Cross-Validation with FRESA.CAD

José Tamez-Peña

Aug 15, 2021

Contents

1 Simple Cross-Validation of Common ML Methods

This tutorial will guide users on how to use FRESA.CAD to evaluate the performance of binary classifiers.

1.1 The required libraries

```
library("FRESA.CAD")
library("mlbench")
library("fastAdaboost")
library("gbm")
```

1.2 PimaIndiansDiabetes Data Set

I will use the PimaIndiansDiabetes2 data set from the mlbech package.

```
data("PimaIndiansDiabetes2", package = "mlbench")
```

We have to condition the data set.
FRESA.CAD cross-validation requires a data frame with complete cases. Also, the outcome has to be numeric.
* 0 for Controls, and
* 1 for Cases

```
PimaIndiansDiabetes  <- PimaIndiansDiabetes2[complete.cases(PimaIndiansDiabetes2),]
PimaIndiansDiabetes$diabetes <- 1*(PimaIndiansDiabetes$diabetes == "pos")
```

1.3 Gradient boosting from the gbm package

The cross-validation with FRESA.CAD can be done on any R function that fits binary outcomes.
The requirement is that model fit has to done as:

```
>model <- fit(formula,data),
```

and the predict must be called as:

```
>pre <- predict(model,testdata)
```

If the fitting function does not conform to the requirements, you can always create a wrapper.
Here we will show how to create a wrapper to the gradient boost method of the gbm package.

The following code shows the gbm wrapper function:

```
GBM_fit <- function(formula = formula, data=NULL, distribution = "bernoulli", n.trees = 1000,
                  shrinkage = 0.01, interaction.depth = 4, ...)
{
  fit <- gbm(formula = formula,data = data,distribution = distribution,n.trees = n.trees,
                  shrinkage = shrinkage, interaction.depth = interaction.depth,...);
  selectedfeatures <- summary(fit,plotit = FALSE);
  sum <- 0;
  sfeat = 1;
  while (sum < 90) {
    sum <- sum + selectedfeatures[sfeat,2];
  sfeat <- sfeat + 1;
  } #keep the ones that add to 90%

    result <- list(fit = fit,n.trees = n.trees,
                   selectedfeatures = rownames(selectedfeatures[1:sfeat,]))
    class(result) <- "GBM_FIT";
    return(result)
}
```

We also need a proper predict function for the boosting algorithm:

```
predict.GBM_FIT <- function(object,...)
{
        parameters <- list(...);
        testData <- parameters[[1]];
        n.trees = seq(from = (0.1*object$n.trees),
                      to = object$n.trees,
                      by = object$n.trees/25) #no of trees-a vector of 25 values
        pLS <- predict(object$fit,testData,n.trees = n.trees);
        pLS <- 1.0/(1.0 + exp(-apply(pLS,1,mean)))
        return(pLS);
}
```

Let me check that fitting and prediction functions are working:

```
gfit <- GBM_fit(formula = diabetes ~ .,PimaIndiansDiabetes)
pr <- predict(gfit,PimaIndiansDiabetes)
pander::pander(table(pr > 0.5,PimaIndiansDiabetes$diabetes),
               caption="Training: Gradient Boost Confusion Matrix")
```

Table 1: Training: Gradient Boost Confusion Matrix

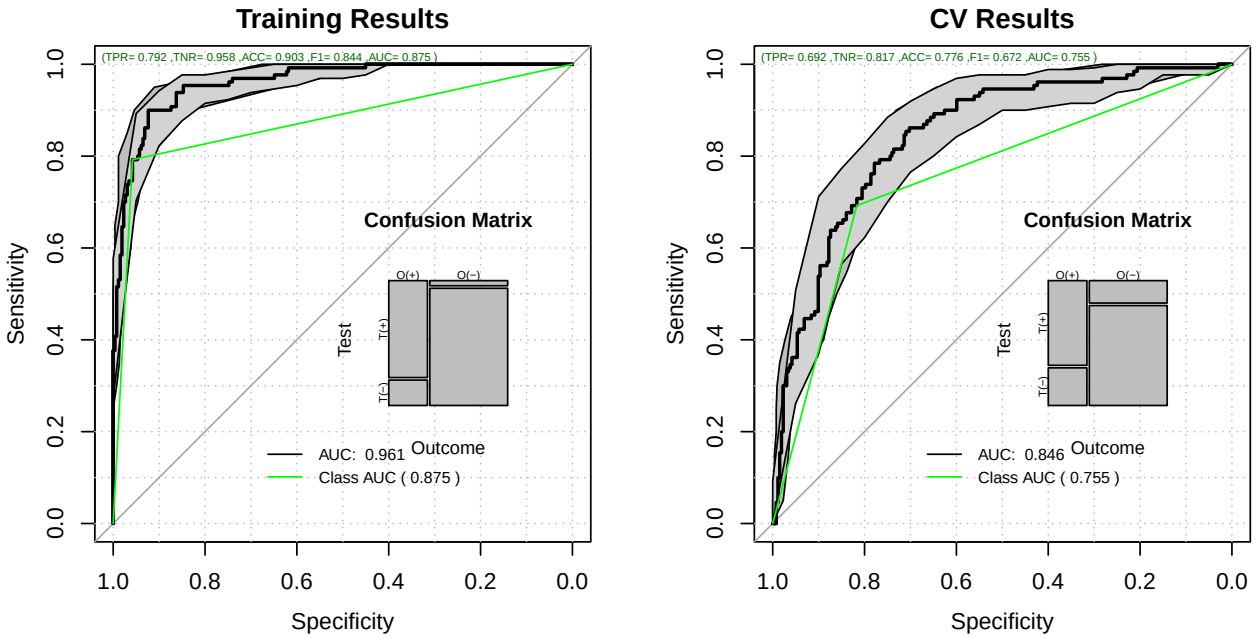
	0	1
FALSE	251	27
TRUE	11	103

Now I can check the test ensembles performance of the gradient boosting method.
The following code shows five alternatives for the cross-validation.

```
op <- par(no.readonly = TRUE)

GradBOOSTcv <- randomCV(PimaIndiansDiabetes,
                        "diabetes",
                        GBM_fit,
                        trainFraction = 0.5,
                        repetitions = 100
                        )
```

I'll use the `plotModels.ROC()` function to plot the ROC curves

```
par(mfrow = c(1,2),cex = 0.5);
pmr1 <- plotModels.ROC(cbind(PimaIndiansDiabetes$diabetes,pr),
                       main="Training Results",cex = 0.8)
pmr2 <- plotModels.ROC(GradBOOSTcv$medianTest,
                       main="CV Results",cex = 0.8)
```




```
par(mfrow = c(1,1),cex = 1.0);
```

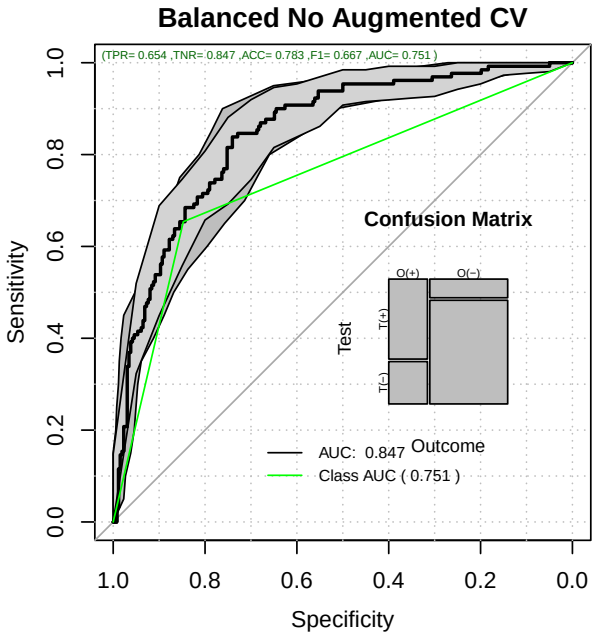
FRESA.CAD provides different alternatives for selecting the training sample inside the Cross-validation. The default setting uses a balanced scheme that randomly add samples from the under represented class (`classSamplingType = "Augmented"`). Other options are class-proportional (`classSamplingType = "Proportional"`), Balanced (`classSamplingType = "NoAugmented"`), and Leave One Out per class (`classSamplingType = "LOO"`). Bootstrap (`trainFraction = "Bootstrap"`) sampling can be used on all the sampling schemes.

