A demonstration of the nproc package

Yang Feng, Jingyi Jessica Li and Xin Tong

2017-09-26

We provide a detailed demo of the usage for the `nproc` package.

Introduction

Let (X, Y) be random variables where $X \in \mathcal{X}$ is a vector of d features and $Y \in \{0, 1\}$ represents a binary class label. A data set that contains independent observations $\{(x_i, y_i)\}_{i=1}^{n+m}$ sampled from the joint distribution of (X, Y) are often divided into training data $\{(x_i, y_i)\}_{i=1}^{n}$ and test data $\{(x_i, y_i)\}_{i=n+1}^{n+m}$.

Based on training data, a *classifier* $\phi(\cdot)$ is a mapping $\phi : \mathcal{X} \to \{0, 1\}$ that returns the predicted class label given X. Classification error occurs when $\phi(X) \neq Y$, and the binary loss is defined as $1(\phi(X) \neq Y)$, where $1(\cdot)$ denotes the indicator function. The risk is defined as $R(\phi) = E[1(\phi(X) \neq Y)] = P(\phi(X) \neq Y)$, which can be expressed as a weighted sum of type I and II errors: $R(\phi) = P(Y = 0)R_0(\phi) + P(Y = 1)R_1(\phi)$, where $R_0(\phi) = P(\phi(X) \neq Y|Y = 0)$ denotes the type I error, and $R_1(\phi) = P(\phi(X) \neq Y|Y = 1)$ denotes the type II error.

The classical classification paradigm aims to find an oracle classifier ϕ^* by minimizing the risk, $\phi^* = \arg\min_\phi R(\phi)$.

In contrast, the NP classification paradigm aims to mimic the NP oracle classifier ϕ_α^* with respect to a pre-specified type I error upper bound α, $\phi_\alpha^* = \arg\min_{\phi:\ R_0(\phi) \leq \alpha} R_1(\phi)$, where α reflects users' conservative attitude (priority) towards the type I error. In practice, since we do not know the class distirbution, we do not expect to enforce that the type I error is bounded from above strictly; instead, we wish the type I error is bounded by α with probability at least $1 - \delta$, where $\delta \in (0, 1)$ is another user-specified hyper-parameter.

The `nproc` package provides two essential tools for NP classification: i). a function `npc` that adapts popular classification algorithms, such as linear discriminant analysis, logistic regression, support vector machines and random forrests to the NP paradigm; and ii) a function `nproc` that constructs NP-ROC bands, an alternative to the popular ROC curves, that allows visualization and comparison of NP classifiers.

Installation and Package Loading

The simplest way to obtain the `nproc` package is to install it directly from CRAN. Type the following command in R console:

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

Users may change the `repos` options depending on their locations and preferences. Other options such as the directories to which we want to install the packages can be altered in the command. For more details, see `help(install.packages)`.

Here the `nproc` package has been downloaded and installed to the default directories.

Alternatively, users can download the package source at http://cran.r-project.org/web/packages/nproc/index.html and type Unix commands to install it to the desired location.

