Critical Thinking Group 4 - HW2

Sreejaya, Suman, Vuthy

September 27, 2016

1. Download the classification output data set

```
classification.orig <- read.csv("https://raw.githubusercontent.com/Nguyver/DATA621-HW/master/HW2/classi
    header = TRUE, sep = ",", stringsAsFactors = FALSE)
```

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

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

Use the table() function to get 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?

In our case, the columns are the predictions and the rows are the actuals.

```
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
```

```
class.df <- classification.orig
class.df$class <- factor(class.df$class)
class.df$scored.class <- factor(class.df$scored.class)

glimpse(class.df)
```

```
## Observations: 181
## Variables: 11
## $ pregnant           <int> 7, 2, 3, 1, 4, 1, 9, 8, 1, 2, 5, 5, 13, 0, ...
## $ glucose            <int> 124, 122, 107, 91, 83, 100, 89, 120, 79, 12...
## $ diastolic          <int> 70, 76, 62, 64, 86, 74, 62, 78, 60, 48, 78,...
## $ skinfold           <int> 33, 27, 13, 24, 19, 12, 0, 0, 42, 32, 30, 4...
## $ insulin            <int> 215, 200, 48, 0, 0, 46, 0, 0, 48, 165, 0, 7...
## $ bmi                <dbl> 25.5, 35.9, 22.9, 29.2, 29.3, 19.5, 22.5, 2...
## $ pedigree           <dbl> 0.161, 0.483, 0.678, 0.192, 0.317, 0.149, 0...
## $ age                <int> 37, 26, 23, 21, 34, 28, 33, 64, 23, 26, 37,...
## $ class              <fctr> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ scored.class       <fctr> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ scored.probability <dbl> 0.32845226, 0.27319044, 0.10966039, 0.05599...
```

```r
summary(class.df)
```

```
##     pregnant         glucose        diastolic        skinfold
##  Min.   : 0.000   Min.   : 57.0   Min.   : 38.0   Min.   : 0.0
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 64.0   1st Qu.: 0.0
##  Median : 3.000   Median :112.0   Median : 70.0   Median :22.0
##  Mean   : 3.862   Mean   :118.3   Mean   : 71.7   Mean   :19.8
##  3rd Qu.: 6.000   3rd Qu.:136.0   3rd Qu.: 78.0   3rd Qu.:32.0
##  Max.   :15.000   Max.   :197.0   Max.   :104.0   Max.   :54.0
##     insulin            bmi           pedigree          age        class
##  Min.   :  0.00   Min.   :19.40   Min.   :0.0850   Min.   :21.00   0:124
##  1st Qu.:  0.00   1st Qu.:26.30   1st Qu.:0.2570   1st Qu.:24.00   1: 57
##  Median :  0.00   Median :31.60   Median :0.3910   Median :30.00
##  Mean   : 63.77   Mean   :31.58   Mean   :0.4496   Mean   :33.31
##  3rd Qu.:105.00   3rd Qu.:36.00   3rd Qu.:0.5800   3rd Qu.:41.00
##  Max.   :543.00   Max.   :50.00   Max.   :2.2880   Max.   :67.00
##  scored.class scored.probability
##  0:149        Min.   :0.02323
##  1: 32        1st Qu.:0.11702
##               Median :0.23999
##               Mean   :0.30373
##               3rd Qu.:0.43093
##               Max.   :0.94633
```

```r
table(class.df$class, class.df$scored.class)
```

```
##
##       0   1
##   0 119   5
##   1  30  27
```

```r
# Rows represent Actual class and the columns represent Predicted class.

# Verifying Table results = Predict-Actual True Positive = 1-1 False Positive =
# 1-0 True Negative = 0-0 False Negative = 0-1
sum(class.df$class == class.df$scored.class & class.df$class == 1)  #TP (2,2)
```

```
## [1] 27
```

```r
sum(class.df$class != class.df$scored.class & class.df$class == 0)  #FN (1,2)
```

```
## [1] 5
```

```r
sum(class.df$class == class.df$scored.class & class.df$class == 0)  #TN (1,1)
```

```
## [1] 119
```

```r
sum(class.df$class != class.df$scored.class & class.df$class == 1)  #FP (2,1)
```

```
## [1] 30
```

