hw3_starter.R (Problem 3.4 to Problem 4.5)

Erica Zhou

Nov 16, 2022

```r
rm(list = ls())

## You should set the working directory to the folder of hw3_starter by
## uncommenting the following and replacing YourDirectory by what you have
## in your local computer / labtop

setwd("~/STA314/sta314-hw3")

## Load utils.R and penalized_logistic_regression.R

source("utils.R")
source("penalized_logistic_regression.R")



## load data sets

train <- Load_data("train.csv")
```

```
## Rows: 600 Columns: 257
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (257): X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15,...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
valid <- Load_data("valid.csv")
```

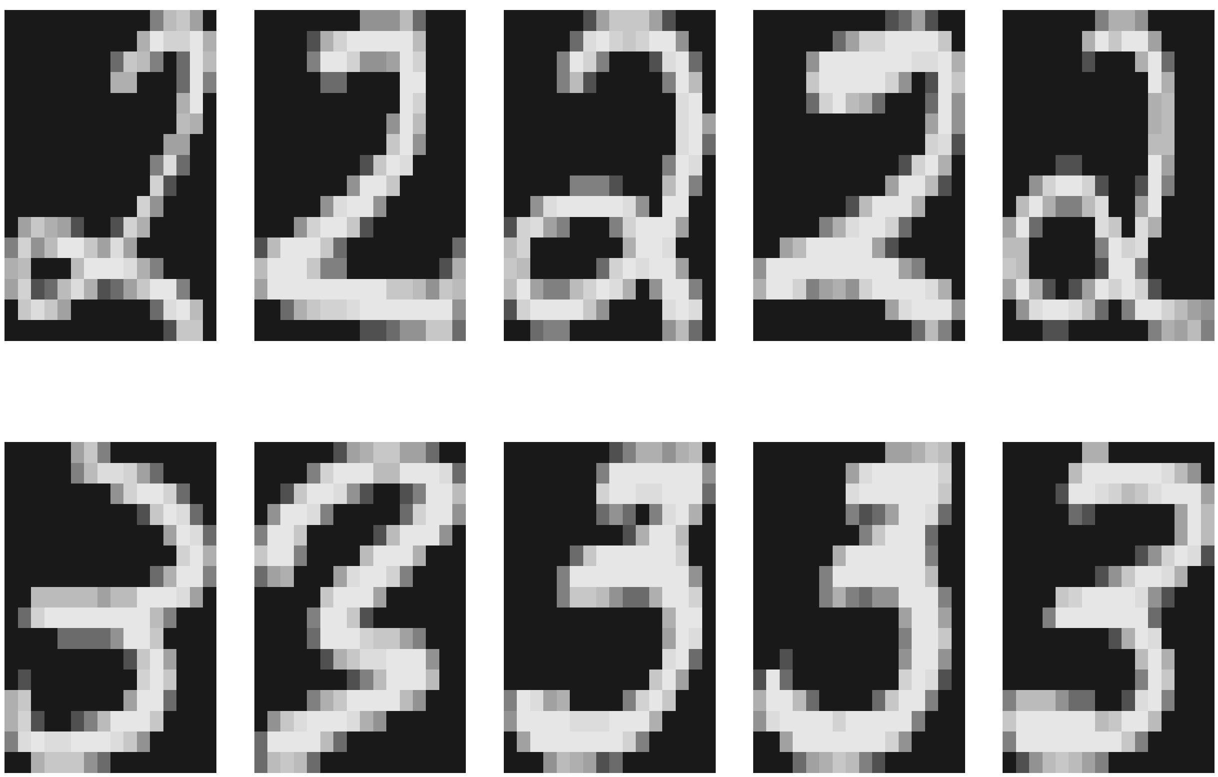
```
## Rows: 200 Columns: 257
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (257): X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15,...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
test <- Load_data("test.csv")
```

```
## Rows: 400 Columns: 257
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (257): X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15,...
##
```

```
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
x_train <- train$x
y_train <- train$y

x_valid <- valid$x
y_valid <- valid$y

x_test <- test$x
y_test <- test$y


### Visualization
## uncomment the following command to visualize the first five and 301th-305th
##  digits in the training data.
Plot_digits(c(1:5, 301:305), x_train)
```



