Q3_starter.R

yuyangchen

2023-11-28

```r
options(encoding = "UTF-8")
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("/Users/yuyangchen/Documents/R/Q3_starter")

## Load utils.R and penalized_logistic_regression.R

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


## load data sets

train <- Load_data("./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("./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("./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
```

```r
######################################################################
#                            Part a.                                 #
# TODO: Find suitable 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             #
######################################################################
library(purrr)
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
```

```r
library(ggplot2)
library(glmnet)
```

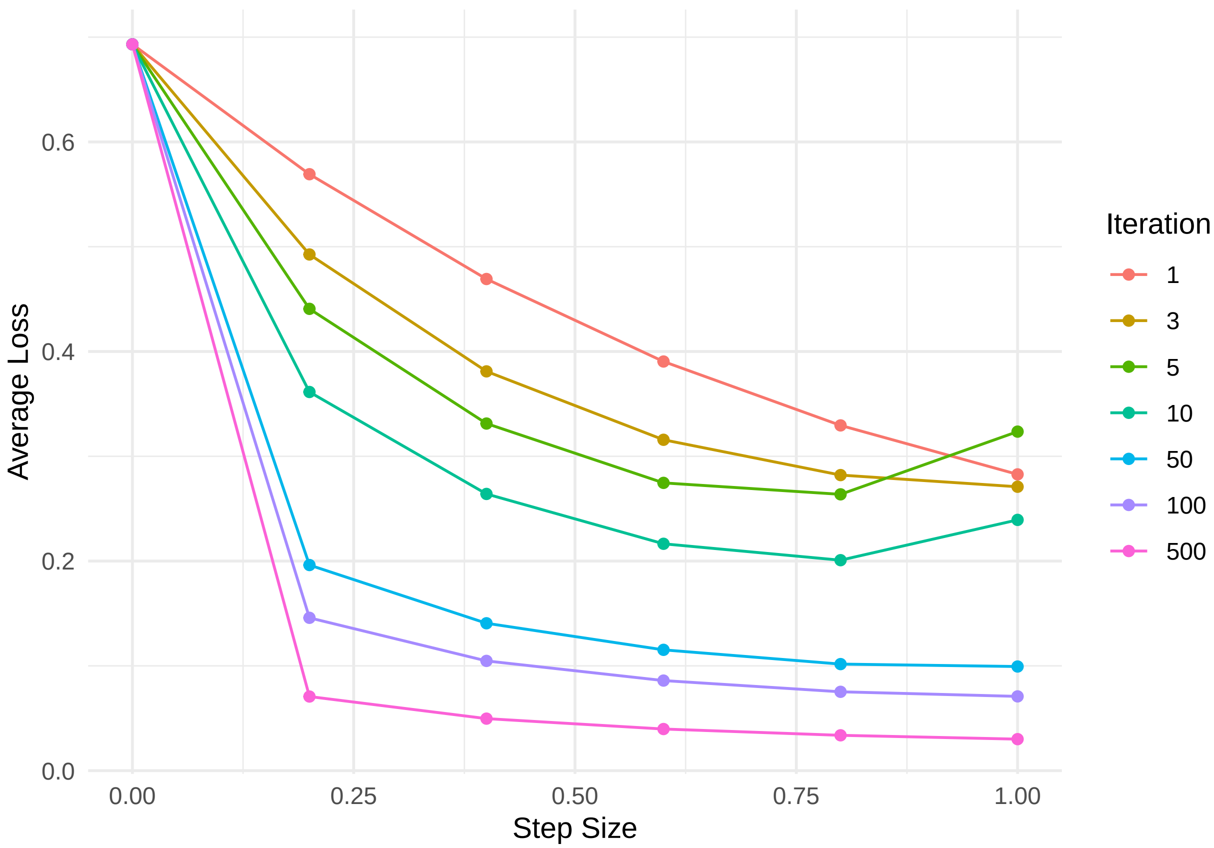
```
## Loading required package: Matrix
## Loaded glmnet 4.1-8
```

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
lbd <- 0
totalIterations <- 500
stepSequence <- seq(0, 1, by = 0.2)
```