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.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Accuracy: Overall, how often is the classifier correct?

```
accuracy <- function(data, actual.idx = 9, predict.idx = 10) {
    confusion <- table(data[, actual.idx], data[, predict.idx])

    value <- (confusion[2, 2] + confusion[1, 1])/(confusion[2, 2] + confusion[2,
        1] + confusion[1, 1] + confusion[1, 2])

    return(value)
}

# Example
accuracy(class.df, 9, 10)
```

```
## [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.

$$ClassificationErrorRate = \frac{FP + FN}{TP + FP + TN + FN}$$

Verify that you get an accuracy and error rate that sums to one.

Error Rate: Overall, how often is it wrong?

```
classErrorRate <- function(data, actual.idx = 9, predict.idx = 10) {
    confusion <- table(data[, actual.idx], data[, predict.idx])

    value <- (confusion[1, 2] + confusion[2, 1])/(confusion[2, 2] + confusion[2,
        1] + confusion[1, 1] + confusion[1, 2])

    return(value)
}

classErrorRate(class.df) + accuracy(class.df)
```

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

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.

$$Precision = \frac{TP}{TP + FP}$$

```
precision <- function(data, actual.idx = 9, predict.idx = 10) {
    confusion <- table(data[, actual.idx], data[, predict.idx])

    value <- confusion[2, 2]/(confusion[2, 2] + confusion[1, 2])

    return(value)
}

precision(class.df)
```

```
## [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.

$$Sensitivity = \frac{TP}{TP + FN}$$

```
sensitivity <- function(data, actual.idx = 9, predict.idx = 10) {
    confusion <- table(data[, actual.idx], data[, predict.idx])

    value <- confusion[2, 2]/(confusion[2, 2] + confusion[2, 1])

    return(value)
}

sensitivity(class.df)
```

```
## [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.

$$Specificity = \frac{TN}{TN + FP}$$

```
specificity <- function(data, actual.idx = 9, predict.idx = 10) {


    confusion <- table(data[, actual.idx], data[, predict.idx])

    value <- confusion[1, 1]/(confusion[1, 1] + confusion[1, 2])

    return(value)
}

specificity(class.df)
```

```
## [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.

$$F1Score = \frac{2XPrecisionXSensitivity}{Precision + Sensitivity}$$

```
f1Score <- function(data, actual.idx = 9, predict.idx = 10) {
    confusion <- table(data[, actual.idx], data[, predict.idx])

    value <- (2 * precision(data) * sensitivity(data))/(precision(data) + sensitivity(data))

    return(value)
}

f1Score(class.df)
```

```
## [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 $ab < a$.)

