Question B: Predicting color name from RGB values, using discriminant analysis

Summer

2025-04-24

Common Function Set Up

We will reuse some quite common functions from to avoid redundancy, but we do implement QDA by hand.

In this question, you are not allowed to use any pre-implemented modules for performing discriminant analysis, but you'll have to implement this yourself.

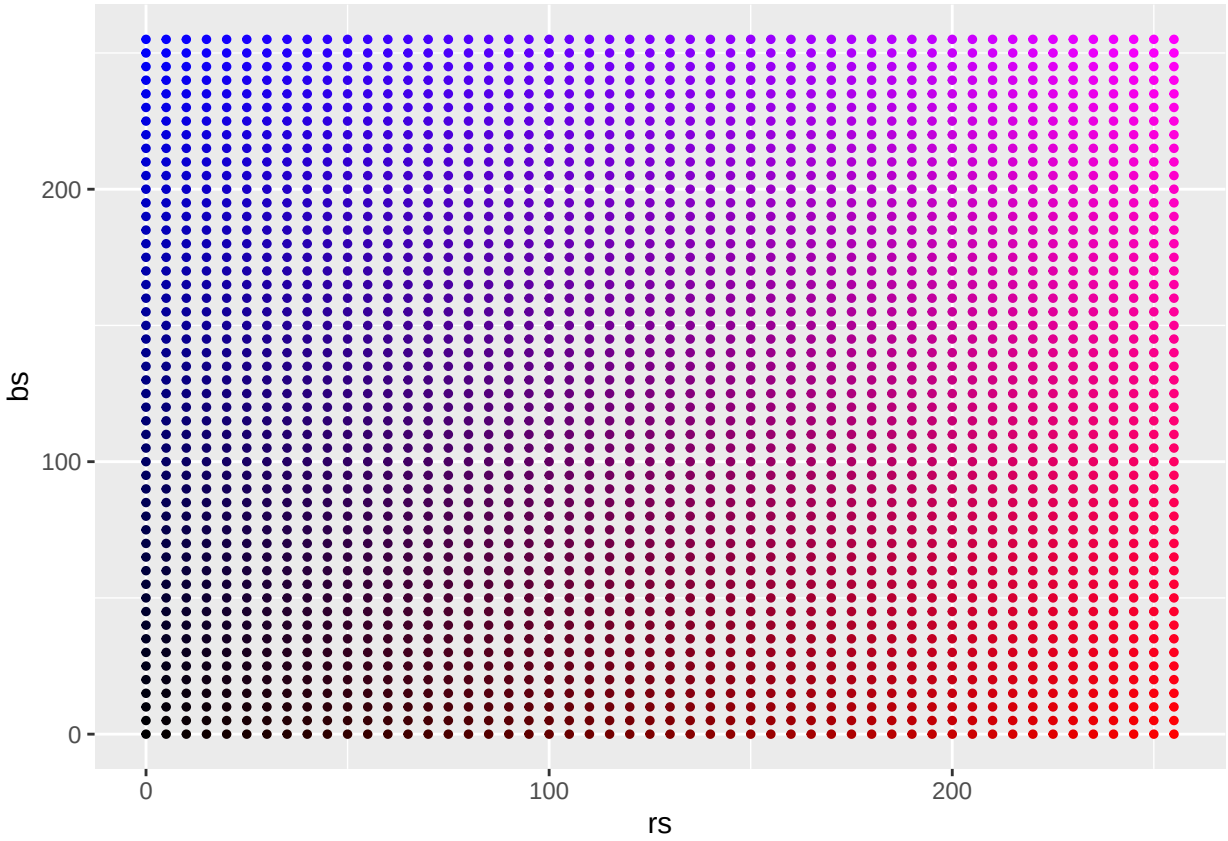
Background Information

Colors can be coded via their RGB (red, green, blue) value in the form `(r,g,b)`, where `r`, `g`, and `b` are integers between 00 and 255. For example, `(255,0,0)` is pure red, and `(128,200,128)` is a shade of green.

In this exercise we will map `(r,g,b)` values to their color names. For example, we want `(255,0,0)` to be classified as red.

In order to make things a bit easier, we focus on the part of color space where `g=0`, i.e. there is no green component. This means the feature space is all combinations `(r,0,b)`, where `r` and `b` are between 0 and 255. For orientation, here is a visualisation of some of these colors, with each circle having the color of its r-b-coordinate.

