ROC and AUC

Sebastian Heucke

9/30/2019

Contents

ROC and AUC in R

This tutorial walks you through, step-by-step, how to draw ROC curves and calculate AUC. We start with basic ROC graph, learn how to extract thresholds for decision making, calculate AUC and partial AUC and how to layer multiple ROC curves on the same graph. The document is based on the https://www.youtube.com/watch?v=qcvAqAH60Yw&list=PLblh5JKOoLUICTaGLRoHQDuF_7q2GfuJF&index=8&t=0s video and I transformed the content into an R Markdown document.

The first thing we need to do is load in **pROC**, the library that will draw **ROC** graphs for us.

```
library(pROC)
```

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

```
##
## Attaching package: 'pROC'
```

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

We're also going to use the **random Forest** package as part of the example. You just need to know that a **Random Forest** is a way to classify samples and we can change the threshold that we use to make those decisions.

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

Since we are going to generate an example dataset, let's set the seed for the random number generator so that we can reproduce our results.

```
set.seed(420)
```

Create a variable for the number of samples, by setting num.samples to 100.

```r
num.samples <- 100
```

Now create **100** measurments and store them in a variable called **weight**. We do this by using the **rnorm()** function to generate **100** random values from a normal distribution with the mean set to 172 and the standard deviation set to 29. Then use the **sort()** function to sort the numbers from low to high.

```r
weight <- sort(rnorm(n=num.samples, mean=172, sd=29))
```

Next we classify an individual as *obese* or *not obese*. The way we are going to classify a sample as *obese* is to start by using the **rank()** function to rank the weights, from lightest to heaviest. The *lightest* sample will have **rank=1**, and the *heaviest* sample will have **rank = 100**. Then we scale the ranks by **100**. This means the *lightest* sample will = **1/100 = 0.01**, and the *heaviest* sample will = **100/100 = 1**. Then we compare the scaled ranks to random numbers between **0** and **1**. If the random number is *smaller* then the scaled rank, then the individual is classified as *obese*, otherwise it is classified as *not obese*. The "if smaller then *obese*, otherwise *not obese*" is performed by the **ifelse()** function and the results are stored in a variable called **obese**.

