DATA 621 – Business Analytics and Data Mining Homework #2

Alain Kuiete Tchoupou

Definition: Confusion Matrix

The confusion matrix of the build model gives elements to test it performance.

		Predicted Values	
		0	**1**
Actual Values	**0**	**True Negative (TN):** cases where the model predicted **no**, and patients **do not have** diabetes	**False Positive (FP)**: cases where the model predicted **no** but the patients do have diabetes, "Type 1 Error"
	1	**False Negative (FN)**: cases where the model predicted **no** but the patients **do have** diabetes.	**True Positive (TP):** cases where the model predicted **yes,** and the patients **do have** diabetes.

Some Metrics

		0	1	
True Negative Rate (TNR): When it is actually 0 how often does the model predict 0. TNR = Specificity = TN/ (TN + FP)	0	TN	FP	**False Positive Rate (FPR)**: When it is actually 0, how often does the model predict 1. FPR = FP/ (TN + FP)
	1	FN	TP	**True Positive Rate (TPR)**: When it is actually 1 how often does the model predict 1. **TPR = Sensitivity = Recall** = TP/ (TP + FN)
			Precision: When the model predicts 1, how often is it correct. Precision = TP/ (FP + TP)	

Accuracy: how often the model is overall correct, accuracy = (TP + TN) / Total

Classification error rate how often the model is overall wrong. (FP + FN)/Total

ROC: curve of the True Positive Rate (TPR) (y-axis) against the False Positive Rate (FPR) (x-axis) when the threshold varies on assigning observation to a given class.

AUC area under the ROC curve. AUC is used to compare the efficacity of different models.

Exploring the Dataset

1. Download the classification output data set (attached in Blackboard to the assignment).

```
class.output <- read.csv("classification-output-data.csv")
 head(class.output)
```

pregnant	glucose	diastolic	skinfold	insulin	bmi	pedigree	age	class	Scored.class	Scored.probability
7	124	70	33	215	25.5	0.161	37	0	0	0.032845226
2	122	76	27	200	35.9	0.483	26	0	0	0.27319044
3	107	62	13	48	22.9	0.678	23	1	0	0.10966039
1	91	64	24	029.2		0.192	21	0	0	0.05599835
4	83	86	19	029.3		0.317	34	0	0	0.10049072
1	100	74	12	46	19.5	0.149	28	0	0	0.05515460

2. The data set has three key columns we will use:

- **class: the actual class for the observation**
- **scored.class: the predicted class for the observation (based on a threshold of 0.5)**
- **scored.probability: the predicted probability of success for the observation**

Use the table() function toget the raw confusion matrix for this scored dataset. Make sure you understand the output. In particular, do the rows represent the actual or predicted class? The columns?

Building the Confusion matrix

```
confusion.matrix <- (table(Actual.value = class.output$class, Predicted.value =
class.output$scored.probability>.5))
 confusion.matrix
```

Confusion Matrix of the Given Classifier

		Predicted. Value	Predicted. Value
		0	1
Actual. Value	0	119	5
Actual. Value	1	30	27

There are 30 patients with diabetes that are predicted to be healthy. This is a lot of patients sick that would be mistakenly taken as normal. This type of error is dangerous since can cause a lot of dead.

We write different functions to compute the metrics

3. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the accuracy of the predictions.

Accuracy function

```
accuracy <- function(class.output){
 accuracy <- (confusion.matrix[1,1] + confusion.matrix[2,2])/sum(confusion.matrix)
 return(accuracy)
}

accuracy(class.output)
```

```
## [1] 0.8066298
```

4. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the classification error rate of the predictions.

Classification Error Rate function

```
class.error.rate <- function(class.output){
 class.er.rate <- (confusion.matrix[2,1] + confusion.matrix[1,2])/sum(confusion.matrix)
 return(class.er.rate)
 }
 class.error.rate(class.output)
```

```
## [1] 0.1933702
```

5. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the precision of the predictions.

Precision

```
precision <- function(class.output){
 precision <- confusion.matrix[2,2]/(confusion.matrix[1,2] + confusion.matrix[2,2])
 return(precision)
 }
 precision(class.output)
```

```
## [1] 0.84375
```

6. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the sensitivity of the predictions. Sensitivity is also known as recall.