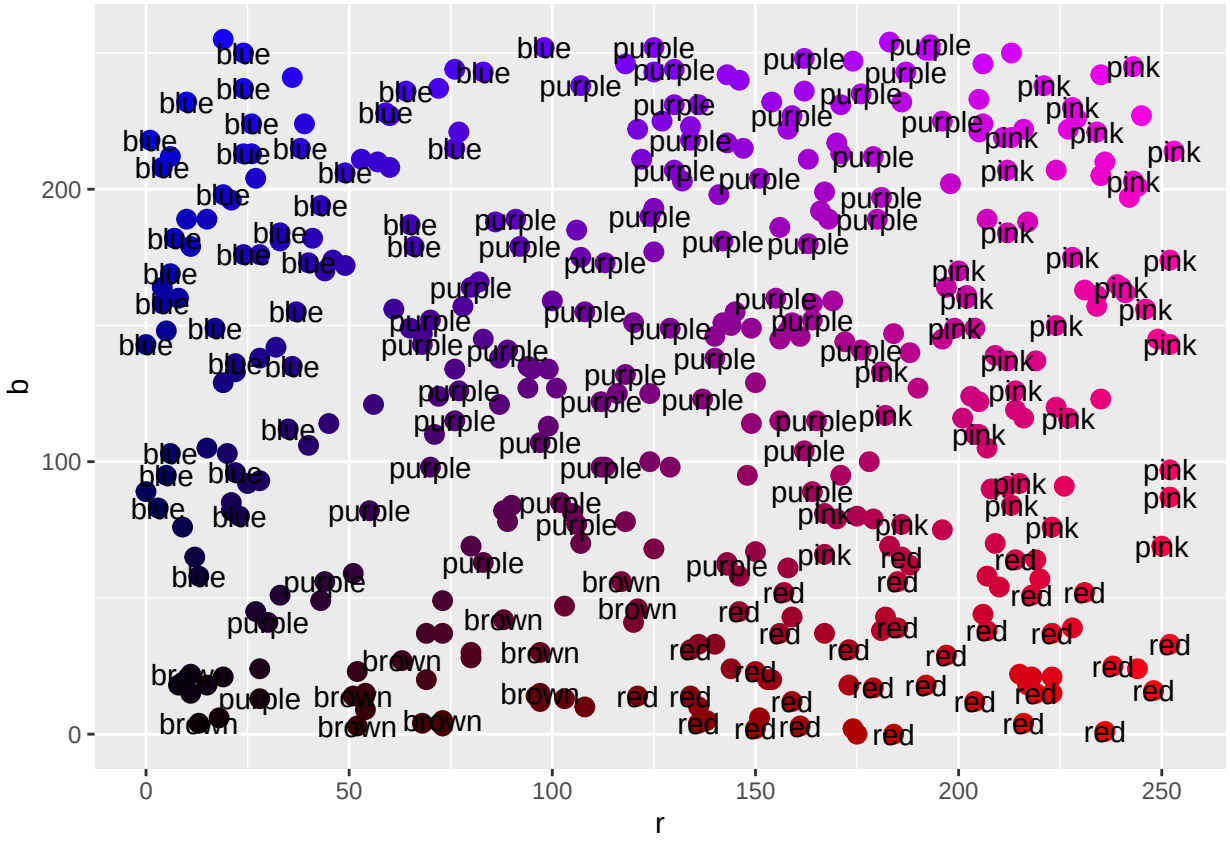
```
rs <- seq(0,256,5)
bs <- seq(0,256,5)
df_plot_colors <- data.frame(rs = rs, bs=bs) %>% tidyr::expand(rs, bs)
ggplot(data=df_plot_colors) +
  geom_point(aes(x=rs, y=bs, color=rgb(rs/256,0,bs/256)), size=1)+
  # R's rgb code works with numbers between 0 and 1 instead of between 0 and 255.
  scale_color_identity() +
  theme(legend.position = "none")
```



Our goal is to create a classification algorithm that takes an `r` value, and a `b` value, and outputs the name of the color this corresponds to.

