A demonstration of the npcs package

Ye Tian and Yang Feng

2021-10-18

We provide an introductory demo of the usage for the `npcs` package. This package implements two multi-class Neyman-Pearson classification algorithms proposed in Tian and Feng (2021).

Installation

`RaSEn` can be installed from CRAN.

```r
install.packages("npcs", repos = "http://cran.us.r-project.org")
```

Then we can load the package:

```r
library(npcs)
```

Introduction

Suppose there are K classes ($K \geq 2$), and we denote them as classes 1 to K. The training samples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ are i.i.d. copies of $(X, Y) \subseteq \mathcal{X} \otimes \{1, \ldots, K\}$, where $\mathcal{X} \subseteq \mathbb{R}^p$. Suggested by Mossman (1999) and Dreiseitl, Ohno-Machado, and Binder (2000), we consider $\mathbb{P}_{X|Y=k}(\phi(X) \neq k|Y = k)$ as the k-th error rate of classifier ϕ for any $k \in \{1, \ldots, K\}$. We focus on the problem which minimizes a weighted sum of $\{\mathbb{P}_{X|Y=k}(\phi(X) \neq k)\}_{k=1}^K$ and controls $\mathbb{P}_{X|Y=k}(\phi(X) \neq k)$ for $k \in \mathcal{A}$, where $\mathcal{A} \subseteq \{1, \ldots, K\}$. The Neyman-Pearson *multi-class* classification (NPMC) problem can be formally presented as

$$\min_{\phi} \quad J(\phi) = \sum_{k=1}^{K} w_k \mathbb{P}_{X|Y=k}(\phi(X) \neq k) \tag{1}$$

$$\text{s.t.} \quad \mathbb{P}_{X|Y=k}(\phi(X) \neq k) \leq \alpha_k, \quad k \in \mathcal{A}, \tag{2}$$

where $\phi : \mathcal{X} \to \{1, \ldots, K\}$ is a classifier, $\alpha_k \in [0, 1)$, $w_k \geq 0$ and $\mathcal{A} \subseteq \{1, \ldots, K\}$ (Tian and Feng (2021)).

This package implements two NPMC algorithms, NPMC-CX and NPMC-ER, which are motivated from cost-sensitive learning. For details about them, please refer to Tian and Feng (2021). Here we just show how to call relative functions to solve NP problems by these two algorithms.

A Simple Example

We take Example 1 in Tian and Feng (2021) as an example. Consider a three-class independent Gaussian conditional distributions $X|Y = k \sim N(\boldsymbol{\mu}_k, \boldsymbol{I}_p)$, where $p = 5$, $\boldsymbol{\mu}_1 = (-1, 2, 1, 1, 1)^T$, $\boldsymbol{\mu}_2 = (1, 1, 0, 2, 0)^T$, $\boldsymbol{\mu}_3 = (2, -1, -1, 0, 0)^T$ and $\boldsymbol{I}_p$ is the p-dimensional identity matrix. The marginal distribution of Y is $\mathbb{P}(Y = 1) = \mathbb{P}(Y = 2) = 0.3$ and $\mathbb{P}(Y = 3) = 0.4$. Training sample size $n = 1000$ and test sample size is 2000.

We would like to solve the following NP problem

$$\min_{\phi} \quad \mathbb{P}_{X|Y=2}(\phi(X) \neq 2) \tag{3}$$

$$\text{s.t.} \quad \mathbb{P}_{X|Y=1}(\phi(X) \neq 1) \leq 0.05, \quad \mathbb{P}_{X|Y=3}(\phi(X) \neq 3) \leq 0.01. \tag{4}$$

Now let's first generate the training data by calling function `generate.data`. We also create variables `alpha` and `w`, representing the target level to control and the weight in the objective function for each error rate. Here the target levels for classes 1 and 3 are 0.05 and 0.01. There is no need to control error rate of class 2, therefore we set the corresponding level as `NA`. The weights for three classes are 0, 1, 0, respectively.

```r
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)
```

We first examine the test error rates of vanilla logistic regression model fitted by training data. We fit the multinomial logistic regression via function `multinom` in package `nnet`. Note that our package provides function `error_rate` to calculate the error rate per class by inputing the predicted response vector and true response vector. The results show that the vanilla logistic regression fails to control the error rates of classes 1 and 3.

```r
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)
```

```
##          1          2          3
## 0.08517888 0.16264090 0.04924242
```

Then we conduct NPMC-CX and NPMC-ER based on multinomial logistic regression by calling function `npcs`. The user can indicate which algorithm to use in parameter `algorithm`. Also note that we need to input the target level `alpha` and weight `w`. For NPMC-ER, the user can decide whether to refit the model using all training data or not by the boolean parameter `refit`. Compared to the previous results of vanilla logistic regression, it shows that both NPMC-CX-logistic and NPMC-ER-logistic can successfully control $\mathbb{P}_{X|Y=1}(\phi(X) \neq 1)$ and $\mathbb{P}_{X|Y=3}(\phi(X) \neq 3)$ around levels 0.05 and 0.01, respectively.

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

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

```
##           1           2           3
## 0.063032368 0.309178744 0.008838384
```

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

```
##          1          2          3
## 0.04940375 0.24798712 0.02020202
```

We can also run NPMC-CX and NPMC-ER with different models, for example, LDA and random forests. They can also successfully control $\mathbb{P}_{X|Y=1}(\phi(X) \neq 1)$ and $\mathbb{P}_{X|Y=3}(\phi(X) \neq 3)$ around levels 0.05 and 0.01, respectively.

```r
fit.npmc.CX.lda <- try(npcs(x, y, algorithm = "CX", classifier = "lda",
    w = w, alpha = alpha))
fit.npmc.ER.lda <- try(npcs(x, y, algorithm = "ER", classifier = "lda",
    w = w, alpha = alpha, refit = TRUE))

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

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

```
##           1           2           3
## 0.056218058 0.341384863 0.007575758
```

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

```
##          1          2          3
## 0.03577513 0.29951691 0.02020202
```

```r
# test error of NPMC-CX-RF
y.pred.CX.rf <- predict(fit.npmc.CX.rf, x.test)
error_rate(y.pred.CX.rf, y.test)
```

```
##          1          2          3
## 0.05792164 0.59259259 0.00000000
```

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

```
##          1          2          3
## 0.03066440 0.44122383 0.01767677
```

Reference

Dreiseitl, Stephan, Lucila Ohno-Machado, and Michael Binder. 2000. “Comparing Three-Class Diagnostic Tests by Three-Way Roc Analysis.” *Medical Decision Making* 20 (3): 323–31.

Mossman, Douglas. 1999. “Three-Way Rocs.” *Medical Decision Making* 19 (1): 78–89.

Tian, Ye, and Yang Feng. 2021. “Neyman-Pearson Multi-Class Classification via Cost-Sensitive Learning.” *Submitted*.