Sensitivity

```
sensivity <- function(class.output){
 sensivity <- confusion.matrix[2,2]/(confusion.matrix[2,1] + confusion.matrix[2,2])
 return(sensivity)
 }
 sensivity(class.output)
```

```
## [1] 0.4736842
```

7. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the specificity of the predictions.

Specificity

```
specificity <- function(class.output){
 specificity <- confusion.matrix[1,1]/(confusion.matrix[1,1] + confusion.matrix[1,2])
 return(specificity)
 }
 specificity(class.output)
```

```
## [1] 0.9596774
```

8. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the F1 score of the predictions.

F1Score

```
F1score <- function(class.output){
 f1score <- (2 * precision(class.output) * sensivity(class.output)) /(precision(class.output) +
sensivity(class.output))
 return(f1score)
 }
 F1score(class.output)
```

```
## [1] 0.6067416
```

9. Before we move on, let's consider a question that was asked: What are the bounds on the F1 score? Show that the F1 score will always be between 0 and 1. (Hint: If 0 < a < 1 and 0 < b < 1 then ab < a.)

Let

a	**b**
c	**d**

Be a confusion matrix where a, b, c, d are positive integers.

If the precision P = a/ (a + b), and the sensitivity S = a/ (a + c), using the formula of F1 score we found

F1 = 2a/ (2a + b + c)

Discussion:

If b = c = 0, then F1 score = 1, since a ≠ 0 and d ≠ 0

If b ≠ 0 or c ≠ 0, then 2a < 2a + b + c. It means F1 score < 1

So, 0 < F1 ≤ 1

10. Write a function that generates an ROC curve from a data set with a true classification column (class in our example) and a probability column (scored.probability in our example). Your function should return a list that includes the plot of the ROC curve and a vector that contains the calculated area under the curve (AUC). Note that I recommend using a sequence of thresholds ranging from 0 to 1 at 0.01 intervals.

ROC and AUC

```
roc_auc <- function(){
 library(ROCR)
 #ROC
 thresholds <- seq(0.1,0.99,0.1)
 false.positive.rate <- NULL
 true.positive.rate <- NULL
 for (threshold in thresholds) {
 confusion.matrix <- (table(Actual.value = class.output$class, Predicted.value =
class.output$scored.probability>threshold))
 fpr <- confusion.matrix[1,2]/(confusion.matrix[1,1] + confusion.matrix[1,2])
 tpr <- confusion.matrix[2,2]/(confusion.matrix[2,1] + confusion.matrix[2,2])
 false.positive.rate <- c(false.positive.rate, fpr)
 true.positive.rate <- c(true.positive.rate, tpr)
```

```
 }
 cplot <- plot(x=false.positive.rate, y=true.positive.rate, type = "b")

#Area Under the Curbe
 t <- true.positive.rate
 f <- false.positive.rate
 a <- 0
 for (i in c(1,8)) {
 a <- a + (t[i] + t[i+1])*f[i]/2
 }
 return(c(cplot, a))
 }

roc_auc()
```

```
## [1] 0.6692134
```

11. Use your created R functions and the provided classification output data set to produce all of the classification metrics discussed above.

Recapitulation of Metrics

Putting all metrics we computed togheter.

```
metrics <- data.frame(Name = c("Accuracy",
"Classification error rate",
 "Precision",
 "Sensitivity",
 "Specificity ",
 "F1Score"),
 Output.Value = c(accuracy(class.output),
 class.error.rate(class.output),
 precision(class.output),
 sensivity(class.output),
 specificity(class.output),
 F1score(class.output)
 ))
 metrics
```

Table: Performance of the Given Classifier

Metric Name	Value
Accuracy	0.81
Classification Error Rate	0.19
Precision	0.84

Sensitivity	0.47
Specificity	0.96
F1 Score	0.61

This classifier is 81% accurate. It is 84% correct when it predicts a patient as diabetic and 96% correct when it predicts a patient as non-diabetic.

12. Investigate the caret package. In particular, consider the functions confusionMatrix, sensitivity, and specificity. Apply the functions to the data set. How do the results compare with your own functions?

