Solução Lista 05

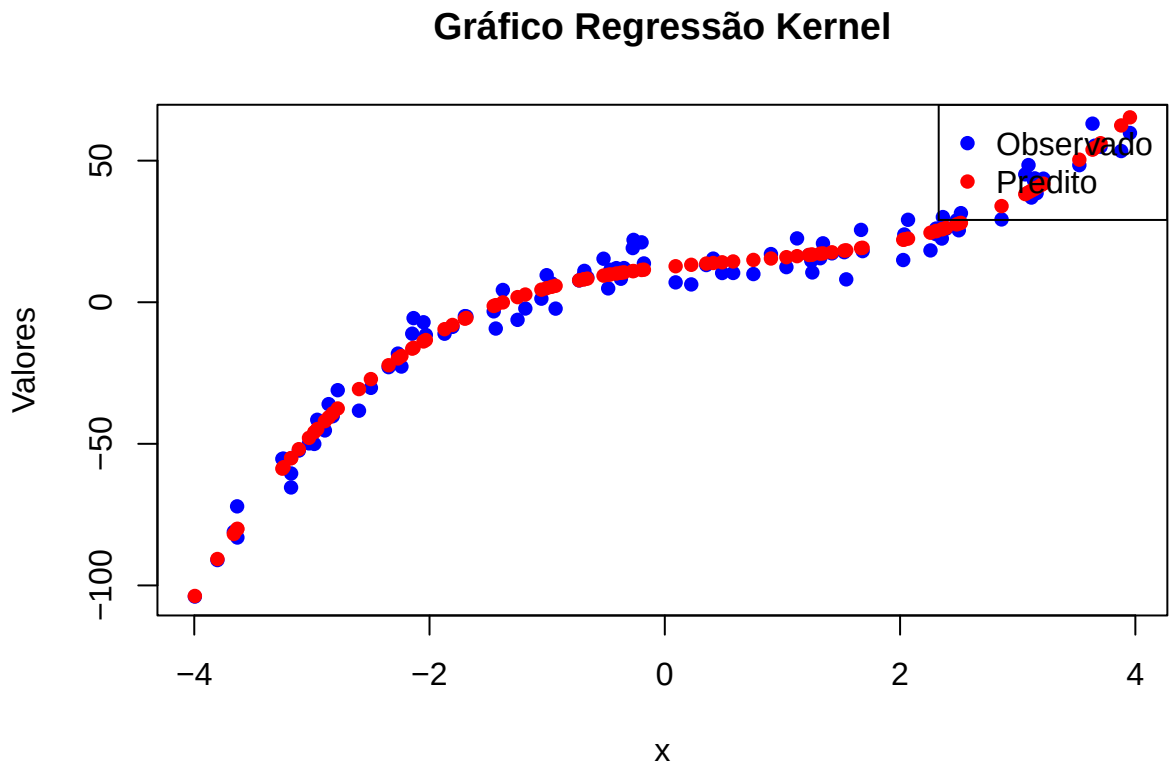
Nome: Vinicius de Oliveira Bezerra
E-mail: v.bezerra@aluno.ufabc.edu.br
Nome: Deyved Kevyn Alves Lima
E-mail: deyved.lima@aluno.ufabc.edu.br

25 March, 2025

Resolução Exercício 1

```r
# Gerando os dados
set.seed(123) # Para resultados reproduzíveis
library(dplyr)
library(tibble)
df <- tibble(
  x1 = runif(100, -4, 4),
  y = x1^3 - 2 * x1^2 + 5 * x1 + 13 + rnorm(100, 0, 5.0)
)

# (a) Função para calcular a matriz kernel polinomial de grau 3
calc_kernel_matrix <- function(x) {
  n <- length(x)
  K <- matrix(0, n, n)
  for (i in 1:n) {
    for (j in 1:n) {
      K[i, j] <- (x[i] * x[j] + 1)^3
    }
  }
  return(K)
}

# (b) Função para calcular os coeficientes alpha com Ridge Regression
calc_alpha <- function(K, y, lambda) {
  n <- nrow(K)
  I <- diag(n) # Matriz identidade
  alpha <- solve(K + lambda * I) %*% y
  return(alpha)
}

# (c) Predições e plot do resultado
predict_ridge <- function(alpha, K) {
  return(K %*% alpha)
}
```