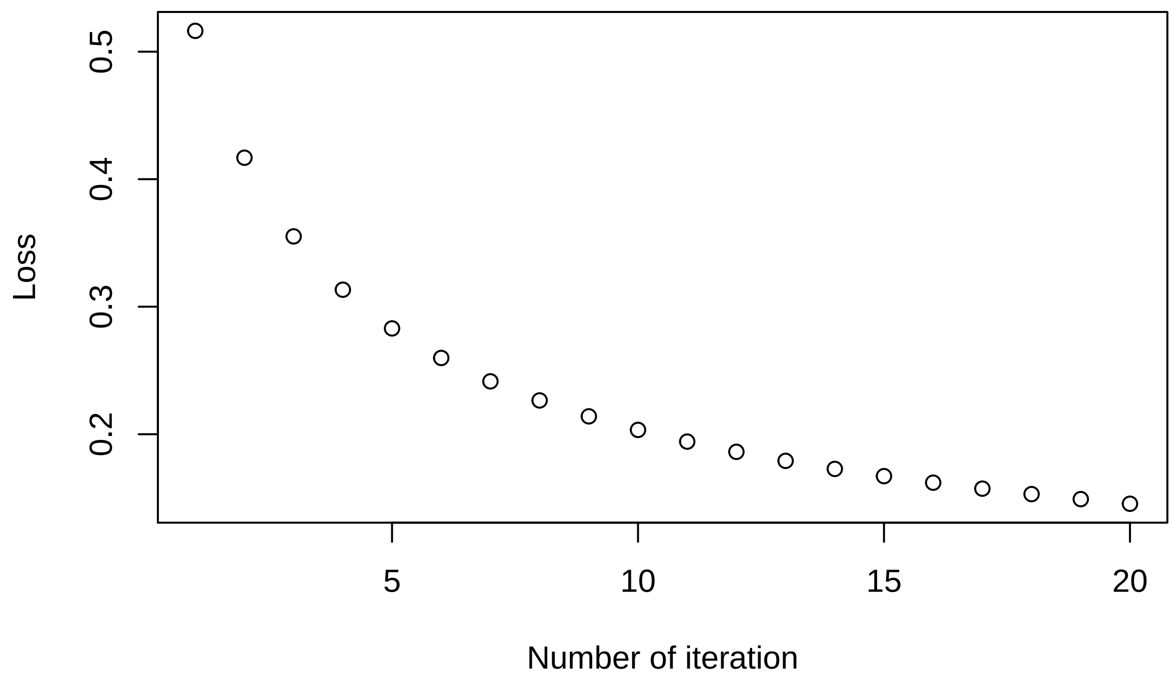
```r
####################################################################
#                          Part a.                                 #
# TODO: Find the best choice of the hyperparameters:               #
#     - stepsize (i.e. the learning rate)                          #
#     - max_iter (the maximal number of iterations)                #
#   The regularization parameter, lbd, should be set to 0          #
#   Draw plot of training losses and training 0-1 errors           #
####################################################################
```

```r
#initial settings

lbd = 0
# My choice of the alpha grid
alpha = c(.7, .5, .3, .1, .05, .03, .012, .01, .008, .005)
loss <- rep(c(), 10)
error <- rep(c(),10)

for (i in 1:20){
  for (j in 1:length(alpha)){
    loss[[j]] <- c(Penalized_Logistic_Reg(x_train,y_train, lbd,alpha[j],
                                          i)$loss)
    error[[j]] <- c(Penalized_Logistic_Reg(x_train,y_train, lbd,alpha[j],
                                           i)$error)
  }
}

df_loss <- as.data.frame(loss, row.names = NULL, optional = FALSE,
                         cut.names = FALSE, col.names = alpha,
                         fix.empty.names = TRUE,
                         stringsAsFactors = default.stringsAsFactors())

# choose 0.015 as a standard point of the change in loss, and find the ones that
# are the nearest to 0.015

#abs_diff <- abs(abs(df_loss - lag(df_loss))-0.015)
#print(abs_diff)

#which(abs_diff == min(abs_diff, na.rm = TRUE), arr.ind = TRUE)

# Codes above gives row 8 (iteration) and col 3 (alpha)
#df_loss[8,3]

######################################################################
#                       END OF YOUR CODE                             #
######################################################################
```