```r
iterationSteps <- c(1, 3, 5, 10, 50, 100, 500)

resultsList <- lapply(stepSequence, function(currentStep) {
  Penalized_Logistic_Reg(x_train, y_train, lbd, currentStep, totalIterations)
})
names(resultsList) <- stepSequence

averageLossData <- data.frame(
  StepSize = rep(stepSequence, each = length(iterationSteps)),
  Iteration = rep(iterationSteps, times = length(stepSequence)),
  Loss = unlist(lapply(resultsList, function(result) sapply(iterationSteps, function(iter) mean(result$l
  ))

ggplot(averageLossData, aes(x = StepSize, y = Loss, color = as.factor(Iteration))) +
  geom_line() +
  geom_point() +
  labs(y = "Average Loss", x = "Step Size", color = "Iteration") +
  theme_minimal()
```

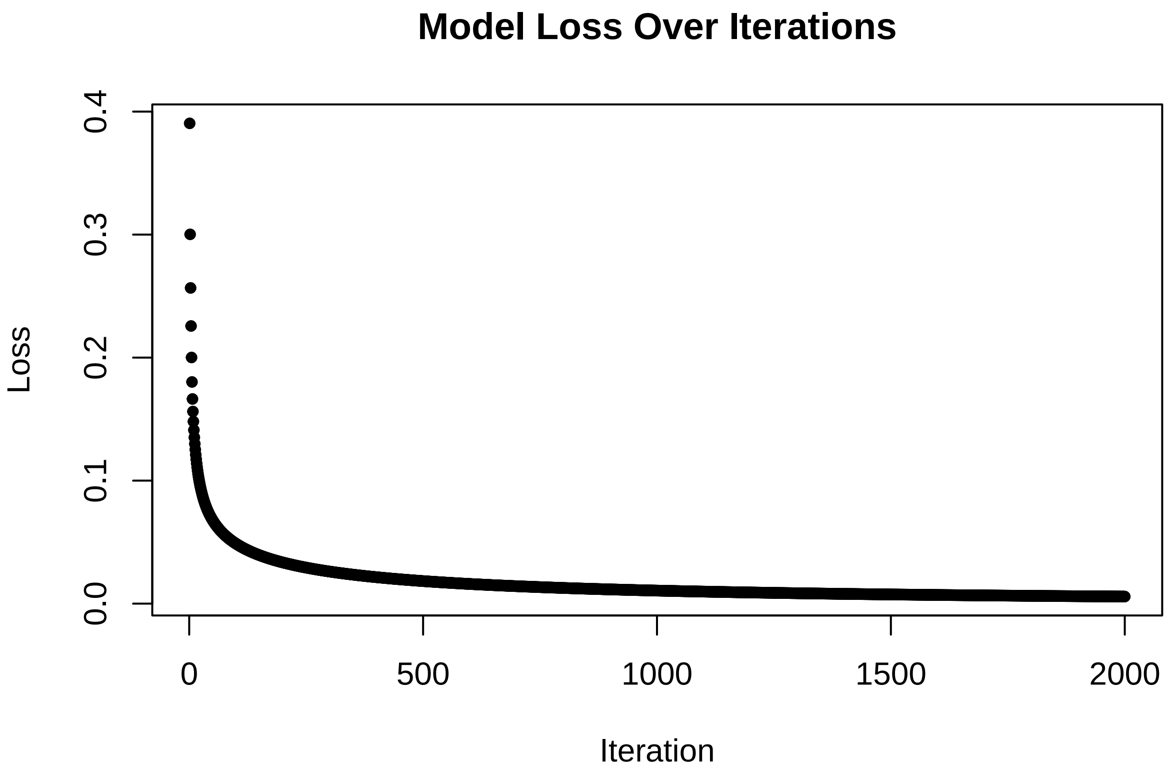



```r
#result
minLossAtIndex5 <- sapply(resultsList, function(result) result$loss[5])
stepsizeFinal <- stepSequence[which.min(minLossAtIndex5)]
stepsizeFinal
```

