Lista 02 - Aprendizado de Máquinas

04 March, 2025

Solução Exercício 01

```
library(tidyverse) # Manipulação dos dados
library(tidymodels) # Modelagem
library(kknn) # kNN

set.seed(42) # Repodutibilidade

# Função f(x)
f <- function(x) {x^2 - 2*x^3 + 1}

# Gerando dados para treinamento (50 pontos)
x_train <- runif(50, min = -1, max = 1)
y_train <- f(x_train) + rnorm(50, mean = 0, sd = 0.5)
train <- tibble(x = x_train, y = y_train)

# Gerando dados para teste (100 prontos)
x_test <- runif(100, min = -1, max = 1)
y_test <- f(x_test) + rnorm(100, mean = 0, sd = 0.5)
test <- tibble(x = x_test, y = y_test)

# ------------------------------------------------------------------------------
# (i) Criando o modelo de Regressção Linear
# ------------------------------------------------------------------------------
lin.model <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

# Treinamento
lin.fit <- lin.model %>%
  fit(y ~ x, data = train)

# Predição e cálculo do erro RMSE
test.pred.lin <- lin.fit %>%
  predict(new_data = test) %>% bind_cols(test)
  rmse_lin <- rmse(test.pred.lin, truth = y, estimate = .pred)
  print(rmse_lin)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.648
```

```r
# ------------------------------------------------------------------------------
# (ii) Criando modelo kNN com k = 1
# ------------------------------------------------------------------------------
knn1.model <- nearest_neighbor(neighbors = 1, weight_func = "rectangular", dist_power = 2) %>%
  set_engine("kknn") %>%
  set_mode("regression")

# Treinamento
knn1.fit <- knn1.model %>%
  fit(y ~ x, data = train)

# Predição e cálculo do erro RMSE
test.pred.knn1 <- knn1.fit %>% predict(new_data = test) %>% bind_cols(test)
rmse_knn1 <- rmse(test.pred.knn1, truth = y, estimate = .pred)
print(rmse_knn1)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.674
```

```r
# ------------------------------------------------------------------------------
# (iii) Criando modelo kNN com k = 10
# ------------------------------------------------------------------------------
knn10.model <- nearest_neighbor(neighbors = 10, weight_func = "rectangular", dist_power = 2) %>%
  set_engine("kknn") %>%
  set_mode("regression")

# Treinamento
knn10.fit <- knn10.model %>% fit(y ~ x, data = train)

# Predição e cálculo do erro RMSE
test.pred.knn10 <- knn10.fit %>% predict(new_data = test) %>% bind_cols(test)
rmse_knn10 <-rmse(test.pred.knn10, truth = y, estimate = .pred)
print(rmse_knn10)
```