I chose to set the hyperparameter $\alpha = 0.3$ and max_iter $= 8$. According to the calculation, the value of the loss function decrease as the number of iteration increases. It is because we are gradually approaching to the regression that maximizes the likelihood. However, it is unreasonable that we always need as many iterations as possible. Therefore, we can select an appropriate learning rate so that the loss would finally be respectively low and acceptable. A lower learning rate takes longer to converge, but a too large learning rate may miss to catch the maximizing point. Therefore, we need to select a learning rate neither too low nor too high. We also need to decide the number of iteration. I decide to choose the a combination of learning rate and number of iteration that cause a 0.015 difference between the loss of this iteration and the previous iteration because it seems to be an acceptable value of for convergence. Also, since we selected the initial beta and beta0 at random, we want to avoid too much convergence because we do not know how far the initial coefficients are from the true ones. At $\alpha = 0.3$ and max_iter $= 8$, the loss is about 0.23, which is also acceptable.

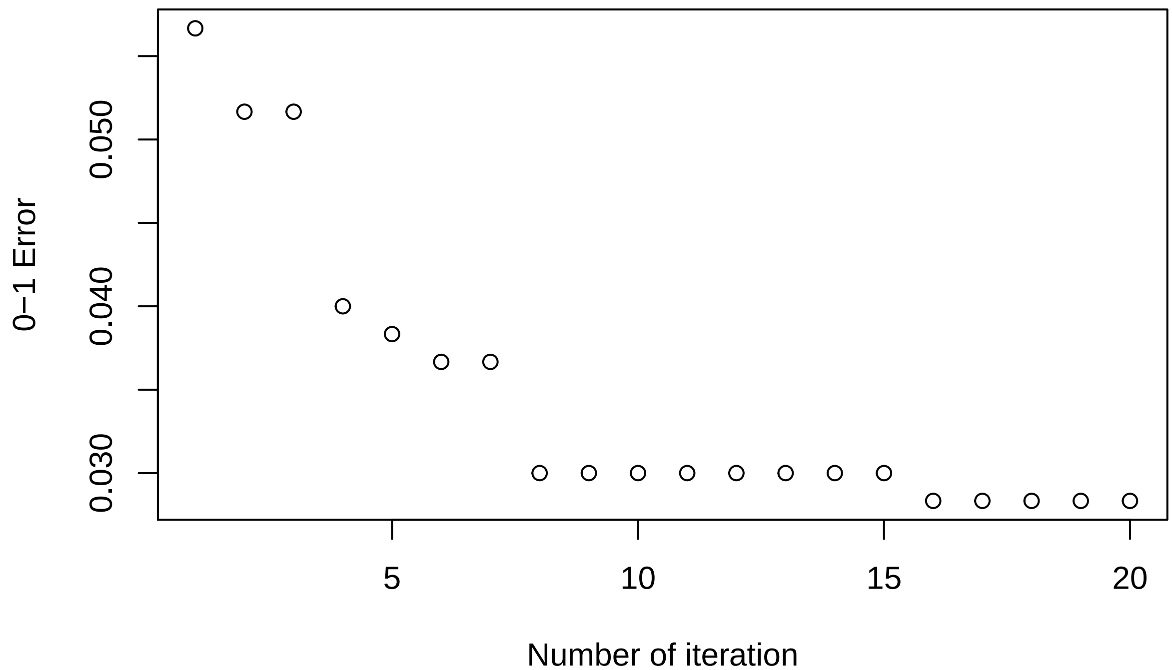
```r
df_error <- as.data.frame(error, row.names = NULL, optional = FALSE,
                          cut.names = FALSE, col.names = alpha,
                          fix.empty.names = TRUE,
                          stringsAsFactors = default.stringsAsFactors())
```

```r
plot(df_loss$X0.3, ylab = "Loss", xlab = "Number of iteration")
```