```
## [1] 0.6
```

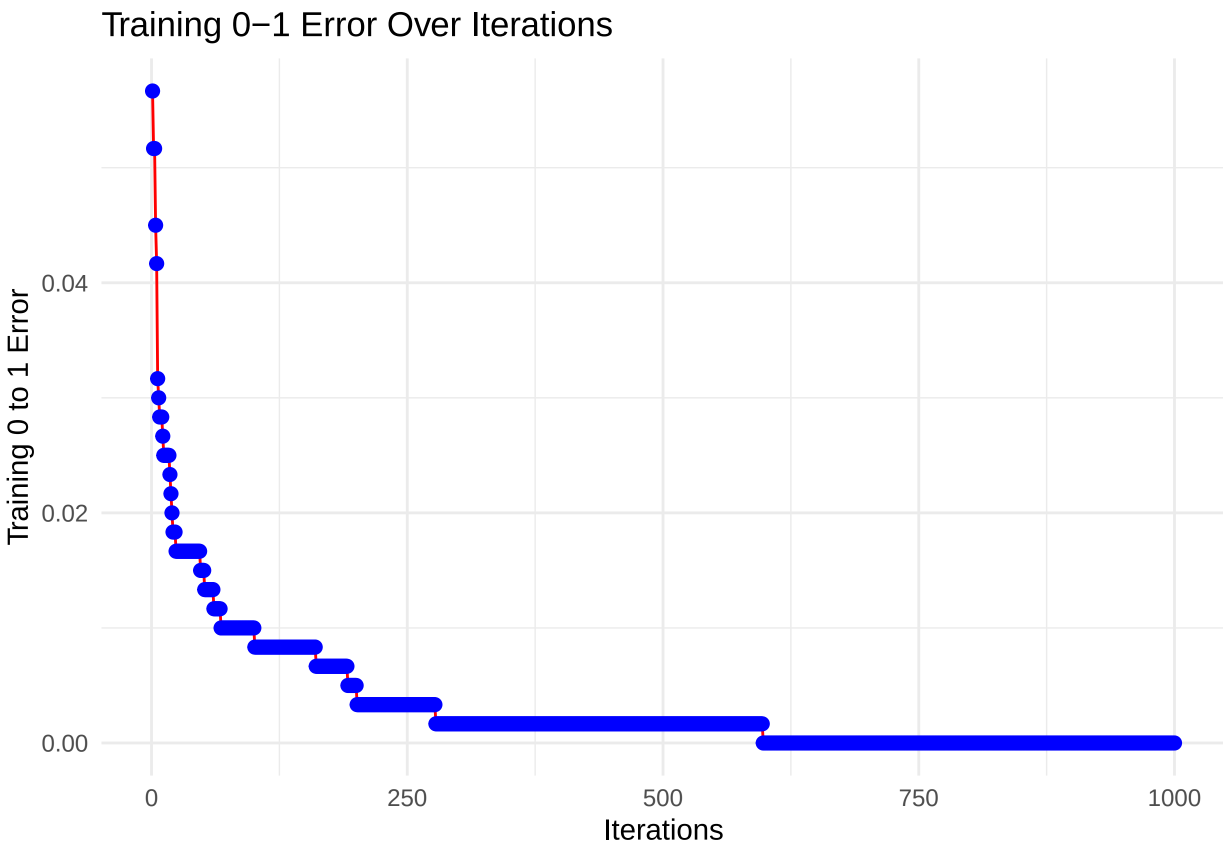
```r
increasedIterations <- 2000
```

```r
regressionResult <- Penalized_Logistic_Reg(x_train, y_train,
                                           lbd, stepsizeFinal,
                                           increasedIterations)

plot(regressionResult$loss, type = "p", xlab = "Iteration", ylab = "Loss",
     main = "Model Loss Over Iterations", col = "black", pch = 20)
```



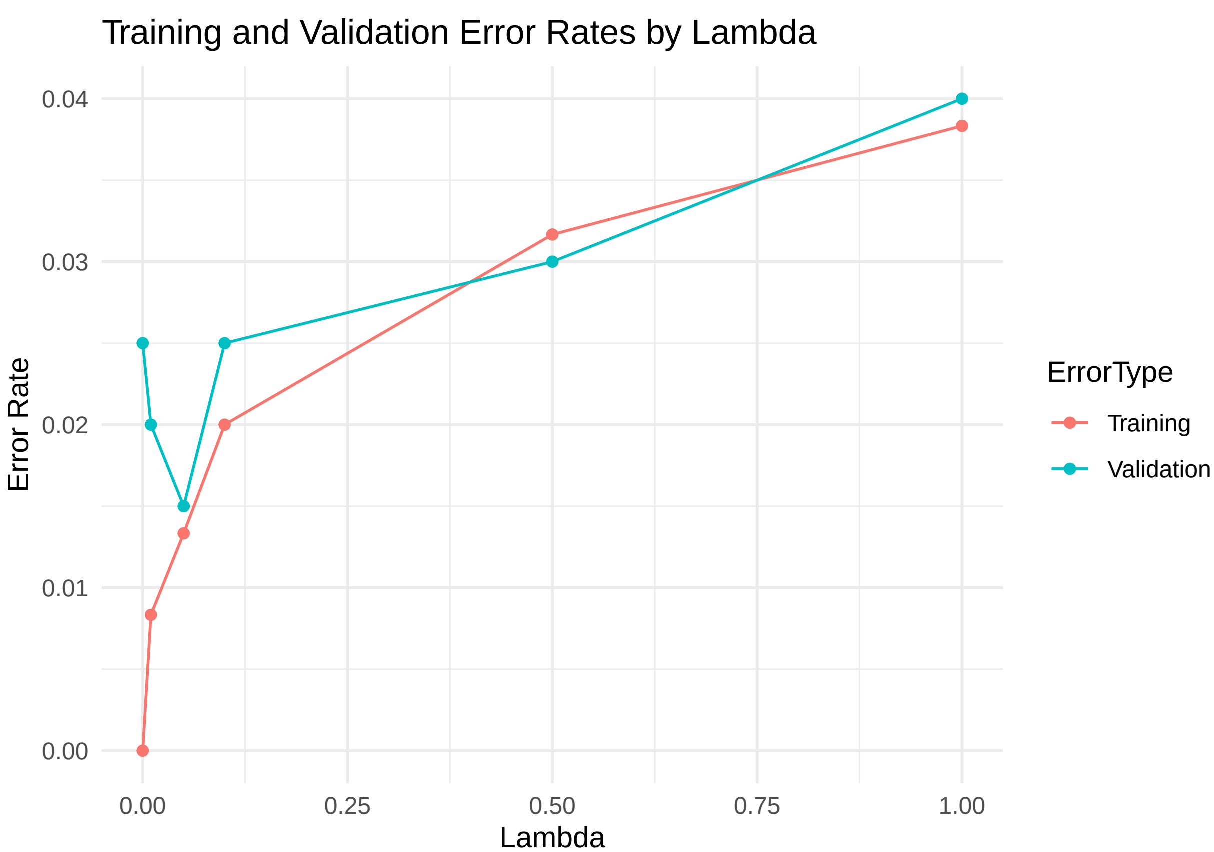