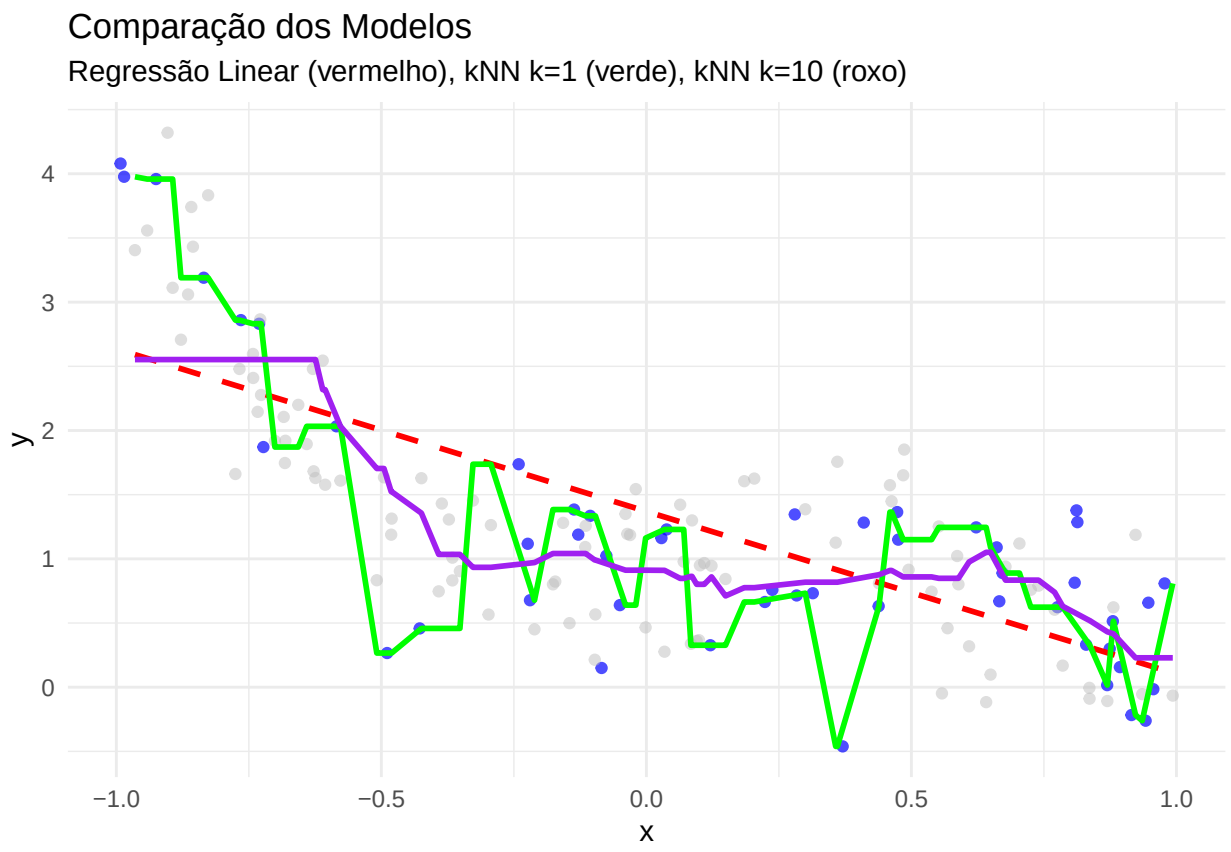
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.569
```

```r
# ------------------------------------------------------------------------------
# (iv) Comparando os resultados
# ------------------------------------------------------------------------------
resultados <- tibble(
  Modelo = c("Regressão Linear", "kNN (k = 1)", "kNN (k = 10)"),
  RMSE = c(rmse_lin$.estimate, rmse_knn1$.estimate, rmse_knn10$.estimate)
)
print(resultados)
```

```
## # A tibble: 3 x 2
##   Modelo            RMSE
```

```
##   <chr>             <dbl>
## 1 Regressão Linear 0.648
## 2 kNN (k = 1)      0.674
## 3 kNN (k = 10)     0.569
```

```r
# ------------------------------------------------------------------------------
# (v) Visualização dos Modelos
# ------------------------------------------------------------------------------

# Gráfico com os dados reais e as previsões dos modelos
ggplot() +
  geom_point(data = train, aes(x, y), color = "blue", alpha = 0.7) + # Dados de treino
  geom_point(data = test, aes(x, y), color = "gray", alpha = 0.5) + # Dados de teste
  geom_line(data = test.pred.lin, aes(x, .pred), color = "red", size = 1, linetype = "dashed") + # Reg.
  geom_line(data = test.pred.knn1, aes(x, .pred), color = "green", size = 1) + # kNN (k=1)
  geom_line(data = test.pred.knn10, aes(x, .pred), color = "purple", size = 1) + # kNN (k=10)
  labs(title = "Comparação dos Modelos",
       x = "x",
       y = "y",
       subtitle = "Regressão Linear (vermelho), kNN k=1 (verde), kNN k=10 (roxo)") +
  theme_minimal()
```



Comentários sobre Viés-Variância (Bias-Variance tradeoff)

A regressão linear apresenta um alto viés, pois assume uma relação linear entre as variáveis, o que pode não representar bem a complexidade dos dados. No entanto, seu erro de variância é baixo, tornando-a menos sensível a mudanças nos dados de treino. O modelo kNN com k=1 tem um viés extremamente baixo, pois ajusta-se completamente aos dados de treino, mas apresenta alta variância, tornando-o suscetível a overfitting. O modelo kNN com k=10 reduz a variância ao suavizar as previsões, mas apresenta um viés maior em comparação ao kNN com k=1. Em resumo, há um compromisso entre viés e variância: modelos mais complexos (como kNN com k=1) podem ter baixa generalização devido à alta variância, enquanto modelos mais simples (como regressão linear) podem ter dificuldade em capturar padrões complexos devido ao alto viés.