Then we can load the `nproc` package:

```
library(nproc)
```

Neyman-Pearson Classifier

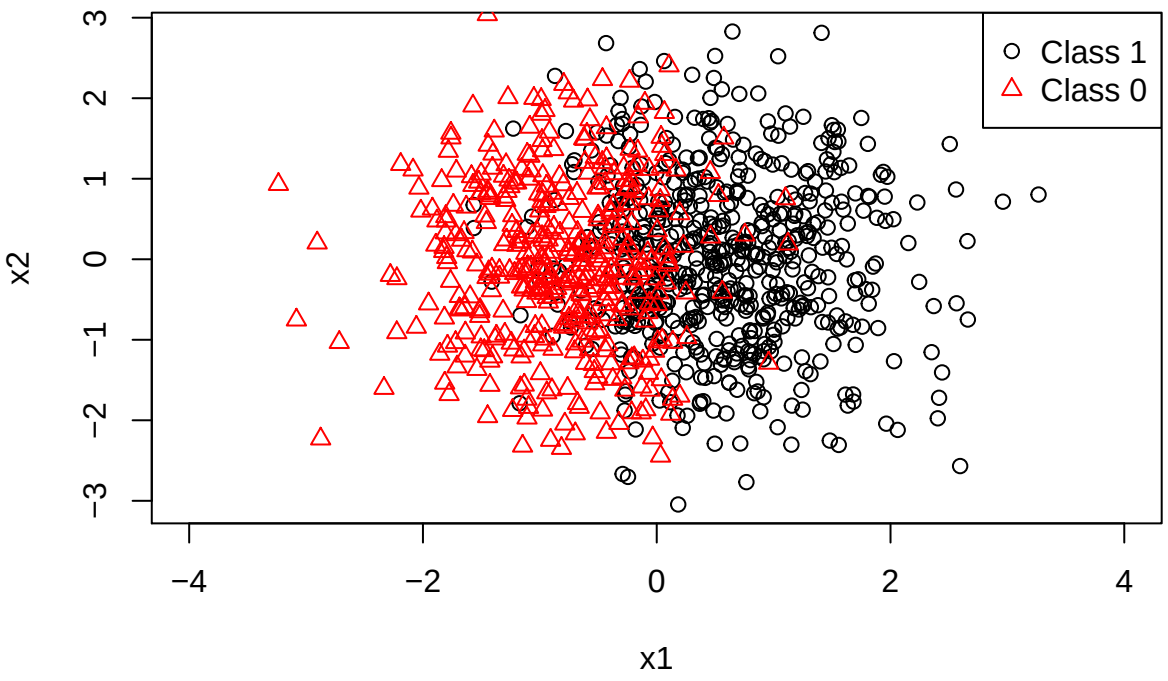
Here, we demonstrate how to construct a Neyman-Pearson classifier when a user has priority to control the type I error. Note that a user can also control the type II error under specific levels by switching the class labels and applying the npc function.

In the first step, we create a dataset (x,y) from a logistic regression model with 2 features and sample size 1000.

```
n = 1000
set.seed(0)
x = matrix(rnorm(n*2),n,2)
c = 1+3*x[,1]
y = rbinom(n,1,1/(1+exp(-c)))
```

A visualization of the two classes.

```
plot(x[y==1,],col=1,xlim=c(-4,4),xlab='x1',ylab='x2')
points(x[y==0,],col=2,pch=2)
legend("topright",legend=c('Class 1','Class 0'),col=1:2,pch=c(1,2))
```



Then, we call the `npc` function to construct Neyman-Pearson classifiers. Suppose we want to use linear discriminant analysis, we set `method` = "lda". The default type I error upper bound is `alpha=0.05`, while the `alpha` value can be changed to any user specified value in $(0, 1)$. The tolerance upper bound δ also has the default value at 0.05.

```
fit = npc(x, y, method = "lda", alpha = 0.05)
```

We can now evaluate the prediction performance of the NP classifier `fit` on a test set (xtest, ytest) generated as follows.

```
xtest = matrix(rnorm(n*2),n,2)
ctest = 1+3*xtest[,1]
ytest = rbinom(n,1,1/(1+exp(-ctest)))
```

We calculate the overall accuracy of the classifier as well as the realized type I error on test data. It is shown that the type I error is smaller than `alpha`.Strictly speaking, to demonstrate the effectiveness of the `fit` classifier under the NP paradigm, we should repeat this experiment many times, and show that in $1-\delta$ of these repetitions, type I error (approximated by empirical type I error on a large test data set) is smaller than `alpha`.

```
pred = predict(fit,xtest)
fit.score = predict(fit,x)
accuracy = mean(pred$pred.label==ytest)
cat("Overall Accuracy: ",  accuracy,'\n')
```

```
## Overall Accuracy:  0.682
```

```
ind0 = which(ytest==0)
typeI = mean(pred$pred.label[ind0]!=ytest[ind0]) #type I error on test set
cat('Type I error: ', typeI, '\n')
```

```
## Type I error:  0.02631579
```

The base classification method implemented in the `npc` function includes the following options.

