Customed Wrapper

Yehu Chen

2018/1/28

```r
library(nnls)
library(SuperLearner)
```

```
## Super Learner

## Version: 2.0-22

## Package created on 2017-07-18
```

```r
# Review available models.
listWrappers()
```

```
## All prediction algorithm wrappers in SuperLearner:

##  [1] "SL.bartMachine"      "SL.bayesglm"         "SL.biglasso"
##  [4] "SL.caret"            "SL.caret.rpart"      "SL.cforest"
##  [7] "SL.dbarts"           "SL.earth"            "SL.extraTrees"
## [10] "SL.gam"              "SL.gbm"              "SL.glm"
## [13] "SL.glm.interaction"  "SL.glmnet"           "SL.ipredbagg"
## [16] "SL.kernelKnn"        "SL.knn"              "SL.ksvm"
## [19] "SL.lda"              "SL.leekasso"         "SL.lm"
## [22] "SL.loess"            "SL.logreg"           "SL.mean"
## [25] "SL.nnet"             "SL.nnls"             "SL.polymars"
## [28] "SL.qda"              "SL.randomForest"     "SL.ranger"
## [31] "SL.ridge"            "SL.rpart"            "SL.rpartPrune"
## [34] "SL.speedglm"         "SL.speedlm"          "SL.step"
## [37] "SL.step.forward"     "SL.step.interaction" "SL.stepAIC"
## [40] "SL.svm"              "SL.template"         "SL.xgboost"

##
## All screening algorithm wrappers in SuperLearner:

## [1] "All"
## [1] "screen.corP"           "screen.corRank"        "screen.glmnet"
## [4] "screen.randomForest"   "screen.SIS"            "screen.template"
## [7] "screen.ttest"          "write.screen.template"
```

Read binomial dataset

Separate data into 50% training, 25% holdout validation and 25% test sets

```r
filename = "Binomial_dataset_5_final.txt"
data = read.csv(filename)

# First divide data into positive and negative sets
data_pos = subset(data,Ytemp==1)
```

```r
data_neg = subset(data,Ytemp==0)

# select the 50%, 25%, 25% to be training, holdout and testing data
data_train_pos = data_pos[c(2:as.integer(nrow(data_pos)/2)),]
data_holdout_pos = data_pos[c(as.integer(nrow(data_pos)/2+1):as.integer(3*nrow(data_pos)/4)),]
data_test_pos = data_pos[c(as.integer(3*nrow(data_pos)/4+1):nrow(data_pos)),]

data_train_neg = data_neg[c(2:as.integer(nrow(data_neg)/2)),]
data_holdout_neg = data_neg[c(as.integer(nrow(data_neg)/2+1):as.integer(3*nrow(data_neg)/4)),]
data_test_neg = data_neg[c(as.integer(3*nrow(data_neg)/4+1):nrow(data_neg)),]

# stack the pos and neg datasets
data_train = rbind(data_train_pos,data_train_neg)
data_holdout = rbind(data_holdout_pos,data_holdout_neg)
data_test = rbind(data_test_pos,data_test_neg)

# split datasets to X and Y
X_train = data_train[,c(2:ncol(data_train))]
X_test = data_test[,c(2:ncol(data_train))]
X_holdout = data_holdout[,c(2:ncol(data_train))]
Y_train = data_train[,c(1)]
Y_test = data_test[,c(1)]
Y_holdout = data_holdout[,c(1)]
```

Notice that the final training, holdout, and testing Y are still imbalanced.

We only split the dataset to training, holdout, and testing in the same proportion as the original dataset. This might require extra work, but can eliminate the pitfall where the data is imbalanced to positive but the training data is imbalanced to negative.