#??caret

Using caret package

Finding the splitting rate

We split the data into training and test sets. First, we set the splitting rate to .75

```
library(caret)

### precision, specificity

library(mlbench)
 diabetics <- class.output[,1:9]
 diabetics$class <- factor(diabetics$class, labels = c("F", "T"))

set.seed(127)
 inTrain <- createDataPartition(
 y = diabetics$class,
 ## the outcome data are needed
 p = .75,
 ## The percentage of data in the
 ## training set
 list = FALSE

)
 ## The format of the results

 str(inTrain)

## int [1:136, 1] 1 2 3 4 5 6 7 8 10 11 ...
 ## - attr(*, "dimnames")=List of 2
```

```
 ## ..$ : NULL
 ## ..$ : chr "Resample1"
```

structure of our dataset

```
str(diabetics)
```

```
## 'data.frame': 181 obs. of 9 variables:
 ## $ pregnant : int 7 2 3 1 4 1 9 8 1 2 ...
 ## $ glucose : int 124 122 107 91 83 100 89 120 79 123 ...
 ## $ diastolic: int 70 76 62 64 86 74 62 78 60 48 ...
 ## $ skinfold : int 33 27 13 24 19 12 0 0 42 32 ...
 ## $ insulin : int 215 200 48 0 0 46 0 0 48 165 ...
 ## $ bmi : num 25.5 35.9 22.9 29.2 29.3 19.5 22.5 25 43.5 42.1 ...
 ## $ pedigree : num 0.161 0.483 0.678 0.192 0.317 0.149 0.142 0.409 0.678 0.52 ...
 ## $ age : int 37 26 23 21 34 28 33 64 23 26 ...
 ## $ class : Factor w/ 2 levels "F","T": 1 1 2 1 1 1 1 1 1 1 ...
```

Splitting the data

```
training <- diabetics[ inTrain,]
 testing <- diabetics[-inTrain,]


nrow(training)
```

```
## [1] 136
```

```
nrow(testing)
```

```
## [1] 45
```

Tunning the model

```
ctrl <- trainControl(
 method = "repeatedcv",
repeats = 3,
 classProbs = TRUE,
summaryFunction = twoClassSummary
 )
```

Here we train the model with partial least square .

```
plsFit <- train(
 class ~ .,
 data = training,
 method = "pls",
 ## Center and scale the predictors for the training
 ## set and all future samples.
 preProc = c("center", "scale"),
 trControl = ctrl,
 metric = "ROC"
 )
ggplot((plsFit))
```

