Solution to the coding part of HW 4

2023-11-08

Problem 2

Setup

```
rm(list = ls())

# You should set the working directory to the folder of hw4_starter by
# uncommenting the following and replacing YourDirectory by the one you use

setwd("~/Documents/Mike/Teaching/Sta314-2023-fall/hws/hw4/hw4_solution/")

source("utils.R")
source("discriminant_analysis.R")


train <- Load_data("../Q2_starter/data/digits_train.txt")
```

```
## Rows: 7000 Columns: 65
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (65): 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.
```

```
test <- Load_data("../Q2_starter/data/digits_test.txt")
```

```
## Rows: 4000 Columns: 65
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (65): 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.
```

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

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

### Visualization
## uncomment the following command to visualize the first ten digits in the training data.
# Plot_digits(1:10, x_train)
```

Part a: train LDA on the training data and classify the test data

```
lda_start <- Sys.time()

K <- length(unique(y_train))
priors <- Comp_priors(y_train, K)
means <- Comp_cond_means(x_train, y_train, K)
covs <- Comp_cond_covs(x_train, y_train, K, method = "LDA")
prob_mat <- Predict_posterior(x_test, priors, means, covs, method = "LDA")
lda_end <- Sys.time()
pred_test <- Predict_labels(prob_mat)
acc_lda <- mean(pred_test != y_test)
cat("The test misclassification rate of LDA is", acc_lda, "\n")
```

```
## The test misclassification rate of LDA is 0.10225
```

Part b: train QDA on the training data and classify the test data

```
qda_start <- Sys.time()
priors <- Comp_priors(y_train, K)
means <- Comp_cond_means(x_train, y_train, K)
covs <- Comp_cond_covs(x_train, y_train, K, method = "QDA")
prob_mat <- Predict_posterior(x_test, priors, means, covs, method = "QDA")
qda_end <- Sys.time()
pred_test <- Predict_labels(prob_mat)
acc_qda <- mean(pred_test != y_test)
cat("The test misclassification rate of QDA is", acc_qda, "\n")
```

```
## The test misclassification rate of QDA is 0.04075
```

Part c: train Naive Bayes on the training data and classify the test data

```
nb_start <- Sys.time()
priors <- Comp_priors(y_train, K)
means <- Comp_cond_means(x_train, y_train, K)
covs <- Comp_cond_covs(x_train, y_train, K, method = "NB")
prob_mat <- Predict_posterior(x_test, priors, means, covs, method = "NB")
nb_end <- Sys.time()
pred_test <- Predict_labels(prob_mat)
acc_nb <- mean(pred_test != y_test)
cat("The test misclassification rate of NB is", acc_nb, "\n")
```

```
## The test misclassification rate of NB is 0.17025
```

Part d: compare with `lda` and `qda`

Compare misclassification rates.

```
library(MASS)

lda_r_start <- Sys.time()
```

```
fitted_lda <- lda(x_train, grouping = y_train)
lda_r_end <- Sys.time()
qda_r_start <- Sys.time()
fitted_qda <- qda(x_train, grouping = y_train)
qda_r_end <- Sys.time()

pred_test <- predict(fitted_lda, x_test)$class
pred_test_qda <- predict(fitted_qda, x_test)$class

acc_lda_r <- mean(pred_test != y_test)
cat("The test misclassification rate of lda is", acc_lda_r, "\n")
```

```
## The test misclassification rate of lda is 0.10225
```

```
acc_qda_r <- mean(pred_test_qda != y_test)
cat("The test misclassification rate of qda is", acc_qda_r, "\n")
```

```
## The test misclassification rate of qda is 0.04075
```

Compare running time.