Customize SVM wrappers

```r
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```r
library(lattice)
library(ggplot2)
SL.svm.1 = function(..., kernel = "linear") {
  SL.svm(..., kernel = kernel)
}

SL.svm.2 = function(..., kernel = "polynomial", degree = 2, coef0 = 1) {
  SL.svm(..., kernel = kernel, degree = degree, coef0 = coef0)
}
```

```r
SL.svm.3 = function(..., kernel = "polynomial", degree = 2, coef0 = 10) {
  SL.svm(..., kernel = kernel, degree = degree, coef0 = coef0)
}

SL.svm.4 = function(..., kernel = "polynomial", degree = 4, coef0 = 1) {
  SL.svm(..., kernel = kernel, degree = degree, coef0 = coef0)
}

SL.svm.5 = function(..., kernel = "polynomial", degree = 4, coef0 = 10) {
  SL.svm(..., kernel = kernel, degree = degree, coef0 = coef0)
}

SL.svm.6 = function(..., kernel = "sigmoid",coef0 = 0) {
  SL.svm(..., kernel = kernel,coef0 = coef0)
}

SL.svm.7 = function(..., kernel = "sigmoid",coef0 = 1) {
  SL.svm(..., kernel = kernel,coef0 = coef0)
}

SL.svm.10 = function(...,type.class = "C-classification", cost = 3) {
  SL.svm(...,type.class = type.class, cost = cost )
}

SL.svm.11 = function(..., kernel = "radial", coef0 = 1) {
  SL.svm(..., kernel = kernel, coef0 = coef0)
}

SL.svm.12 = function(..., kernel = "radial", coef0 = 10) {
  SL.svm(..., kernel = kernel, coef0 = coef0)
}
SL.glmnet.0 <- function(..., alpha = 0,family="binomial"){
  SL.glmnet(..., alpha = alpha , family = family)
}

SL.glmnet.1 <- function(..., alpha = 1,family="binomial"){
  SL.glmnet(..., alpha = alpha , family = family)
}

SL.glmnet.0.25 <- function(..., alpha = 0.25,family="binomial"){
  SL.glmnet(..., alpha = alpha, family = family)
}

SL.glmnet.0.50 <- function(..., alpha = 0.50,family="binomial"){
  SL.glmnet(..., alpha = alpha, family = family)
}

SL.glmnet.0.75 <- function(..., alpha = 0.75,family="binomial"){
  SL.glmnet(..., alpha = alpha, family = family)
}


my_library = c("SL.svm.4",
               "SL.svm.1","SL.svm.2","SL.svm.3",
"SL.svm.5","SL.svm.6","SL.svm.12",
"SL.svm.7","SL.glmnet","SL.svm.10","SL.svm.11",
               "SL.glmnet.0","SL.glmnet.1","SL.glmnet.0.50",
               "SL.glmnet.0.25","SL.glmnet.0.75",
               "SL.knn","SL.randomForest","SL.lm","SL.mean","SL.glmnet","SL.glm","SL.nnls")
```

The values we are changing are nu and cost(C)

We are dealing with an almost balanced dataset, so we leave the class weights as default.