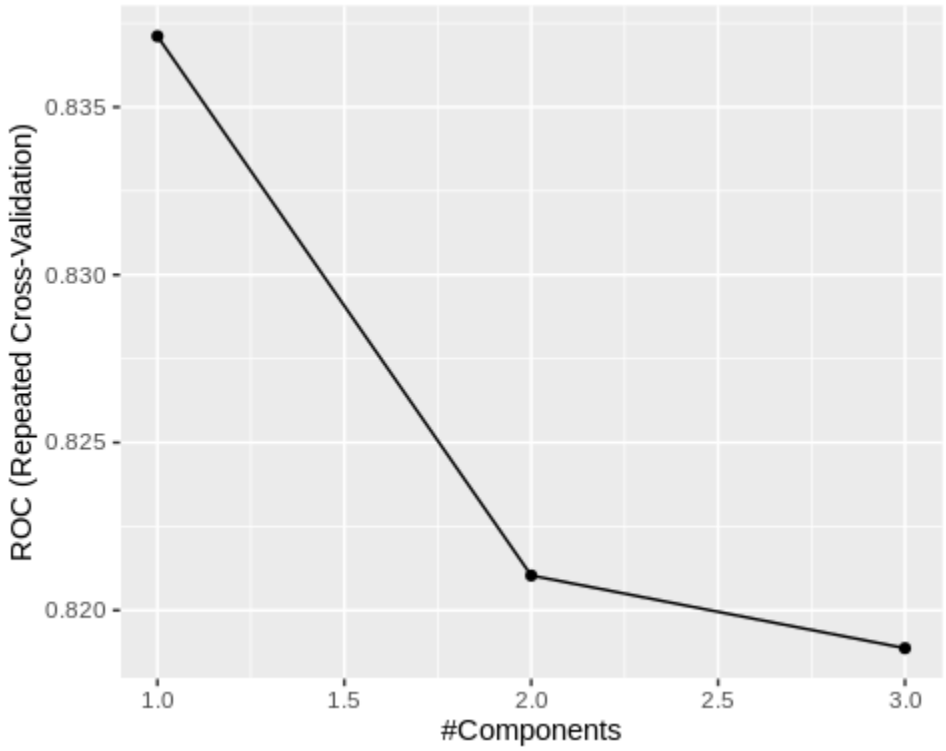

Fig 1. ROC curve from tuning model of caret package

confusion matrix with the training set

```
caret.cm.train <- confusionMatrix(training$class, predict(plsFit), positive = 'T')

caret.metrics1 <- data.frame(caret.train = c(caret.cm.train$overall["Accuracy"],
 1-caret.cm.train$overall["Accuracy"],
 caret.cm.train$byClass["Precision"],
 caret.cm.train$byClass["Sensitivity"],
 caret.cm.train$byClass["Specificity"],
 caret.cm.train$byClass["F1"]
```

```
 ))
caret.metrics1

## caret.train
 ## 1 0.8382353
 ## 2 0.1617647
 ## 3 0.6511628
 ## 4 0.8000000
 ## 5 0.8514851
 ## 6 0.7179487
```

Different model using Partial Least Squares

Here we train the model with partial least square using 10-fold cross validation

```
ctrl <- trainControl(method = "repeatedcv", repeats = 3)

plsFit <- train(
 class ~ .,
 data = training,
 method = "pls",
 preProc = c("center", "scale"),
 tuneLength = 15,
 ## added:
 trControl = ctrl
 )

ctrl <- trainControl(
 method = "repeatedcv",
repeats = 3,
 classProbs = TRUE,
summaryFunction = twoClassSummary
 )

set.seed(123)
 plsFit2 <- train(
 class ~ .,
 data = training,
 method = "pls",
 preProc = c("center", "scale"),
 tuneLength = 15,
 trControl = ctrl,
 metric = "ROC"
 )
 plsFit2
```

Partial Least Squares
##
136 samples
8 predictor
2 classes: 'F', 'T'
##
Pre-processing: centered (8), scaled (8)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 122, 123, 123, 123, 122, 123, ...
Resampling results across tuning parameters:
##

Table: Resampling results across tuning parameters:

ncomp	ROC	Sensitivity	Specificity
1	0.8416111	0.9048148	0.5716667
2	0.8185000	0.9074074	0.5200000
3	0.8208889	0.9114815	0.5400000
4	0.8212407	0.9111111	0.5566667
5	0.8205000	0.9077778	0.5483333
6	0.8189259	0.9077778	0.5483333
7	0.8189259	0.9077778	0.5566667

ROC was used to select the optimal model using the largest value.
The final value used for the model was ncomp = 1.

We can now obtain predicted values with the testing set

```
plsClasses <- predict(plsFit2, newdata = testing)
str(plsClasses)
```

Factor w/ 2 levels "F","T": 1 1 2 2 2 1 2 1 1 1 ...

#> Factor w/ 2 levels "M","R": 2 1 1 1 2 2 1 2 2 2 ...

```
plsProbs <- predict(plsFit2, newdata = testing, type = "prob")
head(plsProbs)
```

Table. Probability obtained from 10-cross validation

	F	T
9	0.6540304	0.3459696
12	0.5586304	0.4413696
15	0.3458017	0.6541983
18	0.4724670	0.5275330
19	0.2104277	0.7895723
21	0.5829847	0.4170153

We can also plot the ROC of the new model using ggplot
ggplot((plsFit2))