```
table_data <- data.frame(c("Implemented LDA", "Implemented QDA", "R LDA", "R QDA"),
                    as.numeric(c(lda_end-lda_start, qda_end-qda_start,
                                 lda_r_end-lda_r_start,
                                 qda_r_end-qda_r_start)),
                    c(acc_lda, acc_qda, acc_lda_r, acc_qda_r))

knitr::kable(table_data,
             col.names=c("Method", "Computational Speed", "Error Rate"),
             digits= 3)
```

Method	Computational Speed	Error Rate
Implemented LDA	8.860	0.102
Implemented QDA	7.234	0.041
R LDA	0.190	0.102
R QDA	0.107	0.041

Problem 3

Setup

```
rm(list = ls())

# You should set the working directory to the folder of hw3_starter by
# uncommenting the following and replacing YourDirectory by the one you use
# setwd("YourDirectory/Q3_starter")

setwd("~/Documents/Mike/Teaching/Sta314-2023-fall/hws/hw4/hw4_solution/")


# Load utils.R and penalized_logistic_regression.R
```

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



# load data sets

train <- Load_data("../Q3_starter/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("../Q3_starter/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("../Q3_starter/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.
```

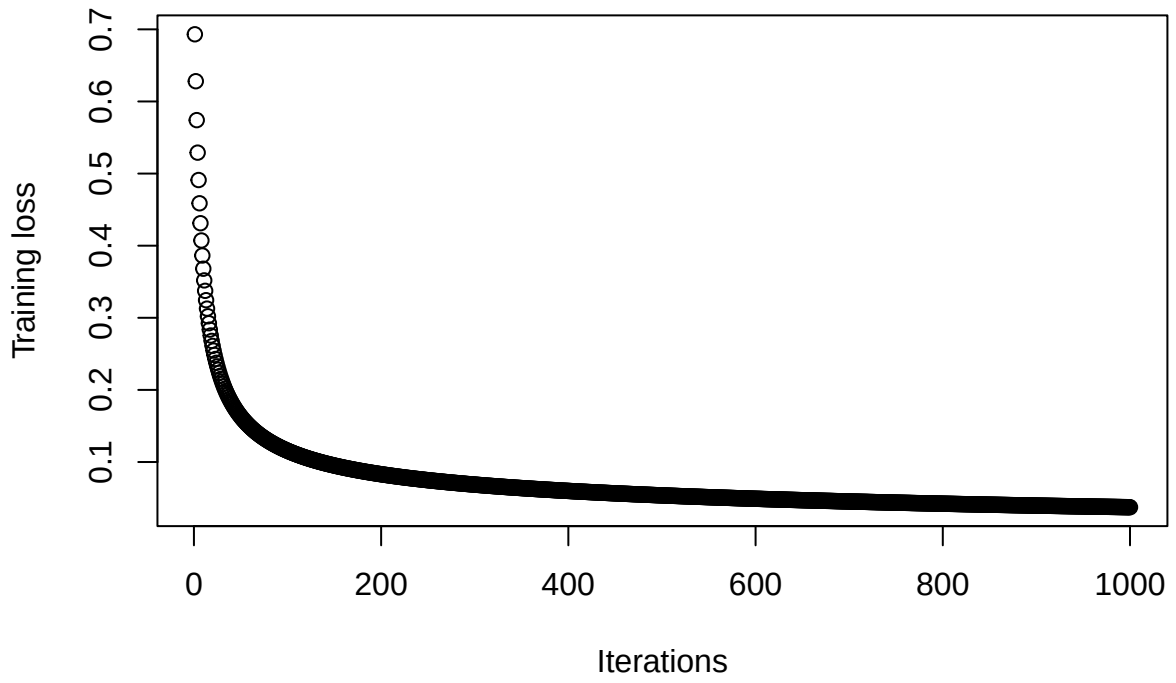
```r
x_train <- train$x
y_train <- train$y

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

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

Part a

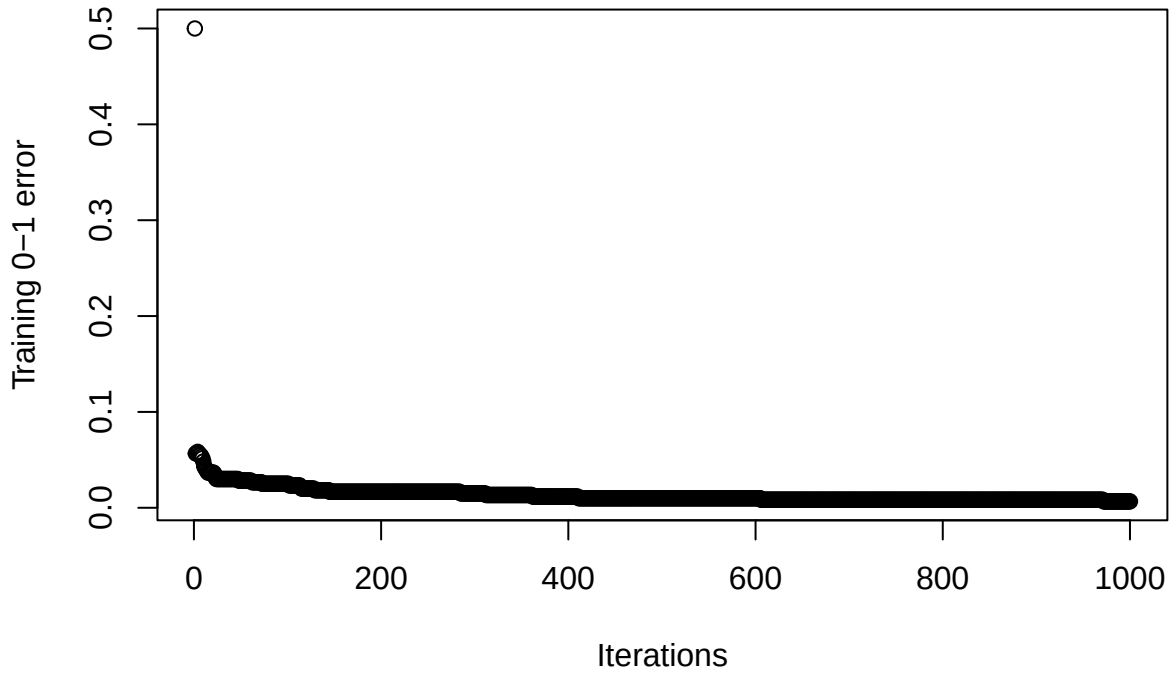
Find a suitable choice of `stepsize` and `max_iter`, draw the plots of training losses and training 0-1 errors.

