PPCA0026 - Tarefa de Casa: Validação Cruzada e Bootstrap

Code

Análise de SVM e k-NN no dataset Iris

AUTHOR
Denard Costa Soares

PUBLISHED
June 20, 2025

Prazo de Entrega: 2025-06-29 23:59

Introdução

Nesta tarefa, você aplicará os conceitos de Validação Cruzada (CV) e Bootstrap para selecionar e avaliar modelos de classificação. O objetivo é ir além da simples aplicação de funções prontas, focando na implementação dos mecanismos subjacentes para garantir um entendimento profundo dos métodos.

Objetivos de Aprendizagem:

1. Observar a instabilidade da abordagem de validação com uma única divisão (treino/validação).
2. Implementar um loop de 5-fold Cross-Validation estratificado para selecionar hiperparâmetros para modelos SVM e k-NN.
3. Visualizar os resultados da busca por hiperparâmetros usando heatmaps e gráficos de slice.
4. Utilizar o Bootstrap em um conjunto de teste para quantificar a incerteza na estimativa do erro dos modelos finais.
5. Comparar modelos de forma robusta, analisando a distribuição dos seus rankings de performance sob reamostragem.

Instruções Gerais:

- Este arquivo serve como template. Você deve preencher as seções marcadas com seu código, saídas e respostas.
- Para esta tarefa, usaremos o pacote `e1071` para SVM, `class` para k-NN, e o `tidyverse` para manipulação de dados e gráficos.
- **Entrega:** Envie dois arquivos: este `.qmd` completo e o arquivo `.html` auto-contido resultante.

---

0. O Problema da Variabilidade de uma Única Divisão

Tarefa 0

▼ Code

```r
# Carregar pacotes
library(tidyverse)
library(class) # Para knn()
library(e1071) # Para svm()
library(caret) # Para createDataPartition

# Filtrar o dataset iris para as duas espécies
iris_duas_classes <- iris %>%
  filter(Species %in% c("versicolor", "virginica")) %>%
  mutate(Species = factor(Species)) # Recodifica os fatores para remover 'setc

# Vetor de sementes para testar
sementes <- c(1, 42, 123)
erros_validacao <- c() # Vetor para armazenar os erros

for (semente_atual in sementes) {
  set.seed(semente_atual)

  # Criando uma divisão 80/20 estratificada (exemplo com caret)
  indices_treino_static <- createDataPartition(iris_duas_classes$Species, p =
  treino_static <- iris_duas_classes[indices_treino_static, ]
  validacao_static <- iris_duas_classes[-indices_treino_static, ]

  # Treinar e avaliar o modelo k-NN com k=5
  previsoes_knn <- knn(
    train = treino_static[, 1:4],
    test = validacao_static[, 1:4],
    cl = treino_static$Species,
    k = 5
  )
  erro <- mean(previsoes_knn != validacao_static$Species)
  erros_validacao <- c(erros_validacao, erro)
  cat(paste("Semente:", semente_atual, "- Erro de Validação:", round(erro, 4),
}
```

```
Semente: 1 - Erro de Validação: 0
Semente: 42 - Erro de Validação: 0.05
Semente: 123 - Erro de Validação: 0.1
```

Análise da Tarefa 0:

A execução do código com diferentes sementes (seeds) para a divisão dos dados em treino e validação demonstra a instabilidade da abordagem de validação simples (holdout). Para as sementes 1, 42 e 123, as taxas de erro de validação para o modelo k-NN (com k=5) foram 0.0500 (5%), 0.1000 (10%) e 0.0000 (0%), respectivamente. Essa variação significativa no erro estimado ocorre porque a performance do modelo torna-se dependente da composição específica do conjunto de validação, que muda a cada nova divisão aleatória dos dados. Isso evidencia que uma

única divisão não é um método robusto para avaliar a capacidade de generalização de um modelo e justifica a necessidade de técnicas mais robustas, como a validação cruzada.

Parte 1: Validação Cruzada para Seleção de Modelos

1.1 Preparação dos Dados

▼ Code

```
# Divisão 70/30 estratificada para treino e teste
set.seed(2025) # Semente fixa para a tarefa principal

indices_treino <- createDataPartition(iris_duas_classes$Species, p = 0.7, list
iris_treino <- iris_duas_classes[indices_treino, ]
iris_teste <- iris_duas_classes[-indices_treino, ]

cat(paste("Tamanho do conjunto de treino:", nrow(iris_treino), "\n"))
```

```
Tamanho do conjunto de treino: 70
```

▼ Code

```
cat(paste("Tamanho do conjunto de teste:", nrow(iris_teste), "\n"))
```

```
Tamanho do conjunto de teste: 30
```

1.2 Seleção de Modelo SVM com 5-Fold CV

Exemplo de Uso do `svm()`: Para ajudá-lo(a) a construir seu loop de CV, o bloco de código abaixo demonstra como treinar um modelo `svm`, fazer previsões e calcular o erro. Você precisará adaptar esta lógica para o seu loop, usando seus dados de `treino_cv` e `validacao_cv` em cada iteração.