```
library(tidymodels)

## Cria o modelo knn
knn.model <- nearest_neighbor(neighbors = 10,
                              weight_func = "rectangular",
                              dist_power = 2) %>%
             set_engine("kknn") %>%
             set_mode("regression")

## Faz o treinamento
knn.fit <- knn.model %>%
    fit( y ~ x, data = train)
```

A expressão `y ~ x` passada como primeiro argumento da função `fit` é conhecida como "fórmula" na linguagem `R`. Ela representa que queremos predizer a coluna y da tabela `train` a partir da coluna x.

Para fazer a predição, usaremos a função `predict` sobre a `tibble` test e depois mediremos a raiz do erro médio quadrático (RMSE) usando a biblioteca `yardstick` incluída no `tidymodels`.

```
## Faz a predição
test.pred <- knn.fit %>%
    predict( new_data = test) %>%
    bind_cols( test ) ## Adiciona as predicoes como coluna
                      ##  da tabela test

## Calcula o erro
rmse(test.pred, y, .pred)
```

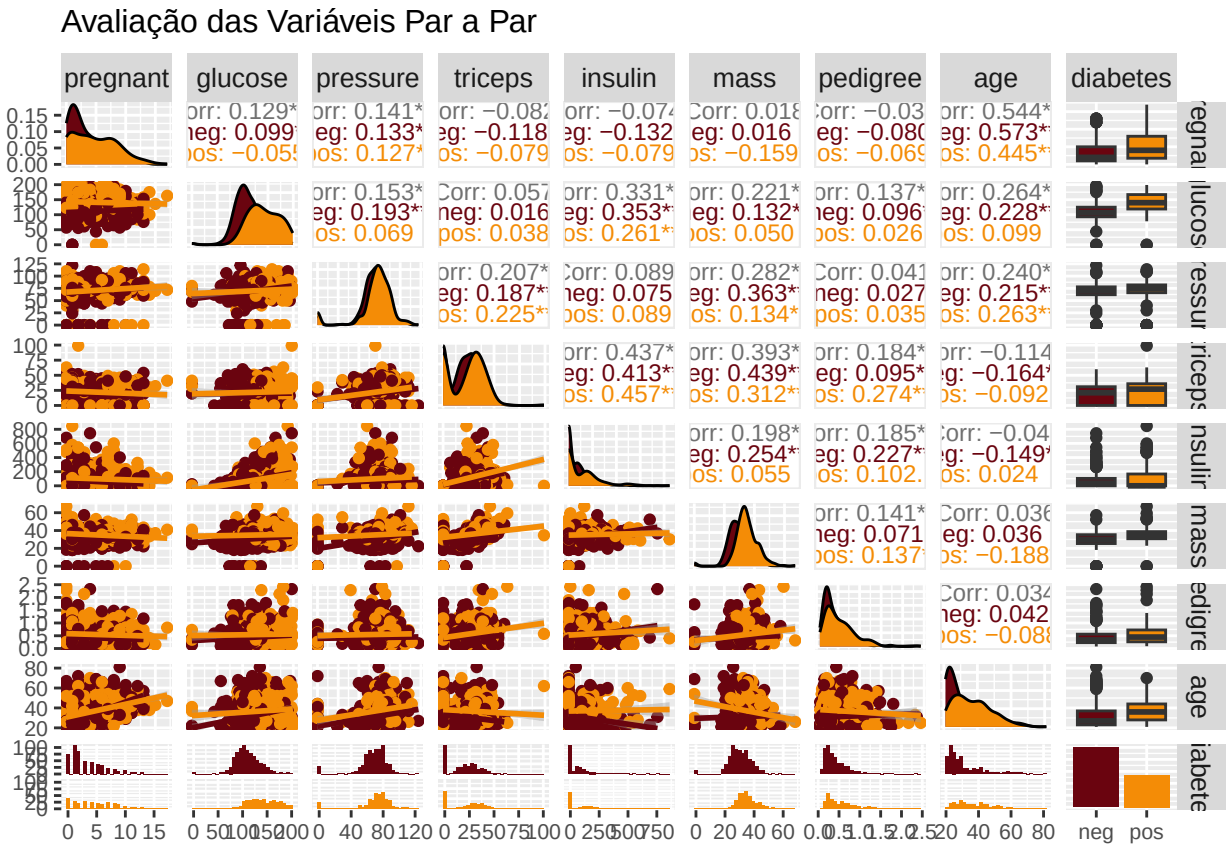
Para rodar a regressão linear, teríamos o seguinte:

