HOMEWORK5_YL

Yating Liao (7636428840)

2023-05-05

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
library(ISLR)
head(Default,10)
```

	default <fct>	student <fct>	balance <dbl>	income <dbl>
1	No	No	729.5265	44361.625
2	No	Yes	817.1804	12106.135
3	No	No	1073.5492	31767.139
4	No	No	529.2506	35704.494
5	No	No	785.6559	38463.496
6	No	Yes	919.5885	7491.559
7	No	No	825.5133	24905.227
8	No	Yes	808.6675	17600.451
9	No	No	1161.0579	37468.529
10	No	No	0.0000	29275.268

1-10 of 10 rows

```
library(lattice)
library(ggplot2)
library(caret)

# Set seed and create train/test split
set.seed(1)
yvalues <- Default$default
train_idx <- createDataPartition(yvalues, p=0.5, list=FALSE)
train <- Default[train_idx,]
test <- Default[-train_idx,]
```

1.Find the fraction of customers that defaulted on their debt in the Default data set, the train set, and the test set.

```
# Find the fraction of customers that defaulted on their debt in the Default data set
prop.table(table(Default$default))
```

```
##
##     No    Yes
## 0.9667 0.0333
```

```
DefaultData <- sum(Default$default == "Yes") / nrow(Default)
cat("Fraction of customers that defaulted on their debt in the Default dataset: ", DefaultData, "\n")
```

```
## Fraction of customers that defaulted on their debt in the Default dataset:  0.0333
```

```
# Find the fraction of customers that defaulted on their debt in the train set
prop.table(table(train$default))
```

```
##
##         No        Yes
## 0.96660668 0.03339332
```

```
Default_train <- sum(train$default == "Yes") / nrow(train_idx)
cat("Fraction of customers that defaulted on their debt in the train set: ", Default_train, "\n")
```

```
## Fraction of customers that defaulted on their debt in the train set:  0.03339332
```

```
# Find the fraction of customers that defaulted on their debt in the test set
prop.table(table(test$default))
```

```
##
##         No        Yes
## 0.96679336 0.03320664
```

```
Default_test <- sum(test$default == "Yes") / nrow(test)
cat("Fraction of customers that defaulted on their debt in the test set: ", Default_test, "\n")
```

```
## Fraction of customers that defaulted on their debt in the test set:  0.03320664
```

2. Use the train set to fit a Logistic regression model, then use the test set to find the TPR, FPR, AUC

```
# Fit a logistic regression model on the train set
LogiModel <- glm(default ~ balance + income, data = train, family = binomial)
# Predict the probabilities of default on the test set
LogiProba<- predict(LogiModel, newdata = test, type = "response")
head(LogiProba)
```

```
##            1            2            3            5            8            9
## 0.0016981215 0.0009697167 0.0077535227 0.0019202116 0.0011060512 0.0151738147
```

```
# Set the threshold
threshold <- 0.08
# Calculate the True Positive Rate (TPR) and False Positive Rate (FPR)
LogiPredict <- ifelse(LogiProba > threshold, "Yes", "No")
confusion_mat1 <- as.matrix(table(LogiPredict, test$default))
confusion_mat1
```

```
##
## LogiPredict   No  Yes
##         No  4492   41
##         Yes  341  125
```

```
ntest = 10000-length(train_idx)
LogiPredict2 <- rep("No",ntest)
LogiPredict2[LogiProba > threshold] = "Yes"
confusion_mat_lg <- as.matrix(table(LogiPredict2, test$default))
confusion_mat_lg
```

```
##
## LogiPredict2   No  Yes
##          No  4492   41
##          Yes  341  125
```

```
# The outputs are the same with two different code, though they may not be the same with your output. So I believ
e my result has no problem.