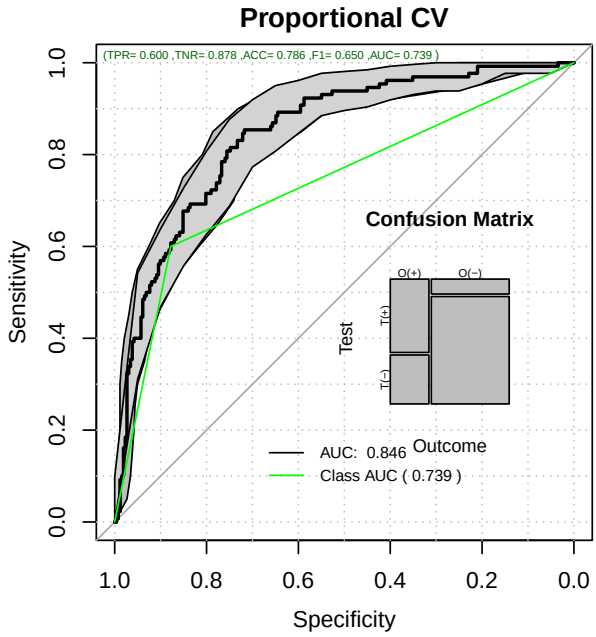
```
GradBOOST_NoAugmentedcv <- randomCV(PimaIndiansDiabetes,
                                    "diabetes",
                                    GBM_fit,
                                    trainFraction = 0.5,
                                    repetitions = 100,
                                    classSamplingType = "NoAugmented"
```

```
                                    )

GradBOOST_Proportionaldcv <- randomCV(PimaIndiansDiabetes,
                                      "diabetes",
                                      GBM_fit,
                                      trainFraction = 0.5,
                                      repetitions = 100,
                                      classSamplingType = "Proportional"
                                      )

GradBOOST_ProportionalBootstrapcv <- randomCV(PimaIndiansDiabetes,
                                              "diabetes",
                                              GBM_fit,
                                              trainFraction = "Bootstrap",
                                              repetitions = 100,
                                              classSamplingType = "Proportional"
                                              )

GradBOOST_NoAugmentedBootstrapcv <- randomCV(PimaIndiansDiabetes,
                                             "diabetes",
                                             GBM_fit,
                                             trainFraction = 0.5,
                                             repetitions = 100,
                                             classSamplingType = "NoAugmented"
                                             )

GradBOOST_LOOcv <- randomCV(PimaIndiansDiabetes,
                                             "diabetes",
                                             GBM_fit,
                                             repetitions = 100,
                                             classSamplingType = "LOO"
                                             )
```

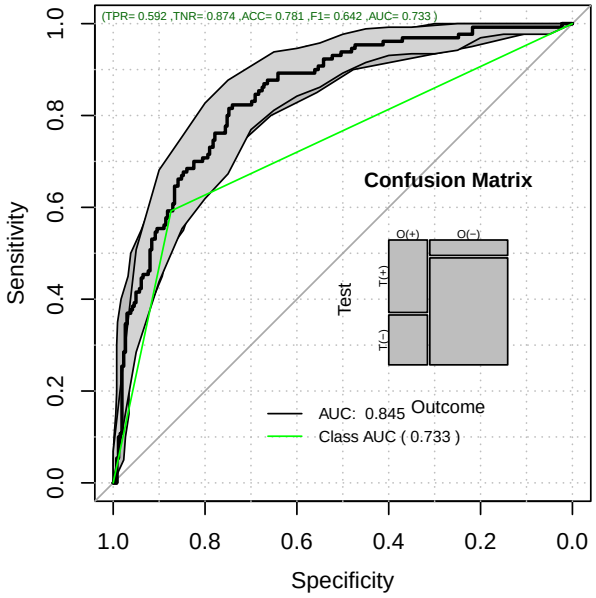
Once cross-validated, the performance results can be analyzed and plotted using the `predictionStats_binary()` function.

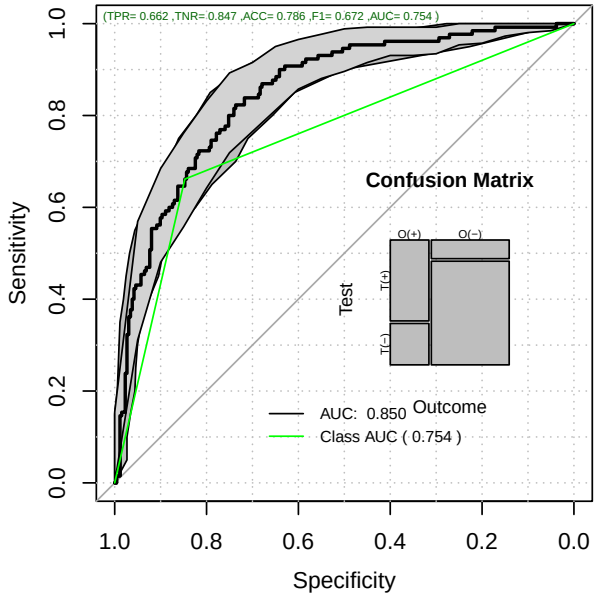
```
par(mfrow = c(1,2),cex = 0.5);
bs1 <- predictionStats_binary(cbind(PimaIndiansDiabetes$diabetes,pr)) #No plotting
bs2 <- predictionStats_binary(GradBOOSTcv$medianTest,cex = 0.8) #No plotting
bs3 <- predictionStats_binary(GradBOOST_NoAugmentedcv$medianTest,
                              "Balanced No Augmented CV",cex = 0.8)
bs4 <- predictionStats_binary(GradBOOST_Proportionaldcv$medianTest,
                              "Proportional CV",cex = 0.8)
```



