LDA, QDA and Naive Bayes for handwritten digits recognition

Alexandre H.

Part 1: Computation Functions

This section contains functions for computing priors, conditional means, and conditional covariance matrices for the classification models. It includes implementations for Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), and Naive Bayes (NB), with the flexibility to switch between these methods.

```r
library("mvtnorm") # for multivariate Gaussian density
```

```
## Warning: package 'mvtnorm' was built under R version 4.3.2
```

```r
Comp_priors <- function(train_labels, K) {
#' Compute the priors of each class label
#'
#' @param train_labels a vector of labels with length equal to n
#' @param K the number of classes in the response
#'
#' @return a probability vector of length K

pi_vec <- rep(0, K)

classes <- sort(unique(train_labels))

for (k in 1:K) {
pi_vec[k] <- sum(train_labels == classes[k]) / length(train_labels)
}

return(pi_vec)
}



Comp_cond_means <- function(train_data, train_labels, K) {
#' Compute the conditional means of each class
#'
#' @param train_data a n by p matrix containing p features of n training points
#' @param train_labels a vector of labels with length equal to n
#' @param K the number of levels in the response
#'
#' @return a p by K matrix, each column represents the conditional mean given
#'   each class.

p <- ncol(train_data)
mean_mat <- matrix(0, p, K)

classes <- sort(unique(train_labels))

for (k in 1:K) {
mean_mat[, k] <- colMeans(train_data[train_labels == classes[k], ])
}

return(mean_mat)
}



Comp_cond_covs <- function(train_data, train_labels, K, method = "LDA") {
#' Compute the conditional covariance matrix of each class
#'
#' @param train_data a n by p matrix containing p features of n training points
#' @param train_labels a vector of labels with length equal to n
#' @param K the number of levels in the response
#' @param method one of the methods in "LDA", "QDA" and "NB"
#'
#' @return
#'  if \code{method} is "QDA", return an array with dimension (p, p, K),
#'    containing p by p covariance matrices of each class;
#'  else if \code{method} is "NB", return a p by K matrix containing the
#'    diagonal covariance entries of each class;
#'  else return a p by p covariance matrix.

p <- ncol(train_data)

classes <- sort(unique(train_labels))

if (method == "QDA") {
cov_arr <- array(0, dim = c(p, p, K))
for (k in 1:K) {
cov_arr[, , k] <- cov(train_data[train_labels == classes[k], ])
}
} else if (method == "NB") {
cov_arr <- matrix(0, p, K)
for (k in 1:K) {
cov_arr[, k] <- diag(cov(train_data[train_labels == classes[k], ]))
}
} else {
cov_arr <- cov(train_data)
}

return(cov_arr)
}
```

Part 2: Prediction Functions

In this part, two functions are defined for making predictions. The first function predicts posterior probabilities for each class using LDA, QDA, or NB methods. The second function determines the final predicted labels based on these posterior probabilities.

```r
Predict_posterior <- function(test_data, priors, means, covs, method = "LDA") {

#' Predict the posterior probabilities of each class
#'
#' @param test_data a n_test by p feature matrix
#' @param priors a vector of prior probabilities with length equal to K
#' @param means a p by K matrix containing conditional means given each class
#' @param covs covariance matrices of each class, depending on \code{method}
#' @param method one of the methods in "LDA", "QDA" and "NB"
#'
#' @return a n_test by K matrix: each row contains the posterior probabilities
#'   of each class.

n_test <- nrow(test_data)
K <- length(priors)

posteriors <- matrix(0, n_test, K)

    if (method == "QDA") {
        for (k in 1:K) {
        posteriors[, k] <- dmvnorm(test_data, mean = means[, k], sigma = covs[, , k]) * priors[k]
        }
    } else if (method == "NB") {
        for (k in 1:K) {
        posteriors[, k] <- dmvnorm(test_data, mean = means[, k], sigma = diag(covs[, k])) * priors[k]
        }
    } else {
        for (k in 1:K) {
        posteriors[, k] <- dmvnorm(test_data, mean = means[, k], sigma = covs) * priors[k]
        }
    }

return(posteriors)
}


Predict_labels <- function(posteriors) {

#' Predict labels based on the posterior probabilities over K classes
#'
#' @param posteriors A n by K posterior probabilities
#'
#' @return A vector of predicted labels with length equal to n

n_test <- nrow(posteriors)
pred_labels <- rep(NA, n_test)
for (i in 1:n_test){
pred_labels[i] <- which.max(posteriors[i, ]) - 1
}

return(pred_labels)
}
```

Part 3: Application to Digits Data

This section focuses on applying the previously defined functions to a dataset of handwritten digits. It includes loading the data, plotting some digit examples, and implementing the LDA method for classifying the digits. Performance metrics such as misclassification rates and computation time are also calculated and displayed.

```
train <- Load_data(train_data_digits_directory)
```

```
## 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.
```