```
library(tidymodels)

## Cria o modelo linear
lin.model <- linear_reg() %>%
             set_engine("lm") %>%
             set_mode("regression")

## Faz o treinamento
lin.fit <- lin.model %>%
    fit( y ~ x, data = train)
```

A predição seria similar ao que foi feito acima para o kNN.

Solução Exercicio 02

```r
## Exercicio 2

# Importar o banco de dados
data("PimaIndiansDiabetes")
pima_indians <- as_tibble(PimaIndiansDiabetes)

# Gerar o gráfico
g = ggpairs(
  data = pima_indians,
  mapping = aes(colour = diabetes),
  title = "Avaliação das Variáveis Par a Par",
  upper = list(
    continuous = wrap("cor", size = 3),
    combo = "box_no_facet",
    discrete = "count",
    na = "na"
  ),
  lower = list(
    continuous = "smooth",
    combo = "facethist",
    discrete = "facetbar",
    na = "na"
  ),
  diag = list(
    continuous = "densityDiag",
    discrete = "barDiag",
    na = "naDiag"
  )
) +
  scale_fill_manual(values = c("#6a040f", "#f48c06")) +  # Definindo as cores de preenchimento
  scale_color_manual(values = c("#6a040f", "#f48c06")) +  # Definindo as cores das linhas/contornos
  theme(
    # Ajustando os tamanhos das fontes
    axis.text = element_text(size = 8),  # Tamanho do texto nos eixos
    axis.title = element_text(size = 10),  # Tamanho do título dos eixos
    strip.text = element_text(size = 10),  # Tamanho do texto nas faixas (strip)
    plot.title = element_text(size = 12),  # Tamanho do título
    legend.text = element_text(size = 8),  # Tamanho do texto da legenda
    legend.title = element_text(size = 10)  # Tamanho do título da legenda
  )

# Imprime o gráfico
print(g)
```




```
# Salvar o gráfico ajustado como um arquivo PDF com mais espaço
ggsave("grafico_de_pares.pdf", plot = g, width = 14, height = 14, dpi = 300)
```

É possível visualizar melhor esse gráfico no arquivo grafico_de_pares.pdf

Descrição do gráfico

O gráfico de pares exibe uma matriz de gráficos que mostra todas as combinações possíveis entre as variáveis numéricas de um conjunto de dados. Na diagonal principal, são exibidas as distribuições dos valores de cada variável, geralmente com histogramas ou gráficos de densidade. Fora da diagonal, na parte inferior, os gráficos de dispersão ilustram como as variáveis se relacionam entre si, destacando correlações lineares ou padrões não-lineares. Já na parte superior, é possível observar a correlação entre as variáveis, quantizando o nível de associação entre elas. Além disso, esses dados podem ser mapeados de acordo com variáveis categóricas, permitindo identificar diferenças entre os grupos distintos No contexto do banco de dados "PimaIndiansDiabetes", os indivíduos são classificados com base na presença ou ausência de diabetes. A partir dessa categorização, é possível analisar e identificar quais características estão associadas ao diabetes entre esses indivíduos, permitindo avaliar padrões e fatores que podem influenciar o desenvolvimento da doença.

Solução Exercicio 03

```
## Exercicio 3
```

```r
#Importar o banco de dados
data("PimaIndiansDiabetes")
pima_indians <- as_tibble(PimaIndiansDiabetes)

#Dividir os dados de treino e teste

set.seed(0)  #Define a seed para reprodutibilidade
tt.split = initial_split(pima_indians, prop = 0.8)
train = training(tt.split)
test = testing(tt.split)

#Definir os valores utilizados para a predição (todos)
formula = diabetes ~ .

