Cross-Validation with FRESA.CAD

José Tamez-Peña

Dec 6, 2019

Table of Contents

Simple Cross-Validation of Common ML Methods 1
The required libraries 1
PimaIndiansDiabetes Data Set 1
Gradient boosting from the gbm package 2
Cross-Validation of common ML-Methods 7
FRESA.CAD::BinaryBenchmark and Comparing Methods 7
Reporting the results of the Benchmark procedure 9

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.

The required libraries

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

PimaIndiansDiabetes Data Set

I will use the PimaIndiansDiabetes dataset from the mlbech package.

```r
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

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

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)
```

```r
pander::pander(table(pr > 0.5,PimaIndiansDiabetes$diabetes),caption="Training: Gradient Boost Confusion Matrix")
```

Training: Gradient Boost Confusion Matrix

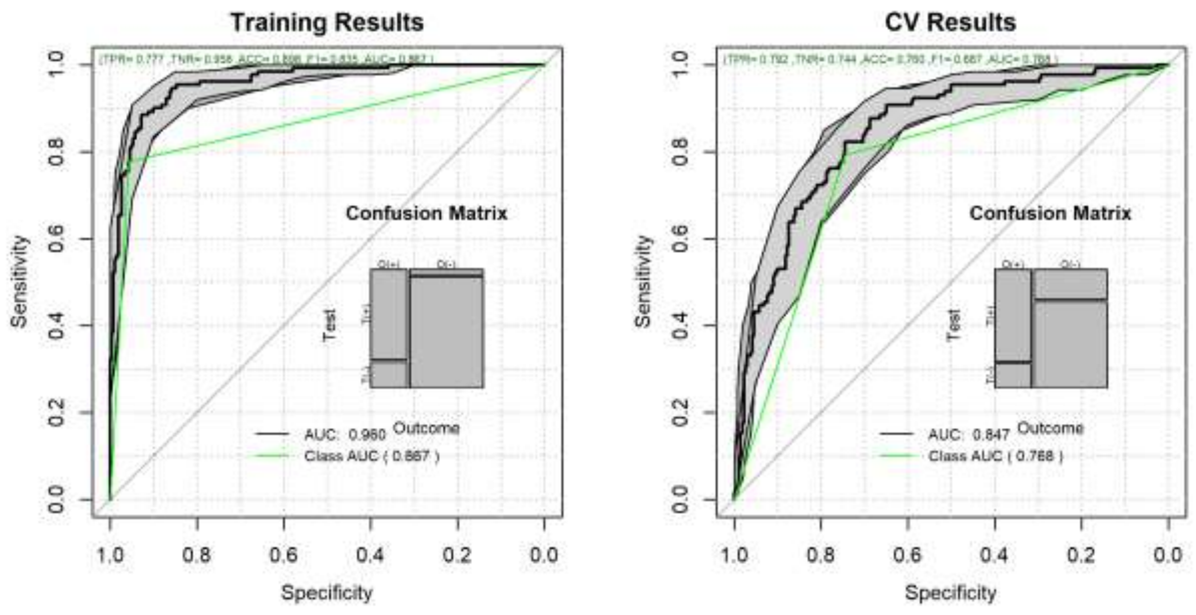
	0	1
FALSE	251	29
TRUE	11	101

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

```r
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

```r

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)
```



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

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

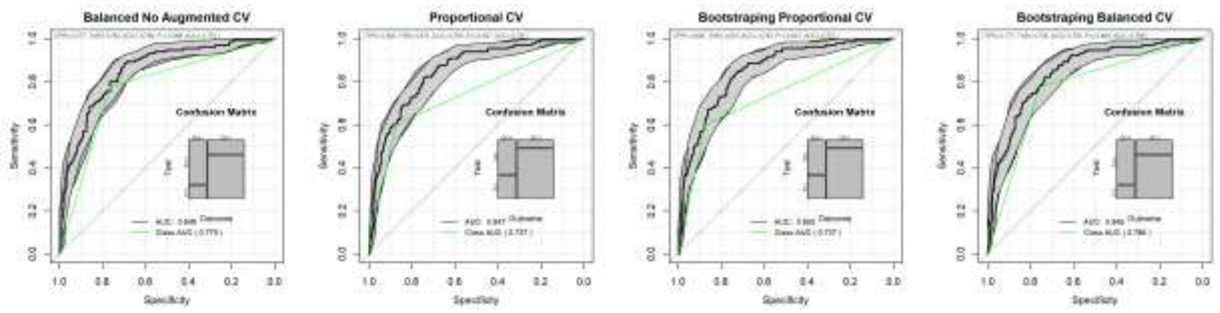
```
GradBOOST_NoAugmentedcv <- randomCV(PimaIndiansDiabetes,
                                    "diabetes",
                                    GBM_fit,
                                    trainFraction = 0.5,
                                    repetitions = 100,
                                  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

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$diabetes,pr)) #No ploting
bs2 <- predictionStats_binary(GradBOOSTcv$medianTest,cex = 0.8) #No ploting
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)
```

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