Thankfully, we have a dataset where some of these labels have been entered. This is visualised below

```
df_colors <- read.csv("./data/unzipped/colors_train.csv")

df_colors_augmented <- df_colors %>% mutate(rgb = rgb(r/256,0,b/256))

ggplot(data=df_colors_augmented, aes( r, b, label = color)) +
  geom_point(aes(x=r, y=b, color=rgb), size=3) +
  geom_text(data = df_colors, check_overlap = TRUE) +
  scale_color_identity() +
  theme(legend.position = "none")
```



Solution

How many classes are there in the dataset?

```
classes <- df_colors_augmented %>% pull("color") %>% unique
class_num <- classes %>% length
class_num
```

```
## [1] 5
```

Therefore, there are 5 classes, which correspond to the following colors:

```
classes
```

```
## [1] "pink"   "purple" "red"    "brown"  "blue"
```

Fit a QDA algorithm to this classification problem and visualise the decision boundaries in a suitable way.

The general discriminant score function takes the following form:

$$\delta_k(\underline{x}) = -\frac{(\underline{x}-\underline{m}_k)^\top C_k^{-1}(\underline{x}-\underline{m}_k)}{2} - \frac{\log\det(C_k)}{2} + \log\pi_k$$

Now we are using 2 features, `red:=df_colors_augmented$r` and `blue:=df_colors_augmented$b`, to predict the class: `color:= df_colors_augmented$color`.

For each class $X = \underline{x}|class = k$, we need to know its *mean vector(m_k)*, *covariance matrix(C_k)* and the prior probability(π_k) . Here $\underline{x}$ represents a vector corresponding to class k.

Dataset Split

Use `split_data` function to split the dataset into a training set and a test set with the ratio of 8:2.

```
# split_data is the same function in Q1. Here we just reuse it.
df_colors <- split_data(data = df_colors_augmented,train_ratio = 0.8)
```

Build up discriminant score function

We have 5 classes as below:

```
df_colors$train %>% pull("color") %>% unique
```

```
## [1] "purple" "pink"   "red"    "brown"  "blue"
```

To improve the efficiency of calculation process, we define a function `stat_cal` to solve m_k, C_k, π_k.

```
# Function name: calculate_category_stats
# Arguments:
#   category_name: Category name (string)
#   df: DataFrame (pandas DataFrame)
# Returns:
#   A list containing the number of rows, column means, and covariance matrix for the category

stat_cal <- function(category_name, df) {
  # Filter the DataFrame for the specified category
  df_filtered <- df %>% filter(color == category_name)

  # Calculate the number of rows in the category
  n <- nrow(df_filtered)

  # Calculate the portion of rows in the category
  pi <- n/nrow(df)

  # Calculate the column means for 'r' and 'b'
  m <- df_filtered %>% select(r, b) %>% colMeans

  # Calculate the covariance matrix for 'r' and 'b'
  Sigma <- df_filtered %>% select(r, b) %>% cov

  return(list(pi = pi, m = m, Sigma = Sigma))
}
```

For all 5 classes, we can apply a `sapply` function to above calculation process to obtain the `stat_values` matrix, which combines n, π, m, σ for each class.

```
color_names=c("purple","pink","red","brown","blue")
color_stat_values <- sapply(color_names, function(color) {
  stat_cal(color, df = df_colors$train)
})

color_stat_values
```

```
##       purple    pink      red       brown     blue
## pi    0.371875  0.190625  0.159375  0.075     0.203125
## m     numeric,2 numeric,2 numeric,2 numeric,2 numeric,2
## Sigma numeric,4 numeric,4 numeric,4 numeric,4 numeric,4
```

N.B. We can verify the statistic values above by summing the `pi` values; the sum should equal 1. `pi` represents the prior probability, indicated by each class's frequency over all events.

```
sum(unlist(color_stat_values["pi",]))
```

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

Then we define a function `delta` to calculate the delta value, given class input ($X = x$) and its mean *vector*(m_k), prior probability(π_k) and *covariance* matrix(C_k).

```
# Function: Calculate the discriminant function value (delta) for a given sample.
#
# Args:
#   X: A numeric vector representing an observation.
#   mean: The mean vector of the class.
#   Sigma: The covariance matrix of the class.
#   pi: The prior probability of the class.
#
# Returns:
#   The discriminant function value for the sample belonging to the class.

# Corrected delta function
delta <- function(X, mean, Sigma, pi){
  X_minus_mean <- X - mean
  Sigma_inv <- solve(Sigma)
  dv <- -t(X_minus_mean) %*% Sigma_inv %*% X_minus_mean / 2 - log(det(Sigma))/2 + log(pi)
  return(dv)
}
```

Now, we can apply function `delta` to calculate classes' delta values of a given class.

```
X_purple <- df_colors$train %>% filter(color == "purple") %>% select(r, b)
X_purple_stat <- stat_cal("purple",df = df_colors$train)
delta_purple <- apply(X_purple, 1, function(row) {
  delta(X = as.numeric(row),
        mean = X_purple_stat$m,
        Sigma = X_purple_stat$Sigma,
        pi = X_purple_stat$pi)
})
```