The loss decreases at a decreasing rate as the number of iteration increases because we are approaching to the likelihood maximizing setting and we becomes slower as we are closer to that setting.

```r
plot(df_error$X0.3, ylab = "0-1 Error", xlab = "Number of iteration")
```



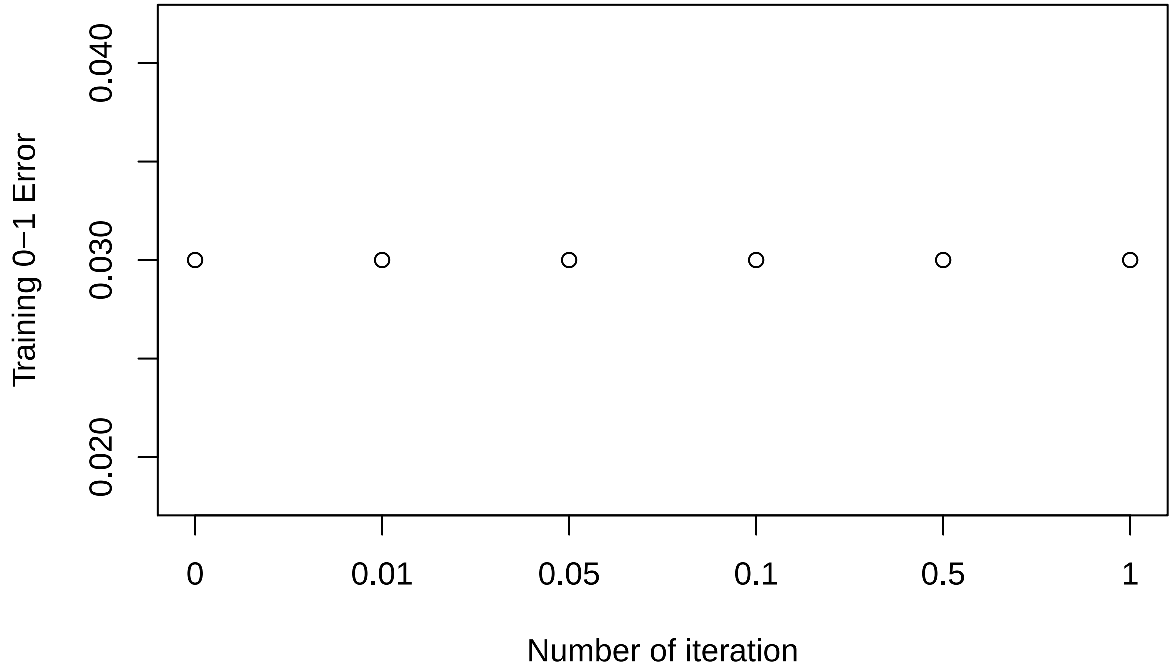
The 0-1 error (fraction) also decreases at a decreasing rate. These are the fractions of the cases where the predicted label is not the same as the true label among all predicted vs. true. It makes sense that the fraction decreases as we are approaching to the likelihood maximizing settings. The training 0-1 error had the same pattern as the training loss. It makes sense because both of them represent behavior of the regression. As we are approaching to the likelihood maximizing parameters (at a decreasing rate), the regression is performing better, and thus, we have both less loss and less 0-1 error (at a decreasing rate).

```
#####################################################################
#                        END OF YOUR CODE                           #
#####################################################################






#####################################################################
#                           Part b.                                 #
# TODO: Identify the best stepsize and max_iter for each lambda      #
#       from the given grid. Draw the plots of training and         #
#       validation 0-1 errors versus different values of lambda     #
#####################################################################


stepsize <- .3  # this should be replaced by your answer in Part a
max_iter <- 8  # this should be replaced by your answer in Part a

lbd_grid <- c(0, 0.01, 0.05, 0.1, 0.5, 1)
```

```r
loss_b <- rep(c(), length(lbd_grid))
error_train <- rep(c(), length(lbd_grid))
error_valid <- rep(c(), length(lbd_grid))



for (q in 1:length(lbd_grid)){


  loss_b[[q]] <- c(Penalized_Logistic_Reg(x_train,y_train, lbd_grid[q]
                                          ,stepsize,max_iter)$loss)
  error_train[[q]] <- c(Penalized_Logistic_Reg(x_train,y_train, lbd_grid[q]
                                               ,stepsize,max_iter)$error)
  error_valid[[q]] <- c(Penalized_Logistic_Reg(x_valid,y_valid, lbd_grid[q]
                                               ,stepsize,max_iter)$error)
}

df_loss_b <- as.data.frame(loss_b, row.names = NULL, optional = FALSE,
                           cut.names = FALSE, col.names = lbd_grid,
                           fix.empty.names = TRUE,
                           stringsAsFactors = default.stringsAsFactors())

df_error_train <- as.data.frame(error_train, row.names = NULL, optional = FALSE,
                                cut.names = FALSE, col.names = lbd_grid,
                                fix.empty.names = TRUE,
                                stringsAsFactors = default.stringsAsFactors())

df_error_valid <- as.data.frame(error_valid, row.names = NULL, optional = FALSE,
                                cut.names = FALSE, col.names = lbd_grid,
                                fix.empty.names = TRUE,
                                stringsAsFactors = default.stringsAsFactors())

#df_error_train[8,]
#df_error_valid
```