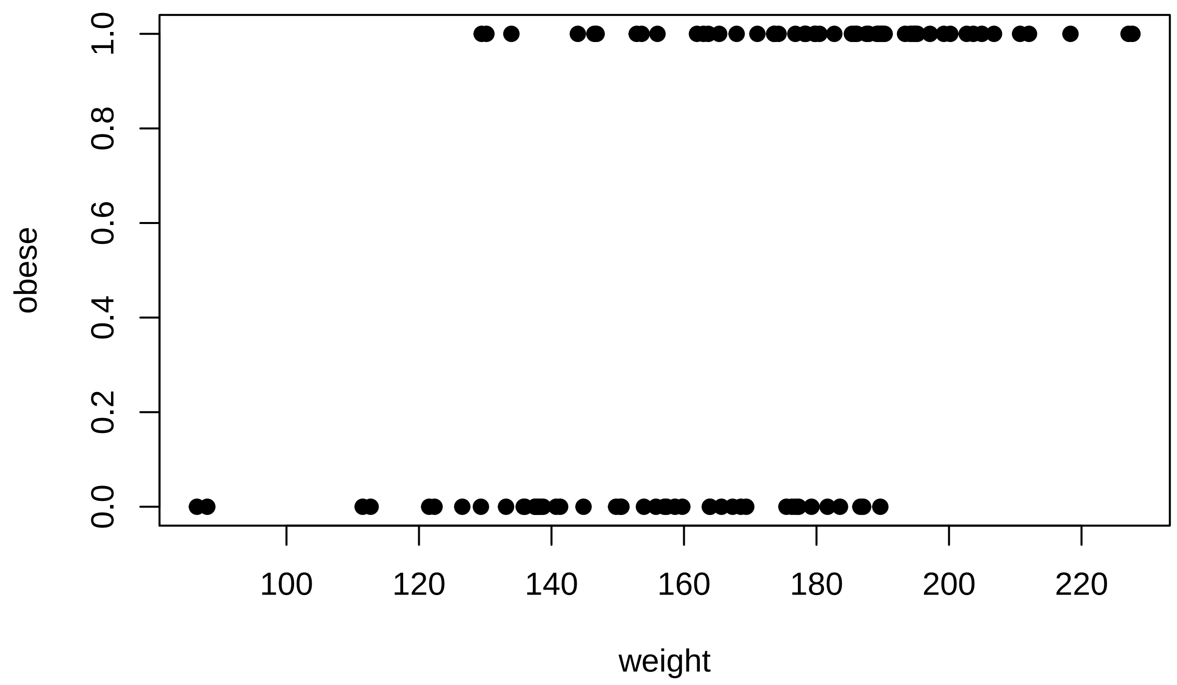
```r
obese <- ifelse(test=(runif(n=num.samples) < (rank(weight)/num.samples)), yes=1, no=0)
# print out the content of obese
obese
```

```
##   [1] 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1 1 0 0 1 0
##  [36] 0 0 0 1 1 1 0 0 1 0 0 1 0 0 1 1 1 1 0 0 1 0 0 1 1 1 0 1 1 1 0 1 0 1 1
##  [71] 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

The **0**'s stand for *not obese* and the **1**'s stand for *obese*. The lighter samples are mostly **0**s (not obese) and the heavier samples are mostly **1**s (obese).

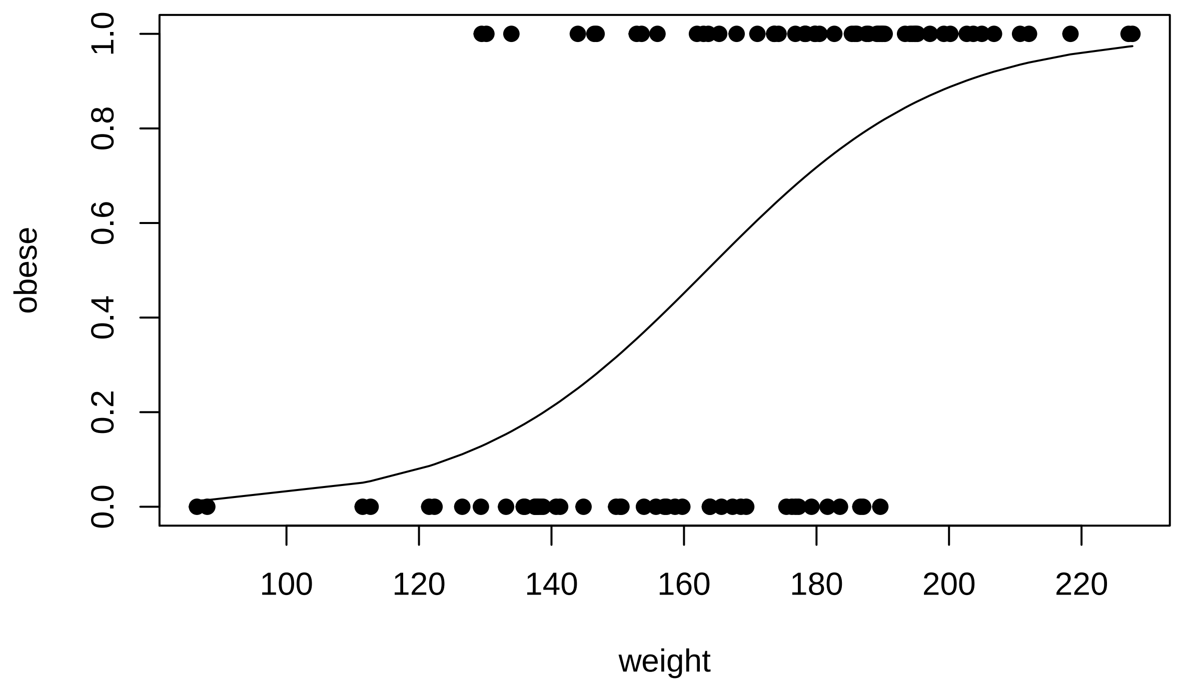
Now we can plot the data to see what it looks like.

