Package ‘npcs’

October 18, 2021

Type Package

Title Neyman-Pearson Classification via Cost-sensitive Learning

Version 0.1.0

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), which will be posted on arXiv soon.

Imports dfoptim, nnet, randomForest, e1071, magrittr, MASS, smotefamily, rpart, foreach, naivebayes, caret, formatR

License GPL-2

Depends R (>= 3.5.0)

Encoding UTF-8

LazyData TRUE

RoxygenNote 7.1.2

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Ye Tian [aut, cre],
Yang Feng [aut]

Maintainer Ye Tian <ye.t@columbia.edu>

R topics documented:

error_rate 2
gamma_smote 3
generate_data 5
npcs 6
predict.npcs 9

Index **10**

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: case 1 in Tian, Y., & Feng, Y. (2021) with p = 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 with multinomial logistic regression, and vanilla multinomial
## logistic regression
fit.npmc.CX <- try(npcs(x, y, algorithm = "CX", classifier = "logistic", w = w, alpha = alpha))
fit.npmc.ER <- try(npcs(x, y, algorithm = "ER", classifier = "logistic", 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
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)

# 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)
```

---

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 = "logistic", w = w, alpha = alpha))
fit.npmc.ER <- try(npcs(x, y, algorithm = "ER", classifier = "logistic", 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 = "logistic", w = w, alpha = alpha))
fit.npmc.ER <- try(npcs(x, y, algorithm = "ER", classifier = "logistic", 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 = c("logistic", "knn", "randomforest", "tree", "neuralnet", "svm", "lda",
    "qda", "nb", "nnb"),
  w,
  alpha,
  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. • logistic: multinomial logistic regression, which is implemented via `multinom` in package `nnet`. • knn: k-nearest neighbor, which is implemented via `knn3` in package `caret`. An addition parameter (number of nearest neighbors) "k" is needed, which is 5 in default. • randomforest: random forests, which is implemented via `randomForest` in package `randomForest`. • tree: decition trees, which is implemented via `rpart` in package `rpart`.

- neuralnet: single-layer neural networks, which is implemented via `nnet` in package `nnet`.
- svm: support vector machines, which is implemented via `svm` in package `e1071`.
- lda: linear discriminant analysis, which is implemented via `lda` in package `MASS`.
- qda: quadratic discriminant analysis, which is implemented via `qda` in package `MASS`.
- nb: naive Bayes classifier with Gaussian marginals, which is implemented via `naiveBayes` in package `e1071`.
- nnb: naive Bayes classifier with non-parametric-estimated marginals (kernel-based), which is implemented via `nonparametric_naive_bayes` in package `naivebayes`. The default kernel is the Gaussian kernel. Check the documentation of function `nonparametric_naive_bayes` to see how to change the estimation settings.

`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.

`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".

`...` additional arguments. Will be passed to the function which fits the model indicated in `classifier`. For example, when `classifier` = "knn", the number of nearest neightbors `k` should be inputed. When `classifier` = "neuralnets"

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.npmc.CX <- try(npcs(x, y, algorithm = "CX", classifier = "logistic", w = w, alpha = alpha))
fit.npmc.ER <- try(npcs(x, y, algorithm = "ER", classifier = "logistic", 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
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)

# 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)
```

`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 fitted NPMC classifier from function `npcs`, which is an object of S3 class "npcs".
`newx`	the new observations. Should be a matrix or a data frame, where each row and column represents an observation and predictor, respectively.
`...`	additional arguments.

Value

the predicted labels.

References

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

See Also

`npcs`, `error_rate`, `generate_data`, `gamma_smote`.

Index

error_rate, 2, 4, 5, 8, 9

gamma_smote, 2, 3, 5, 8, 9
generate_data, 2, 4, 5, 8, 9

hjkb, 7

knn3, 6

lda, 7

multinom, 6

naiveBayes, 7
nmkb, 7
nnet, 7
nonparametric_naive_bayes, 7
npcs, 2, 4, 5, 6, 9

predict.npcs, 2, 4, 5, 8, 9

qda, 7

randomForest, 6
rpart, 6

svm, 7