```r
finalMaxIterations <- 1000

finalModelResult <- Penalized_Logistic_Reg(x_train, y_train, lbd, stepsizeFinal, finalMaxIterations)

plot(finalModelResult$loss, type = "o", col = "darkblue", xlab = "Iterations",
     ylab = "Training Loss",
     main = "Training Loss per Iteration")
```


```r
errorData <- data.frame(Iterations = 1:length(finalModelResult$error), TrainingError = finalModelResult$

ggplot(errorData, aes(x = Iterations, y = TrainingError)) +
  geom_line(color = "red") +
  geom_point(color = "blue", size = 2) +
  labs(x = "Iterations", y = "Training 0 to 1 Error",
       title = "Training 0-1 Error Over Iterations") +
  theme_minimal()
```

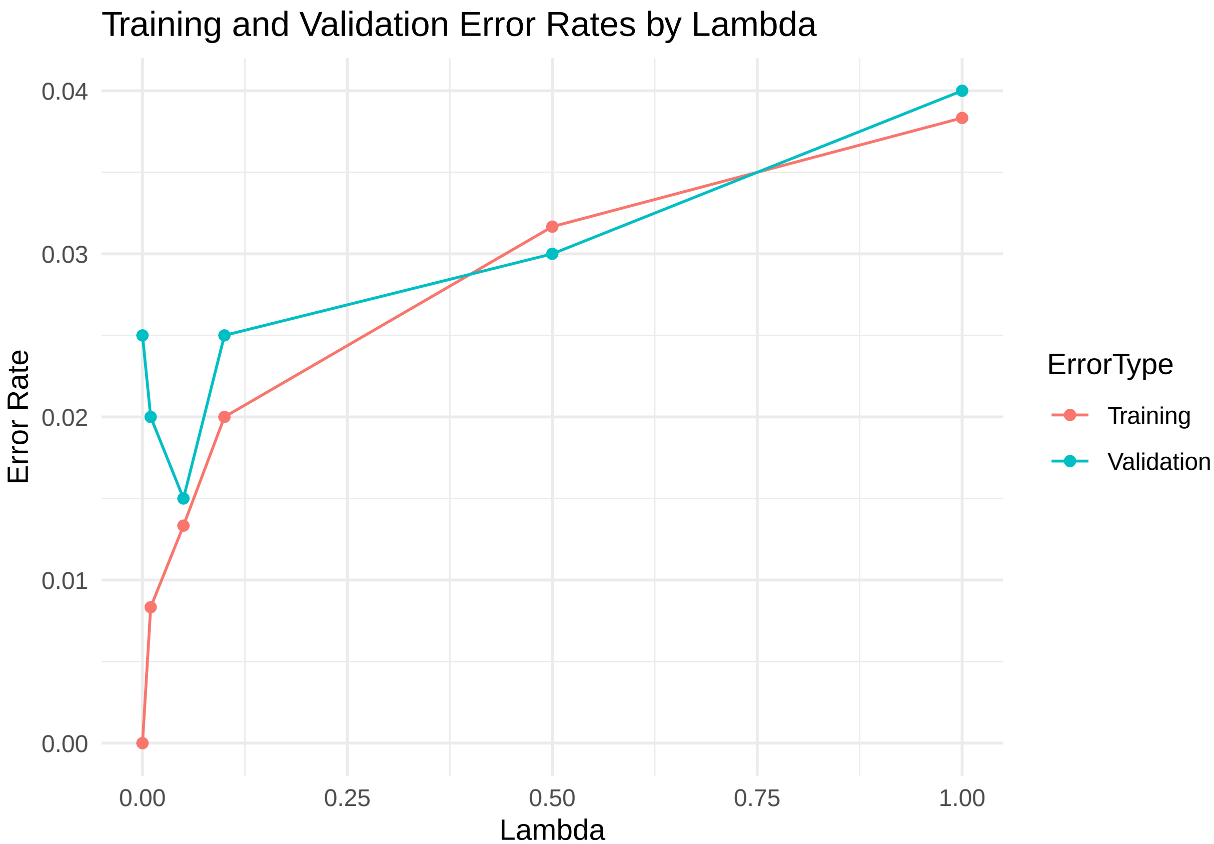