The hyperparameters guarantee convergence for all λs as the changes in loss are respectively small for all λs.

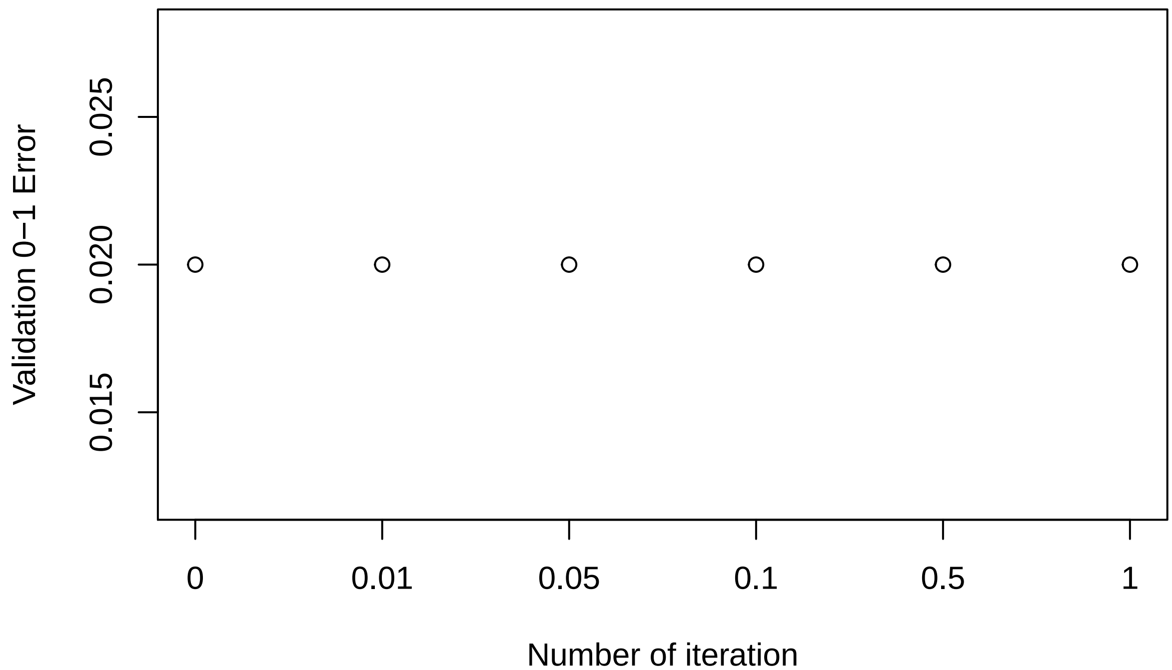
```r
plot(t(df_error_train)[,8], xaxt = "n",ylab = "Training 0-1 Error",
     xlab = "Number of iteration")
axis(1, at = 1:6,labels = lbd_grid)
```




```r
plot(t(df_error_valid)[,8], xaxt = "n",ylab = "Validation 0-1 Error",
     xlab = "Number of iteration")
axis(1, at = 1:6,labels = lbd_grid)


library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```