- logistic: Logistic regression. `glm` function with family = 'binomial'
- penlog: Penalized logistic regression with LASSO penalty. `glmnet` in `glmnet` package
- svm: Support Vector Machines. `svm` in `e1071` package
- randomforest: Random Forest. `randomForest` in `randomForest` package
- lda: Linear Discriminant Analysis. `lda` in `MASS` package
- nb: Naive Bayes. `naiveBayes` in `e1071` package
- ada: Ada-Boost. `ada` in `ada` package

Now, we can change the classification method of choice to logistic regression and change `alpha` to 0.1.

```
fit = npc(x, y, method = "logistic", alpha = 0.1)
pred = predict(fit,xtest)
accuracy = mean(pred$pred.label == ytest)
cat("Overall Accuracy: ",  accuracy,'\n')
```

```
## Overall Accuracy:  0.769
```

```
ind0 = which(ytest == 0)
typeI = mean(pred$pred.label[ind0] != ytest[ind0]) #type I error on test set
cat('Type I error: ', typeI, '\n')
```

```
## Type I error:  0.09473684
```

To construct each NP classifier, `npc` function randomly splits class 0 sample into two equal halves; the first half, together with the class 1 sample, is used to train the base classification method, and the second half is reserved to find the threshold on the scores. To increase stability, the `nproc` package provides implementation of ensembled classifiers, which are majority votes of the NP classifiers constructed from different sample splitting. The parameter value `split` is the number of random splits. In the following codes, we change its number to 11.

```
fit = npc(x, y, method = "logistic", alpha = 0.1, split = 11)
pred = predict(fit,xtest)
```

```
accuracy = mean(pred$pred.label == ytest)
cat("Overall Accuracy: ",  accuracy,'\n')
```

```
## Overall Accuracy:  0.77
```

```
ind0 = which(ytest == 0)
typeI = mean(pred$pred.label[ind0] != ytest[ind0]) #type I error on test set
cat('Type I error: ', typeI, '\n')
```

```
## Type I error:  0.09473684
```

Let's compare the performance of different methods on our simulated dataset.

```
methodlist = c("logistic", "penlog", "randomforest", "svm",
                                        "lda", "nb", "ada")
for(method in methodlist){
fit = npc(x, y, method = method, alpha = 0.05)
pred = predict(fit,xtest)
accuracy = mean(pred$pred.label==ytest)
cat(method, ': Overall Accuracy: ',  accuracy,'\n')
ind0 = which(ytest==0)
typeI = mean(pred$pred.label[ind0]!=ytest[ind0]) #type I error on test set
cat(method, ': Type I error: ', typeI, '\n')
}
```

```
## logistic : Overall Accuracy:  0.681
## logistic : Type I error:  0.02631579
## penlog : Overall Accuracy:  0.682
## penlog : Type I error:  0.02631579
## randomforest : Overall Accuracy:  0.58
## randomforest : Type I error:  0.02105263
## svm : Overall Accuracy:  0.505
## svm : Type I error:  0.06052632
## lda : Overall Accuracy:  0.682
## lda : Type I error:  0.02631579
## nb : Overall Accuracy:  0.685
## nb : Type I error:  0.02631579
## ada : Overall Accuracy:  0.72
## ada : Type I error:  0.05
```