```
# F1 score reaches its best value at 1 and worst at 0. since both presition and
# sensitivity are in between 0 and 1 we know 0 < p*s <=1 and 2*p*s <=p +s
# (maximum value p and s can have is 1 ) so 2*p*s/p +s can have values in between
# 0 and 1
```

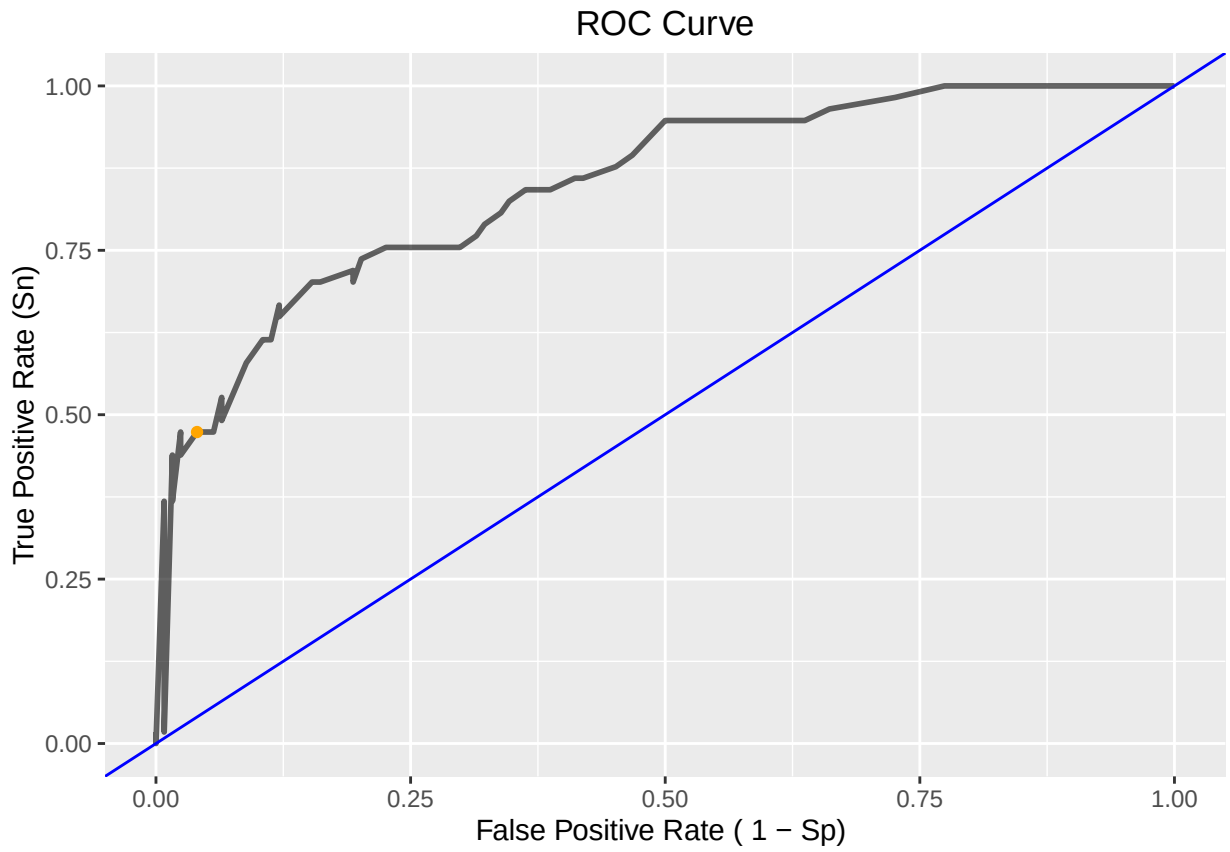
10. Write a function that generates an ROC curve from a data set with a true classification column (class in our example) and a probability column (scored.probability in our example). Your function should return a list that includes the plot of the ROC curve and a vector that contains the calculated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

```
library(ggplot2)
library(dplyr)


getRoCData <- function(data) {

    rocResults <- data.frame(Threshold = c(), Sn = c(), Sp = c())
    Sn.vec <- c()
    Sp.vec <- c()
    thresholds <- seq(0, 1, by = 0.01)
    for (threshold in thresholds) {

        data <- data %>% mutate(predicted = as.numeric(scored.probability >= threshold))
        data$predicted <- factor(data$predicted, levels = c(0, 1))

        Sn <- sensitivity(data, actual.idx = 9, predict.idx = 12)
        Sp <- specificity(data, actual.idx = 9, predict.idx = 12)
        Sn.vec <- append(Sn, Sn.vec)
        Sp.vec <- append(Sp, Sp.vec)