Similarly, we apply a loop function to above delta calculation process.

```
# Create an empty list to store the results
delta_values <- list()

# Loop through each color category
for (color in color_names) {
  # 1. Extract data for the specific color
  X_color <- df_colors$train %>%
    filter(color == !!color) %>%
    select(r, b)

  # 2. Calculate statistics
  X_color_stat <- stat_cal(color, df = df_colors$train)

  # 3. Calculate discriminant values for each sample in this class
  delta_color <- apply(X_color, 1, function(row) {
    delta(X = as.numeric(row),
          mean = X_color_stat$m,
          Sigma = X_color_stat$Sigma,
          pi = X_color_stat$pi)
  })

  # 4. Store the results in the list
  delta_values[[color]] <- list(
    X = X_color,
    stat = X_color_stat,
    delta = delta_color
  )
}
```

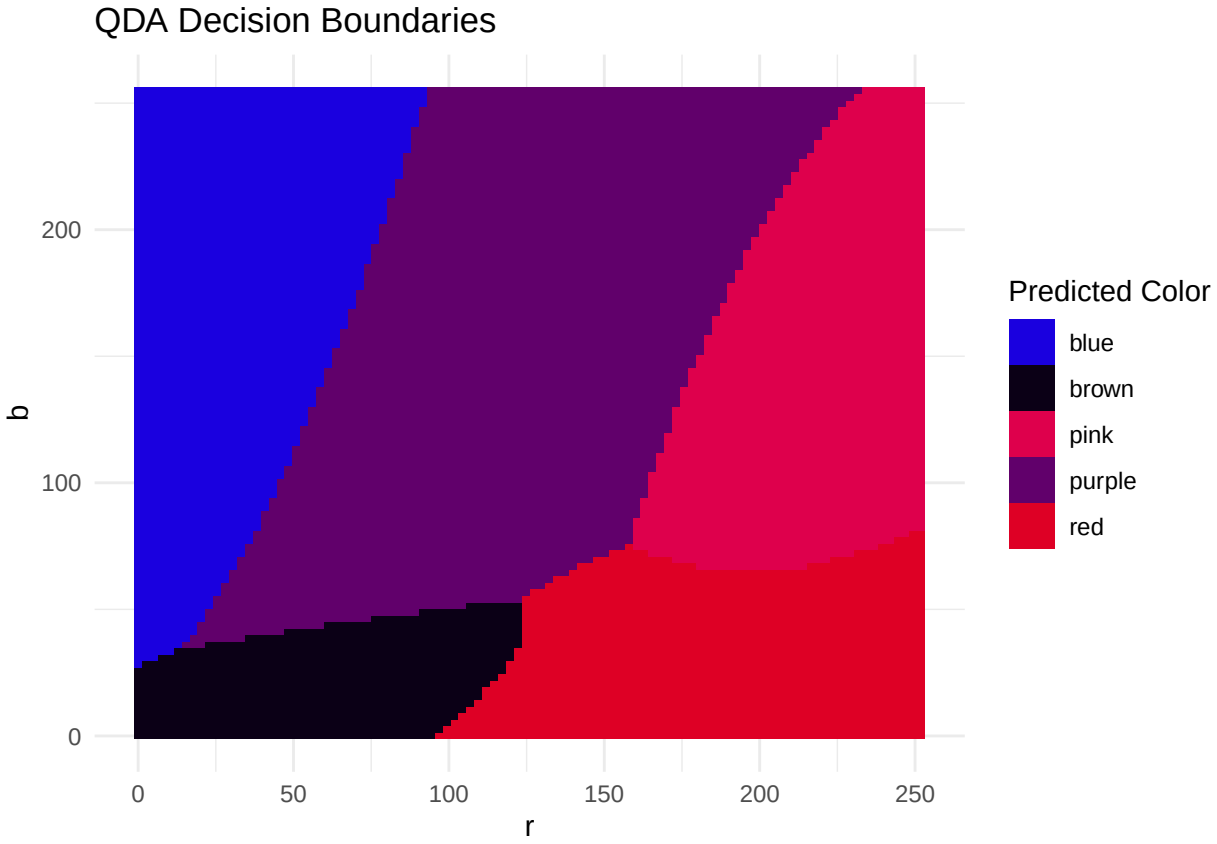
Visualise The Decision Boundaries

Now we have stored all classes' delta values in list `delta_values` and we can call them by using `delta_values[["purple"]]$delta`. Then, we use these 5 delta values to visualise the decision boundary.

```
# 1. Generate a 2D grid of (r, b) values to evaluate decision boundaries
plot_grid <- expand.grid(
  r = seq(min(df_colors$train$r), max(df_colors$train$r), length.out = 100),
  b = seq(min(df_colors$train$b), max(df_colors$train$b), length.out = 100)
)

# 2. Predict the color class for each grid point using QDA
plot_grid <- plot_grid %>%
  rowwise() %>%
  mutate(pred_by_hand = {
    # For each point (r, b), compute the discriminant score for each class
    deltas <- sapply(color_names, function(color) {
      delta(
        X = c(r, b),
        mean = delta_values[[color]]$stat$m,
        Sigma = delta_values[[color]]$stat$Sigma,
        pi = delta_values[[color]]$stat$pi
```

```
      )
    })
    # Assign the class with the highest discriminant score
    color_names[which.max(deltas)]
  }) %>%
  ungroup()
```

```
library(tidyr)

# Create a named vector that maps each class label to a representative RGB color
color_map <- df_colors_augmented %>%
  group_by(color) %>%
  summarise(rgb_color = first(rgb), .groups = "drop") %>%
  { setNames(.$rgb_color, .$color) }

# Plot the QDA decision boundaries using the predicted class for each point on the grid
ggplot(plot_grid, aes(x = r, y = b, fill = pred_by_hand)) +
  geom_tile() +
  scale_fill_manual(values = color_map) +
  theme_minimal() +
  labs(title = "QDA Decision Boundaries", fill = "Predicted Color")
```



Testing point(200,0,200)

```