```r
plot(x=weight,y=obese)
```



Now we will use the **glm()** function to fit a logistic regression curve to the data. The results of the **glm()** function are stored in a variable called glm.fit. Then pass the **weight** and **fitted.values** stored in **glm.fit** into the **lines()** function to draw a curve that tells us the predicted probability that an individual is *obese* or *not obese*.

```r
# fit logistiv regression
plot(x=weight,y=obese)
glm.fit=glm(obese ~ weight, family=binomial)
lines(weight, glm.fit$fitted.values)
```

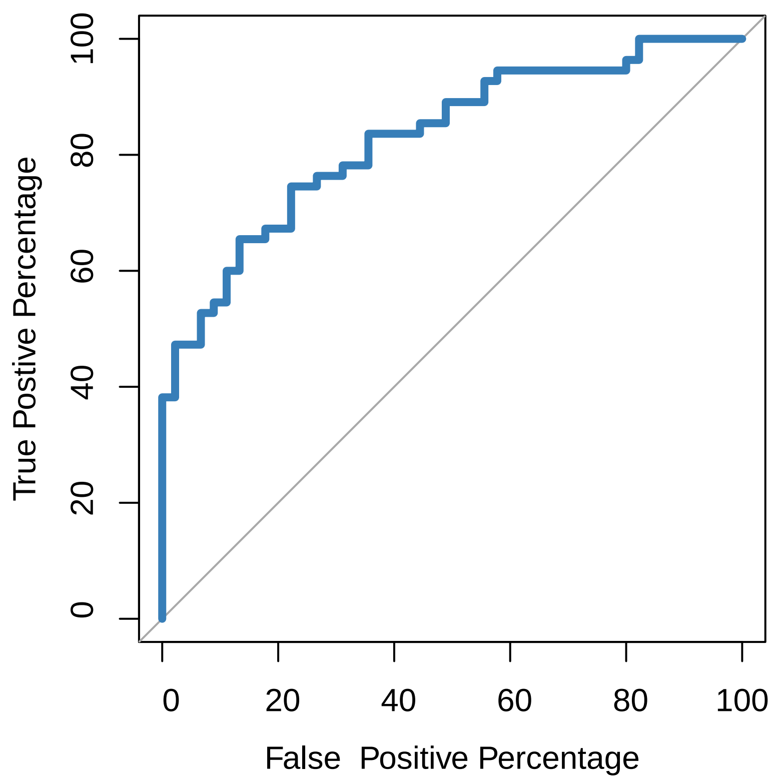


glm.fit.fitted.values contains the y-axis coordinates along the curve for each sample. In other words glm.fit.fitted.values contains estimated probabilities that each sample is *obese*.

We will you the known classification and the estimated probabilities to draw an **ROC** curve. We use the **roc()** function from the **pROC** library to draw the **ROC** graph. We pass in the known classifications, *obese* or *not obese*, for each sample and the estimated probabilities that each sample is *obese*. We tell the **roc()** function to draw the graph, not just calculate all of the numbers used to draw the graph.

```
par(pty = "s") # sets the aspect ratio of plot to a square plotting region

## NOTE: By default, roc() uses specificity on the x-axis and the values range
## from 1 to 0. This makes the graph look like what we would expect, but the
## x-axis itself might induce a headache. To use 1-specificity (i.e. the
## False Positive Rate) on the x-axis, set "legacy.axes" to TRUE.

roc(obese, glm.fit$fitted.values, plot=TRUE, legacy.axes=TRUE, percent=TRUE,
xlab="False  Positive Percentage", ylab="True Postive Percentage", col="#377eb8", lwd=4)
```

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

## Setting direction: controls < cases
```



```
##
## Call:
## roc.default(response = obese, predictor = glm.fit$fitted.values,     percent = TRUE, plot = TRUE, leg
##
## Data: glm.fit$fitted.values in 45 controls (obese 0) < 55 cases (obese 1).
## Area under the curve: 82.91%
```

The first part of the output just echos what you typed in, and isn't very interesting. The second part is a little more interesting - it tells us how may samples were *not obese* and how many were *obese*. The third part is the most interesting of all, it tells us the Area Under the Curve, or the **AUC**.