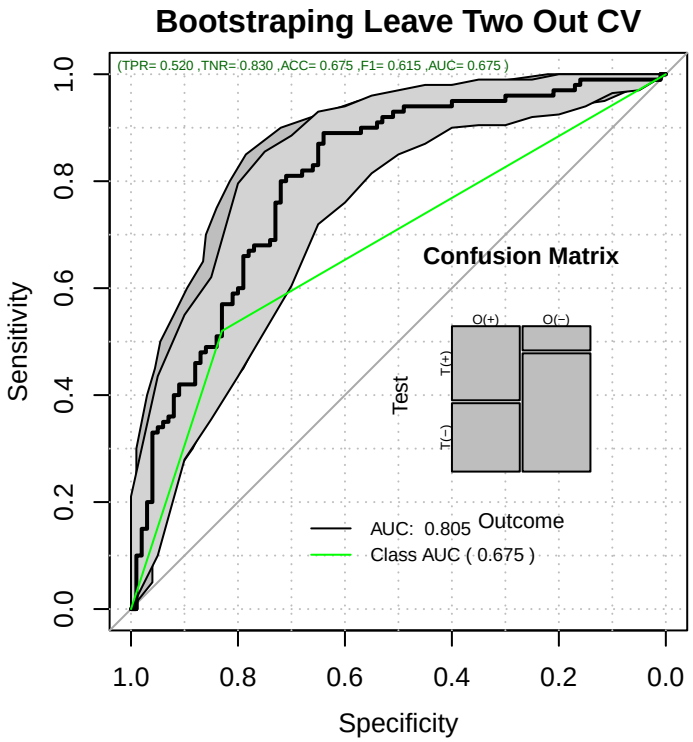



```
bs5 <- predictionStats_binary(GradBOOST_ProportionalBootstrapcv$medianTest,
                              "Bootstraping Proportional CV",cex = 0.8)
bs6 <- predictionStats_binary(GradBOOST_NoAugmentedBootstrapcv$medianTest,
                              "Bootstraping Balanced CV",cex = 0.8)
```






```
bs7 <- predictionStats_binary(GradBOOST_LOOcv$medianTest,
                              "Bootstraping Leave Two Out CV",cex = 0.8)
par(mfrow = c(1,1),cex = 1.0);
```



The output of the predictionStats_binary() function provides key performance metrics with their corresponding 95% confidence intervals

```
pander::pander(bs2$accc,caption = "Accuracy")
```

Table 2: Accuracy

est	lower	upper
0.7755	0.7309	0.8159

```
pander::pander(bs2$berror,caption = "Balanced Error")
```

50%	2.5%	97.5%
0.2454	0.2009	0.2894

```
pander::pander(bs2$aucs,caption = "AUC")
```

est	lower	upper
0.846	0.8057	0.8863

```
pander::pander(bs2$sensitivity,caption = "Sensitivity")
```

Table 5: Sensitivity

est	lower	upper
0.6923	0.6054	0.7702

```
pander::pander(bs2$specificity,caption = "Specificity")
```

Table 6: Specificity

est	lower	upper
0.8168	0.7645	0.8617

```
pander::pander(bs2$ClassMetrics,caption = "All Metrics")
```

- **accci**:

50%	2.5%	97.5%
0.7755	0.7372	0.8163

- **senci**:

50%	2.5%	97.5%
0.7546	0.7106	0.7991

- **aucci**:

50%	2.5%	97.5%
0.7546	0.7106	0.7991

- **berci**:

50%	2.5%	97.5%
0.2454	0.2009	0.2894

- **preci**:

50%	2.5%	97.5%
0.7482	0.7036	0.7919

- **F1ci**:

50%	2.5%	97.5%
0.7505	0.7065	0.7931

1.4 Cross-Validation of common ML-Methods

Now I will compare the performance to other **R** methods that already have a handy fit and predict methods.

```
ADABOOSTcv <- randomCV(fittingFunction = fastAdaboost::adaboost,
                  trainSampleSets = GradBOOSTcv$trainSamplesSets,
                  asFactor = TRUE,
                  nIter=10)

QDAcv <- randomCV(fittingFunction = MASS::qda,
                  trainSampleSets = GradBOOSTcv$trainSamplesSets,
                  method = "mve")


LDAcv <- randomCV(fittingFunction = MASS::lda,
                  trainSampleSets = GradBOOSTcv$trainSamplesSets)



logisticCV <- randomCV(fittingFunction = glm,
                  trainSampleSets = GradBOOSTcv$trainSamplesSets,
                  family="binomial")
```

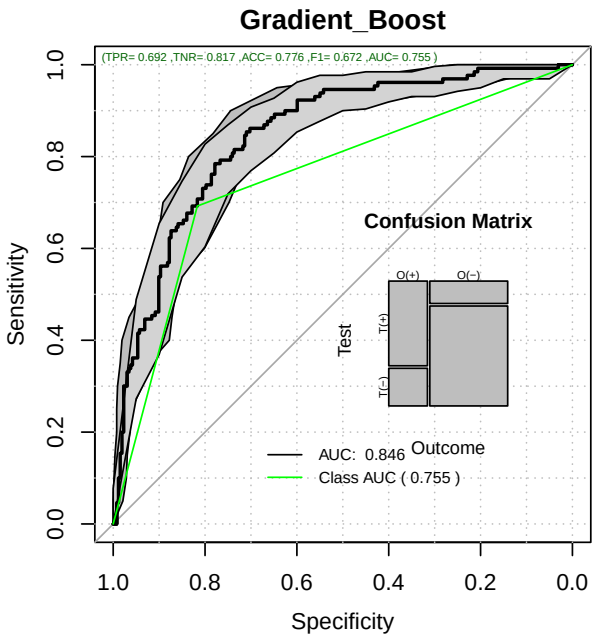
1.5 FRESA.CAD::BinaryBenchmark and Comparing Methods

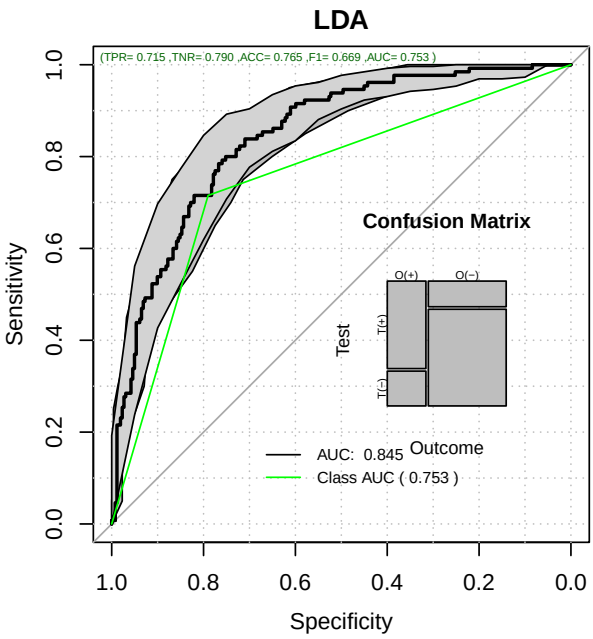
Once all the cross-validation have been completed, we can compare their performance to five common ML methods:

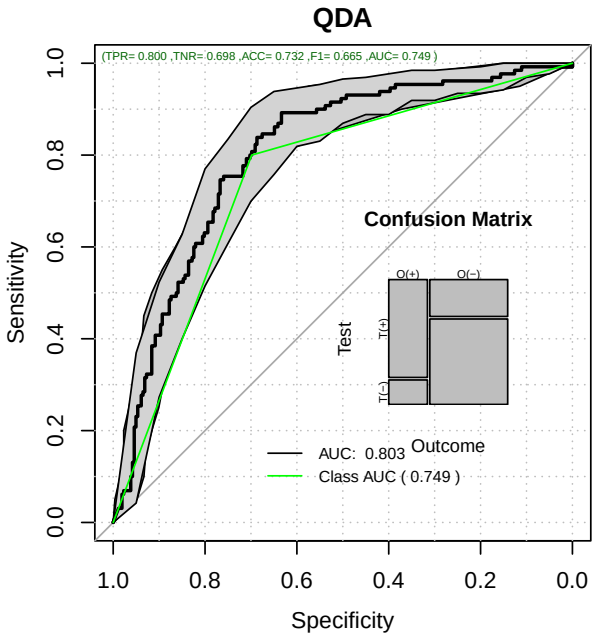
- KNN,
- Random Forest,
- RPART,
- SVM, and
- LASSO

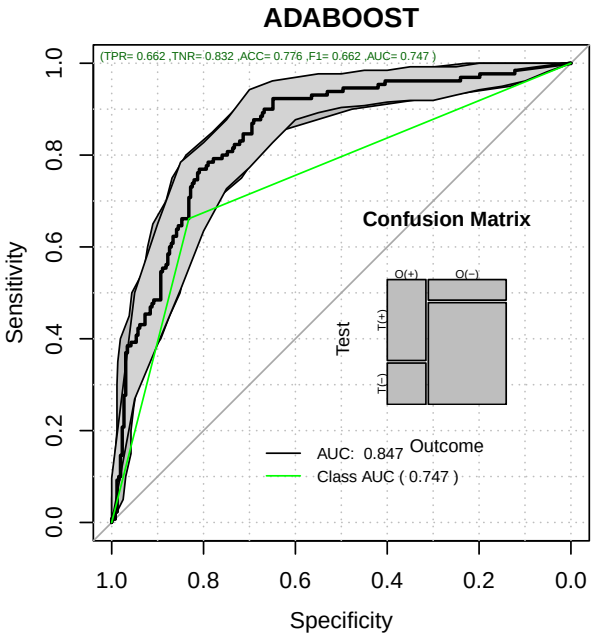
These methods are fitted using their default parameters inside the BinaryBenchmark function:

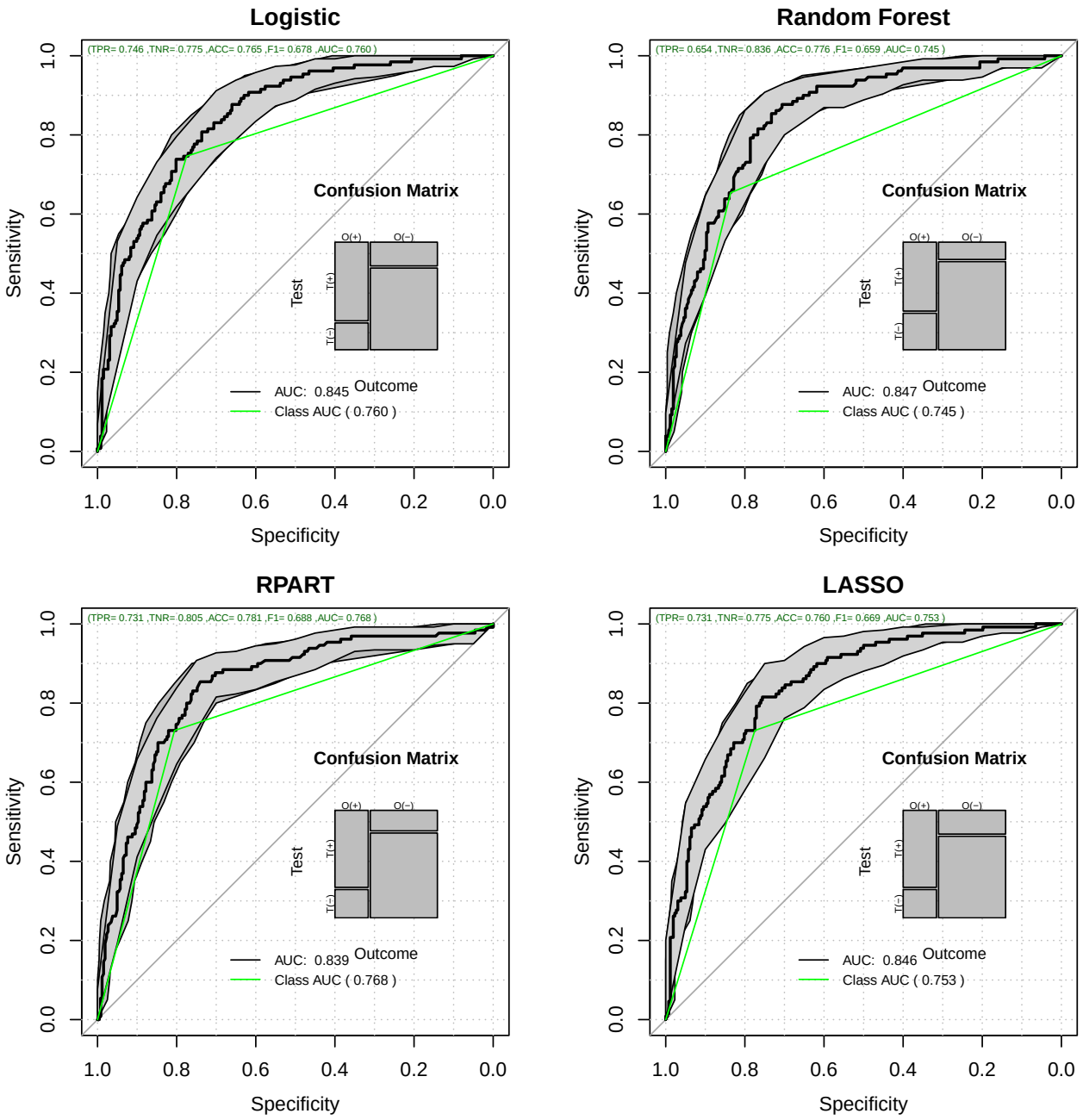
```
par(op);

par(mfrow = c(2,2),cex = 0.6);

cp <- BinaryBenchmark(referenceCV = list(Gradient_Boost = GradBOOSTcv,
                                         LDA = LDAcv,
                                         QDA = QDAcv,
                                         ADABOOST = ADABOOSTcv,
                                         Logistic = logisticCV))
```


Gradient_Boost
(TPR= 0.692 ,TNR= 0.817 ,ACC= 0.776 ,F1= 0.672 ,AUC= 0.755)
Confusion Matrix
O(+)
O(−)
T(+)
T(−)
Test
Outcome
AUC: 0.846
Class AUC (0.755)
Sensitivity
Specificity
1.0
0.8
0.6
0.4
0.2
0.0