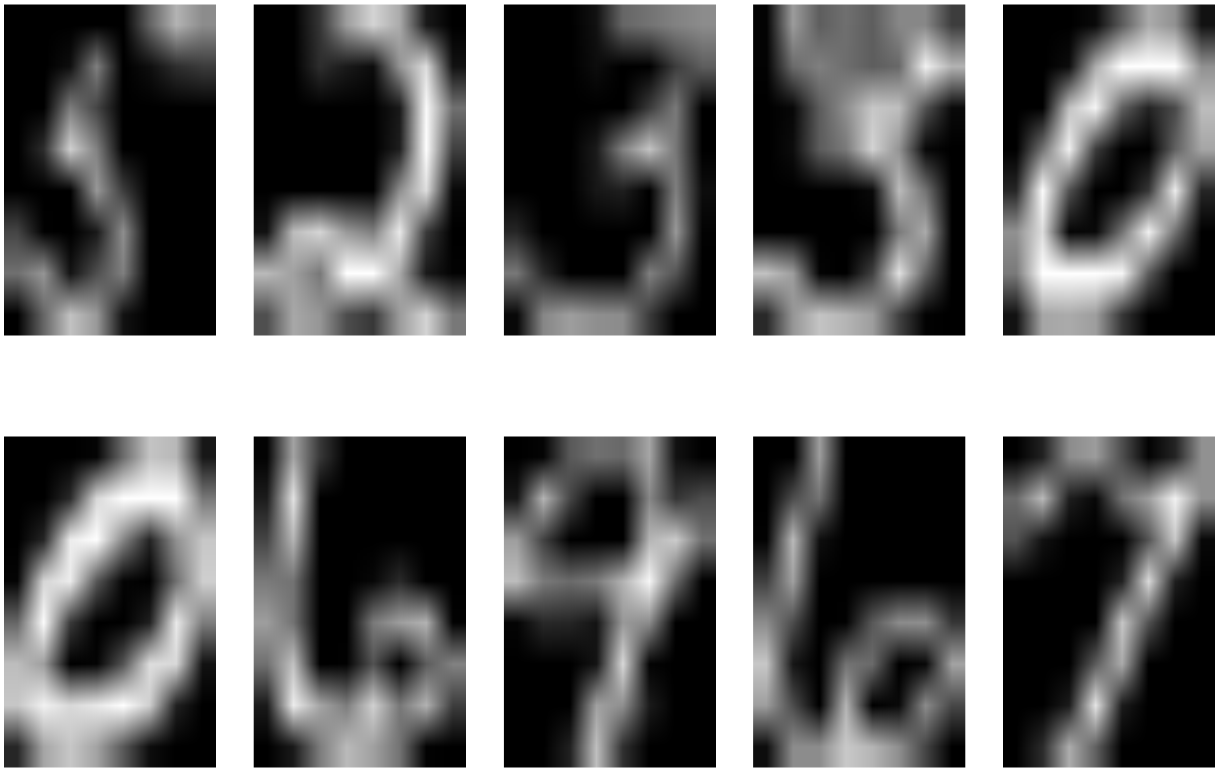
```
test <- Load_data(test_data_digits_directory)
```

```
## 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

Plot_digits(1:10, x_train)
```



```r
start_time <- Sys.time()

priors <- Comp_priors(train_labels = y_train, K = 10)
means <- Comp_cond_means(train_data = x_train, train_labels = y_train, K = 10)
covs <- Comp_cond_covs(train_data = x_train, train_labels = y_train, K = 10, method = "LDA")
posterior_train <- Predict_posterior(test_data = x_train, priors = priors,
                                     means = means, covs = covs, method = "LDA")
predicted_train <- Predict_labels(posterior_train)
posterior_test <- Predict_posterior(test_data = x_test, priors = priors,
                                    means = means, covs = covs, method = "LDA")
predicted_test <- Predict_labels(posterior_test)
miss_rate_train <- mean(predicted_train != y_train)
miss_rate_test <- mean(predicted_test != y_test)

end_time <- Sys.time()

results <- data.frame(model= "function LDA",
train_error=miss_rate_train,
test_error=miss_rate_test,
time=end_time - start_time)

pander::pander(tail(results, 1))
```