#Variáveis para armazenar o desempenho do modelo
results = tibble(k = integer(), accuracy = numeric())

#testar diferentes valores de k
for (k in seq(1, 20, by = 2)) {  # Variando k de 1 a 20 pulando de 2 em 2
  knn_model = knn(
    train = select(train, -diabetes),
    test = select(test, -diabetes),
    cl = train$diabetes,
    k = k
  )

  #Calcular a acurácia
  accuracy = mean(knn_model == test$diabetes)

  #Salvar os resultados
  results = results %>% add_row(k = k, accuracy = accuracy)
}

#Mostrar os resultados
print(results)
```

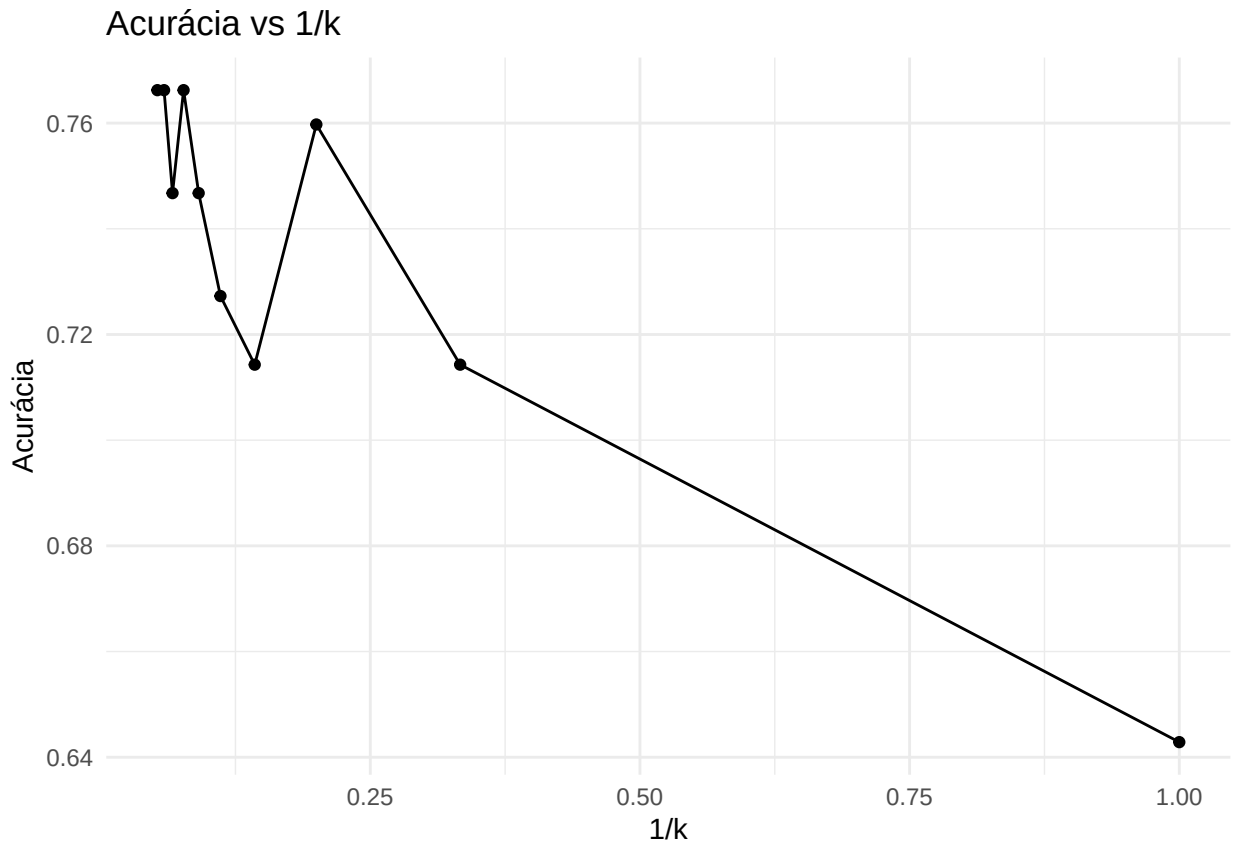
```
## # A tibble: 10 x 2
##        k accuracy
##    <dbl>    <dbl>
##  1     1    0.643
##  2     3    0.714
##  3     5    0.760
##  4     7    0.714
##  5     9    0.727
##  6    11    0.747
##  7    13    0.766
##  8    15    0.747
##  9    17    0.766
## 10    19    0.766
```