```r
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

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

Accuracy

est	lower	upper
0.7602	0.7148	0.8016

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

50%	2.5%	97.5%
0.2317	0.1886	0.2769

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

est	lower	upper
0.8467	0.8063	0.887

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

Sensitivity

est	lower	upper
0.7923	0.7124	0.8584

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

Specificity

est	lower	upper
0.7443	0.687	0.796

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

- **accci**:

50%	2.5%	97.5%
0.7602	0.7168	0.801

- **senci**:

50%	2.5%	97.5%
0.7683	0.7231	0.8114

- **aucci**:

50%	2.5%	97.5%
0.7683	0.7231	0.8114

- **berci**:

50%	2.5%	97.5%
0.2317	0.1886	0.2769

- **preci**:

50%	2.5%	97.5%
0.7425	0.6976	0.7839

- **F1ci**:

50%	2.5%	97.5%
0.7457	0.6984	0.7894

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")
```

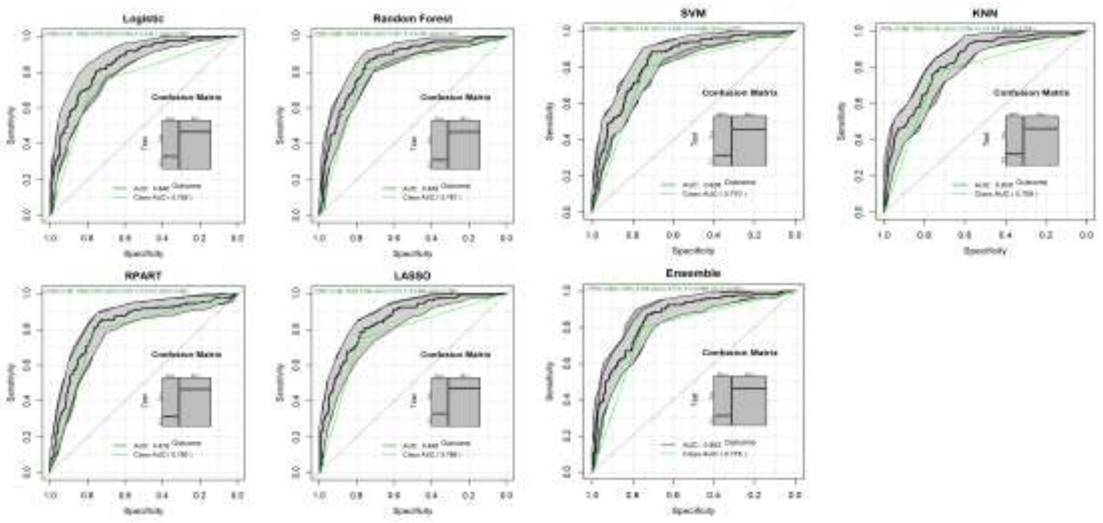
FRESA.CAD::BinaryBenchmark and Comparing Methods

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))
```