```
##########################################################################
#                          END OF YOUR CODE                              #
##########################################################################


##########################################################################
#                              Part b.                                   #
# TODO: Identify suitable 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          #
##########################################################################
max_iter <- 1000
lbd_grid <- c(0, 0.01, 0.05, 0.1, 0.5, 1)
stepsize <- 0
stepsizes <- seq(0, 1, 0.2)

res_list <- lapply(lbd_grid, function(lambda) {
  lapply(stepsizes, function(stepsize) {
    Penalized_Logistic_Reg(x_train, y_train, lambda, stepsize, max_iter)
  })
})


stepsizes_final <- sapply(1:length(lbd_grid), function(i) {
  minLossIndex <- which.min(sapply(res_list[[i]], function(model) model$loss[5]))
  stepsizes[minLossIndex]
})
```

```r
errorResults <- lapply(1:length(lbd_grid), function(i) {
  model <- Penalized_Logistic_Reg(x_train, y_train, lbd_grid[i],
                                  stepsizes_final[i], max_iter)
  list(
    TrainError = Evaluate(y_train, Predict_logis(x_train, model$beta,
                                                 model$beta0, type = "class")),
    ValidError = Evaluate(y_valid, Predict_logis(x_valid, model$beta,
                                                 model$beta0, type = "class"))
  )
})

#data
errorData <- data.frame(
  Lambda = rep(lbd_grid, each = 2),
  ErrorType = rep(c("Training", "Validation"), times = length(lbd_grid)),
  ErrorRate = unlist(lapply(errorResults, function(res) c(res$TrainError, res$ValidError)))
)

#plot
ggplot(errorData, aes(x = Lambda, y = ErrorRate, color = ErrorType, group = ErrorType)) +
  geom_line() +
  geom_point() +
  labs(x = "Lambda", y = "Error Rate", title = "Training and Validation Error Rates by Lambda") +
  theme_minimal()
```



```r
#5.2
ggplot(errorData, aes(x = Lambda, y = ErrorRate, color = ErrorType, group = ErrorType)) +
  geom_line() +
  geom_point() +
  labs(x = "Lambda", y = "Error Rate", title = "Training and Validation Error Rates by Lambda") +
  theme_minimal()
```




```r
#5.3
validationErrorVec <- errorData$ErrorRate[errorData$ErrorType == "Validation"]

lambdaFinal <- lbd_grid[which.min(validationErrorVec)]
lambdaFinal
```

```
## [1] 0.05
```

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

indexLambda <- which(lbd_grid == lambdaFinal)
stepsize <- stepsizes_final[indexLambda]
max_iter <- 1000
lbd <- 0
finalLambda <- lambdaFinal
PenLogisModel <- Penalized_Logistic_Reg(x_train, y_train, finalLambda, stepsize, max_iter)
predictedTest <- Predict_logis(x_test, PenLogisModel$beta, PenLogisModel$beta0, type = "class")
```

```
testError <- Evaluate(y_test, predictedTest)
print(paste("The test error is", testError))
```

```
## [1] "The test error is 0.025"
```

```
fit_glmnet <- glmnet(x_train, y_train, family = "binomial",
                     lambda = 0.05, alpha = 0)
pred_glmnet_test <- predict(fit_glmnet, x_test, type = "class")
cat("The test error of after using glmnet is",
    Evaluate(y_test, as.numeric(pred_glmnet_test)))
```

```
## The test error of after using glmnet is 0.0225
```

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


#######################################################################
#                            Part d.                                  #
# TODO: Use your implementation of LDA and Naive Bayes in Problem 2 #
#   to classify the test data points and report test 0-1 errors     #
#######################################################################

source("discriminant_analysis.R")
uniqueClasses <- unique(y_train)
numClassesLDA <- length(uniqueClasses)

# Compute priors
classPriorsLDA <- Comp_priors(y_train)

# Compute means
conditionalMeansLDA <- Comp_cond_means(x_train, y_train)

# Compute covariances
conditionalCovsLDA <- Comp_cond_covs(x_train, y_train, numClassesLDA, "LDA")

# Predict posteriors
posteriorProbsLDA <- Predict_posterior(x_test, classPriorsLDA,
                                       conditionalMeansLDA,
                                       conditionalCovsLDA, "LDA")