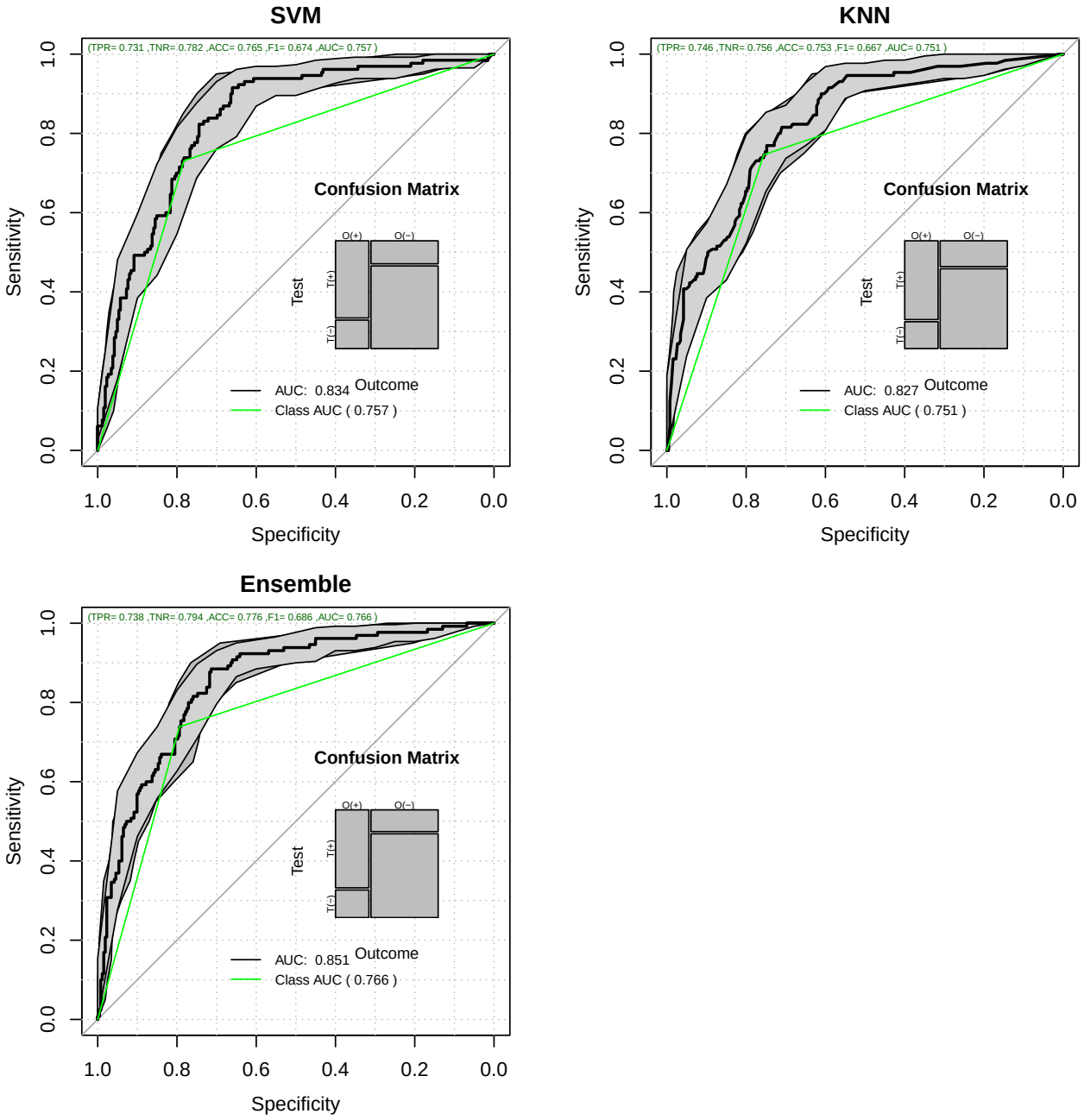

LDA
(TPR= 0.715 ,TNR= 0.790 ,ACC= 0.765 ,F1= 0.669 ,AUC= 0.753)
Confusion Matrix
O(+)
O(−)
T(+)
T(−)
Test
Outcome
AUC: 0.845
Class AUC (0.753)
Sensitivity
Specificity
1.0
0.8
0.6
0.4
0.2
0.0


QDA
(TPR= 0.800 ,TNR= 0.698 ,ACC= 0.732 ,F1= 0.665 ,AUC= 0.749)
Confusion Matrix
O(+)
O(−)
T(+)
T(−)
Test
Outcome
AUC: 0.803
Class AUC (0.749)
Sensitivity
Specificity
1.0
0.8
0.6
0.4
0.2
0.0


ADABOOST
(TPR= 0.662 ,TNR= 0.832 ,ACC= 0.776 ,F1= 0.662 ,AUC= 0.747)
Confusion Matrix
O(+)
O(−)
T(+)
T(−)
Test
Outcome
AUC: 0.847
Class AUC (0.747)
Sensitivity
Specificity
1.0
0.8
0.6
0.4
0.2
0.0




```
par(mfrow = c(1,1),cex = 1.0);
```



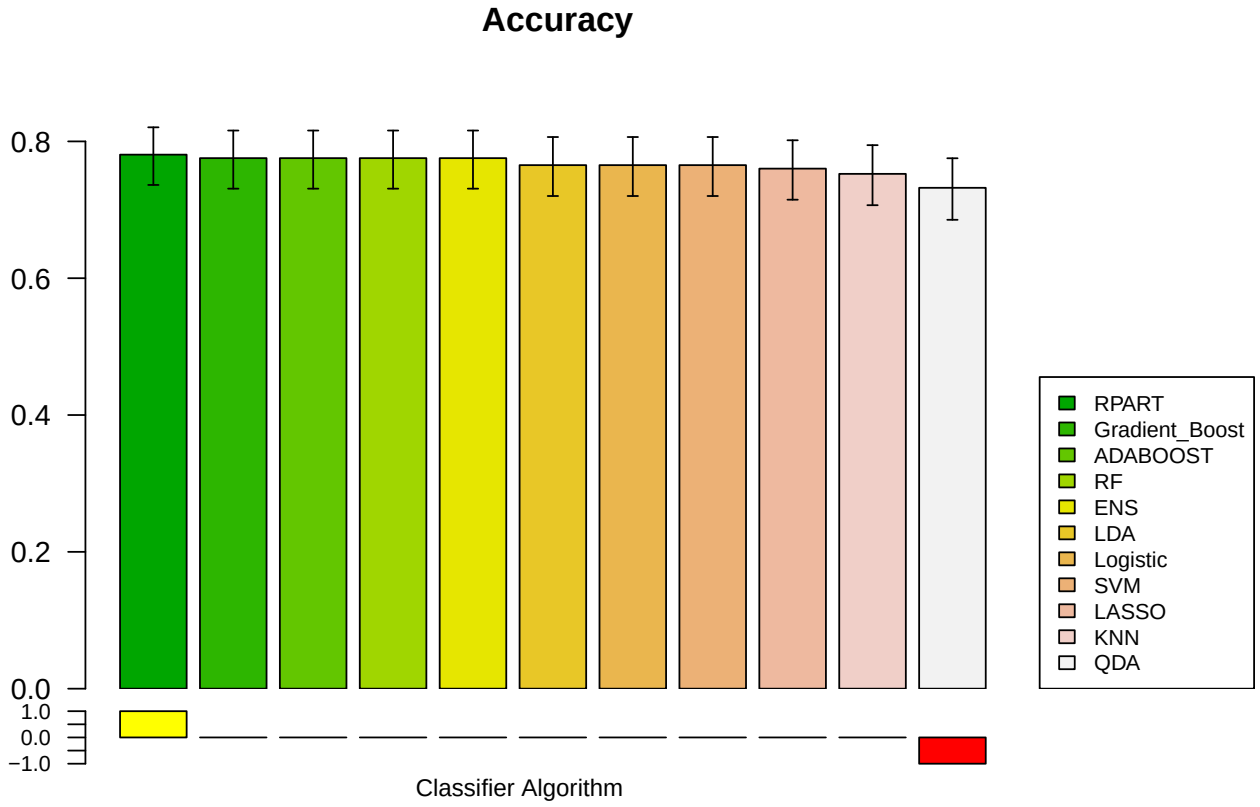
1.6 Reporting the results of the Benchmark procedure

