_rate	duplicate rate of original data. Default = 1, which finally leads to a new data set with twice sample size.
gamma	the upper bound of uniform distribution used when generating synthetic data points in SMOTE. Can be any number between 0 and 1. Default = 0.5. When it equals to 1, gamma-SMOTE is equivalent to the original SMOTE (Chawla, N. V. et. al (2002)).
k	the number of nearest neighbors during sampling process in SMOTE. Default = 5.

Value

A list consisting of merged original and synthetic data, with two components x and y. x is the predictor matrix and y is the label vector.

References

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16, 321-357.

Tian, Y., & Feng, Y. (2021). Neyman-Pearson Multi-class Classification via Cost-sensitive Learning. Submitted. Available soon on arXiv.

See Also

npcs, predict.npcs, error_rate, and generate_data.

Examples

```
## Not run:
set.seed(123, kind = "L'Ecuyer-CMRG")
train.set <- generate_data(n = 200, model.no = 1)
x <- train.set$x
y <- train.set$y

test.set <- generate_data(n = 1000, model.no = 1)
x.test <- test.set$x
y.test <- test.set$y

# contruct the multi-class NP problem: case 1 in Tian, Y., & Feng, Y. (2021)
alpha <- c(0.05, NA, 0.01)
w <- c(0, 1, 0)

## try NPMC-CX, NPMC-ER based on multinomial logistic regression, and vanilla multinomial
## logistic regression without SMOTE. NPMC-ER outputs the infeasibility error information.
fit.npmc.CX <- try(npcs(x, y, algorithm = "CX", classifier = "multinom", w = w, alpha = alpha))
```

```
fit.npmc.ER <- try(npcs(x, y, algorithm = "ER", classifier = "multinom", w = w, alpha = alpha,
refit = TRUE))
fit.vanilla <- nnet::multinom(y~., data = data.frame(x = x, y = factor(y)), trace = FALSE)

# test error of NPMC-CX based on multinomial logistic regression without SMOTE
y.pred.CX <- predict(fit.npmc.CX, x.test)
error_rate(y.pred.CX, y.test)

# test error of vanilla multinomial logistic regression without SMOTE
y.pred.vanilla <- predict(fit.vanilla, newdata = data.frame(x = x.test))
error_rate(y.pred.vanilla, y.test)


## create synthetic data by 0.5-SMOTE
D.syn <- gamma_smote(x, y, dup_rate = 1, gamma = 0.5, k = 5)
x <- D.syn$x
y <- D.syn$y

## try NPMC-CX, NPMC-ER based on multinomial logistic regression, and vanilla multinomial logistic
## regression with SMOTE. NPMC-ER can successfully find a solution after SMOTE.
fit.npmc.CX <- try(npcs(x, y, algorithm = "CX", classifier = "multinom", w = w, alpha = alpha))
fit.npmc.ER <- try(npcs(x, y, algorithm = "ER", classifier = "multinom", w = w, alpha = alpha,
refit = TRUE))
fit.vanilla <- nnet::multinom(y~., data = data.frame(x = x, y = factor(y)), trace = FALSE)

# test error of NPMC-CX based on multinomial logistic regression with SMOTE
y.pred.CX <- predict(fit.npmc.CX, x.test)
error_rate(y.pred.CX, y.test)

# test error of NPMC-ER based on multinomial logistic regression with SMOTE
y.pred.ER <- predict(fit.npmc.ER, x.test)
error_rate(y.pred.ER, y.test)

# test error of vanilla multinomial logistic regression wit SMOTE
y.pred.vanilla <- predict(fit.vanilla, newdata = data.frame(x = x.test))
error_rate(y.pred.vanilla, y.test)

## End(Not run)
```

---

generate_data *Generate the data.*

Description

Generate the data from two simulation cases in Tian, Y., & Feng, Y. (2021).

Usage

```
generate_data(n = 1000, model.no = 1)
```

Arguments

`n`	the generated sample size. Default = 1000.
`model.no`	the model number in Tian, Y., & Feng, Y. (2021). Can be 1 or 2. Default = 1.

Value

A list with two components x and y. x is the predictor matrix and y is the label vector.

References

Tian, Y., & Feng, Y. (2021). Neyman-Pearson Multi-class Classification via Cost-sensitive Learning. Submitted. Available soon on arXiv.

See Also

`npcs`, `predict.npcs`, `error_rate`, and `gamma_smote`.

Examples

```
set.seed(123, kind = "L'Ecuyer-CMRG")
train.set <- generate_data(n = 1000, model.no = 1)
x <- train.set$x
y <- train.set$y
```

---

npcs — *Fit a multi-class Neyman-Pearson classifier with error controls via cost-sensitive learning.*

Description

Fit a multi-class Neyman-Pearson classifier with error controls via cost-sensitive learning. This function implements two algorithms proposed in Tian, Y. & Feng, Y. (2021). The problem is minimize a linear combination of P(hat(Y)(X) != k| Y=k) for some classes k while controlling P(hat(Y)(X) != k| Y=k) for some classes k. See Tian, Y. & Feng, Y. (2021) for more details.