```r
lbd = 0

stepsize = 0.1
max_iter = 1000
```

```r
res <- Penalized_Logistic_Reg(x_train, y_train, lbd, stepsize, max_iter)

plot(res$loss, xlab = "Iterations", ylab = "Training loss")
```




```r
plot(res$error, xlab = "Iterations", ylab = "Training 0-1 error")
```



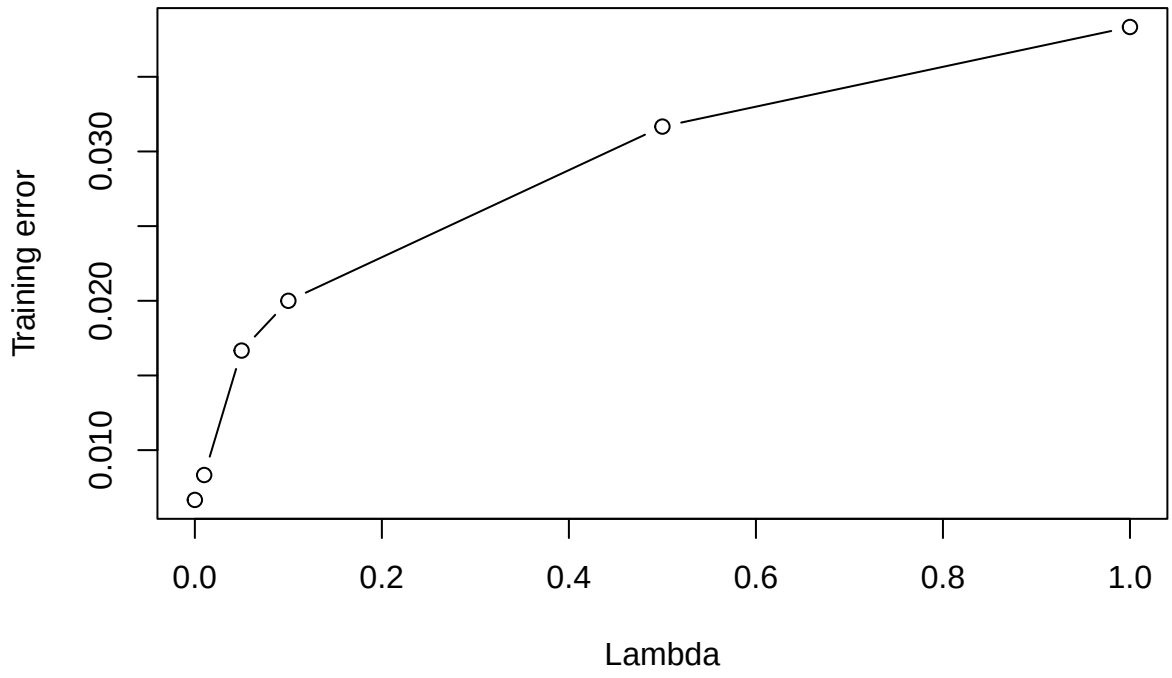
Choice of the `stepsize` and `max_iter` should ensure the convergence of gradient descent. (As long as the choice of hyperparameters ensures algorithmic convergence, we can provide full credit.)

The trend of training losses and training 0-1 errors are decreasing in general, but the training errors dropped much more quickly. Training losses should always be monotonically decreasing while this is not necessarily for the training 0-1 errors in general.

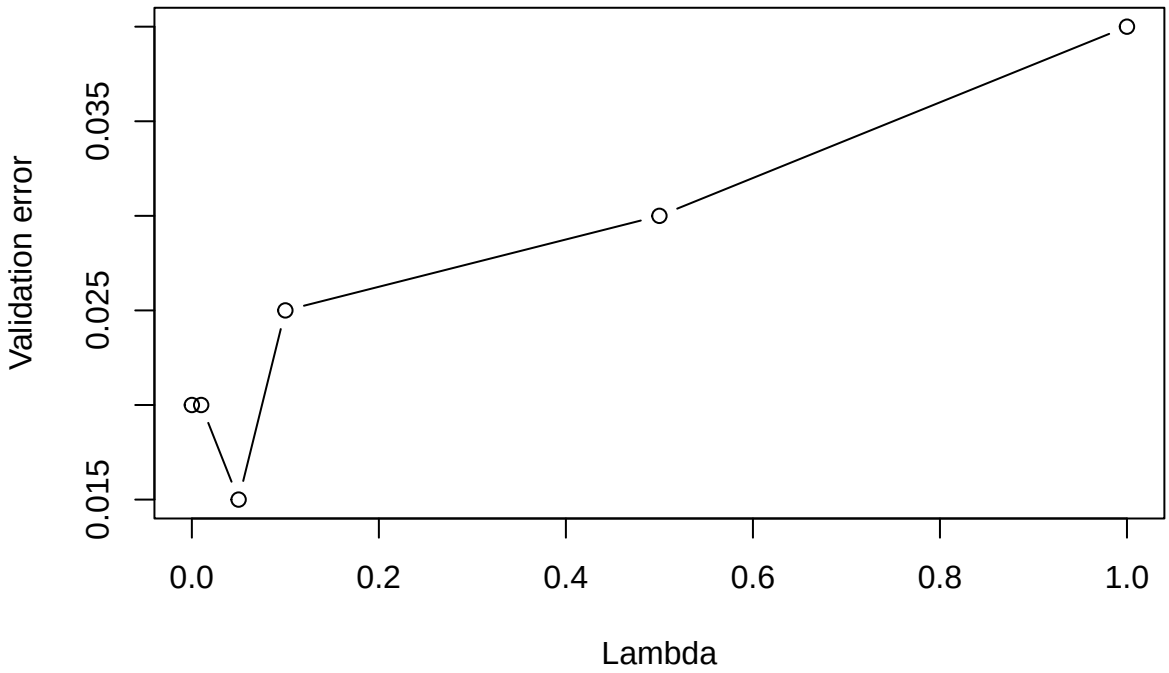
Part b

Identify suitable `stepsize` and `max_iter` for each `lambda` and draw the plots of training and validation 0-1 errors versus different values of `lambda`.