```r
#Escolher o melhor k
best_k = results %>% filter(accuracy == max(accuracy))
```

```r
#Escolher o melhor k
best_k = results %>% slice_max(accuracy, n = 1)  # Retorna apenas o melhor k

#Exibir o melhor k
print(paste("O melhor k é:", best_k$k))
```

```
## [1] "O melhor k é: 13" "O melhor k é: 17" "O melhor k é: 19"
```

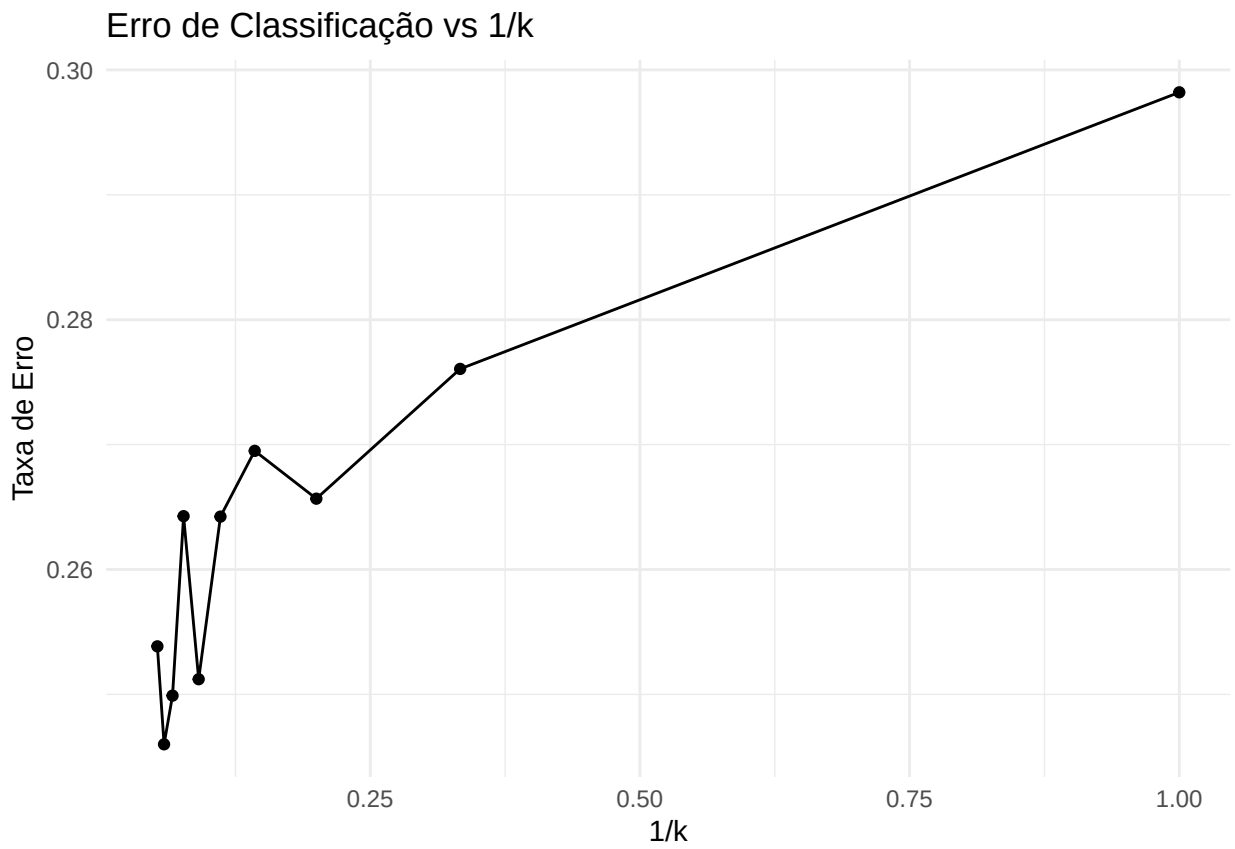
```r
#Plotando o gráfico da acurácia vs 1/k
ggplot(results, aes(x = 1/k, y = accuracy)) +
  geom_line() +
  geom_point() +
  labs(x = "1/k", y = "Acurácia", title = "Acurácia vs 1/k") +
  theme_minimal()
```



Exercício 04