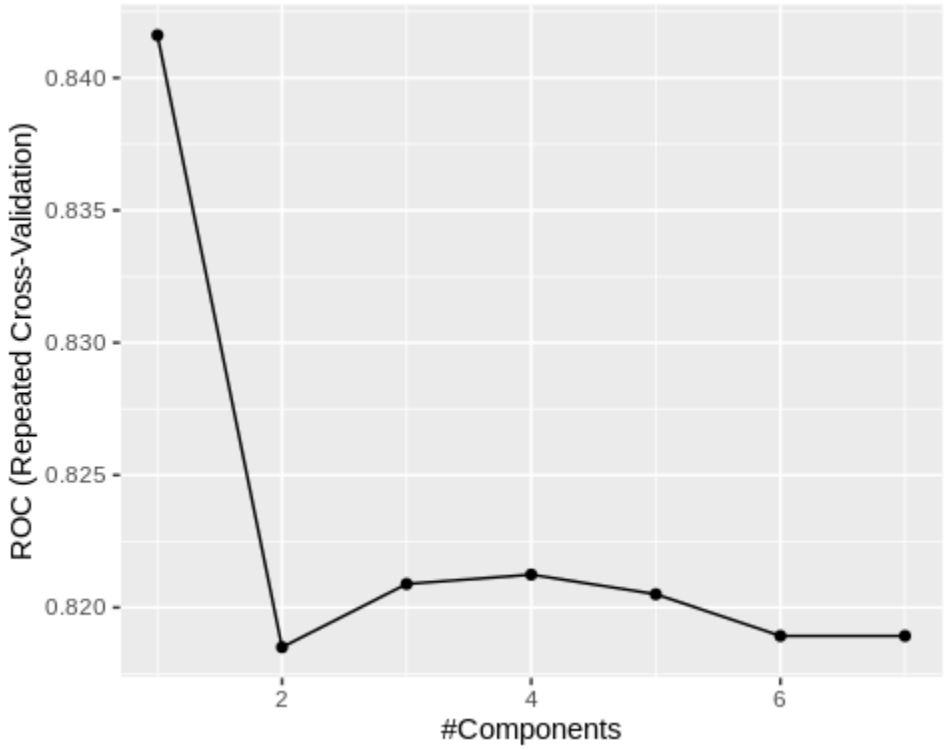

Fig. 2 ROC

C*onfusion matrix of the new model with testing set*

confusionMatrix(data = plsClasses, testing$class, positive = 'T')

Confusion Matrix and Statistics
##
Reference
Prediction F T
F 28 8
T 3 6
##
Accuracy : 0.7556
95% CI : (0.6046, 0.8712)
No Information Rate : 0.6889
P-Value [Acc > NIR] : 0.2127
##
Kappa : 0.3678
##
Mcnemar's Test P-Value : 0.2278

```
##
## Sensitivity : 0.4286
## Specificity : 0.9032
## Pos Pred Value : 0.6667
## Neg Pred Value : 0.7778
## Prevalence : 0.3111
## Detection Rate : 0.1333
## Detection Prevalence : 0.2000
## Balanced Accuracy : 0.6659
##
## 'Positive' Class : T
##
```

Now we extract the performance metrics of the classifier

```
caret.cm.train2 <- confusionMatrix(testing$class, plsClasses, positive = 'T')

caret.metrics2 <- data.frame(caret.test = c(caret.cm.train2$overall["Accuracy"],
 1-caret.cm.train2$overall["Accuracy"],
 caret.cm.train2$byClass["Precision"],
 caret.cm.train2$byClass["Sensitivity"],
 caret.cm.train2$byClass["Specificity"],
 caret.cm.train2$byClass["F1"]
 ))
caret.metrics2
```

```
## caret.test
 ## 1 0.7555556
 ## 2 0.2444444
 ## 3 0.4285714
 ## 4 0.6666667
 ## 5 0.7777778
 ## 6 0.5217391
```

Here we join in a data frame the different values of metrics from first, our given model, the train and test set of tuning models from caret package.

```
metric.compare <- cbind(metrics, caret.metrics1, caret.metrics2)
 metric.compare
```

Table. Comparison of accuracies

Metric Name	Given Model	Training New Model	Testing New Model
Accuracy	0.81	0.84	0.76

Classification Error Rate	0.19	0.16	0.24
Precision	0.84	0.65	0.43
Sensitivity	0.47	0.80	0.67
Specificity	0.96	0.85	0.77
F1 Score	0.61	0.72	0.52

The accuracy of the given classifier is less than the accuracy of the new model's training set but greater than the one of new model testing set. The given classifier has a larger specificity than the model from caret package, but a smaller sensitivity.

 # library(MLeval)
 # x <- evalm(predict(plsFit, type = "prob"))

13. Investigate the pROC package. Use it to generate an ROC curve for the data set. How do the results compare with your own functions?

Using pROC package.

We can plot the ROC curve and extract the AUC value.

```
library(pROC)

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
 ##
## cov, smooth, var

library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
 ##
## filter, lag

## The following objects are masked from 'package:base':
 ##
## intersect, setdiff, setequal, union

roc(class.output$class, class.output$scored.probability, auc = TRUE)
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

##
## Call:
 ## roc.default(response = class.output$class, predictor = class.output$scored.probability, auc = TRUE)
 ##
## Data: class.output$scored.probability in 124 controls (class.output$class 0) < 57 cases
(class.output$class 1).
 ## Area under the curve: 0.8503

## With ggplot2 ##
 library(ggplot2)
 # Create multiple curves to plot
 rocs <- roc(class ~ scored.probability, data = class.output)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

ggroc(rocs)
```

Ours function we created to compute AUC gave us 0.67 less than the 0.85 obtain with pROC.