Fit cv.superlearner

```
sl = SuperLearner(Y = Y_train, X = X_train, family = binomial(),
                  SL.library = my_library)
```

```
## Loading required package: glmnet

## Loading required package: Matrix

## Loading required package: foreach

## Loaded glmnet 2.0-13

## Loading required package: class

## Loading required package: randomForest

## randomForest 4.6-12

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## Loading required package: e1071

## Warning in predict.lm(fit, newdata = newX, type = "response"): prediction
## from a rank-deficient fit may be misleading

## Warning: glm.fit: algorithm did not converge

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(fit, newdata = newX, type = "response"): prediction
## from a rank-deficient fit may be misleading

## Warning: glm.fit: algorithm did not converge

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
## Warning in predict.lm(fit, newdata = newX, type = "response"): prediction
## from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
## Warning in predict.lm(fit, newdata = newX, type = "response"): prediction
## from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
## Warning in predict.lm(fit, newdata = newX, type = "response"): prediction
## from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
## Warning in predict.lm(fit, newdata = newX, type = "response"): prediction
## from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
## Warning in predict.lm(fit, newdata = newX, type = "response"): prediction
## from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
## Warning in predict.lm(fit, newdata = newX, type = "response"): prediction
## from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
## Warning in predict.lm(fit, newdata = newX, type = "response"): prediction
## from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

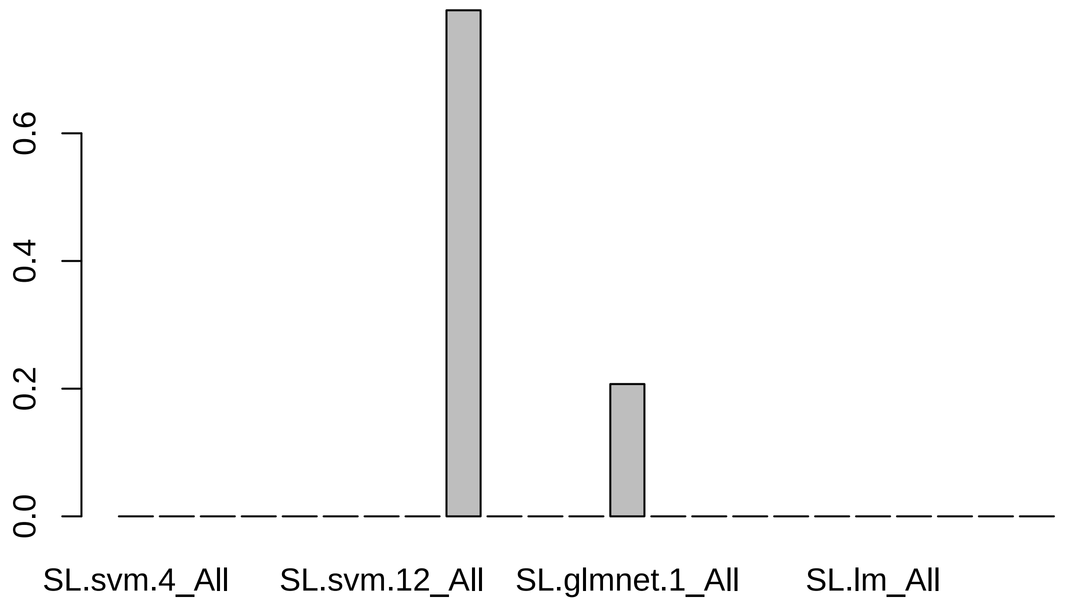
```
## Warning in predict.lm(fit, newdata = newX, type = "response"): prediction
## from a rank-deficient fit may be misleading
## Warning: glm.fit: algorithm did not converge

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
# Review results.
```

```
sl
```

```
##
## Call:
## SuperLearner(Y = Y_train, X = X_train, family = binomial(), SL.library = my_library)
##
##
##
##                           Risk      Coef
## SL.svm.4_All         0.2590367 0.0000000
## SL.svm.1_All         0.2573252 0.0000000
## SL.svm.2_All         0.2576456 0.0000000
## SL.svm.3_All         0.2578379 0.0000000
## SL.svm.5_All         0.2569211 0.0000000
## SL.svm.6_All         0.2639106 0.0000000
## SL.svm.12_All        0.2614928 0.0000000
## SL.svm.7_All         0.2612267 0.0000000
## SL.glmnet_All        0.1405737 0.7927469
## SL.svm.10_All        0.2619820 0.0000000
## SL.svm.11_All        0.2556844 0.0000000
## SL.glmnet.0_All      0.2593691 0.0000000
## SL.glmnet.1_All      0.1411372 0.2072531
## SL.glmnet.0.50_All   0.1611375 0.0000000
## SL.glmnet.0.25_All   0.1911410 0.0000000
## SL.glmnet.0.75_All   0.1469679 0.0000000
## SL.knn_All           0.2814714 0.0000000
## SL.randomForest_All  0.2454540 0.0000000
## SL.lm_All            0.5180213 0.0000000
## SL.mean_All          0.2525362 0.0000000
## SL.glmnet_All        0.1416455 0.0000000
## SL.glm_All           0.5267904 0.0000000
## SL.nnls_All          0.4880549 0.0000000
```