model	train_error	test_error	time
function LDA	0.1351	0.1502	0.3615 secs

Part 4: Comparison with LDA and QDA in MASS

Part 4 compares the custom QDA implementation with the QDA function available in the MASS package. It evaluates the models on the same handwritten digits dataset and appends the results to a summary dataframe for later comparison.

```r
start_time <- Sys.time()

priors <- Comp_priors(train_labels = y_train, K = 10)
means <- Comp_cond_means(train_data = x_train, train_labels = y_train, K = 10)
covs <- Comp_cond_covs(train_data = x_train, train_labels = y_train, K = 10, method = "QDA")
posterior_train <- Predict_posterior(test_data = x_train, priors = priors,
                                     means = means, covs = covs, method = "QDA")
predicted_train <- Predict_labels(posterior_train)
posterior_test <- Predict_posterior(test_data = x_test, priors = priors,
                                    means = means, covs = covs, method = "QDA")
predicted_test <- Predict_labels(posterior_test)
miss_rate_train <- mean(predicted_train != y_train)
miss_rate_test <- mean(predicted_test != y_test)

end_time <- Sys.time()

results <- rbind(results, data.frame(model= "function QDA",
train_error=miss_rate_train,
test_error=miss_rate_test,
time=end_time - start_time))

pander::pander(tail(results, 1))
```

	model	train_error	test_error	time
2	function QDA	0.01871	0.04075	0.3861 secs

Part 5: Comparison with Naive Bayes in MASS

This part introduces the Naive Bayes method for the digits recognition task. It applies the custom Naive Bayes model to the dataset, calculates performance metrics, and adds the results to the summary for comparison with LDA and QDA.

```r
start_time <- Sys.time()

priors <- Comp_priors(train_labels = y_train, K = 10)
means <- Comp_cond_means(train_data = x_train, train_labels = y_train, K = 10)
covs <- Comp_cond_covs(train_data = x_train, train_labels = y_train, K = 10, method = "NB")
posterior_train <- Predict_posterior(test_data = x_train, priors = priors,
                                     means = means, covs = covs, method = "NB")
```

```r
predicted_train <- Predict_labels(posterior_train)
posterior_test <- Predict_posterior(test_data = x_test, priors = priors,
                                   means = means, covs = covs, method = "NB")
predicted_test <- Predict_labels(posterior_test)
miss_rate_train <- mean(predicted_train != y_train)
miss_rate_test <- mean(predicted_test != y_test)

end_time <- Sys.time()

results <- rbind(results, data.frame(model= "function NB",
train_error=miss_rate_train,
test_error=miss_rate_test,
time=end_time - start_time))

pander::pander(tail(results, 1))
```

	model	train_error	test_error	time
3	function NB	0.167	0.1703	0.359 secs

Part 6: Comparison with LDA and QDA in MASS

In the final part, LDA and QDA models from the MASS package are applied to the digits data. It compares the performance of these standard implementations with the custom functions defined earlier, focusing on misclassification rates and computation time.

```r
library(MASS)

start_time <- Sys.time()

fit_lda <- lda(x_train, y_train)
posterior_train <- predict(fit_lda, x_train)$posterior
predicted_train <- Predict_labels(posterior_train)
posterior_test <- predict(fit_lda, x_test)$posterior
predicted_test <- Predict_labels(posterior_test)
miss_rate_train <- mean(predicted_train != y_train)
miss_rate_test <- mean(predicted_test != y_test)

end_time <- Sys.time()

results <- rbind(results, data.frame(model= "MASS LDA",
train_error=miss_rate_train,
test_error=miss_rate_test,
time=end_time - start_time))

start_time <- Sys.time()

fit_qda <- qda(x_train, y_train)
posterior_train <- predict(fit_qda, x_train)$posterior
predicted_train <- Predict_labels(posterior_train)
posterior_test <- predict(fit_qda, x_test)$posterior
```

```r
predicted_test <- Predict_labels(posterior_test)
miss_rate_train <- mean(predicted_train != y_train)
miss_rate_test <- mean(predicted_test != y_test)

end_time <- Sys.time()

results <- rbind(results, data.frame(model= "MASS QDA",
train_error=miss_rate_train,
test_error=miss_rate_test,
time=end_time - start_time))
```

```
pander::pander(results)
```

model	train_error	test_error	time
function LDA	0.1351	0.1502	0.3615 secs
function QDA	0.01871	0.04075	0.3861 secs
function NB	0.167	0.1703	0.359 secs
MASS LDA	0.09571	0.1022	0.2367 secs
MASS QDA	0.01871	0.04075	0.5337 secs

The results show a comparison of various machine learning models, with the Quadratic Discriminant Analysis (QDA) models, both standard and MASS versions, demonstrating the best performance with the lowest train and test errors (0.01871 and 0.04075, respectively). However, the MASS QDA model has a notably longer execution time. Linear Discriminant Analysis (LDA) models, in both standard and MASS implementations, show higher train and test errors but are faster, particularly the MASS LDA. The Naive Bayes (NB) model, while the fastest, exhibits the highest errors, indicating a trade-off between accuracy and speed across these models.