Usage

```
npcs(
  x,
  y,
  algorithm = c("CX", "ER"),
  classifier,
  seed = 1,
  w,
  alpha,
  trControl = list(),
```

```
  tuneGrid = list(),
  split.ratio = 0.5,
  split.mode = c("by-class", "merged"),
  tol = 1e-06,
  refit = TRUE,
  protect = TRUE,
  opt.alg = c("Hooke-Jeeves", "Nelder-Mead")
)
```

Arguments

`x`	the predictor matrix of training data, where each row and column represents an observation and predictor, respectively.
`y`	the response vector of training data. Must be integers from 1 to K for some K >= 2. Can be either a numerical or factor vector.
`algorithm`	the NPMC algorithm to use. String only. Can be either "CX" or "ER", which implements NPMC-CX or NPMC-ER in Tian, Y. & Feng, Y. (2021).
`classifier`	which model to use for estimating the posterior distribution P(Y|X = x). String only.
`seed`	random seed
`w`	the weights in objective function. Should be a vector of length K, where K is the number of classes.
`alpha`	the levels we want to control for error rates of each class. Should be a vector of length K, where K is the number of classes. Use NA if if no error control is imposed for specific classes.
`trControl`	list; resampling method
`tuneGrid`	list; for hyperparameters tuning or setting
`split.ratio`	the proportion of data to be used in searching lambda (cost parameters). Should be between 0 and 1. Default = 0.5. Only useful when `algorithm` = "ER".
`split.mode`	two different modes to split the data for NPMC-ER. String only. Can be either "per-class" or "merged". Default = "per-class". Only useful when `algorithm` = "ER". • per-class: split the data by class. • merged: split the data as a whole.
`tol`	the convergence tolerance. Default = 1e-06. Used in the lambda-searching step. The optimization is terminated when the step length of the main loop becomes smaller than `tol`. See pages of `hjkb` and `nmkb` for more details.
`refit`	whether to refit the classifier using all data after finding lambda or not. Boolean value. Default = TRUE. Only useful when `algorithm` = "ER".
`protect`	whether to threshold the close-zero lambda or not. Boolean value. Default = TRUE. This parameter is set to avoid extreme cases that some lambdas are set equal to zero due to computation accuracy limit. When `protect` = TRUE, all lambdas smaller than 1e-03 will be set equal to 1e-03.
`opt.alg`	optimization method to use when searching lambdas. String only. Can be either "Hooke-Jeeves" or "Nelder-Mead". Default = "Hooke-Jeeves".

Value

An object with S3 class "npcs".

lambda	the estimated lambda vector, which consists of Lagrangian multipliers. It is related to the cost. See Section 2 of Tian, Y. & Feng, Y. (2021) for details.
fit	the fitted classifier.
classifier	which classifier to use for estimating the posterior distribution P(Y|X = x).
algorithm	the NPMC algorithm to use.
alpha	the levels we want to control for error rates of each class.
w	the weights in objective function.
pik	the estimated marginal probability for each class.

References

Tian, Y., & Feng, Y. (2021). Neyman-Pearson Multi-class Classification via Cost-sensitive Learning. Submitted. Available soon on arXiv.

See Also

predict.npcs, error_rate, generate_data, gamma_smote.

Examples

```
# data generation: case 1 in Tian, Y., & Feng, Y. (2021) with n = 1000
set.seed(123, kind = "L'Ecuyer-CMRG")
train.set <- generate_data(n = 1000, model.no = 1)
x <- train.set$x
y <- train.set$y

test.set <- generate_data(n = 1000, model.no = 1)
x.test <- test.set$x
y.test <- test.set$y

# contruct the multi-class NP problem: case 1 in Tian, Y., & Feng, Y. (2021)
alpha <- c(0.05, NA, 0.01)
w <- c(0, 1, 0)

# try NPMC-CX, NPMC-ER, and vanilla multinomial logistic regression
fit.vanilla <- nnet::multinom(y~., data = data.frame(x = x, y = factor(y)), trace = FALSE)
fit.npmc.CX <- try(npcs(x, y, algorithm = "CX", classifier = "multinom",
w = w, alpha = alpha))
fit.npmc.ER <- try(npcs(x, y, algorithm = "ER", classifier = "multinom",
w = w, alpha = alpha, refit = TRUE))
# test error of vanilla multinomial logistic regression
y.pred.vanilla <- predict(fit.vanilla, newdata = data.frame(x = x.test))
error_rate(y.pred.vanilla, y.test)
# test error of NPMC-CX
y.pred.CX <- predict(fit.npmc.CX, x.test)
error_rate(y.pred.CX, y.test)
```

```
# test error of NPMC-ER
y.pred.ER <- predict(fit.npmc.ER, x.test)
error_rate(y.pred.ER, y.test)
```

---

predict.npcs — *Predict new labels from new data based on the fitted NPMC classifier.*

Description

Predict new labels from new data based on the fitted NPMC classifier, which belongs to S3 class "npcs".

Usage

```
## S3 method for class 'npcs'
predict(object, newx, ...)
```

Arguments

object	the model object for prediction
newx	input feature data
...	arguments to pass down

---

print.cv.npcs — *Print the cv.npcs object.*

Description

Print the cv.npcs object.

Usage

```
## S3 method for class 'cv.npcs'
print(x, ...)
```

Arguments

x	fitted cv.npcs object using `cv.npcs`.
...	additional arguments.

Index