```
barplot(coef(sl))
```



```r
pred = predict(sl, X_holdout, onlySL = T)

# Check the structure of this prediction object.
str(pred)
```

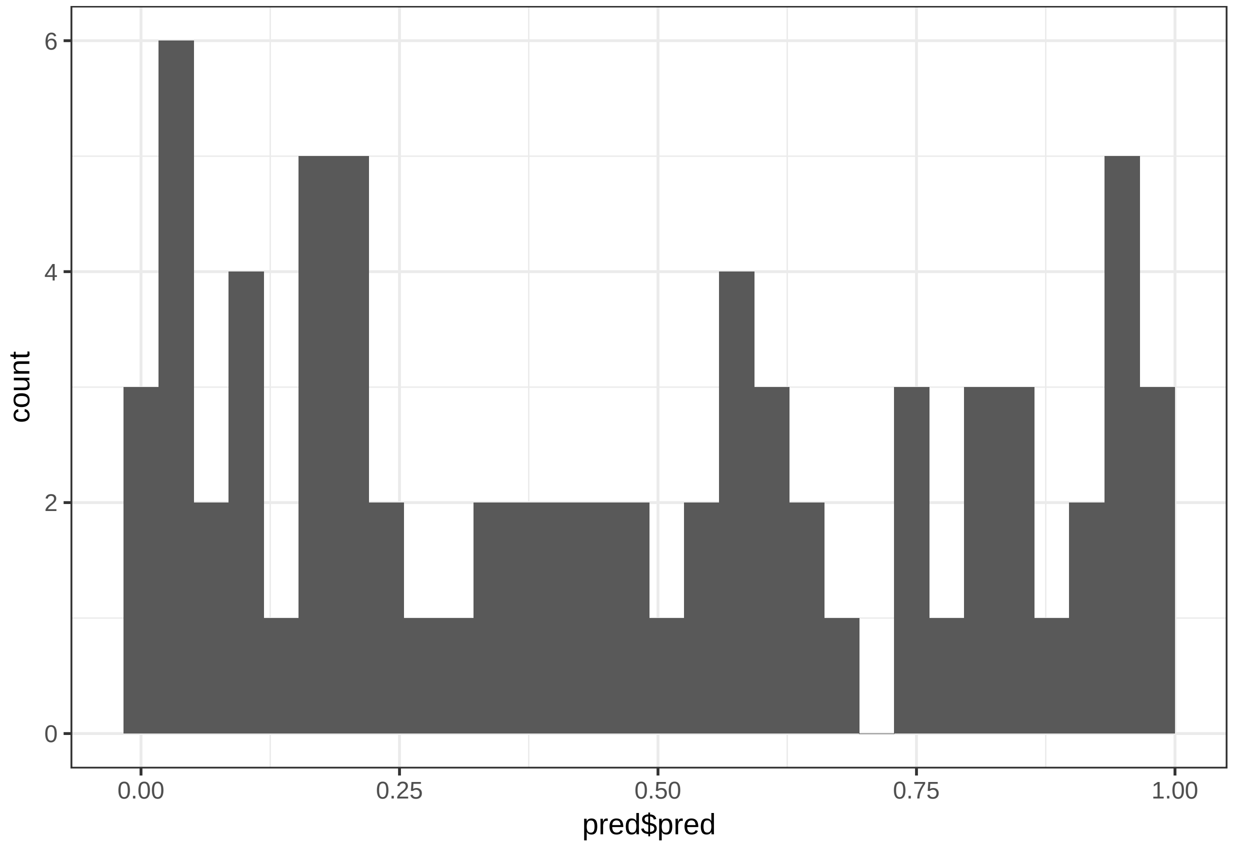
```
## List of 2
##  $ pred           : num [1:74, 1] 0.608 0.453 0.373 0.801 0.899 ...
##  $ library.predict: num [1:74, 1:23] 0 0 0 0 0 0 0 0 0 0 ...
```