```r
##Exercicio 4

#Carregar o banco de dados
data("PimaIndiansDiabetes")
pima_indians <- as_tibble(PimaIndiansDiabetes)
```

```r
#Dividir os dados em conjuntos de treino e teste com validação cruzada
cv.split = vfold_cv(pima_indians, v = 10)  # 10 folds para validação cruzada

#Definir o intervalo de valores de k
k_values = seq(1, 20, by = 2)  # Variando k de 1 a 20 (de 2 em 2)
errors = data.frame(k = k_values, error = numeric(length(k_values)))  # DataFrame para armazenar os res

#Testar diferentes valores de k
for (i in 1:length(k_values)) {
  knn.model = nearest_neighbor(neighbors = k_values[i],  # Número de vizinhos
                               weight_func = "rectangular",   # Função de ponderação retangular
                               dist_power = 2) %>%            # Distância Euclidiana
    set_engine("kknn") %>% set_mode("classification")  # Definindo o modelo e o modo de operação

  #Treinamento do modelo com validação cruzada
  knn.fits = fit_resamples(knn.model,
                           diabetes ~ .,
                           resamples = cv.split)

  #Coletando as métricas e calculando o erro de classificação
  metrics = collect_metrics(knn.fits)
  accuracy = metrics %>% filter(.metric == "accuracy") %>% pull(mean)  # Acurácia média
  error_rate = 1 - accuracy  # Calculando a taxa de erro
  errors[i, "error"] = error_rate  # Armazenando o erro de classificação
}

#Plotando o gráfico de erro contra 1/k
ggplot(errors, aes(x = 1/k, y = error)) +
  geom_line() +  # Adiciona uma linha para conectar os pontos
  geom_point() + # Adiciona pontos no gráfico
  labs(x = "1/k", y = "Taxa de Erro", title = "Erro de Classificação vs 1/k") +
  theme_minimal()
```



Solução Exercícios 5 e 6

```r
set.seed(42) # Reprodutibilidade

# Nova função f(x)
f <- function(x) { 2*x + 1 }

# Gerando dados para treinamento (50 pontos)
x_train <- runif(50, min = -1, max = 1)
y_train <- f(x_train) + rnorm(50, mean = 0, sd = 0.5)
train <- tibble(x = x_train, y = y_train)

# Gerando dados para teste (100 pontos)
x_test <- runif(100, min = -1, max = 1)
y_test <- f(x_test) + rnorm(100, mean = 0, sd = 0.5)
test <- tibble(x = x_test, y = y_test)

# ------------------------------------------------------------------------------
# (i) Criando o modelo de Regressção Linear
# ------------------------------------------------------------------------------
lin.model <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")
```

```r
lin.fit <- lin.model %>% fit(y ~ x, data = train)
test.pred.lin <- lin.fit %>% predict(new_data = test) %>% bind_cols(test)
rmse_lin <- rmse(test.pred.lin, truth = y, estimate = .pred)

