Solução Lista 06

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

01 April, 2025

Resolução Exercício 1

```
# Carregar pacotes necessários
library(mlbench)
library(tidymodels)

# Carregar o banco de dados
data("BreastCancer")

# Transformar os dados em tibble e remover entradas com NAs
bc_df <- as_tibble(BreastCancer) %>% na.omit()

# Criar receita para pré-processamento dos dados
rec <- recipe(Class ~ ., data = bc_df) %>%
  step_rm(Id) %>%  # Remover a coluna de ID, que não é relevante
  step_ordinalscore(Cl.thickness, Cell.size, Cell.shape, Marg.adhesion, Epith.c.size) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>% # Converter variáveis categóricas em dummies
  step_rm(Mitoses_X6) %>%  # Remover variável sem representantes
  step_center(all_predictors()) %>% # Centralizar os dados
  step_scale(all_predictors()) # Normalizar os dados

# Definir o modelo de regressão logística sem regularização
lr.model <- logistic_reg(penalty = 0, mixture = NULL) %>%
  set_engine("glmnet")

# Dividir os dados em treino (70%) e teste (30%)
set.seed(123)
split <- initial_split(bc_df, prop = 0.70)
train <- training(split)
test <- testing(split)

# Preparar os dados com base no conjunto de treino
rec.prep <- rec %>% prep(train)
train.prep <- juice(rec.prep) # Conjunto de treinamento transformado
test.prep <- bake(rec.prep, test) # Conjunto de teste transformado
```

```r
# Treinar o modelo
lr.fit <- lr.model %>% fit(Class ~ ., data = train.prep)

# Fazer previsões no conjunto de teste
test.pred <- test.prep %>%
  bind_cols(predict(lr.fit, new_data = test.prep))

# Gerar matriz de confusão
conf_matrix <- conf_mat(test.pred, truth = Class, estimate = .pred_class)

# Exibir a matriz de confusão
print(conf_matrix$table)
```

```
##            Truth
## Prediction  benign malignant
##   benign       136         5
##   malignant      3        61
```

```r
# Cálculo das métricas de avaliação
metrics <- tibble(
  recall = recall(test.pred, truth = Class, estimate = .pred_class),
  precision = precision(test.pred, truth = Class, estimate = .pred_class),
  sensitivity = sens(test.pred, truth = Class, estimate = .pred_class),
  specificity = spec(test.pred, truth = Class, estimate = .pred_class)
)

# Exibir as métricas
print(metrics)