```r
# We can see which columns are being populated the library.predict.
summary(pred$library.predict)
```

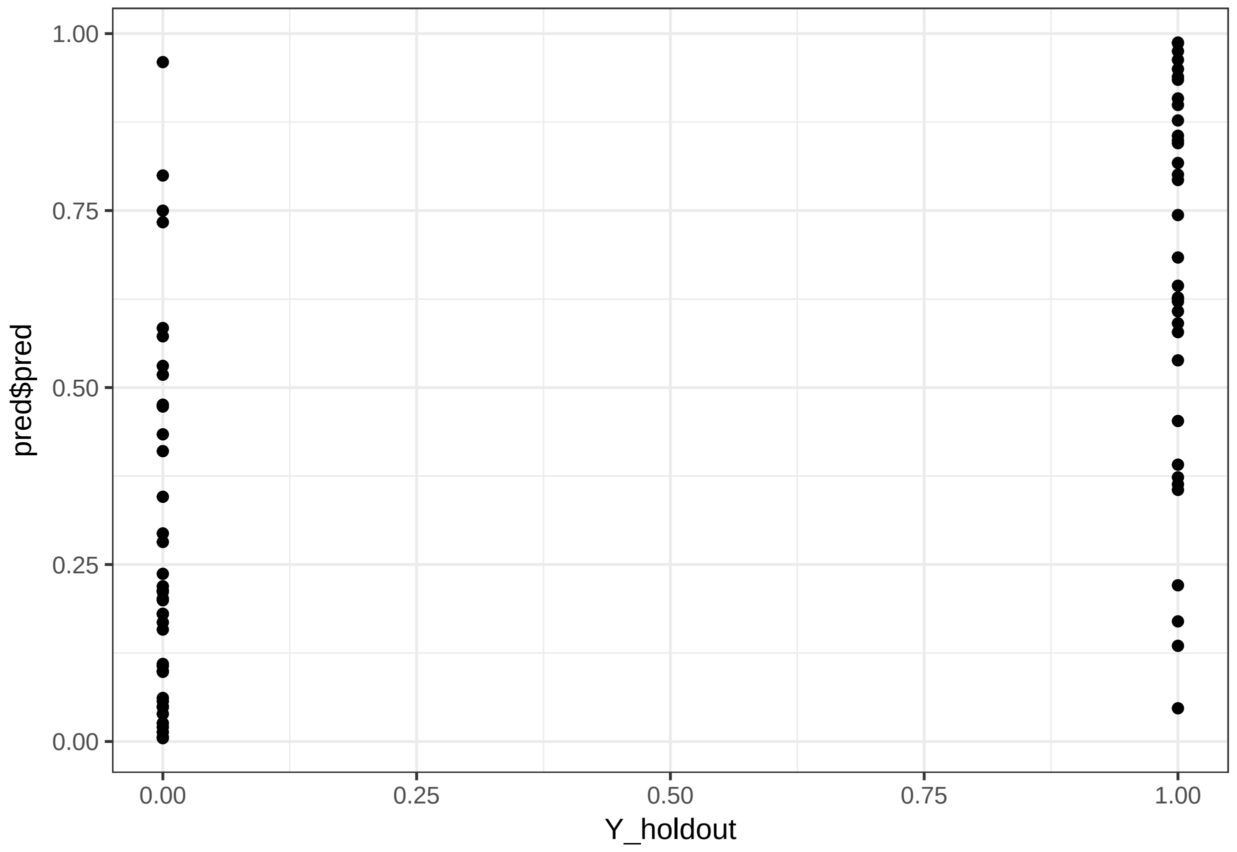
```
##        V1           V2           V3           V4           V5           V6
##  Min.   :0   Min.   :0   Min.   :0   Min.   :0   Min.   :0   Min.   :0
##  1st Qu.:0   1st Qu.:0   1st Qu.:0   1st Qu.:0   1st Qu.:0   1st Qu.:0
##  Median :0   Median :0   Median :0   Median :0   Median :0   Median :0
##  Mean   :0   Mean   :0   Mean   :0   Mean   :0   Mean   :0   Mean   :0
##  3rd Qu.:0   3rd Qu.:0   3rd Qu.:0   3rd Qu.:0   3rd Qu.:0   3rd Qu.:0
##  Max.   :0   Max.   :0   Max.   :0   Max.   :0   Max.   :0   Max.   :0
##        V7           V8           V9                V10          V11
##  Min.   :0   Min.   :0   Min.   :0.003555   Min.   :0   Min.   :0
##  1st Qu.:0   1st Qu.:0   1st Qu.:0.166389   1st Qu.:0   1st Qu.:0
##  Median :0   Median :0   Median :0.440361   Median :0   Median :0
##  Mean   :0   Mean   :0   Mean   :0.458905   Mean   :0   Mean   :0
##  3rd Qu.:0   3rd Qu.:0   3rd Qu.:0.764672   3rd Qu.:0   3rd Qu.:0
##  Max.   :0   Max.   :0   Max.   :0.990171   Max.   :0   Max.   :0
##       V12          V13                V14          V15          V16
##  Min.   :0   Min.   :0.009035   Min.   :0   Min.   :0   Min.   :0
##  1st Qu.:0   1st Qu.:0.201552   1st Qu.:0   1st Qu.:0   1st Qu.:0
##  Median :0   Median :0.448802   Median :0   Median :0   Median :0
##  Mean   :0   Mean   :0.462887   Mean   :0   Mean   :0   Mean   :0
##  3rd Qu.:0   3rd Qu.:0.696155   3rd Qu.:0   3rd Qu.:0   3rd Qu.:0
##  Max.   :0   Max.   :0.976659   Max.   :0   Max.   :0   Max.   :0
##       V17          V18          V19          V20          V21          V22
##  Min.   :0   Min.   :0   Min.   :0   Min.   :0   Min.   :0   Min.   :0
##  1st Qu.:0   1st Qu.:0   1st Qu.:0   1st Qu.:0   1st Qu.:0   1st Qu.:0
##  Median :0   Median :0   Median :0   Median :0   Median :0   Median :0
##  Mean   :0   Mean   :0   Mean   :0   Mean   :0   Mean   :0   Mean   :0
##  3rd Qu.:0   3rd Qu.:0   3rd Qu.:0   3rd Qu.:0   3rd Qu.:0   3rd Qu.:0
##  Max.   :0   Max.   :0   Max.   :0   Max.   :0   Max.   :0   Max.   :0
##       V23
##  Min.   :0
##  1st Qu.:0
##  Median :0
##  Mean   :0
##  3rd Qu.:0
##  Max.   :0
```

```r
# Histogram of our predicted values.
qplot(pred$pred) + theme_bw()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
# Scatterplot of original values (0, 1) and predicted values.
# Ideally we would use jitter or slight transparency to deal with overlap.
qplot(Y_holdout, pred$pred) + theme_bw()
```



```
# Review AUC - Area Under Curve
pred_rocr = ROCR::prediction(pred$pred, Y_holdout)
auc = ROCR::performance(pred_rocr, measure = "auc", x.measure = "cutoff")@y.values[[1]]
auc
```

```
## [1] 0.8461538
```

The auroc value is 0.86, which is pretty good.