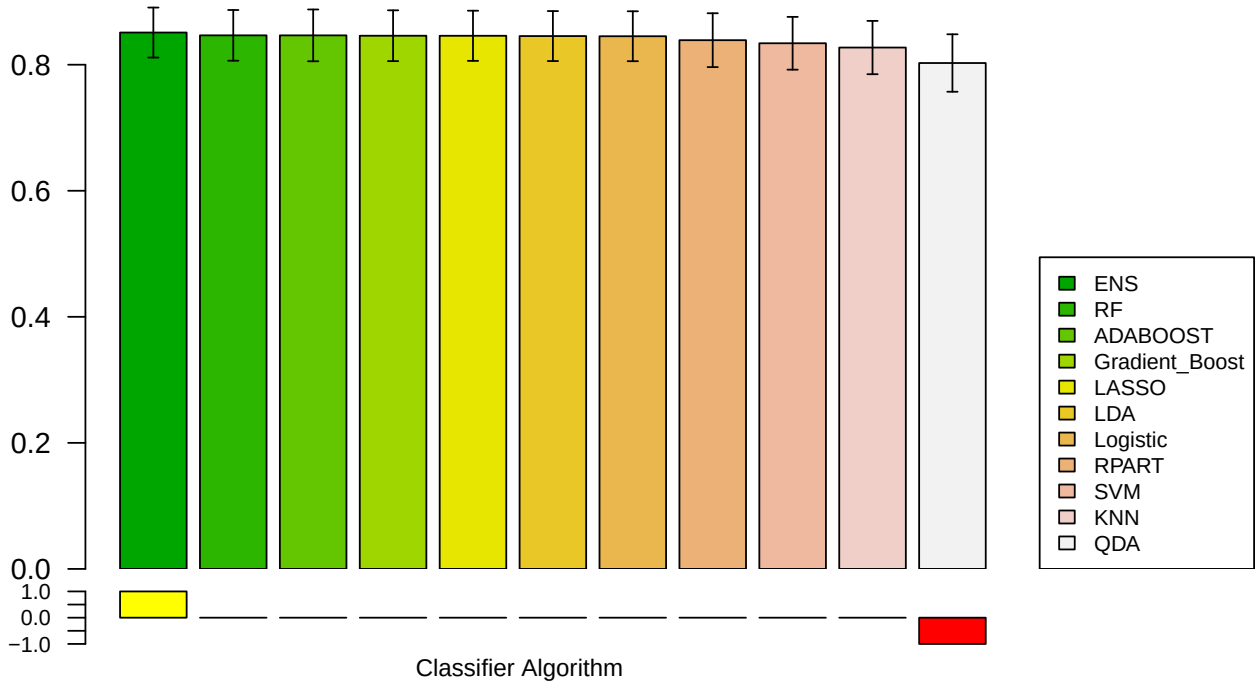
Once done, we can compare the CV test results using the plot() function.
The plot function creates bar plots that compare the balanced error rata, the accuracy, the sensitivity, the specificity, the area under the curve, as well as the report of the concordance index of the individual cross-validation runs.

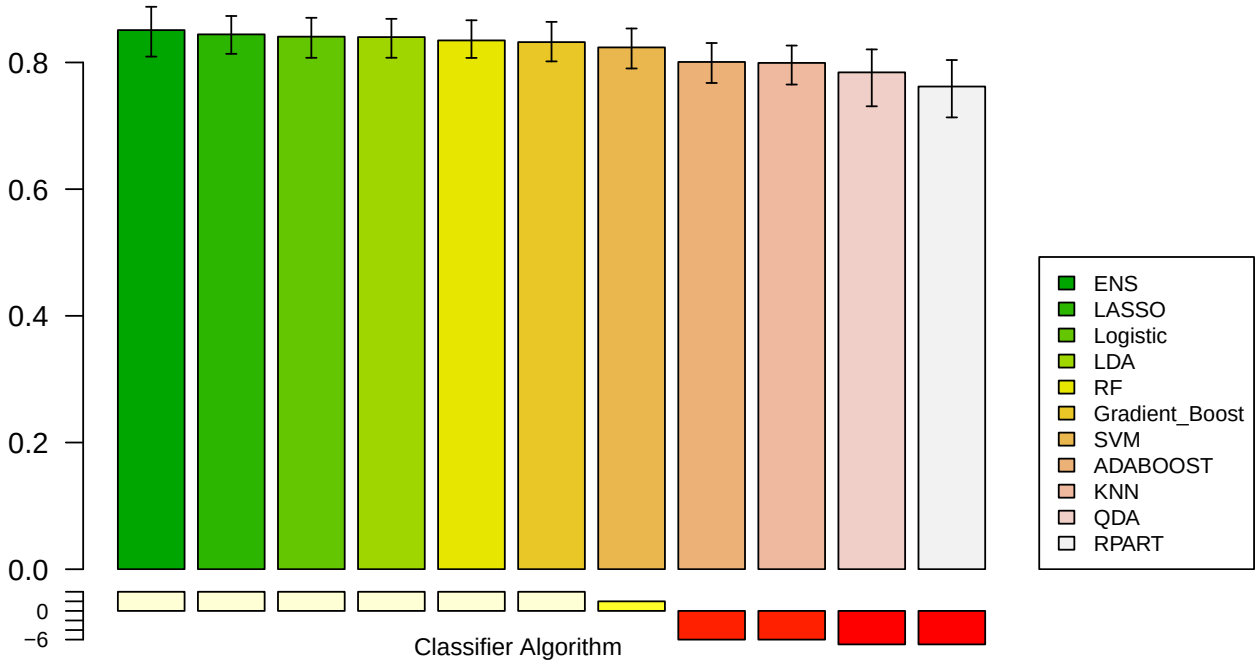
The final two plots provide the heat maps of testing if the methods have similar classification performance and if the methods have larger AUC to the other tested methods.