```

```
## # A tibble: 1 x 4
##   recall$.metric precision$.metric sensitivity$.metric specificity$.metric
##   <chr>          <chr>             <chr>               <chr>
## 1 recall         precision         sens                spec
## # i 8 more variables: recall$.estimator <chr>, $.estimate <dbl>,
## #   precision$.estimator <chr>, $.estimate <dbl>, sensitivity$.estimator <chr>,
## #   $.estimate <dbl>, specificity$.estimator <chr>, $.estimate <dbl>
```

Solução Exercício 2

```r
library(tidymodels)
library(kernlab)

# Criando o modelo de regressão logística com ajuste de hiperparâmetros
lr.model <- logistic_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet")

# Criando o modelo SVM com kernel RBF
svm.model <- svm_rbf(cost = tune(), rbf_sigma = tune()) %>%
  set_engine("kernlab") %>%
  set_mode("classification")
```

```r
# Criando a malha de busca para os hiperparâmetros
svm_grid <- grid_max_entropy(cost(), rbf_sigma(), size = 10)
lr_grid <- grid_max_entropy(penalty(), mixture(), size = 10)

# Criando validação cruzada
folds <- vfold_cv(bc_df, v = 10)

# Definição das métricas de avaliação
metrs <- metric_set(roc_auc, accuracy)

# Ajuste de hiperparâmetros para Regressão Logística
tune_lr <- tune_grid(
  lr.model,
  rec,
  resamples = folds,
  grid = lr_grid,
  metrics = metrs
)

# Ajuste de hiperparâmetros para SVM
tune_svm <- tune_grid(
  svm.model,
  rec,
  resamples = folds,
  grid = svm_grid,
  metrics = metrs
)

# Exibindo os resultados das métricas coletadas
tune_lr %>% collect_metrics()
```

```
## # A tibble: 20 x 8
##     penalty mixture .metric  .estimator  mean     n std_err .config
##       <dbl>   <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1 1.65e- 5  0.0157 accuracy binary     0.962    10 0.00791 Preprocessor1_Model~
##  2 1.65e- 5  0.0157 roc_auc  binary     0.991    10 0.00278 Preprocessor1_Model~
##  3 1.35e-10  0.0368 accuracy binary     0.963    10 0.00761 Preprocessor1_Model~
##  4 1.35e-10  0.0368 roc_auc  binary     0.990    10 0.00338 Preprocessor1_Model~
##  5 8.22e- 1  0.0614 accuracy binary     0.918    10 0.00732 Preprocessor1_Model~
##  6 8.22e- 1  0.0614 roc_auc  binary     0.992    10 0.00240 Preprocessor1_Model~
##  7 6.83e- 9  0.318  accuracy