```
set.seed(1)
cv.net <- cv.glmnet(x_train, y_train, alpha = .3, family = "binomial")
cv.net$lambda.min
```

```
## [1] 0.002126119
```

The 0-1 errors of all the λs are the same. It is because that we have small coefficients and thus the regularization did not penalize or penalized a little on them. Also, the λs are small enough that they won't cause underfitting problem. Thus, the overall 0-1 loss are the same for these λs. Therefore, the given values are all suitable for λ in this case. We then use cross-validation to choose. Notice that cv.net$lambda.min = 0.0021261. Therefore, we can choose 0, which is the nearest to the optimal value of λ.

```
####################################################################
#                        END OF YOUR CODE                          #
####################################################################









####################################################################
#                           Part c.                                #
# TODO: using the best stepsize,  max_iter and lbd you found, fit   #
#  the penalized logistic regression and compute its test 0-1 error #
####################################################################
```

```r
stepsize <- .3  # this should be replaced by your answer in Part a
max_iter <- 8  # this should be replaced by your answer in Part a
lbd <- 0      # this should be replaced by your answer in Part b

#penalized model
p.model <- Penalized_Logistic_Reg(x_train, y_train, lbd, stepsize, max_iter)

p.model$error[max_iter]
```

```
## [1] 0.03
```

```r
#glmnet model
model <- glmnet(x_train, y_train, alpha = stepsize, family = "binomial",
                lambda = lbd)

pred.model <- predict(model, s = lbd, newx = x_test)

mean(pred.model != y_test)
```

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

The test 0-1 error of the penalized logistic regression model is much lower the model using glmnet.

```r
#####################################################################
#                       END OF YOUR CODE                            #
#####################################################################
```

Problem 4 1.

```r
library(ISLR)
library(MASS)
mpg01 <- as.numeric(Auto$mpg > median(Auto$mpg))
Auto$mpg01 <-mpg01
set.seed(0)
#split data
t = sort(sample(nrow(Auto), nrow(Auto)*.7))
train = Auto[t, ]
test = Auto[-t,]
```

2.

```r
# fit lda model
lda.fit <- lda(mpg01 ~ cylinders + displacement + horsepower + weight +
                 acceleration + year, data = train)
#predict on the test set
lda.pred <- predict(lda.fit, test)
#error
lda.class <- lda.pred$class
mean(lda.class != test$mpg01)
```

```
## [1] 0.1186441
```

3.

```r
# qda
qda.fit <- qda(mpg01 ~ cylinders + displacement + horsepower + weight +
                 acceleration + year, data = train)
qda.class <- predict(qda.fit, test)$class
```

```
mean(qda.class != test$mpg01)
```

```
## [1] 0.1101695
```

4.

```
glm.fit <- glm(mpg01 ~ cylinders + displacement + horsepower + weight +
                 acceleration + year, data = train, family = binomial)

glm.probs <- predict(glm.fit, test, type="response")
glm.pred <- rep(0, nrow(test))
glm.pred[glm.probs > 0.5] = 1
mean(glm.pred != test)
```

```
## [1] 0.8940678
```

5.