```
stepsize <- 0.1  # this should be replaced by your answer in Part a
max_iter <- 1000  # this should be replaced by your answer in Part a

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

train_error_vec <- valid_error_vec <- rep(NA, length(lbd_grid))

for (i in 1:length(lbd_grid)) {
  res_i <- Penalized_Logistic_Reg(x_train, y_train, lbd_grid[i], stepsize, max_iter)

  # Predict the classes
  pred_train <- Predict_logis(x_train, res_i$beta, res_i$beta0, type = "class")
  pred_valid <- Predict_logis(x_valid, res_i$beta, res_i$beta0, type = "class")

  # Compute the training and validation errors
  train_error_vec[i] <- Evaluate(y_train, pred_train)
  valid_error_vec[i] <- Evaluate(y_valid, pred_valid)
}


plot(lbd_grid, train_error_vec, xlab = "Lambda", ylab = "Training error", type = "b")
```




```
plot(lbd_grid, valid_error_vec, xlab = "Lambda", ylab = "Validation error", type = "b")
```



(Answers might vary across different choices of hyperparameters. But algorithmic convergence should be ensured.)

Choosing $\lambda > 0$ leads to larger training errors. For validation errors, the choice of $\lambda = 0.05$ yields the smallest.

Part c

Use the best `stepsize`, `max_iter` and `lbd` you found, fit the penalized logistic regression and compute its test 0-1 error.

```
stepsize <- 0.1  # this should be replaced by your answer in Part a
max_iter <- 1000  # this should be replaced by your answer in Part a
lbd <- 0.05       # this should be replaced by your answer in Part b


PenLogisModel <- Penalized_Logistic_Reg(x_train, y_train, lbd, stepsize, max_iter)

# Predict the classes on the test data
pred_test <- Predict_logis(x_test, PenLogisModel$beta, PenLogisModel$beta0, type = "class")
cat("The test error is", Evaluate(y_test, pred_test), "\n")
```

```
## The test error is 0.0225
```

```
# Use the glmnet
library(glmnet)
```

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

```
## Loaded glmnet 4.1-8
```

```
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 using glmnet is", Evaluate(y_test, as.numeric(pred_glmnet_test)), "\n")
```

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

Depending on the choice of hyperparamters, the results could be different slightly.

Part d

Compare with the LDA and Naive Bayes.

```
source("discriminant_analysis.R")

K <- length(unique(y_train))
priors <- Comp_priors(y_train, K)
means <- Comp_cond_means(x_train, y_train, K)
covs <- Comp_cond_covs(x_train, y_train, K, method = "LDA")
prob_mat <- Predict_posterior(x_test, priors, means, covs, method = "LDA")
pred_test <- Predict_labels(prob_mat)
acc_lda <- mean(pred_test != y_test)

covs_nb <- Comp_cond_covs(x_train, y_train, K, method = "NB")
prob_mat_nb <- Predict_posterior(x_test, priors, means, covs_nb, method = "NB")
pred_test_nb <- Predict_labels(prob_mat_nb)
acc_nb <- mean(pred_test_nb != y_test)

cat("The test errors are: LDA", acc_lda, "and NB:", acc_nb, "\n")
```

```
## The test errors are: LDA NaN and NB: 0.065
```

For LDA, the within-class covariance matrix is degenerate. So we don't have a numeric test error.

Optionally for this problem: to deal with issue, we can choose to add δI_p to the estimated within-class covariance matrix. Here $\delta > 0$ is chosen via the validation set.

```
delta_grid = c(0.01, 0.1, 0.2, 0.5)

acc <- c()

for (i in 1:length(delta_grid)) {
  delta_i = delta_grid[i]
  covs_i = covs + delta_i * diag(nrow = ncol(x_train))
  prob_mat_i <- Predict_posterior(x_valid, priors, means, covs_i, method = "LDA")
  pred_valid_i <- Predict_labels(prob_mat_i)
  acc[i] <- mean(pred_valid_i != y_valid)
}

delta_opt <- delta_grid[which.min(acc)]
cat("The best delta is", delta_opt, "\n")
```

```
## The best delta is 0.2
```

```
covs_new <- covs + delta_opt * diag(nrow = ncol(x_train))
prob_mat_new <- Predict_posterior(x_test, priors, means, covs_new, method = "LDA")
pred_test_new <- Predict_labels(prob_mat_new)
acc_lda_new <- mean(pred_test_new != y_test)

cat("The test errors are: LDA (new)", acc_lda_new, "and NB:", acc_nb, "\n")
```

```
## The test errors are: LDA (new) 0.0175 and NB: 0.065
```

Part e

Draw ROC curves of each methods on the test data.