binary     0.959    10 0.00964 Preprocessor1_Model~
##  8 6.83e- 9  0.318  roc_auc  binary     0.989    10 0.00387 Preprocessor1_Model~
##  9 4.84e- 3  0.444  accuracy binary     0.966    10 0.00619 Preprocessor1_Model~
## 10 4.84e- 3  0.444  roc_auc  binary     0.991    10 0.00243 Preprocessor1_Model~
## 11 7.74e- 7  0.588  accuracy binary     0.955    10 0.0105  Preprocessor1_Model~
## 12 7.74e- 7  0.588  roc_auc  binary     0.988    10 0.00389 Preprocessor1_Model~
## 13 4.14e-10  0.615  accuracy binary     0.955    10 0.0105  Preprocessor1_Model~
## 14 4.14e-10  0.615  roc_auc  binary     0.988    10 0.00387 Preprocessor1_Model~
## 15 2.84e- 1  0.797  accuracy binary     0.830    10 0.0143  Preprocessor1_Model~
## 16 2.84e- 1  0.797  roc_auc  binary     0.987    10 0.00351 Preprocessor1_Model~
## 17 2.99e- 8  0.952  accuracy binary     0.955    10 0.0105  Preprocessor1_Model~
## 18 2.99e- 8  0.952  roc_auc  binary     0.988    10 0.00374 Preprocessor1_Model~
## 19 1.29e- 4  0.958  accuracy binary     0.956    10 0.00991 Preprocessor1_Model~
```

```
## 20 1.29e- 4  0.958  roc_auc  binary     0.988    10 0.00386 Preprocessor1_Model~
```

```
tune_svm %>% collect_metrics()
```

```
## # A tibble: 20 x 8
##        cost rbf_sigma .metric  .estimator  mean     n std_err .config
##       <dbl>     <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1  0.0576   1.66e- 6 accuracy binary     0.650    10 0.0147  Preprocessor1_Mod~
##  2  0.0576   1.66e- 6 roc_auc  binary     0.994    10 0.00144 Preprocessor1_Mod~
##  3  8.89     2.30e-10 accuracy binary     0.650    10 0.0147  Preprocessor1_Mod~
##  4  8.89     2.30e-10 roc_auc  binary     0.994    10 0.00142 Preprocessor1_Mod~
##  5  0.00167  4.08e- 1 accuracy binary     0.650    10 0.0147  Preprocessor1_Mod~
##  6  0.00167  4.08e- 1 roc_auc  binary     0.983    10 0.00465 Preprocessor1_Mod~
##  7  0.101    3.56e-10 accuracy binary     0.650    10 0.0147  Preprocessor1_Mod~
##  8  0.101    3.56e-10 roc_auc  binary     0.994    10 0.00131 Preprocessor1_Mod~
##  9  0.598    1.11e- 4 accuracy binary     0.650    10 0.0147  Preprocessor1_Mod~
## 10  0.598    1.11e- 4 roc_auc  binary     0.994    10 0.00144 Preprocessor1_Mod~
## 11  0.00540  1.22e- 4 accuracy binary     