If we want to find out the optimal threshold we can store the data used to make the ROC graph in a variable and then extract just the information that we want from that variable.

```
roc.info <- roc(obese, glm.fit$fitted.values, legacy.axes=TRUE)
```

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

```
## Setting direction: controls < cases
```

```
roc.df <- data.frame(
  tpp=roc.info$sensitivities*100, # tpp = true positive percentage
  fpp=(1 - roc.info$specificities)*100, # fpp = false positive precentage
  thresholds=roc.info$thresholds)
```

We can then use the **head()** function to look at the first six rows of the new **data.frame**.

```
head(roc.df)
```

```
##   tpp       fpp thresholds
## 1 100 100.00000       -Inf
## 2 100  97.77778 0.01349011
## 3 100  95.55556 0.03245008
## 4 100  93.33333 0.05250145
## 5 100  91.11111 0.07017225
## 6 100  88.88889 0.08798755
```

We see that when the threshold is set to **negative infinity**, so that every single sample is called *obese* then the **tpp**, the **True Positive Percentage** is **100** because all of the *obese* samples were *correctly* classified and the **fpp**, the FALSE Positive Percentage, is also **100** because all of the samples that were *not obeses* were *incorrectly* classified. So the first row in **roc.df** corresponds to the upper right-hand corner of the **ROC** curve.

We can use the **tail()** function to look at the last six rows of the **data.frame**.

```r
tail(roc.df)
```

```
##          tpp fpp thresholds
## 96  9.090909   0  0.9275222
## 97  7.272727   0  0.9371857
## 98  5.454545   0  0.9480358
## 99  3.636364   0  0.9648800
## 100 1.818182   0  0.9735257
## 101 0.000000   0        Inf
```

We see that when the threshold is set to positive infinity, so that every single sample is classified *not obeses* then the **tpp** and **fpp** are both **0** because none of the samples were classified, either *correctly* or *incorrectly*, *obese*. So the last row in **roc.df** corresponds to the bottom left-hand corner of the **ROC** curve.

Now we can isolate the **tpp**, the **fpp** and the **tresholds** used when the **True Positive Rate** is between **60** and **80**.

```r
roc.df[roc.df$tpp > 60 & roc.df$tpp < 80,]
```

```
##         tpp      fpp thresholds
## 42 78.18182 35.55556  0.5049310
## 43 78.18182 33.33333  0.5067116
## 44 78.18182 31.11111  0.5166680
## 45 76.36364 31.11111  0.5287933
## 46 76.36364 28.88889  0.5429351
## 47 76.36364 26.66667  0.5589494
## 48 74.54545 26.66667  0.5676342
## 49 74.54545 24.44444  0.5776086
## 50 74.54545 22.22222  0.5946054
## 51 72.72727 22.22222  0.6227449
## 52 70.90909 22.22222  0.6398136
## 53 69.09091 22.22222  0.6441654
## 54 67.27273 22.22222  0.6556705
## 55 67.27273 20.00000  0.6683618
## 56 67.27273 17.77778  0.6767661
## 57 65.45455 17.77778  0.6802060
## 58 65.45455 15.55556  0.6831936
## 59 65.45455 13.33333  0.6917225
## 60 63.63636 13.33333  0.6975300
## 61 61.81818 13.33333  0.6982807
```

If we were interested in choosing a threshold in this range, we could pick the one that had the optimal balance of **True Positives** and **False Positives**.