# Class predictions
maxPosteriorIndices <- apply(posteriorProbsLDA, 1, which.max)
predictionsLDA <- as.numeric(maxPosteriorIndices) - 1

# Test error
testErrorLDA <- Evaluate(y_test, predictionsLDA)
print(paste("The test error of LDA is", testErrorLDA))
```

```
## [1] "The test error of LDA is 0.055"
```

```
# Compute class priors
classPriorsNB <- Comp_priors(y_train)

# Compute conditional means
```

```r
conditionalMeansNB <- Comp_cond_means(x_train, y_train)

# Compute covariance matrices for Naive Bayes
covarianceMatricesNB <- Comp_cond_covs(x_train, y_train, numClassesLDA, "NB")

# Predict posterior probabilities
posteriorProbMatNB <- Predict_posterior(x_test, classPriorsNB,
                                        conditionalMeansNB,
                                        covarianceMatricesNB, "NB")

# Generate predictions
predictionsNB <- Predict_labels(posteriorProbMatNB)

# Output test error
testErrorNB <- Evaluate(y_test, predictionsNB)
print(paste("The test error of Naive Bayes is", testErrorNB))
```

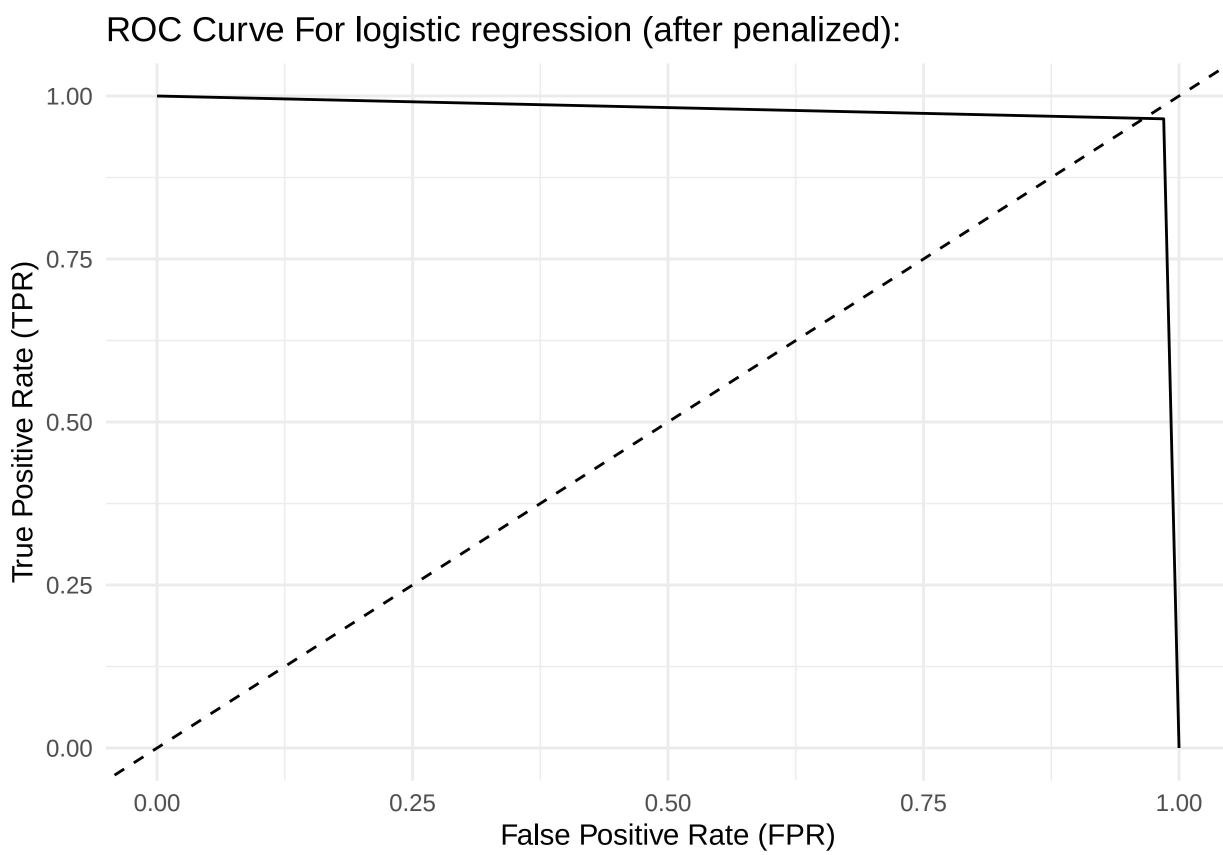
```
## [1] "The test error of Naive Bayes is 0.065"
```

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