The ROC curve from pROC is more elaborate than the one we draw. He has more ramifications and more precision. We use less data points to compute the ROC.

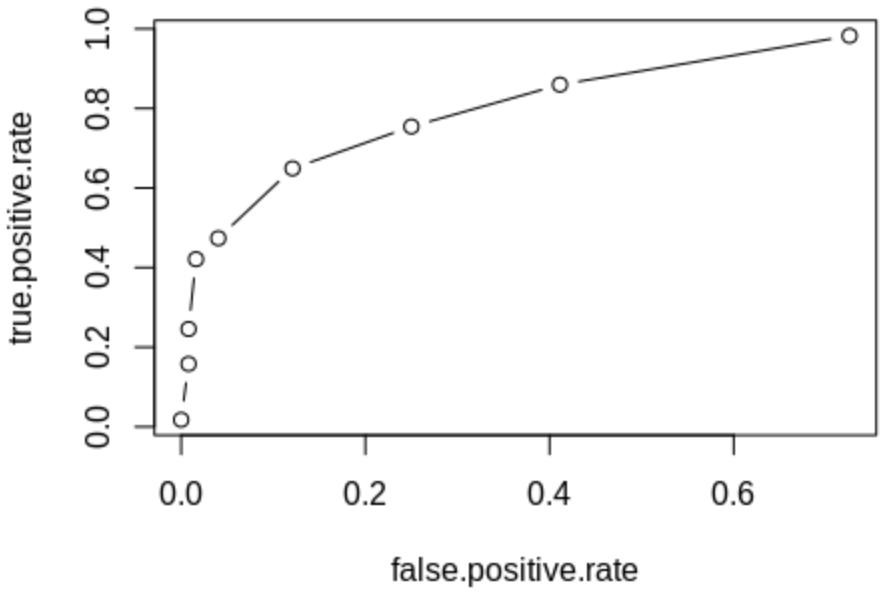

Fig3 ROC curve from the given classifier. Question 10

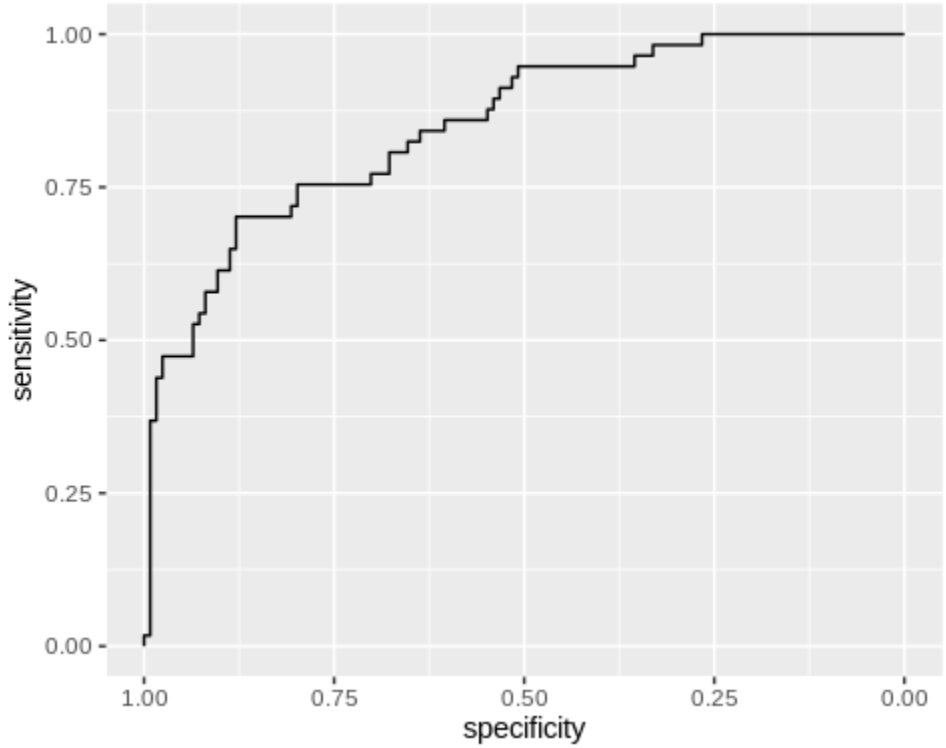

Fig4. ROC Curve from pROC package