# Define the test point
test_point <- c(200, 200)  # (r = 200, b = 200)

# Compute the QDA discriminant score for each color class
test_deltas <- sapply(color_names, function(color) {
  delta(
    X = test_point,
    mean = delta_values[[color]]$stat$m,
    Sigma = delta_values[[color]]$stat$Sigma,
    pi = delta_values[[color]]$stat$pi
  )
})

# Get predicted class: the one with the highest delta value
predicted_qda200 <- color_names[which.max(test_deltas)]
predicted_qda200
```

```
## [1] "purple"
```

Testing on test data set

```
#
predict_qda_manual <- function(point) {
  deltas <- sapply(color_names, function(color) {
    delta(
      X = point,
      mean = delta_values[[color]]$stat$m,
      Sigma = delta_values[[color]]$stat$Sigma,
      pi = delta_values[[color]]$stat$pi
    )
  })
  #   delta
  return(color_names[which.max(deltas)])
}

#
test_data <- df_colors$test %>% select(r,b)

predicted_labels_QDA <- apply(test_data, 1, function(row) {
  point <- c(row['r'], row['b'])
  predict_qda_manual(point)
})
```

kNN-Classification

Algorithm Hypothsis

Compared to the QDA method, kNN uses **Euclidean distance** to measure the similarity between sample points and test points. For a given input feature point $X = x$, we identify x's k nearest neighbors in the

training set. The class label for the test point x is determined by a majority vote among these k neighbors. Specifically, we calculate the distances from x to each of its k neighbors and assign x the label that appears most frequently among them.

```r
# Calculate Euclidean distance
euclidean_distance <- function(x1, x2) {
  sqrt(sum((x1 - x2) ^ 2))
}
```

Euclidean distance

Find Neighbors

1. **Function Purpose**: The function `find_neighbors` is designed to find the k nearest neighbors for a given test sample from the training data.
2. **Distance Calculation**: The `apply` function is used to iterate over each row in the specified columns ("r" and "b") of the `train_data`, calculating the Euclidean distance between each training sample and the `test_sample`.
3. **Return Neighbors**: The `order` function is used to sort the distances and return the indices of the k smallest distances, which correspond to the k nearest neighbors.

```r
# Find k nest neighbours among train data set
find_neighbors <- function(train_data, test_sample, k) {
  distances <- apply(train_data[, c("r", "b")], 1, function(train_sample) {
    euclidean_distance(train_sample, test_sample)
  })
  # Return the index of the minimum distance
  order(distances)[1:k]
}
```

