# A demonstration of the npcs package

Ye Tian, Ching-Tsung Tsai and Yang Feng

2023-04-26

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

`npcs` can be installed from CRAN.

```r
# install.packages("npcs_0.1.1.tar.gz", repos=NULL, type='source')
# 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 $\{(x_i, y_i)\}_{i=1}^n$ 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 $\phi$ 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 $X|Y = k \sim N(\mu_k, I_p)$, where $p = 5$, $\mu_1 = (-1, 2, 1, 1, 1)^T$, $\mu_2 = (1, 1, 0, 2, 0)^T$, $\mu_3 = (2, -1, -1, 0, 0)^T$ and $I_p$ 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.

```
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.

```
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.

```
fit.npmc.CX.logistic <- try(npcs(x, y, algorithm = "CX", classifier = "multinom",
    w = w, alpha = alpha))
fit.npmc.ER.logistic <- try(npcs(x, y, algorithm = "ER", classifier = "multinom",
    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
## 0.056218058 0.333333333 0.007575758
```

```
# 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
## 0.07666099 0.26731079 0.01262626
```

We can use different models to run NPMC-CX and NPMC-ER. Our framework is built on top of the caret::train function from the Caret (Classification And Regression Training) package, which enables users to implement a wide range of classification methods using the `classifier` parameter. For instance, both LDA and Gradient Boosting Machine have effectively controlled the probabilities $\mathbb{P}_{X|Y=1}(\phi(X) \neq 1)$ and $\mathbb{P}_{X|Y=3}(\phi(X) \neq 3)$ at levels of approximately 0.05 and 0.01, respectively.

```
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))
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

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

```
# 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
## 0.02555366 0.32045089 0.02398990
```

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

```
##           1           2           3
## 0.034071550 0.410628019 0.008838384
```

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

```
##           1           2           3
## 0.100511073 0.394524960 0.003787879
```

By default, the base model may perform tuning on certain tuning parameters before running the NPMC-CX and NPMC-ER algorithms. However, we can modify the set of tuning parameter candidates using the `tuneGrid` parameter and specify the resampling method using the `trControl` parameter. The `tuneGrid` parameter accepts values that are transformed into a data.frame with all possible combinations of tuning parameters and passed to the base model's `tuneGrid` parameter. Meanwhile, values in the `trControl` parameter are provided to the `caret::trainControl` function and passed to the base model's `trControl` parameter. For more information on hyperparameters in different models, please see chapters 5.5.2 and 5.5.4 in the Caret documentation.

In the below example, to implement a NPMC-CX model using knn as the base model, we specify k values as 5,7,9 and performs a five-fold cross-validation.

```
# 5-fold cross validation with tuning parameters k = 5,7,9
fit.npmc.CX.knn <- npcs(x, y, algorithm = "CX", classifier = "knn", w = w,
                        alpha = alpha,seed = 1,
                        trControl = list(method="cv", number=3),
                        tuneGrid = list(k=c(5,7,9)))
# the optimal hypterparameter is k=9
fit.npmc.CX.knn$fit
```

```
## k-Nearest Neighbors
##
## 1000 samples
##    5 predictor
##    3 classes: '1', '2', '3'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 666, 666, 668
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.8910071  0.8356724
##   7  0.8990092  0.8477220
##   9  0.9020152  0.8522763
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

```
y.pred.CX.knn <- predict(fit.npmc.CX.knn, x.test)
error_rate(y.pred.CX.knn, y.test)
```