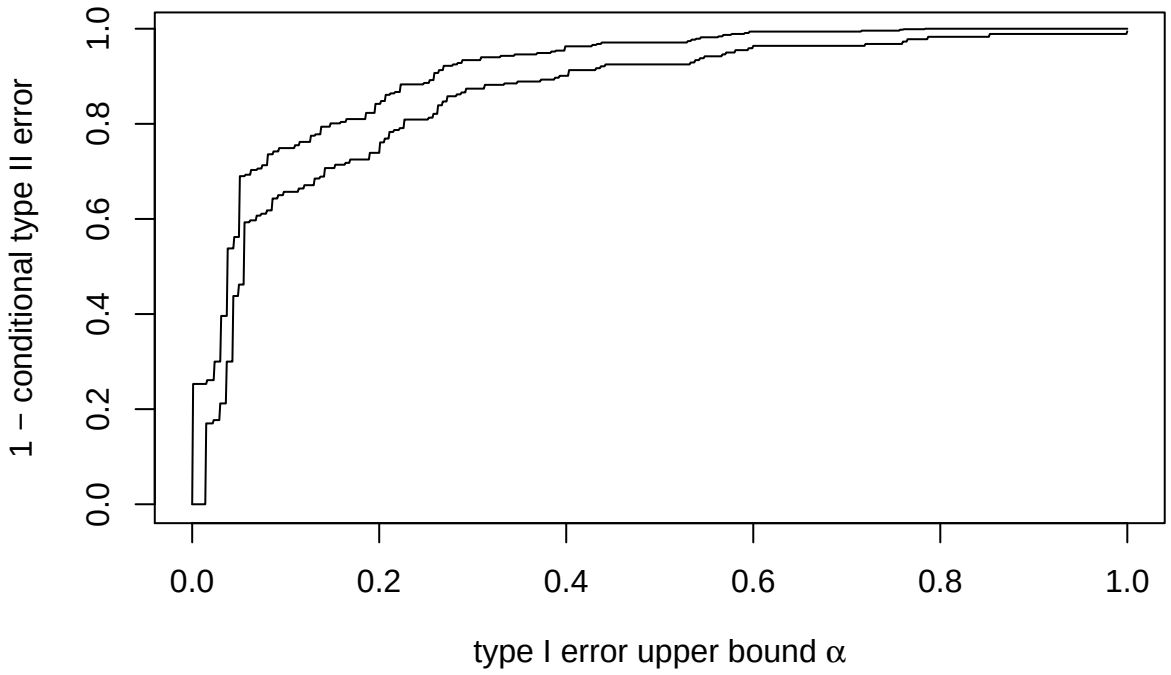
Back to Top

Neyman-Pearson Receiver Operator Characteristic (NP-ROC) Band

The function `nproc` constructs Neyman-Pearson Receiver Operator Characteristic (NP-ROC) band for an NP classifier. When we plot an NP-ROC band, the horizontal axis is the `alpha` value, and the vertical axis is the type II error conditioning on training data. When we section the band vertically at each `alpha` value, the lower point on the segment is a high probability lower bound of the (1-type II error) of the NP classifier corresponding to that `alpha` value, and the the upper point on the segment is a high probability upper bound of the (1-type II error) of the NP classifier corresponding to that `alpha` value.

In the following demo, we use the same data as in the previous section.

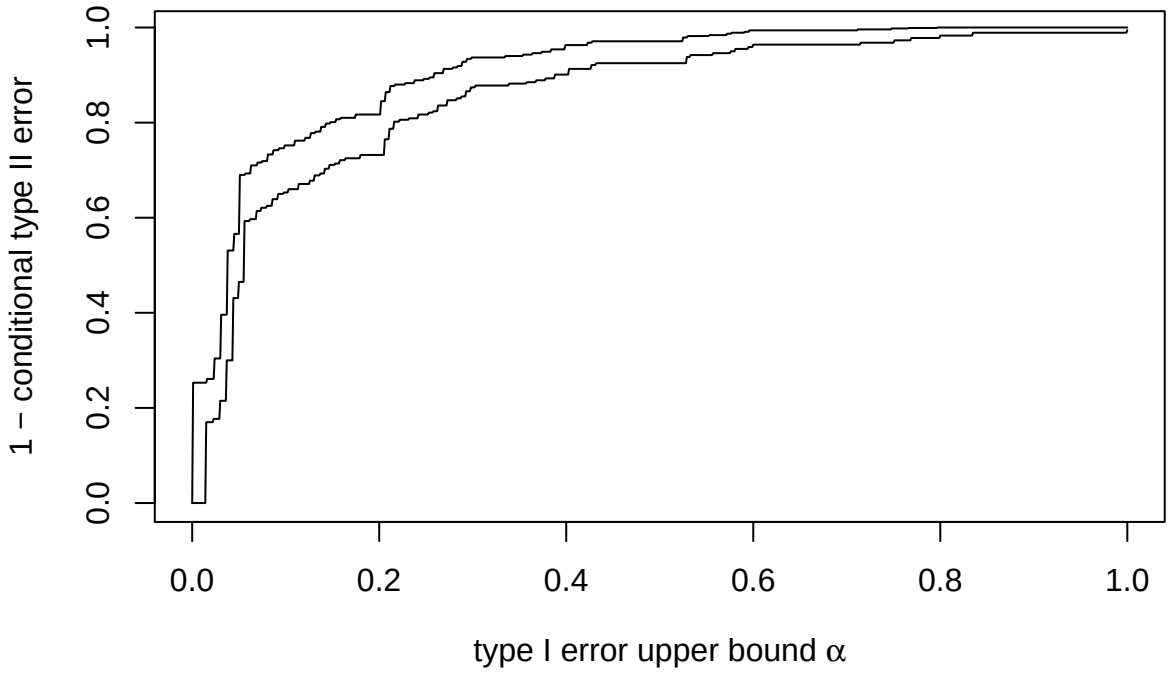
```
fit = nproc(x, y, method = "nb")
plot(fit)
```



Note that, to create an NP-ROC band, there is no need to construct an NP classifier using the `npc` function first.