```r
#####################################################################
#                            Part e.                                #
# TODO: based on the test data, draw the ROC curve and compute AUC  #
#  of your implemented penalized logistic regression, LDA and NB    #
#####################################################################
rocCurveData <- roc(y_test, predictedTest)
```

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

```r
rocDataFrame <- data.frame(
  TPR = rocCurveData$sensitivities,
  FPR = rocCurveData$specificities,
  Thresholds = rocCurveData$thresholds
)

ggplot(rocDataFrame, aes(x = FPR, y = TPR)) +
  geom_line() +
  geom_abline(linetype = "dashed") +
  labs(x = "False Positive Rate (FPR)", y = "True Positive Rate (TPR)",
       title = "ROC Curve For logistic regression (after penalized):") +
  theme_minimal()
```



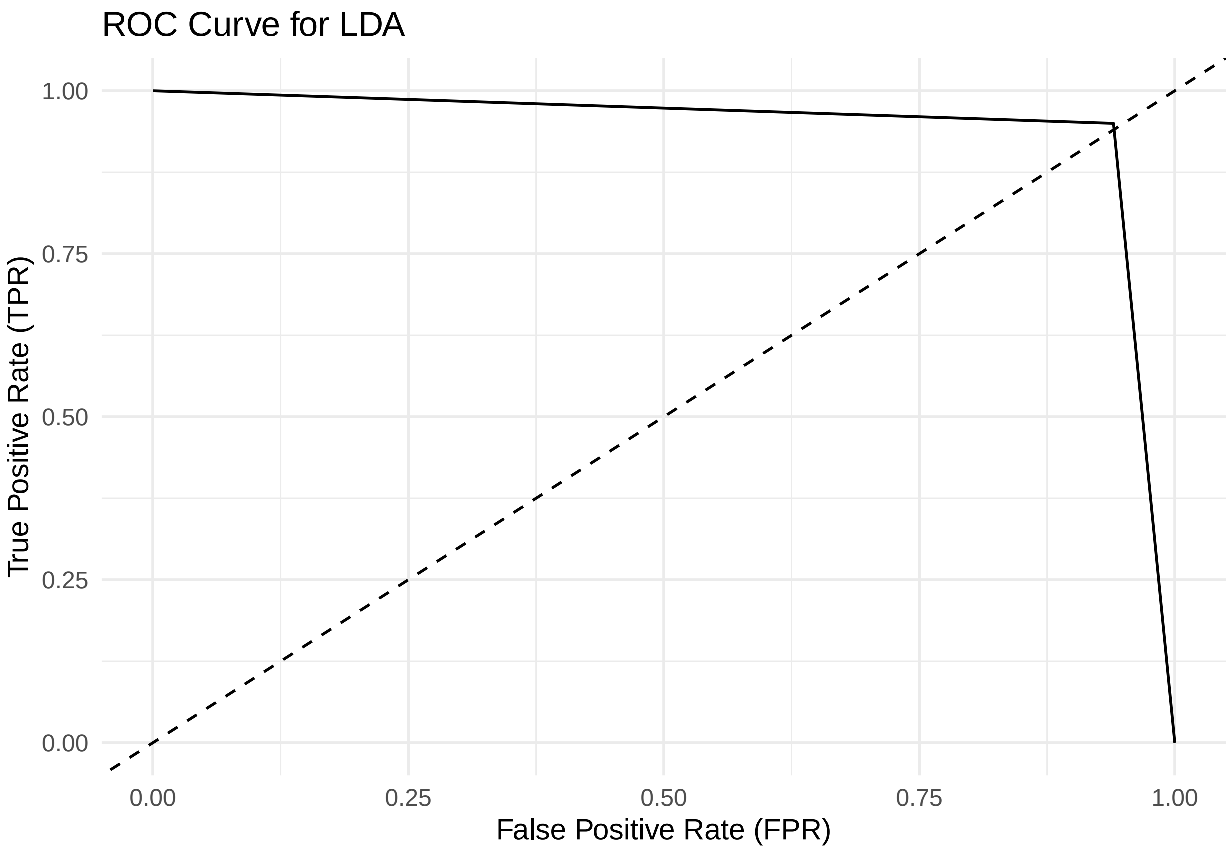

```
auc_value <- auc(rocCurveData)
print(paste("AUC:", auc_value))
```

```
## [1] "AUC: 0.975"
```

```
#LDA
rocCurveLDA <- roc(y_test, predictionsLDA)
```

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

```
rocDataLDA <- data.frame(
  TPR = rocCurveLDA$sensitivities,
  FPR = rocCurveLDA$specificities,
  Thresholds = rocCurveLDA$thresholds
)