        # FPR vs. TPR

        rocResults <- rbind(rocResults, data.frame(Threshold = c(threshold), Sn = c(Sn),
            Sp = c(Sp)))
    }
```

```
    rocPlot <- ggplot(rocResults, aes(x = 1 - Sp, y = Sn)) + geom_line(size = 1,
        alpha = 0.6) + geom_point(data = rocResults[rocResults$Threshold == 0.5,
        ], aes(x = 1 - Sp, y = Sn), colour = "orange") + geom_abline(slope = 1, intercept = 0,
        colour = "blue", size = 0.5) + xlim(0, 1) + ylim(0, 1) + labs(title = "ROC Curve",
        x = "False Positive Rate ( 1 - Sp) ", y = "True Positive Rate (Sn)")

    # TODO - How to calculate the area under the curve ?
    fpr <- 1 - Sp.vec
    auc <- cumsum(Sn.vec - fpr) * 0.01
    print(auc)

    # or
    height = (Sn.vec[-1] + Sn.vec[-length(Sn.vec)])/2
    width = diff(fpr)  # = diff(rev(fpr))
    auc <- sum(height * width)
    # auc <- c()
    results <- list(rocPlot, auc)

    return(results)
}

getRoCData(class.df)
```

```
##   [1] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##   [6] 0.0000000000 0.0001754386 0.0003508772 0.0005263158 0.0007017544
##  [11] 0.0008771930 0.0009719864 0.0015930956 0.0022142049 0.0030107527
##  [16] 0.0039827391 0.0051301641 0.0064530277 0.0077758913 0.0092741935
##  [21] 0.0107724958 0.0122707980 0.0139445388 0.0156182796 0.0174674590
##  [26] 0.0193166384 0.0211658178 0.0230149972 0.0250396152 0.0272396718
##  [31] 0.0296151669 0.0321661007 0.0350679117 0.0381451613 0.0413978495
##  [36] 0.0446505376 0.0479032258 0.0515067912 0.0550297114 0.0590789474
##  [41] 0.0631281834 0.0673528580 0.0715775325 0.0758022071 0.0799462366
##  [46] 0.0840902660 0.0884097340 0.0929046406 0.0972382569 0.1015718732
##  [51] 0.1059054895 0.1100778155 0.1143449349 0.1187874929 0.1234054895
##  [56] 0.1283078664 0.1332102434 0.1382074137 0.1432993775 0.1483106961
##  [61] 0.1535922467 0.1590492360 0.1645062252 0.1699915110 0.1753961517
##  [66] 0.1804782117 0.1855602716 0.1908177702 0.1961700623 0.2014558574
##  [71] 0.2064997170 0.2112209960 0.2157809847 0.2203551217 0.2250240521
##  [76] 0.2297071307 0.2344850028 0.2392770232 0.2439883984 0.2485384833
##  [81] 0.2530220713 0.2574250141 0.2616808149 0.2659507640 0.2704244482
##  [86] 0.2746561969 0.2786460102 0.2820713073 0.2851740238 0.2882102434
##  [91] 0.2907767402 0.2930348048 0.2948089983 0.2963412564 0.2973089983
##  [96] 0.2977928693 0.2981154499 0.2983573854 0.2983573854 0.2983573854
## [101] 0.2983573854
```

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