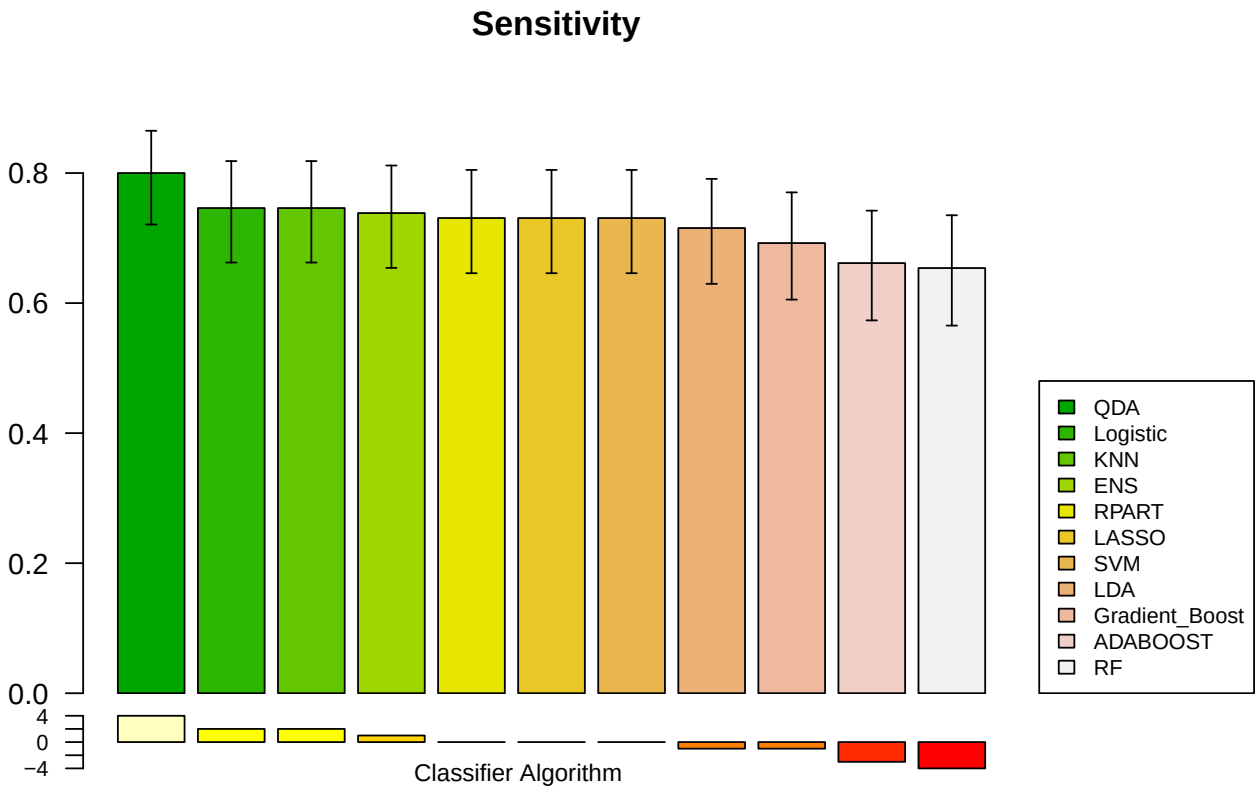
```
par(mfrow = c(1,1),cex = 1.0,xpd = T,pty = 'm', mar = c(3,3,3,10)) # Making space for the legend
prBenchmark <- plot(cp)
```

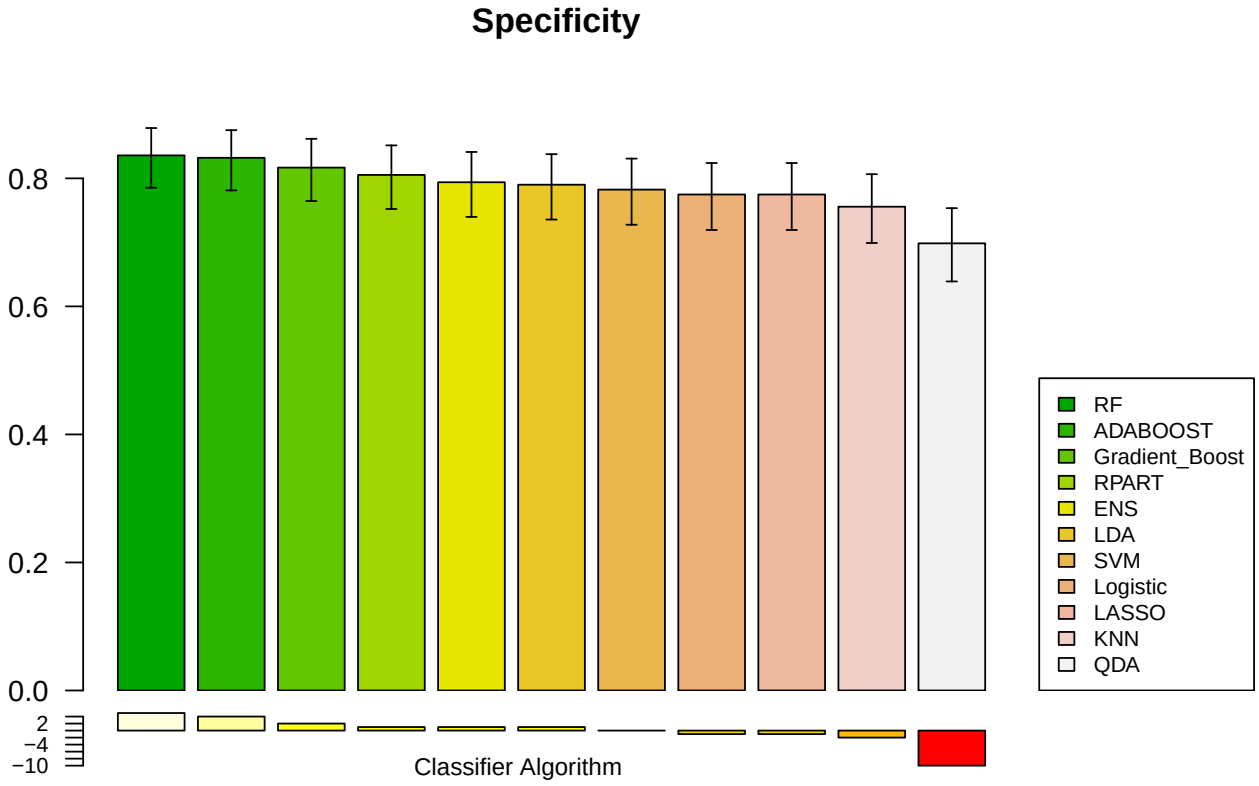

Balanced Error
0.30
0.25
0.20
0.15
0.10
0.05
0.00
1.0
0.0
−1.0
Classifier Algorithm
RPART
ENS
Logistic
SVM
Gradient_Boost
LDA
QDA
LASSO
KNN
ADABOOST
RF