```
##           1           2           3
## 0.040885860 0.475040258 0.006313131
```

We can use the `cv.npcs` function to automatically compare the performance of the NPMC-CX, NPMC-ER, and vanilla models through cross-validation or bootstrapping methods. K-fold cross-validation can be specified using the `fold` parameter and setting `resample` to `"cv"`. Alternatively, bootstrapping can be specified using the `fold` parameter, `partition_ratio` parameter, and setting `resample` to "boot". Here, `fold` represents the number of iterations and `partition_ratio` represents the proportion of data used for constructing the model in each iteration. Additionally, we can use the `stratified` argument to enable stratified sampling.

The `cv.npcs` function produces several outputs. First, a summary of model evaluation is generated, which includes various evaluation metrics such as accuracy, class-specific error rates, Matthews correlation coefficient, micro-F1, macro-F1, Kappa, and balanced accuracy. Class-specific error rates are displayed using a combination of violin and box plots, which can be disabled using `plotit=FALSE`. Other outputs include the training and testing error rates of the two algorithms and base model for each fold, the validation set data indices for each fold, and the arithmetic mean of the confusion matrix and sample size in the validation set.

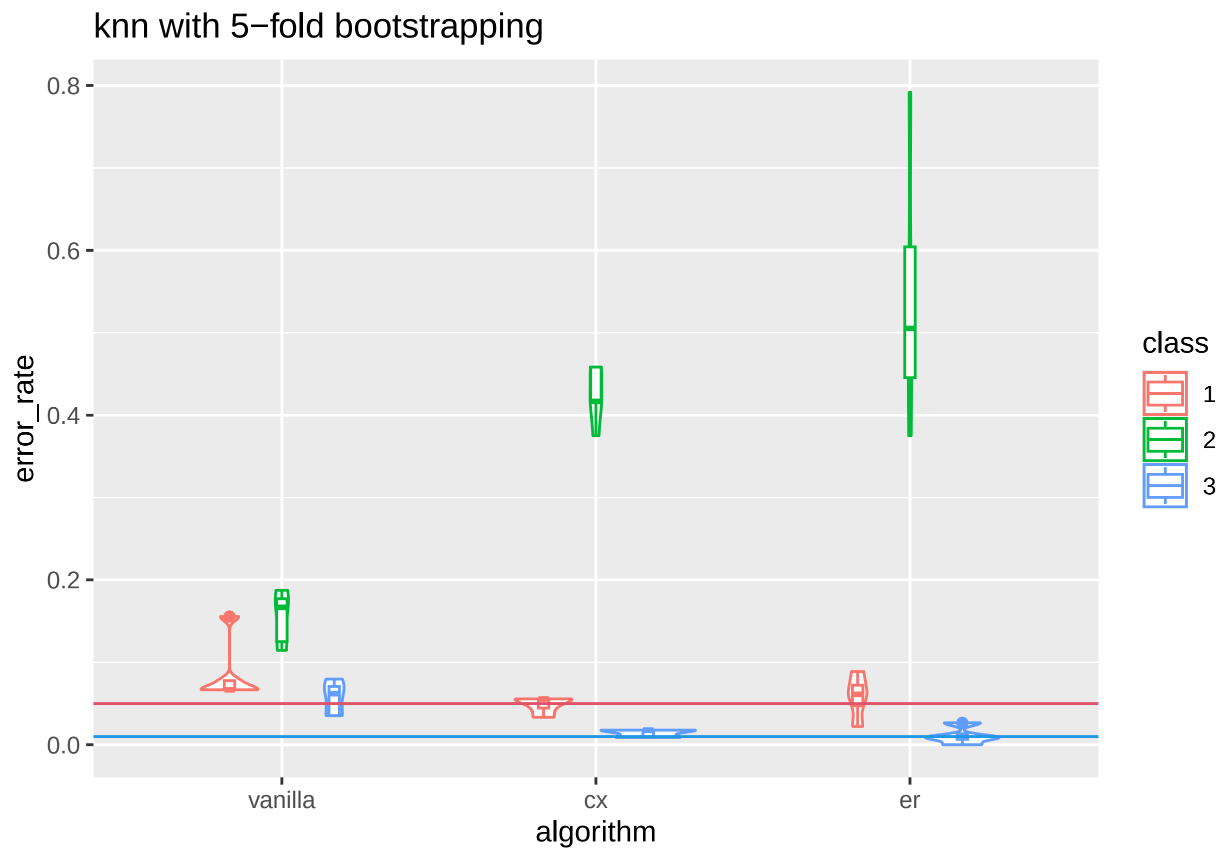
```
cv.npcs.knn <- cv.npcs(x, y, classifier = "knn", w = w, alpha = alpha,
                         # fold=5, stratified=TRUE, partition_ratio = 0.7
                         # resample=c("bootstrapping", "cv"),seed = 1,
                         # plotit=TRUE, trControl=list(), tuneGrid=list(),
     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

```
## Data splitting complete.
## Settings: stratified = TRUE , method = bootstrapping , classifier = knn
## current progress: split 5 of method knn
```

```
cv.npcs.knn$summaries
```

```
##         algorithm training_1 training_2  training_3  testing_1 testing_2
## vanilla   vanilla 0.06540284  0.1089286 0.046616541 0.08666667 0.1541667
## cx             cx 0.03317536  0.3607143 0.006015038 0.04888889 0.4250000
## er             er 0.04620853  0.4520089 0.002819549 0.05833333 0.5442708
##          testing_3 feasibility  accuracy       mcc   microF1   macroF1
## vanilla 0.05663717         1.0 0.9030100 0.8538280 0.9028717 0.9007137
## cx      0.01415929         1.0 0.8434783 0.7776806 0.8328596 0.8317244
## er      0.01106195         0.8 0.8035117 0.7244628 0.7829562 0.7803219
##             Kappa       BAC
## vanilla 0.8538119 0.9008432
## cx      0.7624099 0.8373173
## er      0.7013660 0.7954446
```

```
cv.npcs.knn$plot
```



## 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*.