```
##
## [[2]]
## [1] 0.8488964
```

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

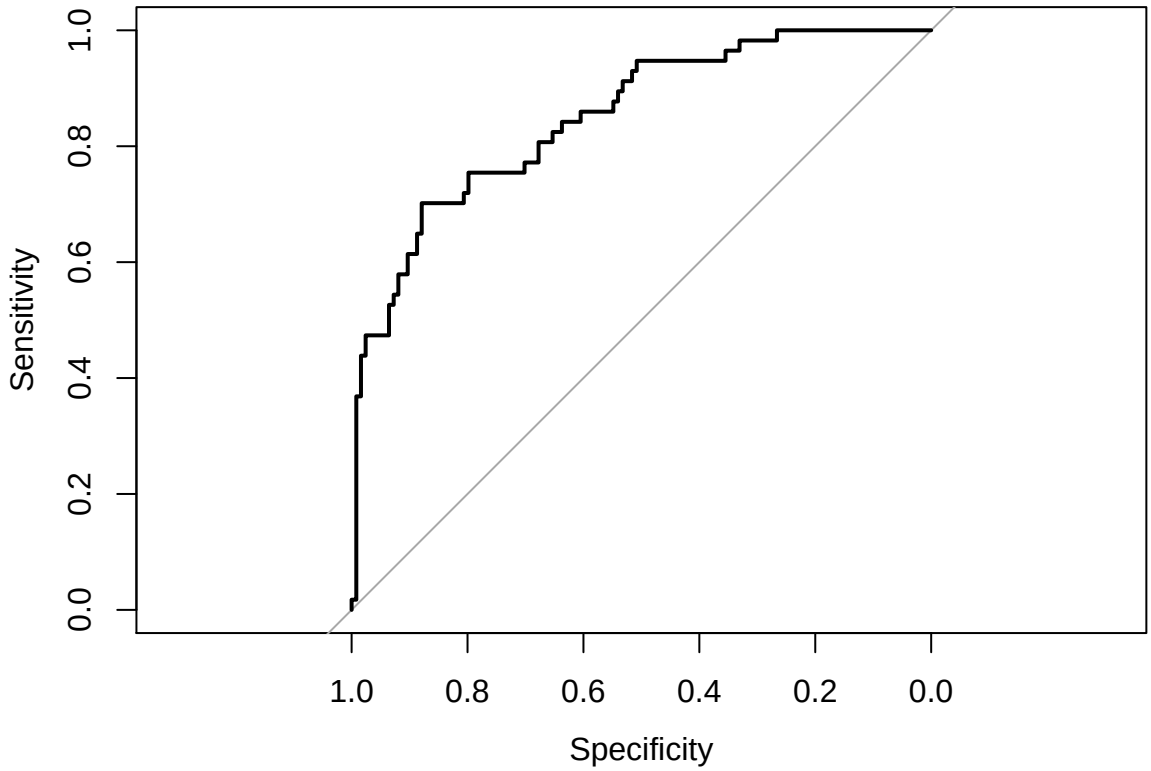
12. Investigate the **caret** package. In particular, consider the functions confusionMatrix, sensitivity, and specificity. Apply the functions to the data set. How do the results compare with your own functions?

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

```
library(pROC)
```

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

##
## Attaching package: 'pROC'

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

```
roc <- roc(factor(class) ~ scored.probability, data = , class.df, plot = TRUE, ci = TRUE)
plot(roc)
```




```
##
## Call:
## roc.formula(formula = factor(class) ~ scored.probability, data = class.df,     plot = TRUE, ci = TRU
##
## Data: scored.probability in 124 controls (factor(class) 0) < 57 cases (factor(class) 1).
## Area under the curve: 0.8503
## 95% CI: 0.7905-0.9101 (DeLong)
```