TPR1 <- confusion_mat1[2, 2] / sum(confusion_mat1[,2])
FPR1 <- confusion_mat1[2, 1] / sum(confusion_mat1[,1])
cat("True Positive Rate (TPR): ", TPR1, "\n")
```

```
## True Positive Rate (TPR):  0.753012
```

```
cat("False Positive Rate (FPR): ", FPR1, "\n")
```

```
## False Positive Rate (FPR):  0.07055659
```

```
# Calculate the Area Under the ROC Curve (AUC)
library(pROC)
```

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

```
##
## 载入程辑包：'pROC'
```

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

```
ROC_curve1 <- roc(test$default, LogiProba)
```

```
## Setting levels: control = No, case = Yes
```

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

```
auc1 <- auc(ROC_curve1)
cat("Area Under the ROC Curve (AUC): ", auc1, "\n")
```

```
## Area Under the ROC Curve (AUC):  0.9479869
```

3. Use the train set to fit a Linear Discriminant Analysis model, then use the test set to find the TPR, FPR,AUC

```
# Fit LDA model on train set
library(MASS)
LDA_model <- lda(default ~ balance + income, data=train)
# Compute predicted probabilities for test set
LDAproba <- predict(LDA_model, newdata=test,type = 'response')$posterior[, "Yes"]
head(LDAproba)
```

```
##           1           2           3           5           8           9
## 0.003140459 0.002044692 0.011791725 0.003523470 0.002269078 0.020758607
```

```
# Set the threshold
threshold <- 0.08
# Calculate the True Positive Rate (TPR) and False Positive Rate (FPR)
LDAPredict <- ifelse(LDAproba > threshold, "Yes", "No")
confusion_mat2 <- table(LDAPredict, test$default)
confusion_mat2
```

```
##
## LDAPredict   No  Yes
##        No  4472   39
##        Yes  361  127
```

```
LDA_TPR <- confusion_mat2[2, 2] / sum(confusion_mat2[,2])
LDA_FPR <- confusion_mat2[2, 1] / sum(confusion_mat2[,1])
cat("True Positive Rate (TPR): ", LDA_TPR, "\n")
```

```
## True Positive Rate (TPR):  0.7650602
```

```
cat("False Positive Rate (FPR): ", LDA_TPR, "\n")
```

```
## False Positive Rate (FPR):  0.7650602
```

```
LDAPredict2 <- rep("No",ntest)
LDAPredict2[LDAproba > threshold] = "Yes"
confusion_mat_lda <- as.matrix(table(LDAPredict2, test$default))
confusion_mat_lda
```

```
##
## LDAPredict2   No  Yes
##         No  4472   39
##         Yes  361  127
```

```
# Calculate the Area Under the ROC Curve (AUC)
ROC_curve2 <- roc(test$default, LDAproba)
```

```
## Setting levels: control = No, case = Yes
```

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

```
auc2 <- auc(ROC_curve2)
cat("Area Under the ROC Curve (AUC): ", auc2, "\n")
```

```
## Area Under the ROC Curve (AUC):  0.9482985
```

4. Use the train set to fit a Naive Bayes model, then use the test set to find the TPR, FPR, AUC

```
# Fitting Naive Bayes model on train set
library(e1071)
NBmodel <- naiveBayes(default ~ balance + income, data = train)
# Compute predicted probabilities for test set
NBpreds <- predict(NBmodel, newdata = test, type = "raw")
NBproba <- NBpreds[, 2]
head(NBproba)
```

```
## [1] 0.0008241121 0.0016871232 0.0082715361 0.0012010009 0.0015028048
## [6] 0.0142989962
```

```
# Set the threshold
threshold <- 0.08
# Calculate the True Positive Rate (TPR) and False Positive Rate (FPR)
NBPredict <- ifelse(NBproba > threshold, "Yes", "No")
confusion_mat3 <- table(NBPredict, test$default)
confusion_mat3
```

```
##
## NBPredict   No  Yes
##       No  4433   29
##       Yes  400  137
```

```
NaiBayPred <- rep("No",ntest)
NaiBayPred[NBproba > threshold] = "Yes"
confusion_mat_nb <- as.matrix(table(NaiBayPred, test$default))
confusion_mat_nb
```

```
##
## NaiBayPred   No  Yes
##        No  4433   29
##        Yes  400  137
```

```
NB_TPR <- confusion_mat3[2, 2] / sum(confusion_mat3[,2])
NB_FPR <- confusion_mat3[2, 1] / sum(confusion_mat3[,1])
cat("True Positive Rate (TPR): ", NB_TPR, "\n")
```