Voting Out Highest Frequency Lable The `get_majority_vote` function determines the most common label among the nearest neighbors.

```r
# get_majority_vote
#
# Purpose:
# Determines the most common label among the nearest neighbors.
#
# Parameters:
# - neighbors: A vector of indices representing the nearest neighbors.
# - train_labels: A vector containing the labels of the training data.
#
# Returns:
# - The label that appears most frequently among the neighbors.
get_majority_vote <- function(neighbors, train_labels) {
  neighbor_labels <- train_labels[neighbors]
  # Return the most frequent label
  names(sort(table(neighbor_labels), decreasing = TRUE))[1]
}
```

```r
# knn_classify
#
# Purpose:
# Classifies test data using the k-Nearest Neighbors algorithm.
#
# Parameters:
# - train_data: A data frame containing the training samples with features.
# - train_labels: A vector containing the labels for the training data.
# - test_data: A data frame containing the test samples to classify.
# - k: The number of nearest neighbors to consider for classification.
#
# Returns:
# - A vector of predicted labels for the test data.
knn_classify <- function(train_data, train_labels, test_data, k) {
  predictions <- sapply(1:nrow(test_data), function(i) {
    test_sample <- test_data[i, c("r", "b")]
    neighbors <- find_neighbors(train_data, test_sample, k)
    get_majority_vote(neighbors, train_labels)
  })
  return(predictions)
}
```

```r
# Classify test data using k-Nearest Neighbors
#
# Assigns labels to test samples based on the training data.
#
# Inputs:
# - df_colors$train: Training data with features ("r", "b") and labels.
# - train_labels: Vector of labels for the training data.
# - test_data: Data frame of test samples with features ("r", "b").
# - k: Number of neighbors to consider.
#
# Output:
# - knn_predictions: Predicted labels for the test data.
knn_predictions <- knn_classify(df_colors$train, train_labels = df_colors$train$color, df_colors$test,
```

KNN classification model training

```r
# Define the test point
test_point <- data.frame(r = 200, b = 200)

# Find the indices of the nearest neighbors
neighbors_indices <- find_neighbors(train_data = df_colors$train, test_sample = test_point, k=5)

# Retrieve the labels of the neighbors
neighbor_labels <- df_colors$train$color[neighbors_indices]
```

```r
# Perform majority voting to determine the predicted label
predicted_knn200 <- get_majority_vote(neighbors_indices, df_colors$train$color)

# Output the prediction result
print(paste("The predicted label for point (200,0,200) is:", predicted_knn200))
```

Test point (200,0,200)

```
## [1] "The predicted label for point (200,0,200) is: pink"
```

Model Comparision On Test Data Set

```r
confusion_metrics_kNN <- confusionMatrix(factor(knn_predictions), factor(df_colors$test$color))
```

kNN Classification Performance

```r
confusion_metrics_QDA <- confusionMatrix(factor(predicted_labels_QDA), factor(df_colors$test$color))
```

```r
#
cm1 <- confusion_metrics_kNN
cm2 <- confusion_metrics_QDA

metrics_df <- data.frame(
  Metric = c("Accuracy", "Kappa", "Sensitivity", "Specificity", "Precision", "Recall", "F1"),
  kNN = c(
    cm1$overall['Accuracy'],
    cm1$overall['Kappa'],
    mean(cm1$byClass[,'Sensitivity'], na.rm = TRUE),
    mean(cm1$byClass[,'Specificity'], na.rm = TRUE),
    mean(cm1$byClass[,'Precision'], na.rm = TRUE),
    mean(cm1$byClass[,'Recall'], na.rm = TRUE),
    mean(cm1$byClass[,'F1'], na.rm = TRUE)
  ),
  QDA = c(
    cm2$overall['Accuracy'],
    cm2$overall['Kappa'],
    mean(cm2$byClass[,'Sensitivity'], na.rm = TRUE),
    mean(cm2$byClass[,'Specificity'], na.rm = TRUE),
    mean(cm2$byClass[,'Precision'], na.rm = TRUE),
    mean(cm2$byClass[,'Recall'], na.rm = TRUE),
    mean(cm2$byClass[,'F1'], na.rm = TRUE)
  )
)

#
kable(metrics_df, caption = "      ")
```

QDA Classification Performance

Table 1:

Metric	kNN	QDA
Accuracy	0.9750000	0.9750000
Kappa	0.9663866	0.9663866
Sensitivity	0.9835484	0.9835484
Specificity	0.9943922	0.9943922
Precision	0.9512821	0.9512821
Recall	0.9835484	0.9835484
F1	0.9654113	0.9654113