```
library(pROC)
```

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

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

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

```
plot(roc(y_test, Predict_logis(x_test, PenLogisModel$beta, PenLogisModel$beta0, type = "prob")[,1]), col
     print.auc=TRUE)
```

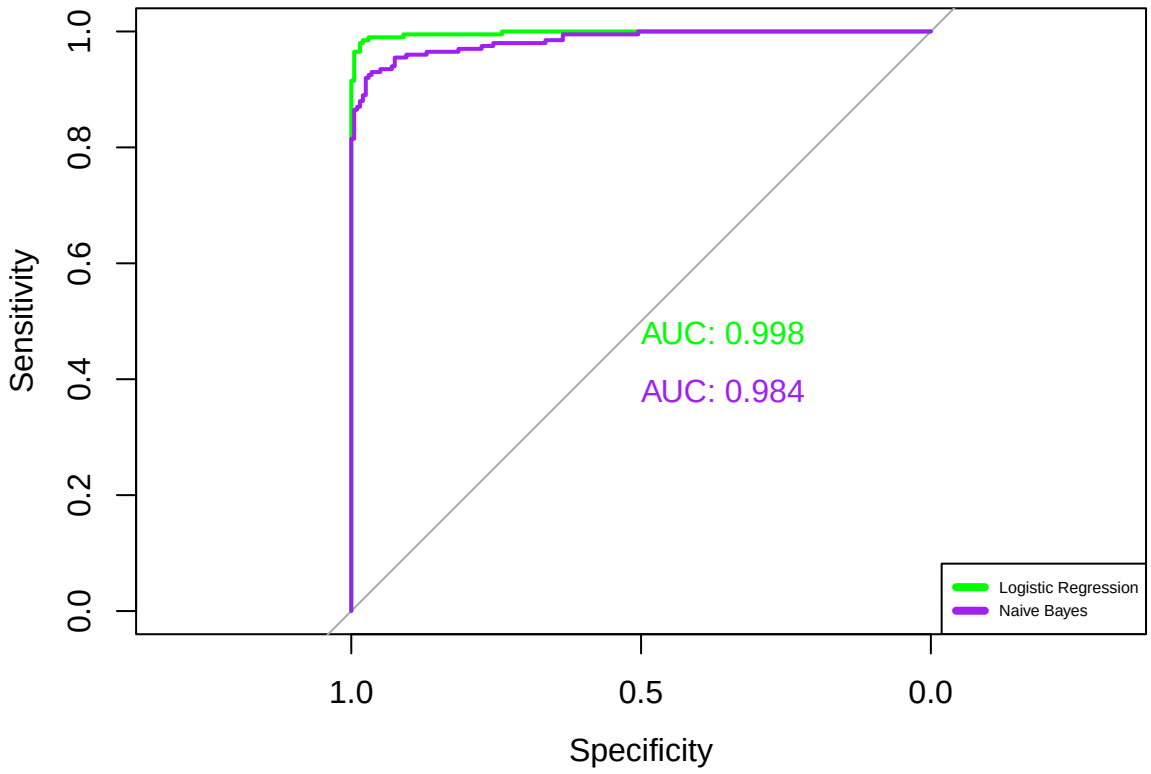
```
## Setting levels: control = 0, case = 1
```

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

```
# plot(roc(y_test, prob_mat[,2]), col="blue", add=TRUE, print.auc=TRUE,
#      print.auc.y=0.4)
plot(roc(y_test, prob_mat_nb[,2]), col="purple", add=TRUE, print.auc=TRUE,
     print.auc.y=0.4)
```

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

```
legend("bottomright",
       legend=c("Logistic Regression",  "Naive Bayes"),
       col=c("green",  "purple"),
       lwd=4, cex =0.5)
```



Since LDA is not applicable, we exclude it for drawing the ROC curve.

Optionally, if you add δI_p to the estimated within-class covariance matrix, we have the following plot.