0.650    10 0.0147  Preprocessor1_Mod~
## 12  0.00540  1.22e- 4 roc_auc  binary     0.994    10 0.00140 Preprocessor1_Mod~
## 13  0.0513   2.63e- 1 accuracy binary     0.851    10 0.0131  Preprocessor1_Mod~
## 14  0.0513   2.63e- 1 roc_auc  binary     0.982    10 0.00484 Preprocessor1_Mod~
## 15  5.91     3.03e- 1 accuracy binary     0.947    10 0.00787 Preprocessor1_Mod~
## 16  5.91     3.03e- 1 roc_auc  binary     0.983    10 0.00465 Preprocessor1_Mod~
## 17 25.6      8.06e- 6 accuracy binary     0.941    10 0.00435 Preprocessor1_Mod~
## 18 25.6      8.06e- 6 roc_auc  binary     0.993    10 0.00149 Preprocessor1_Mod~
## 19  0.00158  6.82e- 9 accuracy binary     0.650    10 0.0147  Preprocessor1_Mod~
## 20  0.00158  6.82e- 9 roc_auc  binary     0.994    10 0.00139 Preprocessor1_Mod~
```

```
# Comparação dos melhores modelos
show_best(tune_lr, metric = "roc_auc", n = 3)
```

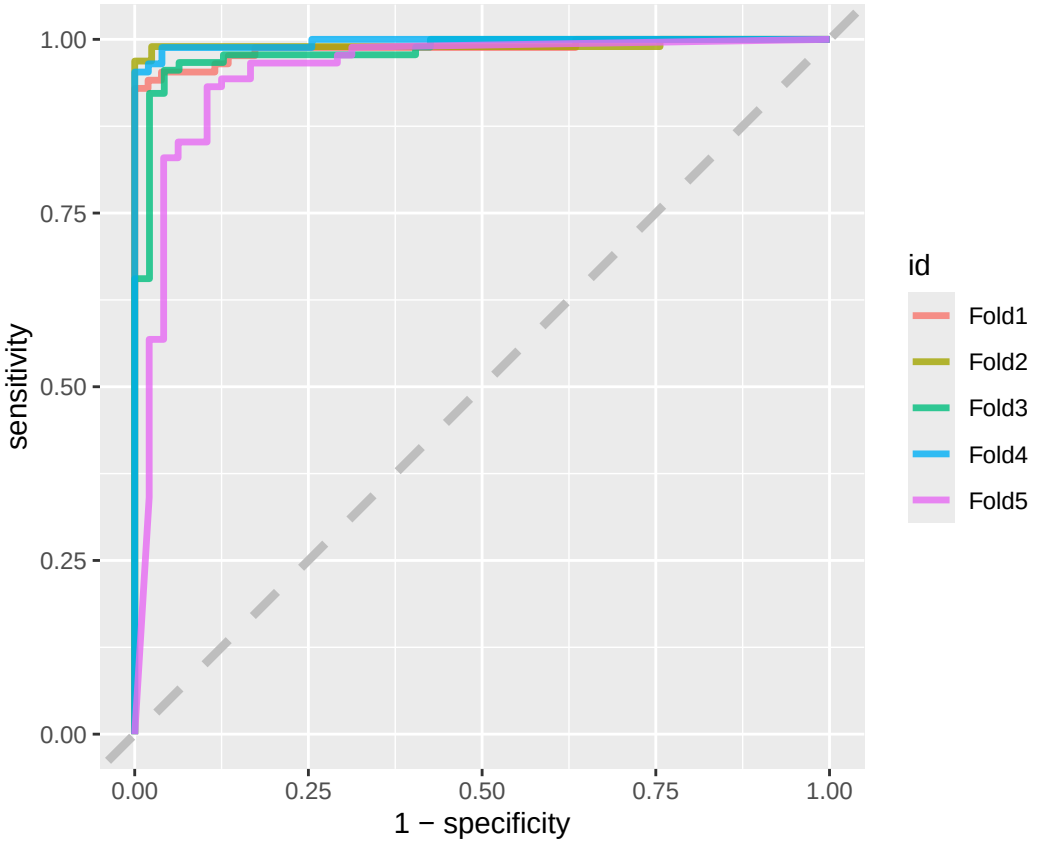
```
## # A tibble: 3 x 8
##     penalty mixture .metric .estimator  mean     n std_err .config
##       <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.822      0.0614 roc_auc binary     0.992    10 0.00240 Preprocessor1_Model03
## 2 0.00484    0.444  roc_auc binary     0.991    10 0.00243 Preprocessor1_Model05
## 3 0.0000165  0.0157 roc_auc binary     0.991    10 0.00278 Preprocessor1_Model01
```

```
show_best(tune_svm, metric = "roc_auc", n = 3)
```

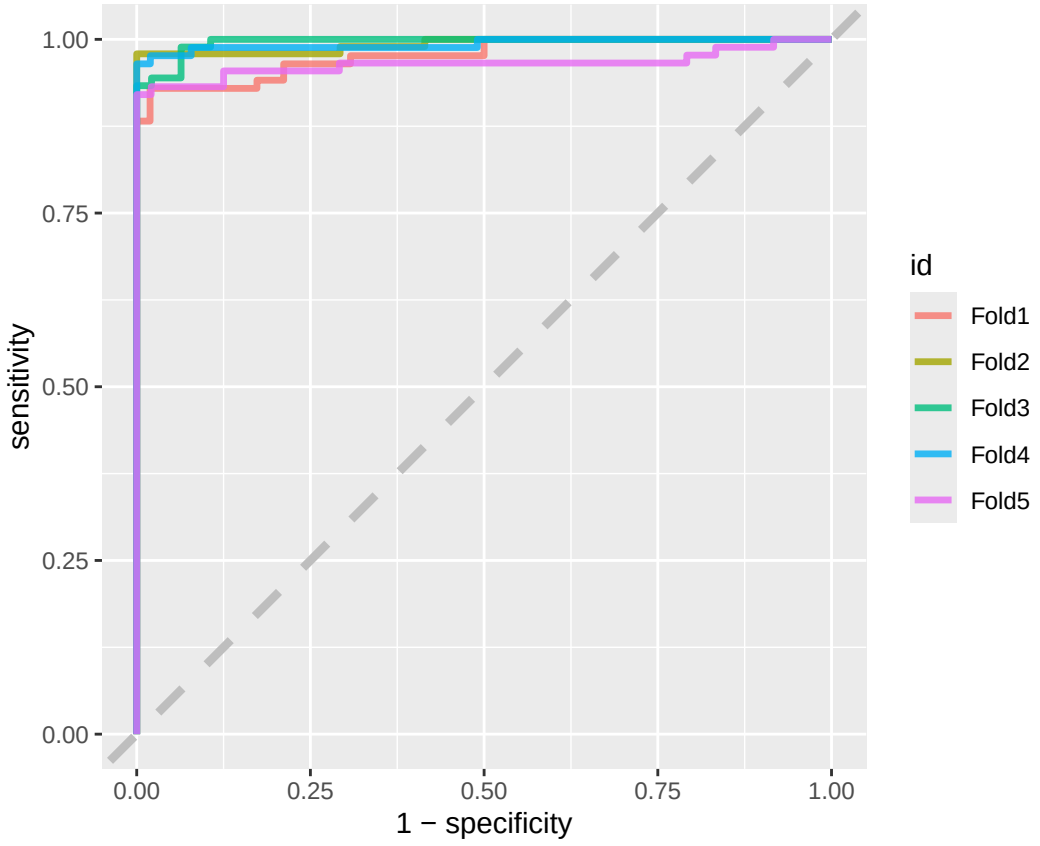
```
## # A tibble: 3 x 8
##      cost rbf_sigma .metric .estimator  mean     n std_err .config
##     <dbl>     <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.101    3.56e-10 roc_auc binary     0.994    10 0.00131 Preprocessor1_Model04
## 2 0.00540  1.22e- 4 roc_auc binary     0.994    10 0.00140 Preprocessor1_Model06
## 3 8.89     2.30e-10 roc_auc binary     0.994    10 0.00142 Preprocessor1_Model02
```

Solução Exercício 3

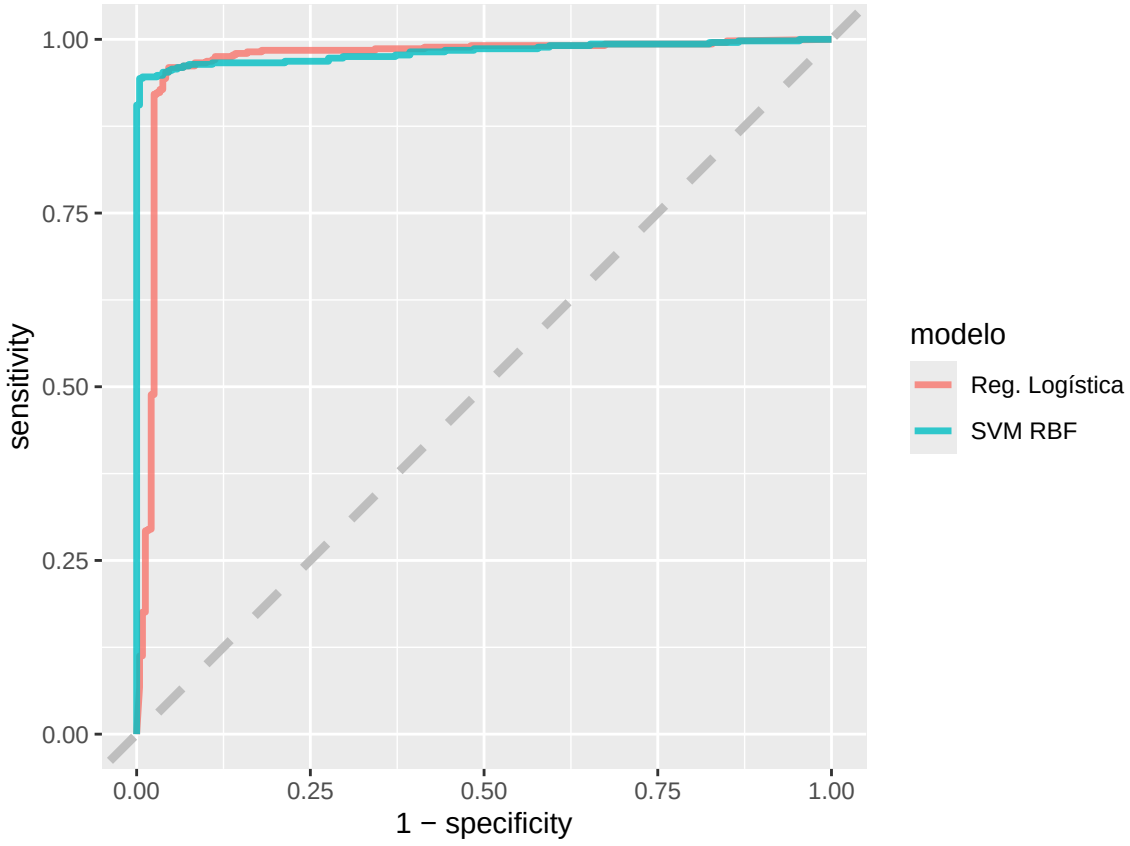
```r
library(tidymodels)

# Definição dos modelos com os melhores parâmetros
lr_model <- logistic_reg(penalty = 0, mixture = NULL) %>%
  set_engine("glmnet") %>%
  set_mode("classification")

svm_model <- svm_rbf(cost = 1, rbf_sigma = 0.1) %>%  # Ajuste os melhores parâmetros
  set_engine("kernlab") %>%
  set_mode("classification")