¨

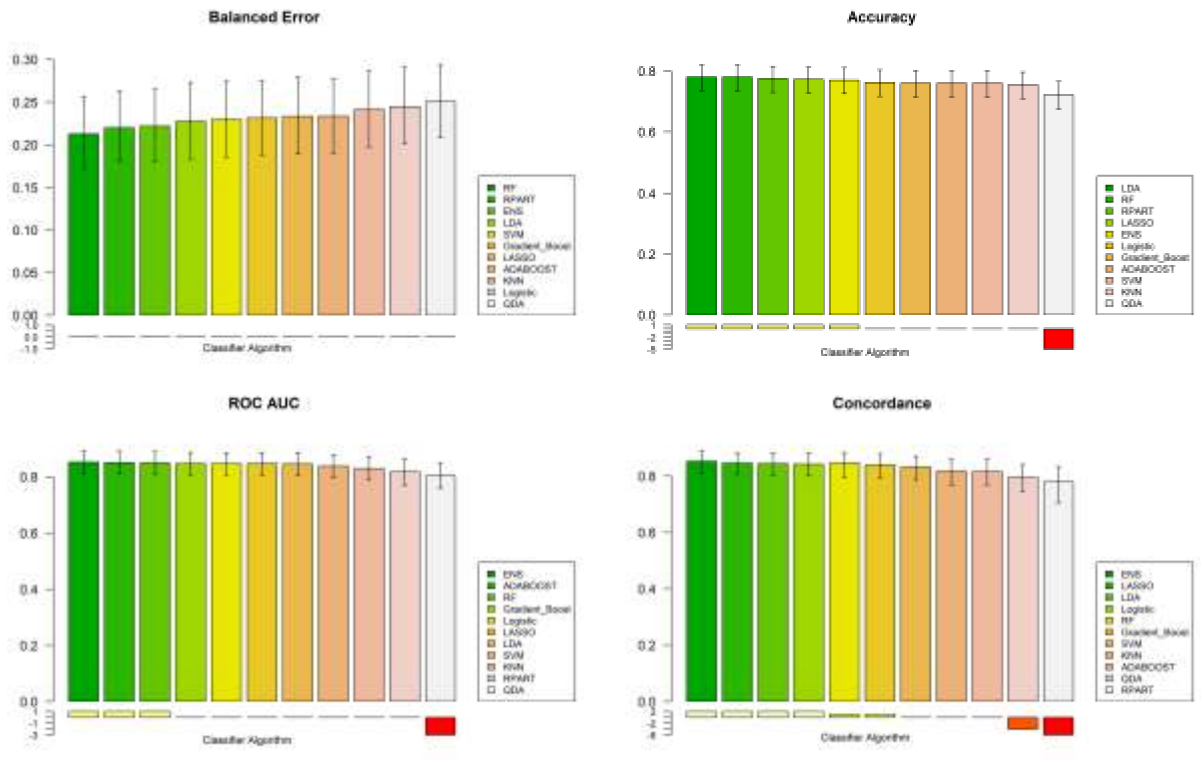
```
par(mfrow = c(1,1),cex = 1.0);
```

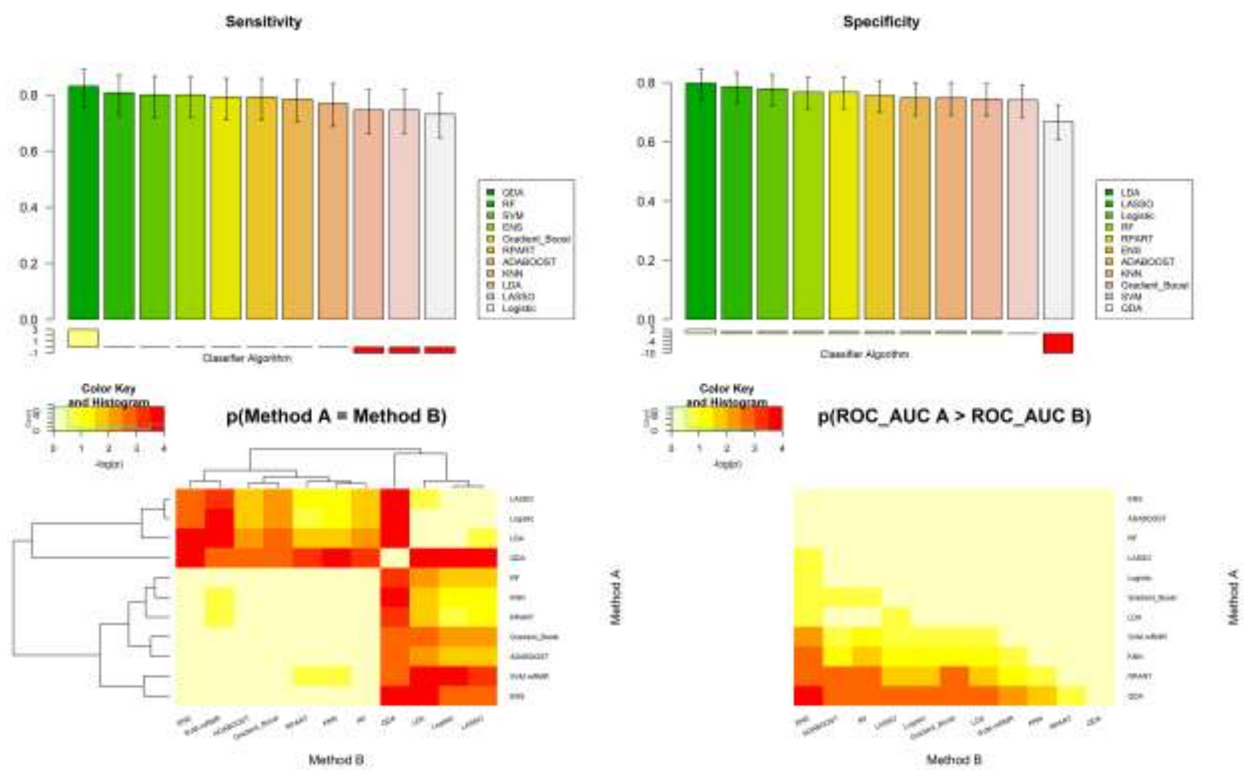
Reporting the results of the Benchmark procedure

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 heatmaps of testing if the methods have similar classification performance and if the methods have larger AUC to the other tested methods.

```r
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)
```





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

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

Classifier Performance

	RF	RPART	ENS	LDA	SVM	Gradient Boost	LASSO	ADABOOST	KNN	Logistic	QDA
BER	0.213	0.22	0.222	0.228	0.23	0.231	0.233	0.234	0.241	0.245	0.251
ACC	0.781	0.776	0.77	0.781	0.76	0.76	0.773	0.76	0.755	0.763	0.722
AUC	0.849	0.818	0.852	0.846	0.838	0.847	0.846	0.85	0.83	0.846	0.804
SEN	0.808	0.792	0.8	0.746	0.8	0.792	0.746	0.785	0.769	0.731	0.831
SPE	0.767	0.767	0.756	0.798	0.74	0.744	0.786	0.748	0.748	0.779	0.668
CIDX	0.846	0.779	0.852	0.844	0.832	0.839	0.846	0.816	0.817	0.842	0.795