```
auc(factor(class) ~ scored.probability, class.df)
```

```
## Area under the curve: 0.8503
```

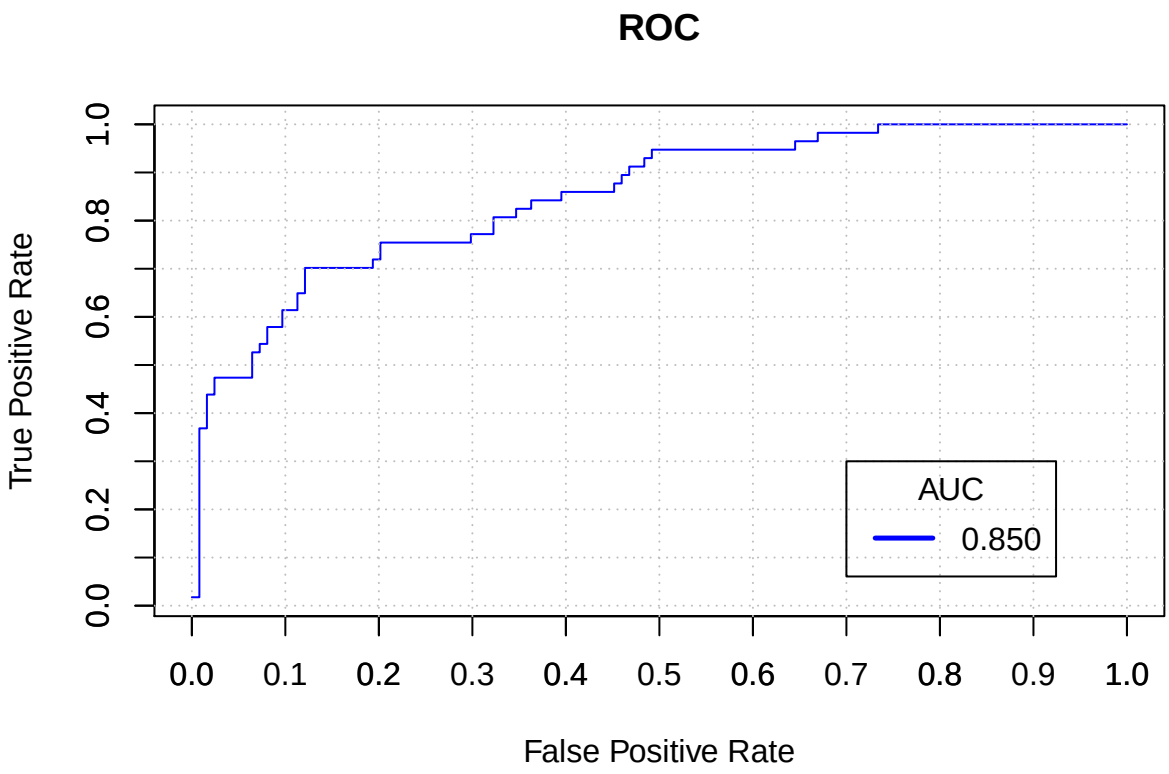
```
## discussion?? x axis should be 1- specificity
```

```
true_Y = class.df$class
probs = class.df$scored.probability

getROC_AUC = function(probs, true_Y) {
    probsSort = sort(probs, decreasing = TRUE, index.return = TRUE)
    val = unlist(probsSort$x)
```

```
    idx = unlist(probsSort$ix)

    roc_y = true_Y[idx]
    stack_x = cumsum(roc_y == 0)/sum(roc_y == 0)
    stack_y = cumsum(roc_y == 1)/sum(roc_y == 1)

    auc = sum((stack_x[2:length(roc_y)] - stack_x[1:length(roc_y) - 1]) * stack_y[2:length(roc_y)])
    return(list(stack_x = stack_x, stack_y = stack_y, auc = auc))
}

aList = getROC_AUC(probs, true_Y)

stack_x = unlist(aList$stack_x)
stack_y = unlist(aList$stack_y)
auc = unlist(aList$auc)
auc
```

```
## [1] 0.8503113
```

```
plot(stack_x, stack_y, type = "l", col = "blue", xlab = "False Positive Rate", ylab = "True Positive Ra
    main = "ROC")
axis(1, seq(0, 1, 0.1))
axis(2, seq(0, 1, 0.1))
abline(h = seq(0, 1, 0.1), v = seq(0, 1, 0.1), col = "gray", lty = 3)
legend(0.7, 0.3, sprintf("%3.3f", auc), lty = c(1, 1), lwd = c(2.5, 2.5), col = "blue",
    title = "AUC")
```


ROC
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
AUC
0.850