# Criando validação cruzada
folds <- vfold_cv(bc_df, v = 5)

# Ajuste dos modelos
fit_res_lr <- fit_resamples(
  lr_model, rec, resamples = folds,
  control = control_resamples(save_pred = TRUE)
)

fit_res_svm <- fit_resamples(
  svm_model, rec, resamples = folds,
  control = control_resamples(save_pred = TRUE)
)

# Extraindo previsões
pred_lr <- fit_res_lr %>% unnest(.predictions)
pred_svm <- fit_res_svm %>% unnest(.predictions)

# Criando curvas ROC por fold
if (nrow(pred_lr) > 0) {
  pred_lr %>%
    group_by(id) %>%
    roc_curve(Class, .pred_benign) %>%
    ggplot(aes(x = 1 - specificity, y = sensitivity, color = id)) +
    geom_path(size = 1.2, alpha = 0.8) +
    geom_abline(lty = 2, color = "gray", size = 1.5) +
    coord_equal()
}
```




```
if (nrow(pred_svm) > 0) {
  pred_svm %>%
    group_by(id) %>%
    roc_curve(Class, .pred_benign) %>%
    ggplot(aes(x = 1 - specificity, y = sensitivity, color = id)) +
    geom_path(size = 1.2, alpha = 0.8) +
    geom_abline(lty = 2, color = "gray", size = 1.5) +
    coord_equal()
}
```




```r
# Criando gráfico médio dos folds
all_preds <- bind_rows(
  pred_lr %>% mutate(modelo = "Reg. Logística"),
  pred_svm %>% mutate(modelo = "SVM RBF")
)

if (nrow(all_preds) > 0) {
  all_preds %>%
    group_by(modelo) %>%
    roc_curve(Class, .pred_benign) %>%
    ggplot(aes(x = 1 - specificity, y = sensitivity, color = modelo)) +
    geom_path(size = 1.2, alpha = 0.8) +
    geom_abline(lty = 2, color = "gray", size = 1.5) +
    coord_equal()
}
```



Solução Exercício 4

```
# Importar bibliotecas
library(modelr)
library(ggplot2)

# Função para fazer o ajuste do perceptron
perceptron.fit <- function(form,           # uma fórmula para treinar
                           df,             # dataframe para treinar
                           eta = 0.01)     # taxa de aprendizado
  {
    # Obtem a dataframe que precisamos para treinar os dados a partir da formula:
    train_df <- model.frame(form, df)

    # Extrai a coluna dos y e depois colocamos os valores -1 e +1
    classes <- levels(train_df[,1])
    y <- train_df[,1] %>% as.integer()
    y <- (y - 1)*2 - 1  # Converte para -1 e +1

    # Extrai as colunas dos X
    X <- as.matrix(train_df[,-1])

    # Vetor normal do hiperplano e o intercepto