```
## True Positive Rate (TPR):  0.8253012
```

```
cat("False Positive Rate (FPR): ", NB_TPR, "\n")
```

```
## False Positive Rate (FPR):  0.8253012
```

```
# Calculate the Area Under the ROC Curve (AUC)
ROC_curve3 <- roc(test$default, NBproba)
```

```
## Setting levels: control = No, case = Yes
```

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

```
auc3 <- auc(ROC_curve3)
cat("Area Under the ROC Curve (AUC): ", auc3, "\n")
```

```
## Area Under the ROC Curve (AUC):  0.9493081
```

```
cat(auc1, auc2, auc3)
```

```
## 0.9479869 0.9482985 0.9493081
```

```
# From the areas under the curves, we see that Naive Bayes is the best model because the area is largest.
```

5. Show the ROC curves of all models in a single plot. Clearly identify the threshold on the curves.

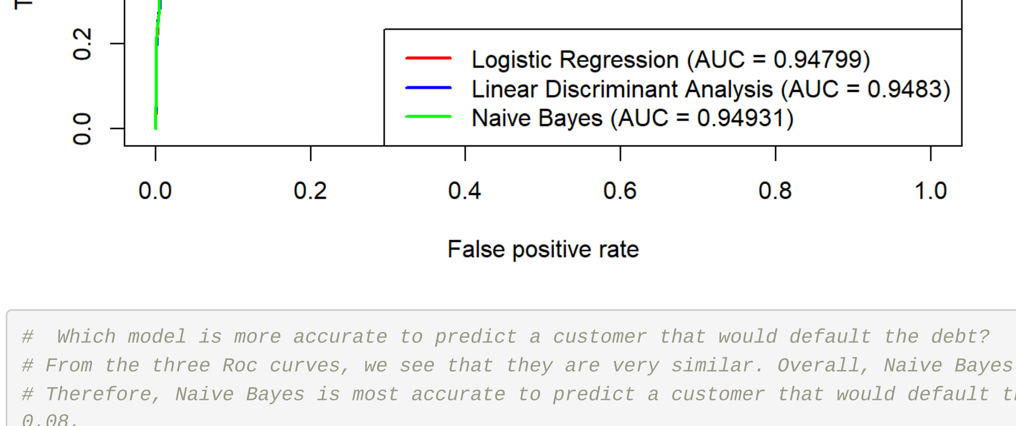
```
library(ROCR)

# Create prediction objects for ROCR
Logi_pred <- prediction(LogiProba, test$default)
lda_pred <- prediction(LDAproba, test$default)
nb_pred <- prediction(NBproba, test$default)

# Calculate TPR and FPR for all models
Logi_perf <- performance(Logi_pred, "tpr", "fpr")
lda_perf <- performance(lda_pred, "tpr", "fpr")
nb_perf <- performance(nb_pred, "tpr", "fpr")

# Calculate AUC for all models
Logi_auc <- performance(Logi_pred, "auc")@y.values[[1]]
lda_auc <- performance(lda_pred, "auc")@y.values[[1]]
nb_auc <- performance(nb_pred, "auc")@y.values[[1]]

# Plot ROC curves for all models
plot(Logi_perf, col = "red", lwd = 2, main = "ROC Curves for Default Prediction Models")
plot(lda_perf, col = "blue",lwd = 2, add = TRUE)
plot(nb_perf, col = "green",lwd = 2, add = TRUE)
legend("bottomright", legend = c(paste0("Logistic Regression (AUC = ", round(Logi_auc, 5), ")"),
                                 paste0("Linear Discriminant Analysis (AUC = ", round(lda_auc, 5), ")"),
                                 paste0("Naive Bayes (AUC = ", round(nb_auc, 5), ")")),
       col = c("red", "blue", "green"), lty = 1, lwd = 2)
text(0.1, 0.8, paste("Threshold =", 0.08), pos = 4, col = "black")
```




```
#  Which model is more accurate to predict a customer that would default the debt?
# From the three Roc curves, we see that they are very similar. Overall, Naive Bayes performs the best.
# Therefore, Naive Bayes is most accurate to predict a customer that would default the debt when the threshold is
0.08.
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.