# ------------------------------------------------------------------------------
# (ii) Criando modelo kNN com k = 1
# ------------------------------------------------------------------------------
knn1.model <- nearest_neighbor(neighbors = 1, weight_func = "rectangular", dist_power = 2) %>%
  set_engine("kknn") %>%
  set_mode("regression")

knn1.fit <- knn1.model %>% fit(y ~ x, data = train)
test.pred.knn1 <- knn1.fit %>% predict(new_data = test) %>% bind_cols(test)
rmse_knn1 <- rmse(test.pred.knn1, truth = y, estimate = .pred)

# ------------------------------------------------------------------------------
# (iii) Criando modelo kNN com k = 10
# ------------------------------------------------------------------------------
knn10.model <- nearest_neighbor(neighbors = 10, weight_func = "rectangular", dist_power = 2) %>%
  set_engine("kknn") %>%
  set_mode("regression")

knn10.fit <- knn10.model %>% fit(y ~ x, data = train)
test.pred.knn10 <- knn10.fit %>% predict(new_data = test) %>% bind_cols(test)
rmse_knn10 <- rmse(test.pred.knn10, truth = y, estimate = .pred)

# ------------------------------------------------------------------------------
# (iv) Comparação dos resultados
# ------------------------------------------------------------------------------
resultados <- tibble(
  Modelo = c("Regressão Linear", "kNN (k = 1)", "kNN (k = 10)"),
  RMSE = c(rmse_lin$.estimate, rmse_knn1$.estimate, rmse_knn10$.estimate)
)
print(resultados)
```

```
## # A tibble: 3 x 2
##   Modelo            RMSE
##   <chr>            <dbl>
## 1 Regressão Linear 0.452
## 2 kNN (k = 1)      0.710
## 3 kNN (k = 10)     0.492
```

Comentário sobre os resultados

Como a função agora é linear (2x + 1), a regressão linear se ajusta perfeitamente ao modelo subjacente, resultando em um erro menor do que no Exercício 01.O kNN com k=1 continua sendo altamente sensível aos dados de treino, apresentando alta variância. O kNN com k=10 reduz essa variância, mas não supera a regressão linear, pois o modelo real é de fato linear.

```r
knitr::opts_chunk$set(echo = TRUE)
```

Definição do Erro Quadrático Esperado

Dado um modelo que tenta estimar uma função $f(x)$ com ruído, o erro esperado da predição $\hat{f}(x)$ pode ser expresso como:

$$E[(y - \hat{f}(x))^2]$$

onde: - $y = f(x) + \epsilon$ (sendo ϵ o ruído com média zero). - $\hat{f}(x)$ é a predição do modelo.

Expandindo o erro:

$$E[(f(x) + \epsilon - \hat{f}(x))^2]$$

Desenvolvendo a equação:

$$E[(f(x) - \hat{f}(x))^2] + 2E[\epsilon(f(x) - \hat{f}(x))] + E[\epsilon^2]$$

Como ϵ tem esperança zero, o termo intermediário desaparece, restando:

$$E[(f(x) - \hat{f}(x))^2] + \sigma_\epsilon^2$$

Decomposição Viés-Variância

Podemos decompor o primeiro termo como:

$$E[(f(x) - E[\hat{f}(x)] + E[\hat{f}(x)] - \hat{f}(x))^2]$$

Expandindo:

$$(f(x) - E[\hat{f}(x)])^2 + E[(\hat{f}(x) - E[\hat{f}(x)])^2]$$

Chamamos esses termos de **viés ao quadrado** e **variância**, respectivamente:

- **Viés ao quadrado**:
$$\text{Bias}^2 = (f(x) - E[\hat{f}(x)])^2$$
- **Variância**:
$$\text{Var} = E[(\hat{f}(x) - E[\hat{f}(x)])^2]$$

Assim, a decomposição final é:

$$E[(y - \hat{f}(x))^2] = \text{Bias}^2 + \text{Var} + \sigma_\epsilon^2$$

Conclusão

Este resultado mostra que o erro total de um modelo é composto por três partes: 1. **Viés ao quadrado**: erro sistemático do modelo. 2. **Variância**: sensibilidade do modelo às variações nos dados. 3. **Ruído irredutível**: variação nos dados que nenhum modelo pode eliminar.

O objetivo em aprendizado de máquina é encontrar um equilíbrio entre viés e variância para minimizar o erro total.