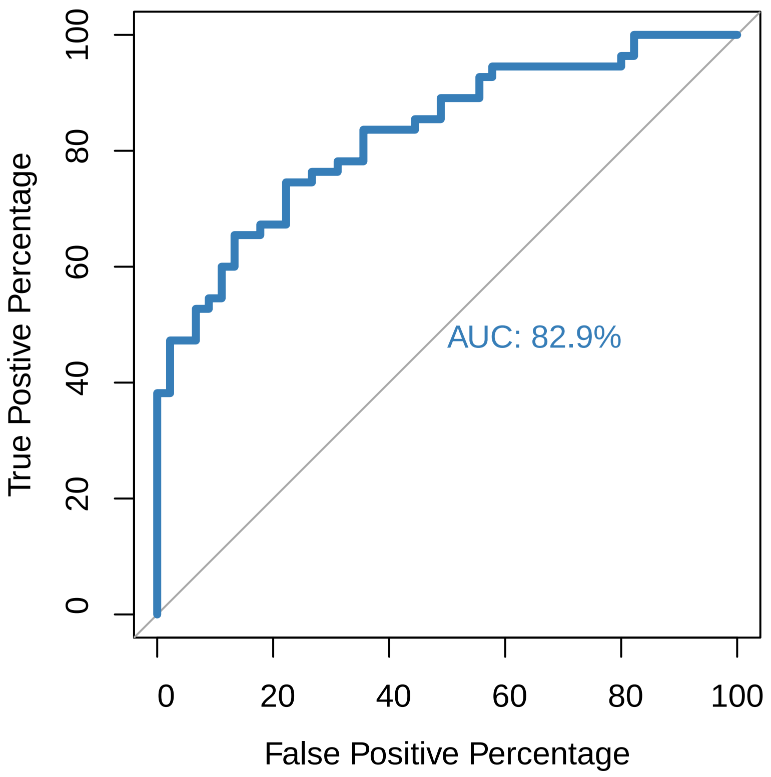
Customizing ROC graph

If we want to print the **AUC** directly on the graph, then we set the **print.auc** parameter to **TRUE**.

```r
par(pty = "s")
roc(obese, glm.fit$fitted.values, plot=TRUE, legacy.axes=TRUE, percent=TRUE,
    xlab="False Positive Percentage", ylab="True Postive Percentage", col="#377eb8", lwd=4,
    print.auc=TRUE)
```