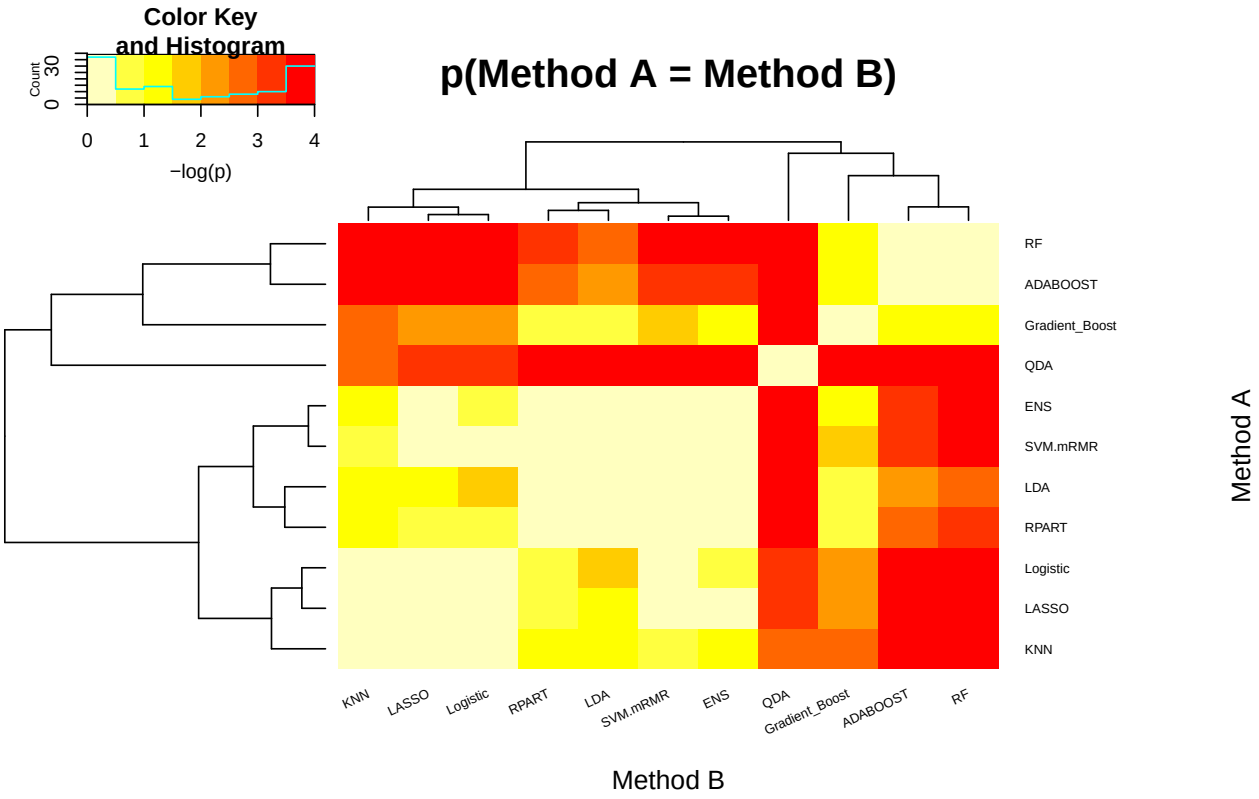

Accuracy
0.8
0.6
0.4
0.2
0.0
1.0
0.0
−1.0
Classifier Algorithm
RPART
Gradient_Boost
ADABOOST
RF
ENS
LDA
Logistic
SVM
LASSO
KNN
QDA

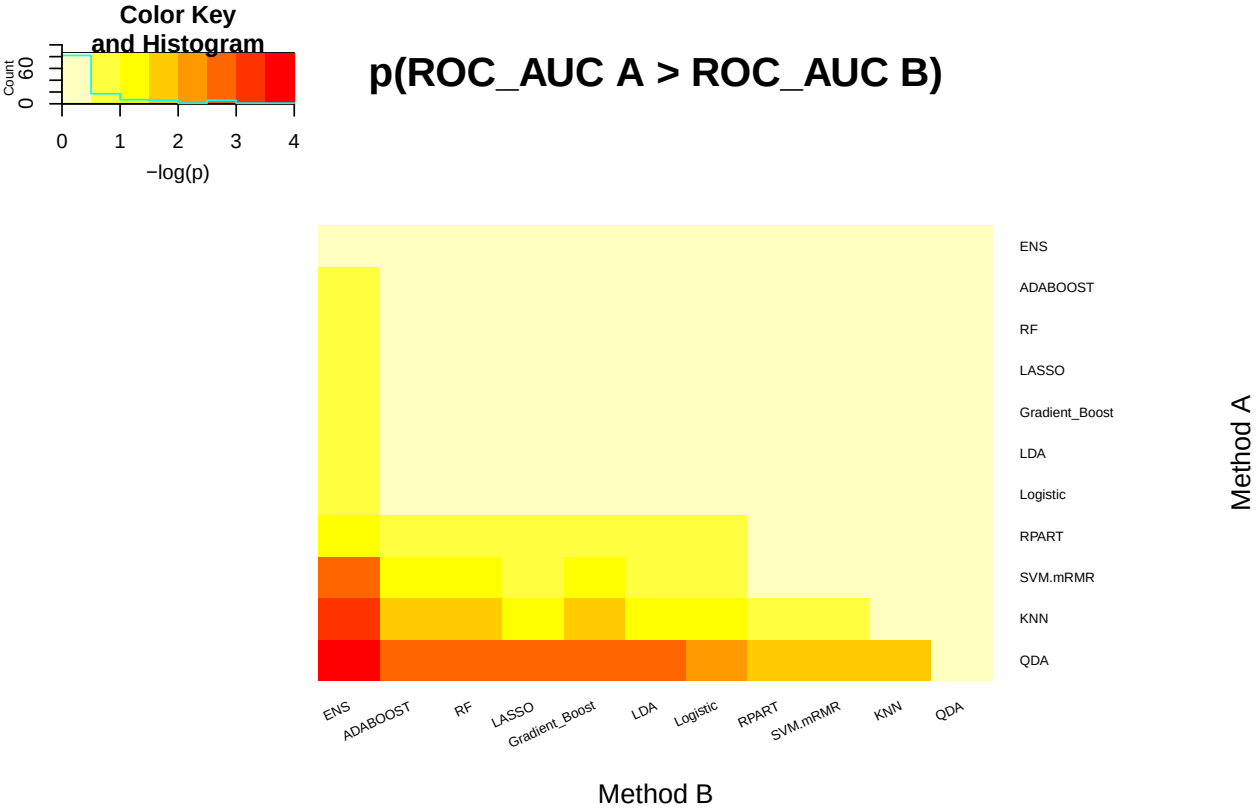

ROC AUC
0.8
0.6
0.4
0.2
0.0
1.0
0.0
−1.0
Classifier Algorithm
ENS
RF
ADABOOST
Gradient_Boost
LASSO
LDA
Logistic
RPART
SVM
KNN
QDA


Concordance
0.8
0.6
0.4
0.2
0.0
0
−6
Classifier Algorithm
ENS
LASSO
Logistic
LDA
RF
Gradient_Boost
SVM
ADABOOST
KNN
QDA
RPART


Sensitivity
0.8
0.6
0.4
0.2
0.0
4
0
−4
Classifier Algorithm
QDA
Logistic
KNN
ENS
RPART
LASSO
SVM
LDA
Gradient_Boost
ADABOOST
RF


Specificity
0.8
0.6
0.4
0.2
0.0
2
−4
−10
Classifier Algorithm
RF
ADABOOST
Gradient_Boost
RPART
ENS
LDA
SVM
Logistic
LASSO
KNN
QDA





The plot function also generates summary tables of the CV results.

```
pander::pander(prBenchmark$metrics,caption = "Classifier Performance",round = 3)
```

Table 13: Classifier Performance (continued below)

	RPART	ENS	Logistic	SVM	Gradient_Boost	LDA	QDA
BER	0.232	0.234	0.24	0.243	0.245	0.248	0.248
ACC	0.781	0.776	0.765	0.765	0.776	0.765	0.732
AUC	0.839	0.851	0.845	0.834	0.846	0.845	0.803
SEN	0.731	0.738	0.746	0.731	0.692	0.715	0.8
SPE	0.805	0.794	0.775	0.782	0.817	0.79	0.698
CIDX	0.762	0.851	0.841	0.824	0.832	0.84	0.784

	LASSO	KNN	ADABOOST	RF
BER	0.249	0.25	0.25	0.256
ACC	0.76	0.753	0.776	0.776
AUC	0.846	0.827	0.847	0.847
SEN	0.731	0.746	0.662	0.654
SPE	0.775	0.756	0.832	0.836
CIDX	0.844	0.799	0.801	0.835