When we choose linear discriminant analysis as the base method, we do as follows.

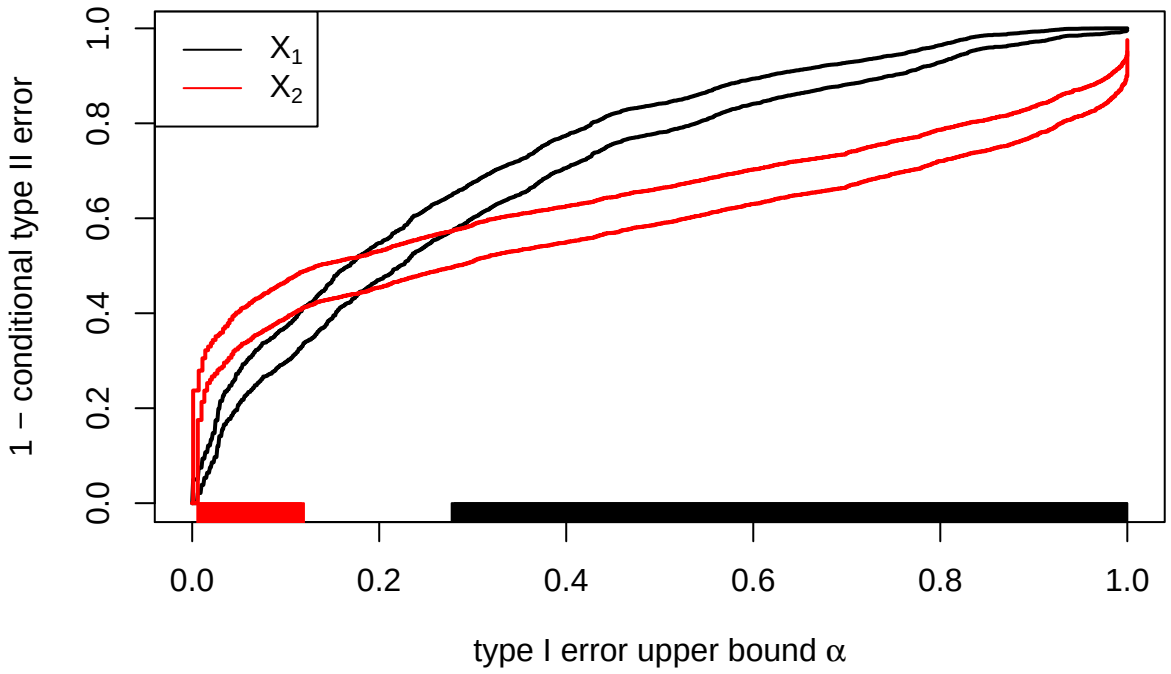
```
fit2 = nproc(x, y, method = "lda")
plot(fit2)
```



The `compare` function compares two NP classifiers at different α values (their δ values have to be the same) by inespecting their NP-ROC bands. The following code displayes the NP-ROC bands generated by logistic regression with the two predictors. The region marked in red is the `alpha` range in which the second NP classifier dominates the first, and the black region means otherwise. The uncolored region means we cannot

claim supriority of one or the other with confidence.

```
n = 1000
set.seed(0)
x1 = c(rnorm(n), rnorm(n) + 1)
x2 = c(rnorm(n), rnorm(n)*sqrt(6) + 1)
y = c(rep(0,n), rep(1,n))
fit1 = nproc(x1, y, split = 11, method = 'lda')
fit2 = nproc(x2, y, split = 11, method = 'lda')
v = compare(fit1, fit2)
legend('topleft',legend = c(expression(X[1]),expression(X[2])),col = 1:2,lty = c(1,1))
```



Lastly, we have implemented a web-based interface using the shiny app. It can be invoked by the following command after the package is loaded.

```
shiny::runApp(system.file('nproc', package='nproc'))
```

Back to Top