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

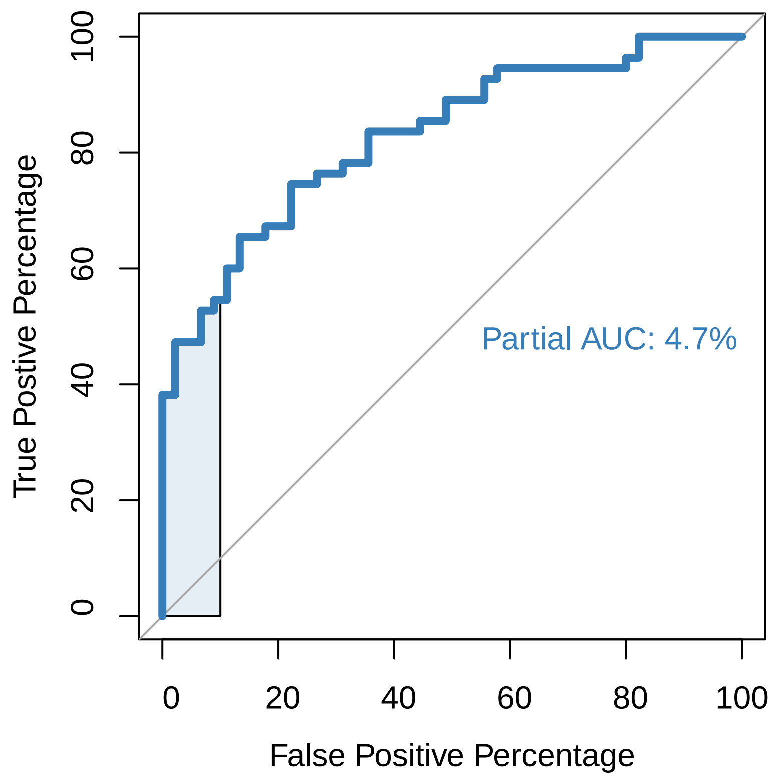
```
## Setting direction: controls < cases
```



```
##
## Call:
## roc.default(response = obese, predictor = glm.fit$fitted.values,     percent = TRUE, plot = TRUE, leg
##
## Data: glm.fit$fitted.values in 45 controls (obese 0) < 55 cases (obese 1).
## Area under the curve: 82.91%
```

```r
par(pty = "s")
# ...and the partial area under the curve.
roc(obese, glm.fit$fitted.values, plot=TRUE, legacy.axes=TRUE, percent=TRUE,
    xlab="False Positive Percentage", ylab="True Postive Percentage", col="#377eb8",
    lwd=4, print.auc=TRUE, print.auc.x=45, partial.auc=c(100, 90), auc.polygon = TRUE,
    auc.polygon.col = "#377eb822")
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```



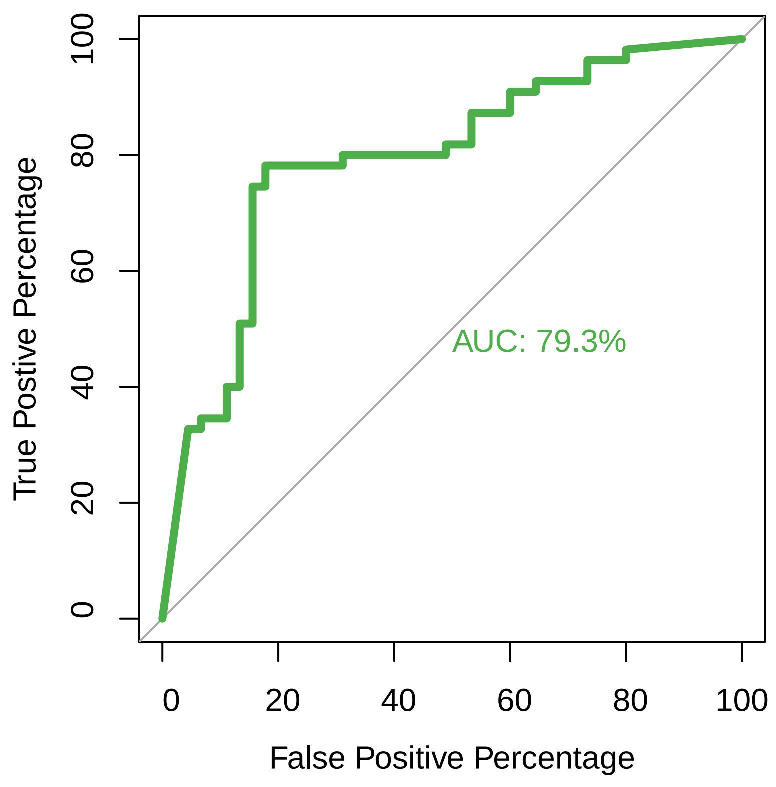
```
##
## Call:
## roc.default(response = obese, predictor = glm.fit$fitted.values,     percent = TRUE, plot = TRUE, leg
##
## Data: glm.fit$fitted.values in 45 controls (obese 0) < 55 cases (obese 1).
## Partial area under the curve (specificity 100%-90%): 4.727%
```

Lastly let's overlap two **ROC** curves so that they are easy to compare. We'll start by making a **Random Forest** classifier with the same dataset. Then drawn the original **ROC** curve for the **Logistic Regression** and add the **ROC** curve for the **Random Forest**.

```
par(pty = "s")
rf.model <- randomForest(factor(obese) ~ weight)

roc(obese, rf.model$votes[,1], plot=TRUE, legacy.axes=TRUE, percent=TRUE,
    xlab="False Positive Percentage", ylab="True Postive Percentage", col="#4daf4a", lwd=4,
    print.auc=TRUE)
```

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