```
#install.packages("pROC")
library(pROC)
```

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

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

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

```
roc_lda <- roc(test$mpg01, predict(lda.fit, test)$posterior[,2])
```

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

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

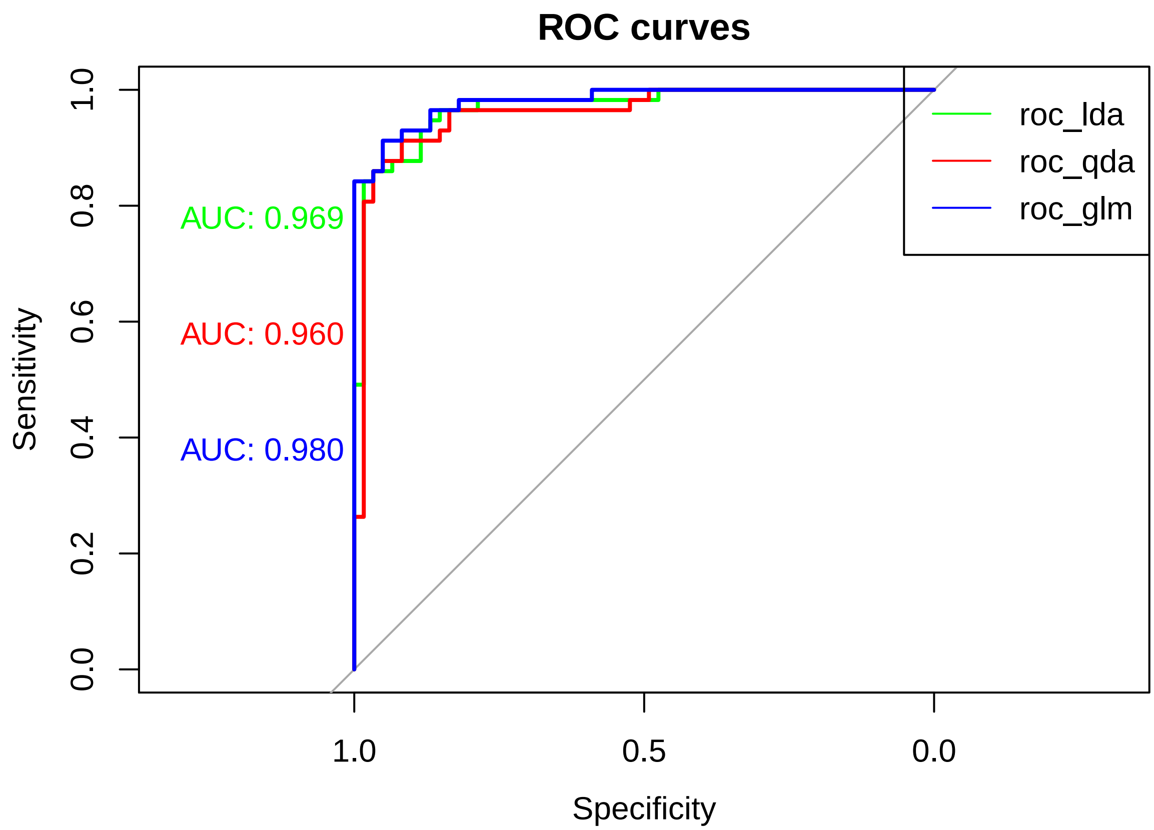
```
roc_qda <- roc(test$mpg01, predict(qda.fit, test)$posterior[,2])
```

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

```
roc_glm <- roc(test$mpg01, glm.probs)
```

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

```
plot(roc_lda, main="ROC curves", col = "green", print.auc = TRUE,
     print.auc.x = 1.3, print.auc.y = .8)
plot(roc_qda, add = TRUE, col = "red", print.auc = TRUE, print.auc.x = 1.3,
     print.auc.y = .6)
plot(roc_glm, add = TRUE, col = "blue", print.auc = TRUE, print.auc.x = 1.3,
     print.auc.y = .4)
legend("topright", legend = c("roc_lda", "roc_qda", "roc_glm"),lty = c(1, 1, 1),
       col = c("green", "red", "blue"))
```



The AUCs are similar for three classifiers. The GLM classifier has the highest AUC but also the highest test error. The QDA classifier has the lowest test error but also lowest AUC. Thus, we may choose the LDA classifier, which has a respectively low test error and high AUC.