```

```
    w <- rep(0, ncol(X))
    b <- 0

    # Implementar o perceptron:
    e <- 1
    while(e != 0) {
        e <- 0

        # Embaralhando o conjunto de indíces
        index <- sample(1:nrow(X))

        for(i in 1:nrow(X)) {
            # pronto para testar
            j <- index[i]
            x <- as.numeric(X[j,])

            # Calcular a predição atual
            pred <- sum(w * x) + b

            # Se a predição estiver errada, ajustar os pesos
            if(y[j] * pred <= 0) {
                w <- w + eta * y[j] * x
                b <- b + eta * y[j]
                e <- e + 1
            }
        }
    }

    return(list("formula" = form,     # formula usada no treinamento
                "classes" = classes,  # nomes das classes
                "normal" = w,
                "intercepto_y" = b))
}

# Função para fazer a predição
perceptron.predict <- function(percep.fit, # saída da função perceptron.fit
                               new_data)   # dados de teste
  {
    form <- percep.fit$formula
    classes <- percep.fit$classes
    w <- percep.fit$normal
    b <- percep.fit$intercepto_y

    # Extrai as colunas do X
    # -- Aqui precisamos gerar a matriz X de acordo com a fórmula,
    # queremos usar a fórmula mesmo se new_data não contêm a # coluna das classes.
    # Extrai as features dos dados de teste
    class_column <- as.character(form[2])
    if(class_column %in% names(new_data)) {
        test_df <- model.frame(form[-2], new_data %>% select(-class_column))
    } else {
        test_df <- model.frame(form[-2], new_data)
    }
```

```
    X <- as.matrix(test_df)

    # Calcular w * x + b, para os pontos em new_data
    # e retorna a classe de cada ponto.
    # Cria um vetor para retornar

    pred <- factor(vector("character", length = nrow(X)), levels = classes)
    for(i in 1:nrow(X)) {
        x <- as.numeric(X[i,])

        output_linear <- sum(w * x) + b

        # Classifica com base no sinal do output linear
        if(output_linear > 0) {
            pred[i] <- classes[2]  # "outra"
        } else {
            pred[i] <- classes[1]  # "setosa"
        }
    }

    return(tibble(.pred = pred))
}

# Carregando e preparando os dados
data("iris")