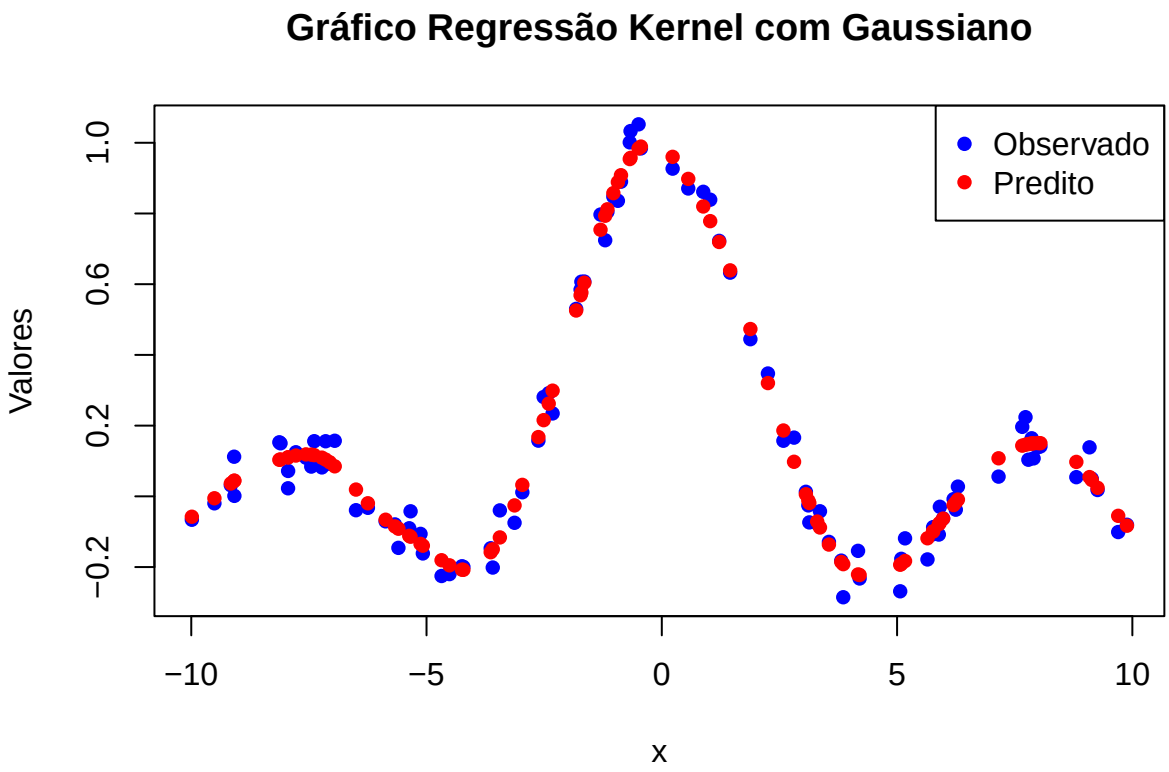
```r
plot_prediction <- function(x, y, y_pred) {
  plot(x, y, main = "Gráfico Regressão Kernel", xlab = "x", ylab = "Valores",
       col = "blue", pch = 16)
  points(x, y_pred, col = "red", pch = 16)
  legend("topright", legend = c("Observado", "Predito"), col = c("blue", "red"), pch = 16)
}

# Etapas do exercício
x <- df$x1
y <- df$y
K <- calc_kernel_matrix(x)
lambda <- 0.001
alpha <- calc_alpha(K, y, lambda)
y_pred <- predict_ridge(alpha, K)

# Gráfico
plot_prediction(x, y, y_pred)
```