```
plot(roc(y_test, Predict_logis(x_test, PenLogisModel$beta, PenLogisModel$beta0, type = "prob")[,1]), col
     print.auc=TRUE)
```

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

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

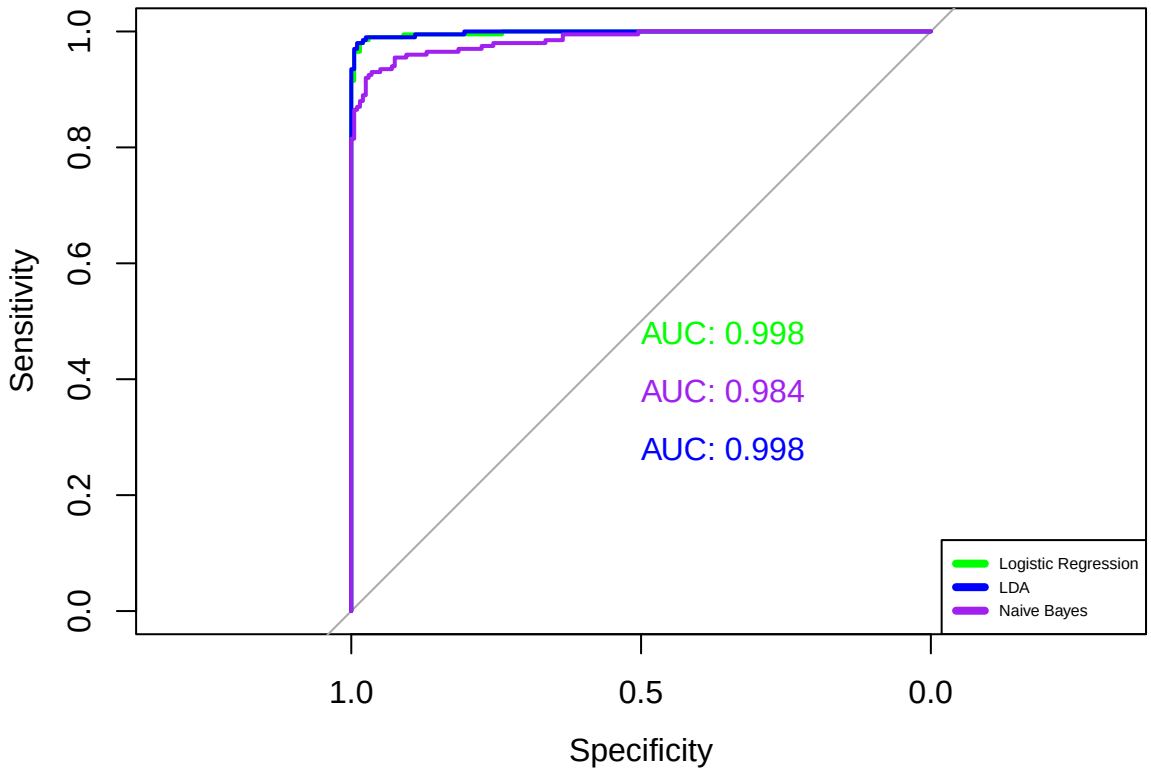
```
plot(roc(y_test, prob_mat_new[,2]), col="blue", add=TRUE, print.auc=TRUE,
     print.auc.y=0.3)
```

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

```
plot(roc(y_test, prob_mat_nb[,2]), col="purple", add=TRUE, print.auc=TRUE,
     print.auc.y=0.4)
```

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

```
legend("bottomright",
       legend=c("Logistic Regression", "LDA", "Naive Bayes"),
       col=c("green",  "blue", "purple"),
       lwd=4, cex =0.5)
```


Sensitivity
1.0
0.8
0.6
0.4
0.2
0.0
AUC: 0.998
AUC: 0.984
AUC: 0.998
Logistic Regression
LDA
Naive Bayes
1.0
0.5
0.0
Specificity