df <- as_tibble(iris) %>%
# Fazendo a coluna Species ser charactere e não
# factor para usarmos o if_else abaixo
mutate( Species = as.character(Species) ) %>%
# A linha abaixo cria a coluna Classe que terá versicolor se
# a planta é dessa espécie e outra caso contrário.
mutate( Classe = if_else( Species == "setosa", Species, "outra")) %>% # Removendo a coluna Species, por
select(-Species) %>%
# Vamos fazer a variável Classe ser um factor
mutate( Classe = factor(Classe, levels=c("setosa","outra")) )
# Para ver se está tudo ok e contar o número de entradas por
# classe, podemos rodar o seguinte:
df %>% count(Classe)
```

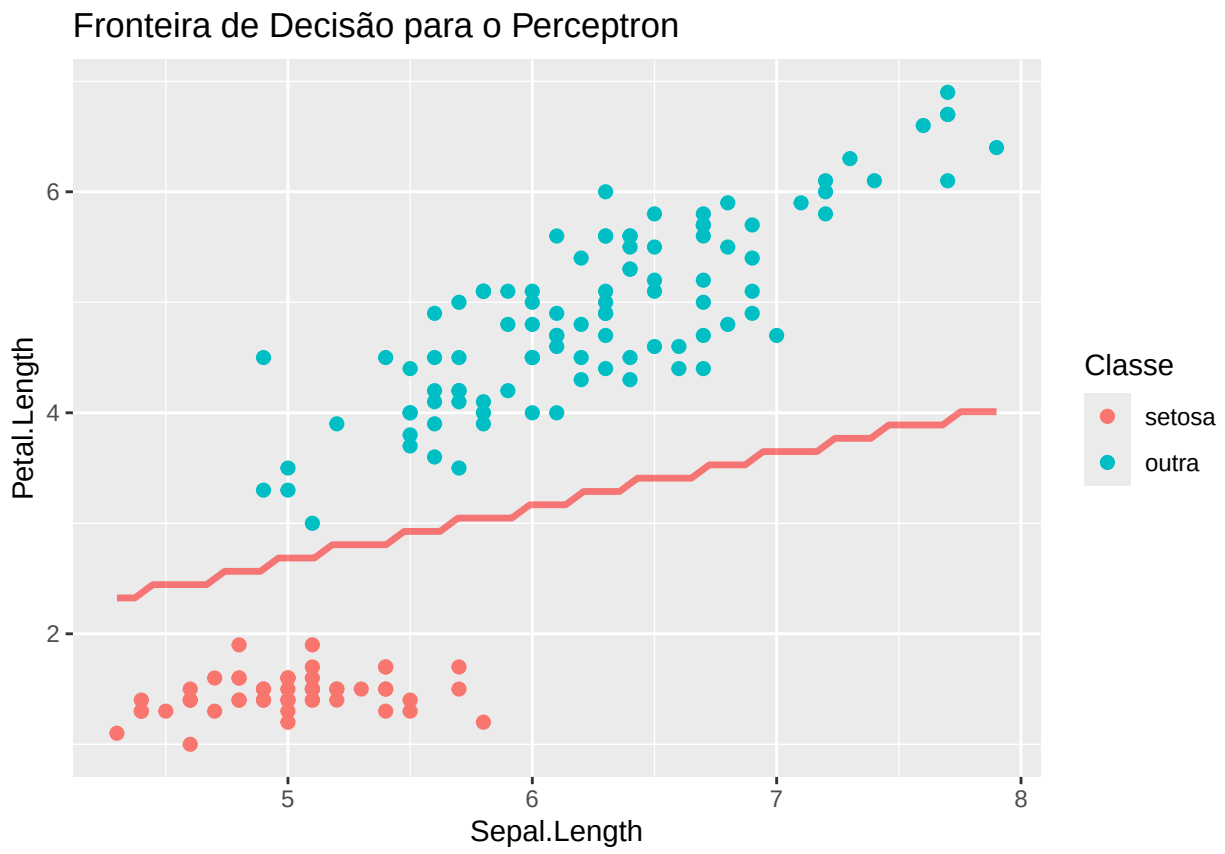
```
## # A tibble: 2 x 2
##   Classe     n
##   <fct>  <int>
## 1 setosa    50
## 2 outra    100
```

```
## Fazendo a validação cruzada com k = 10, repeats = 3 repete a geração de folds 3 vezes, nos dando 30 
splits <- vfold_cv(data = df, v = 10, repeats = 3)

acc_results <- vector("numeric", length = nrow(splits))
for(i in 1:nrow(splits)) {
    s <- splits$splits[[i]]
    train <- analysis(s)
    test <- assessment(s)
```

```
    percep.fit <- perceptron.fit(Classe ~ ., train)
    test_pred <- perceptron.predict(percep.fit, test) %>%
        bind_cols(test) %>%
        accuracy(Classe, .pred)
    acc_results[i] <- test_pred$.estimate
}

cat("Acurácia média = ", mean(acc_results), "\n")
```

```
## Acurácia média =  0.9977778
```

```
# Selecionando apenas duas features para visualização
df_plot <- df %>% select(Sepal.Length, Petal.Length, Classe)

# Gerando grid para plotagem
plot_grid <- expand_grid(
    Sepal.Length = seq_range(df_plot$Sepal.Length, 50),
    Petal.Length = seq_range(df_plot$Petal.Length, 50)
)

# Adicionamos a predição das classes
percep.fit <- perceptron.fit(Classe ~ ., df_plot)
plot_grid_pred <- plot_grid %>%
    mutate(pred = perceptron.predict(percep.fit, plot_grid)$.pred)

# Fazemos o gráfico
ggplot(plot_grid_pred, aes(Sepal.Length, Petal.Length)) +
    geom_contour(aes(z = as.integer(pred)),
                 alpha = 0.5, show.legend = FALSE, breaks = c(1L, 2L),
                 size = 1.2, color = "red") +
    geom_point(data = df_plot, aes(color = Classe), size = 2) +
    labs(title = "Fronteira de Decisão para o Perceptron")
```



Solução Exercício 5

```
library(tidyverse)
library(tidymodels)
library(kernlab)
library(ggplot2)

# Criar problema binário: versicolor vs outras
df_svm = as_tibble(iris) %>%