```
## Setting direction: controls > cases
```



```
##
## Call:
## roc.default(response = obese, predictor = rf.model$votes[, 1],     percent = TRUE, plot = TRUE, legac
##
## Data: rf.model$votes[, 1] in 45 controls (obese 0) > 55 cases (obese 1).
## Area under the curve: 79.29%
```

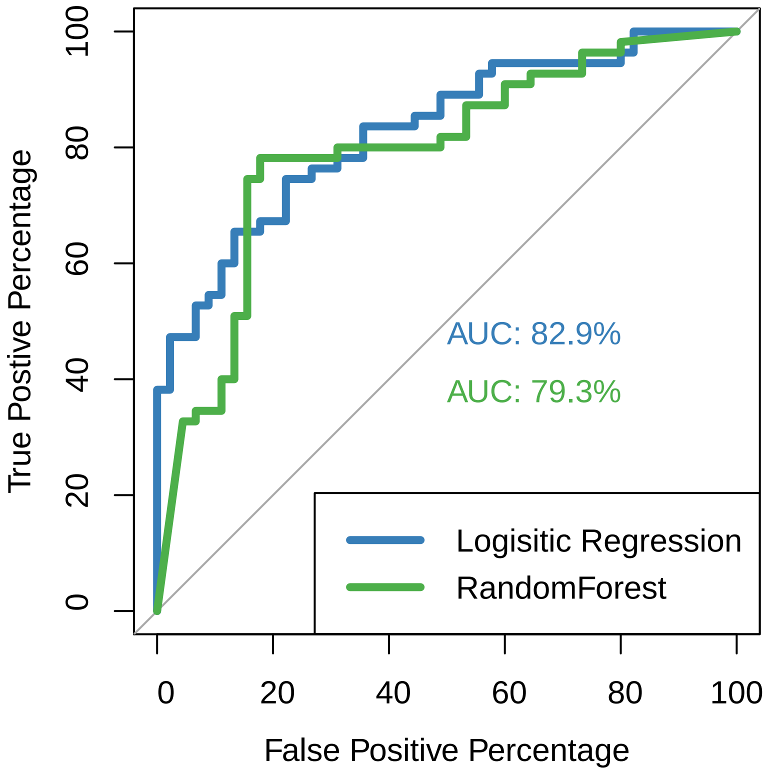
```
roc(obese, glm.fit$fitted.values, plot=TRUE, legacy.axes=TRUE, percent=TRUE,
    xlab="False Positive Percentage", ylab="True Postive Percentage", col="#377eb8", lwd=4,
    print.auc=TRUE)
```

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

## Setting direction: controls < cases

##
## Call:
## roc.default(response = obese, predictor = glm.fit$fitted.values,     percent = TRUE, plot = TRUE, leg
##
## Data: glm.fit$fitted.values in 45 controls (obese 0) < 55 cases (obese 1).
## Area under the curve: 82.91%
```

```
plot.roc(obese, rf.model$votes[,1], percent=TRUE, col="#4daf4a", lwd=4,
         print.auc=TRUE, add=TRUE, print.auc.y=40)
```

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

## Setting direction: controls > cases
```

```
legend("bottomright", legend=c("Logisitic Regression","RandomForest"),
       col=c("#377eb8","#4daf4a"), lwd=4)
```



Session information

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
```

```
## [1] randomForest_4.6-14 pROC_1.15.3         knitr_1.23
## [4] devtools_2.1.0      usethis_1.5.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.1        magrittr_1.5      pkgload_1.0.2
##  [4] R6_2.4.0          rlang_0.4.0       plyr_1.8.4
##  [7] stringr_1.4.0     tools_3.6.1       pkgbuild_1.0.5
## [10] xfun_0.8          sessioninfo_1.1.1 cli_1.1.0
## [13] withr_2.1.2       remotes_2.1.0     htmltools_0.3.6
## [16] rprojroot_1.3-2   yaml_2.2.0        digest_0.6.20
## [19] assertthat_0.2.1  crayon_1.3.4      processx_3.4.1
## [22] callr_3.3.1       fs_1.3.1          ps_1.3.0
## [25] testthat_2.1.1    memoise_1.1.0     glue_1.3.1
## [28] evaluate_0.14     rmarkdown_1.14    stringi_1.4.3
## [31] compiler_3.6.1    backports_1.1.4   desc_1.2.0
## [34] prettyunits_1.0.2
```

The document was processed on 2019-09-30.