Solução Exercício 2

```r
# Gerando os dados
set.seed(123) # Para resultados reproduzíveis
```

```r
library(tibble)
df <- tibble(
  x1 = runif(100, -10, 10),
  y = if_else(x1 == 0, 1, sin(x1) / x1) + rnorm(100, 0, 0.05)
)

# Função para calcular a matriz kernel Gaussiana
calc_kernel_matrix <- function(x, sigma = 1) {
  n <- length(x)
  K <- matrix(0, n, n)
  for (i in 1:n) {
    for (j in 1:n) {
      K[i, j] <- exp(-(x[i] - x[j])^2 / (2 * sigma^2))
    }
  }
  return(K)
}

# Função para calcular os coeficientes alpha com Ridge Regression
calc_alpha <- function(K, y, lambda) {
  n <- nrow(K)
  I <- diag(n) # Matriz identidade
  alpha <- solve(K + lambda * I) %*% y
  return(alpha)
}

# Função para realizar predições
predict_ridge <- function(alpha, K) {
  return(K %*% alpha)
}

# Gráfico das predições
plot_prediction <- function(x, y, y_pred) {
  plot(x, y, main = "Gráfico Regressão Kernel com Gaussiano", xlab = "x", ylab = "Valores",
       col = "blue", pch = 16)
  points(x, y_pred, col = "red", pch = 16)
  legend("topright", legend = c("Observado", "Predito"), col = c("blue", "red"), pch = 16)
}

# Etapas do exercício
x <- df$x1
y <- df$y
sigma <- 1
K <- calc_kernel_matrix(x, sigma)
lambda <- 0.1 # Teste diferentes valores, como 0.001, 0.01, 0.1, etc.
alpha <- calc_alpha(K, y, lambda)
y_pred <- predict_ridge(alpha, K)

# Gráfico
plot_prediction(x, y, y_pred)
```



Solução Exercício 3

A)

O LDA terá a melhor performance tanto no treinamento quanto no teste, pois sua suposição de fronteira linear está alinhada com a verdadeira fronteira de decisão dos dados. O QDA, embora mais flexível, pode sofrer overfitting em conjuntos pequenos devido à sua complexidade desnecessária, reduzindo sua eficiência no teste.

B)

O QDA será melhor, pois consegue capturar relações não-lineares entre as classes, diferente do LDA, que por ser restrito a fronteiras lineares, terá desempenho pior, já que não consegue modelar a complexidade real dos dados. Em ambos os casos, a performance no treinamento e teste seguirá essa tendência, desde que o QDA não sofra overfitting por falta de dados.

Solução Exercício 4

```
# Importações
library(tidyverse)
```

```r
library(tidymodels)
library(discrim)
library(klaR)
library(mlbench)
library(MASS)  # Para LDA/QDA tradicional

# Definir Modelos
model_lda = discrim_linear() %>%
  set_engine("MASS") %>%  # Alterado para MASS
  set_mode("classification")

model_qda = discrim_quad() %>%
  set_engine("MASS") %>%  # Alterado para MASS
  set_mode("classification")

# Naive Bayes continua com klaR
model_nb = naive_Bayes() %>%
  set_engine("klaR") %>%
  set_mode("classification")

# Configurar os dados
data(Ionosphere, package = "mlbench")
df = as_tibble(Ionosphere) %>%
  select(-V1, -V2) %>%  #Remover as primeiras colunas
  mutate(Class = as.factor(Class))

# Validação cruzada (k=5)
folds <- vfold_cv(df, v = 5)

# Avaliar os modelos
avaliar_modelo <- function(modelo) {
  tryCatch({
    workflow() %>%
      add_model(modelo) %>%
      add_formula(Class ~ .) %>%
      fit_resamples(folds, control = control_resamples(save_pred = TRUE)) %>%
      collect_metrics()
  }, error = function(e) {
    message("Erro no modelo: ", e$message)
    return(NULL)
  })
}

# Executar os modelos
resultados = list(
  lda = avaliar_modelo(model_lda),
  qda = avaliar_modelo(model_qda),
  nb = avaliar_modelo(model_nb)
) %>% compact()  # Remove modelos com erro

# Obter os resultados
if (length(resultados) > 0) {
  acuracias = map_dbl(resultados, ~.x %>%
```

```
                        filter(.metric == "accuracy") %>%
                        pull(mean))

  cat("\n RESULTADOS \n")
  imap(resultados, ~cat(.y, ":", .x %>%
                         filter(.metric == "accuracy") %>%
                         pull(mean) %>% round(4), "\n"))

  cat("\n Melhor método:", names(which.max(acuracias)), "\n")
} else {
  cat("Todos os modelos falharam.")
}
```

```
##
##  RESULTADOS
## lda : 0.8433
## qda : 0.8689
## nb : 0.9089
##
##  Melhor método: nb
```

O melhor método entre os 3 foi o Naive Bayes. Isso provavelmente aconteceu porque o método apesar de suas suposições simplificadas, adaptou-se melhor à distribuição dos dados do dataset. Enquanto LDA e QDA assumem distribuições normais subjacentes, o Naive Bayes é mais flexível com distribuições não normais e pode lidar melhor com certos tipos de ruídos ou relações não lineares implícitas nos dados, onde as relações entre variáveis podem não ser perfeitamente lineares ou quadráticas.