  mutate(Species = as.character(Species)) %>%
  mutate(Classe = if_else(Species == "versicolor", Species, "outra")) %>%
  select(Sepal.Length, Petal.Length, Classe) %>%  # Seleciona apenas 2 features + target
  mutate(Classe = factor(Classe, levels = c("versicolor", "outra")))

# Verificar distribuição das classes
df_svm %>% count(Classe)
```

```
## # A tibble: 2 x 2
##   Classe         n
##   <fct>      <int>
## 1 versicolor    50
## 2 outra        100
```

```r
## 2. Definir Receita e Modelo SVM
rec_svm = recipe(Classe ~ Sepal.Length + Petal.Length, data = df_svm) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())

svm_model = svm_rbf(
  cost = tune(),         # Parâmetro de regularização
  rbf_sigma = tune()     # Parâmetro do kernel RBF
) %>%
  set_engine("kernlab") %>%
  set_mode("classification") %>%
  set_args(formula = Classe ~ Sepal.Length + Petal.Length)  # Fórmula explícita

## 3. Tunagem de Parâmetros

grid = grid_max_entropy(
  cost(),           # Valores para o parâmetro de custo
  rbf_sigma(),      # Valores para o parâmetro sigma
  size = 10         # Número de combinações
)

folds = vfold_cv(data = df_svm, v = 5)  # 5-fold CV

# Métricas de avaliação
metrics = metric_set(roc_auc, accuracy)

# Tunagem
svm_tune = tune_grid(
  svm_model,
  rec_svm,
  resamples = folds,
  grid = grid,
  metrics = metrics
)

# Melhores parâmetros
best_params = select_best(svm_tune, metric = "roc_auc")
final_svm = finalize_model(svm_model, best_params)

## 4. Treinar Modelo Final
final_fit = final_svm %>%
  fit(Classe ~ Sepal.Length + Petal.Length, data = df_svm)

## 5. Visualização da Fronteira de Decisão
# Criar grid para plotagem
plot_grid_svm = expand_grid(
  Sepal.Length = seq_range(df_svm$Sepal.Length, 50),
  Petal.Length = seq_range(df_svm$Petal.Length, 50)
)

# Fazer predições no grid
plot_grid_svm_pred = plot_grid_svm %>%
  mutate(pred = predict(final_fit, plot_grid_svm)$.pred_class)
```

```r
# Plotar fronteira de decisão
ggplot(plot_grid_svm_pred, aes(Sepal.Length, Petal.Length)) +
  geom_contour(aes(z = as.integer(pred)),
               alpha = 0.5, show.legend = FALSE,
               breaks = c(1L, 2L), size = 1.2, color = "blue") +
  geom_point(data = df_svm, aes(color = Classe), size = 2) +
  labs(title = "Fronteira de Decisão - SVM com Kernel RBF",
       subtitle = "Classificação: versicolor vs outras",
       x = "Sepal Length", y = "Petal Length") +
  theme_minimal()
```