#plot
ggplot(rocDataLDA, aes(x = FPR, y = TPR)) +
  geom_line() +
  geom_abline(linetype = "dashed") +
  labs(x = "False Positive Rate (FPR)", y = "True Positive Rate (TPR)",
       title = "ROC Curve for LDA") +
  theme_minimal()
```



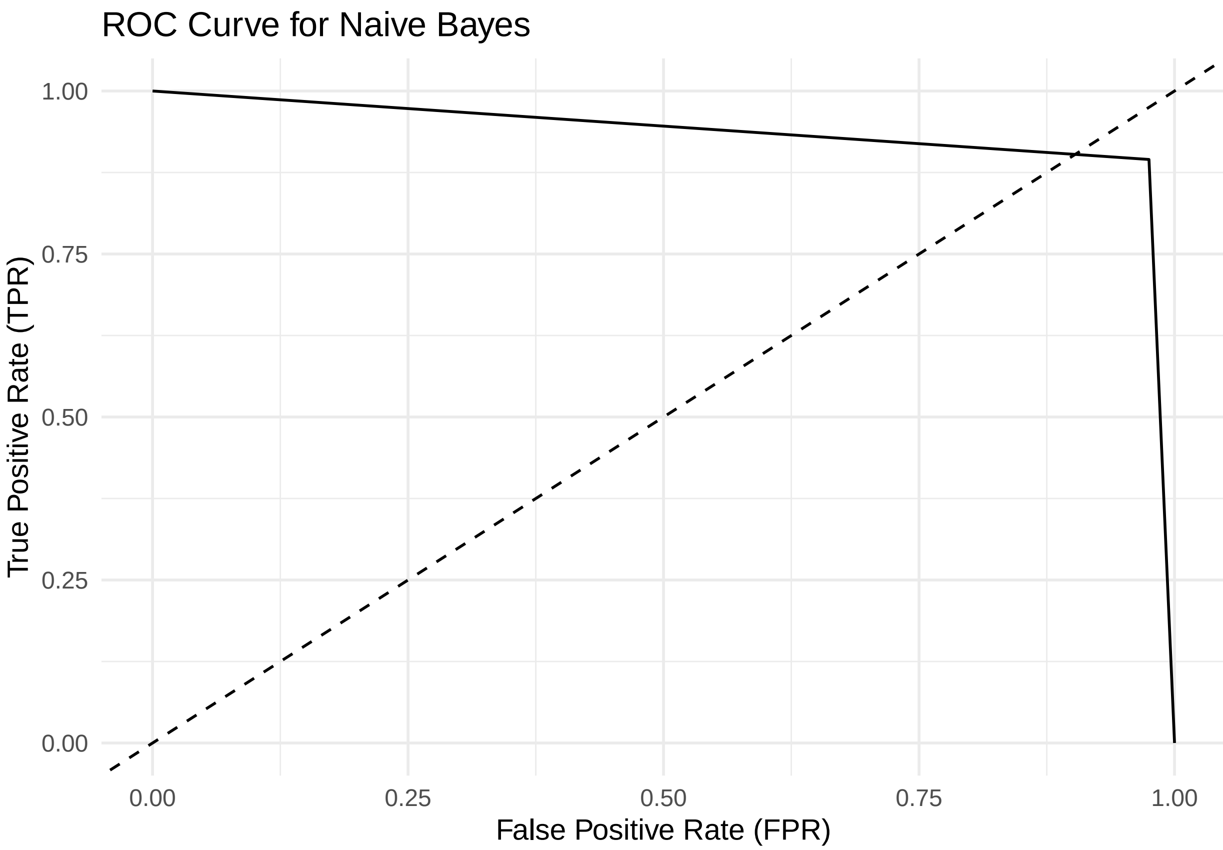

```
auc_value <- auc(rocCurveLDA)
print(paste("AUC:", auc_value))
```

```
## [1] "AUC: 0.945"
```

```
#NB
rocCurveNB <- roc(y_test, predictionsNB)
```

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

```
rocDataNB <- data.frame(
  TPR = rocCurveNB$sensitivities,
  FPR = rocCurveNB$specificities,
  Thresholds = rocCurveNB$thresholds
)

#plot
ggplot(rocDataNB, aes(x = FPR, y = TPR)) +
  geom_line() +
  geom_abline(linetype = "dashed") +
  labs(x = "False Positive Rate (FPR)", y = "True Positive Rate (TPR)",
       title = "ROC Curve for Naive Bayes") +
  theme_minimal()
```




```r
auc_value <- auc(rocCurveNB)
print(paste("AUC:", auc_value))
```

```
## [1] "AUC: 0.935"
```

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