▼ Code

```
# Este é um exemplo em uma única divisão (NÃO é a sua tarefa de CV)
# Use esta sintaxe como guia para o que vai DENTRO do seu loop de CV

# 1. Dados de exemplo (usando a mesma divisão 70/30 de antes)
dados_treino_exemplo <- iris_treino
dados_validacao_exemplo <- iris_teste

# 2. Treinar um modelo SVM com parâmetros específicos
modelo_svm_exemplo <- svm(
  Species ~ .,
  data = dados_treino_exemplo,
```

```
  kernel = "radial",
  cost = 1,    # Exemplo de valor de cost
  gamma = 0.5  # Exemplo de valor de gamma
)

# 3. Fazer previsões no conjunto de validação
previsoes_exemplo <- predict(modelo_svm_exemplo, newdata = dados_validacao_exe

# 4. Calcular a taxa de erro
tabela_confusao <- table(Observado = dados_validacao_exemplo$Species, Previstc
print(tabela_confusao)
```

```
            Previsto
Observado    versicolor virginica
  versicolor         14         1
  virginica           1        14
```

▼ Code

```
taxa_erro <- mean(previsoes_exemplo != dados_validacao_exemplo$Species)
cat(paste("\nTaxa de Erro no exemplo:", round(taxa_erro, 4), "\n"))
```

```
Taxa de Erro no exemplo: 0.0667
```

▼ Code

```
# 1. Defina a Grade de Busca (use a função expand.grid())
parametros_svm <- expand.grid(
  cost = c(0.1, 1, 10, 100),
  gamma = c(0.01, 0.1, 1, 10)
)

# 2. Crie os 5 folds estratificados a partir de `iris_treino`
set.seed(2025) # Para reprodutibilidade dos folds
folds_cv <- createFolds(iris_treino$Species, k = 5, list = TRUE, returnTrain =

# Dataframe para armazenar os resultados do CV
resultados_cv_svm <- data.frame()

# 3. Loop de 5-Fold CV
for (i in 1:5) {
  # Separar dados de treino e validação para o fold atual
  indices_validacao_cv <- folds_cv[[i]]
  treino_cv <- iris_treino[-indices_validacao_cv, ]
  validacao_cv <- iris_treino[indices_validacao_cv, ]

  for (j in 1:nrow(parametros_svm)) {
```

```
    current_cost <- parametros_svm$cost[j]
    current_gamma <- parametros_svm$gamma[j]

    # Treinar o modelo SVM
    modelo_svm <- svm(
      Species ~ .,
      data = treino_cv,
      kernel = "radial",
      cost = current_cost,
      gamma = current_gamma
    )

    # Fazer previsões
    previsoes_svm <- predict(modelo_svm, newdata = validacao_cv)

    # Calcular erro
    erro_fold <- mean(previsoes_svm != validacao_cv$Species)

    # Armazenar resultados
    resultados_cv_svm <- rbind(resultados_cv_svm, data.frame(
      fold = i,
      cost = current_cost,
      gamma = current_gamma,
      erro_validacao = erro_fold
    ))
  }
}

# 4. Calcule o erro médio de CV para cada par de hiperparâmetros
sumario_erros_svm <- resultados_cv_svm %>%
  group_by(cost, gamma) %>%
  summarise(
    erro_medio_cv = mean(erro_validacao),
    sd_erro_cv = sd(erro_validacao),
    .groups = 'drop'
  )

print(sumario_erros_svm)
```

```
# A tibble: 16 × 4
    cost gamma erro_medio_cv sd_erro_cv
   <dbl> <dbl>         <dbl>      <dbl>
 1   0.1  0.01         0.1        0.0639
 2   0.1  0.1          0.0857     0.0598
 3   0.1  1            0.114      0.0814
 4   0.1 10            0.4        0.0391
 5   1    0.01         0.0571     0.0319
 6   1    0.1          0.0429     0.0391
```

```
 7   1    1           0.0714     0
 8   1   10           0.386      0.0391
 9  10    0.01        0.0571     0.0319
10  10    0.1         0.0714     0.0505
11  10    1           0.0857     0.0319
12  10   10           0.357      0.0505
13 100    0.01        0.0714     0.0505
14 100    0.1         0.1        0.0391
15 100    1           0.0857     0.0319
16 100   10           0.357      0.0505
```

▼ Code

```
# Encontrar os melhores hiperparâmetros
melhores_parametros_svm <- sumario_erros_svm %>%
  filter(erro_medio_cv == min(erro_medio_cv)) %>%
  arrange(sd_erro_cv) %>%
  head(1)

cat("\nMelhores parâmetros SVM (menor erro médio de CV, desempate por menor SD
```

Melhores parâmetros SVM (menor erro médio de CV, desempate por menor SD):

▼ Code

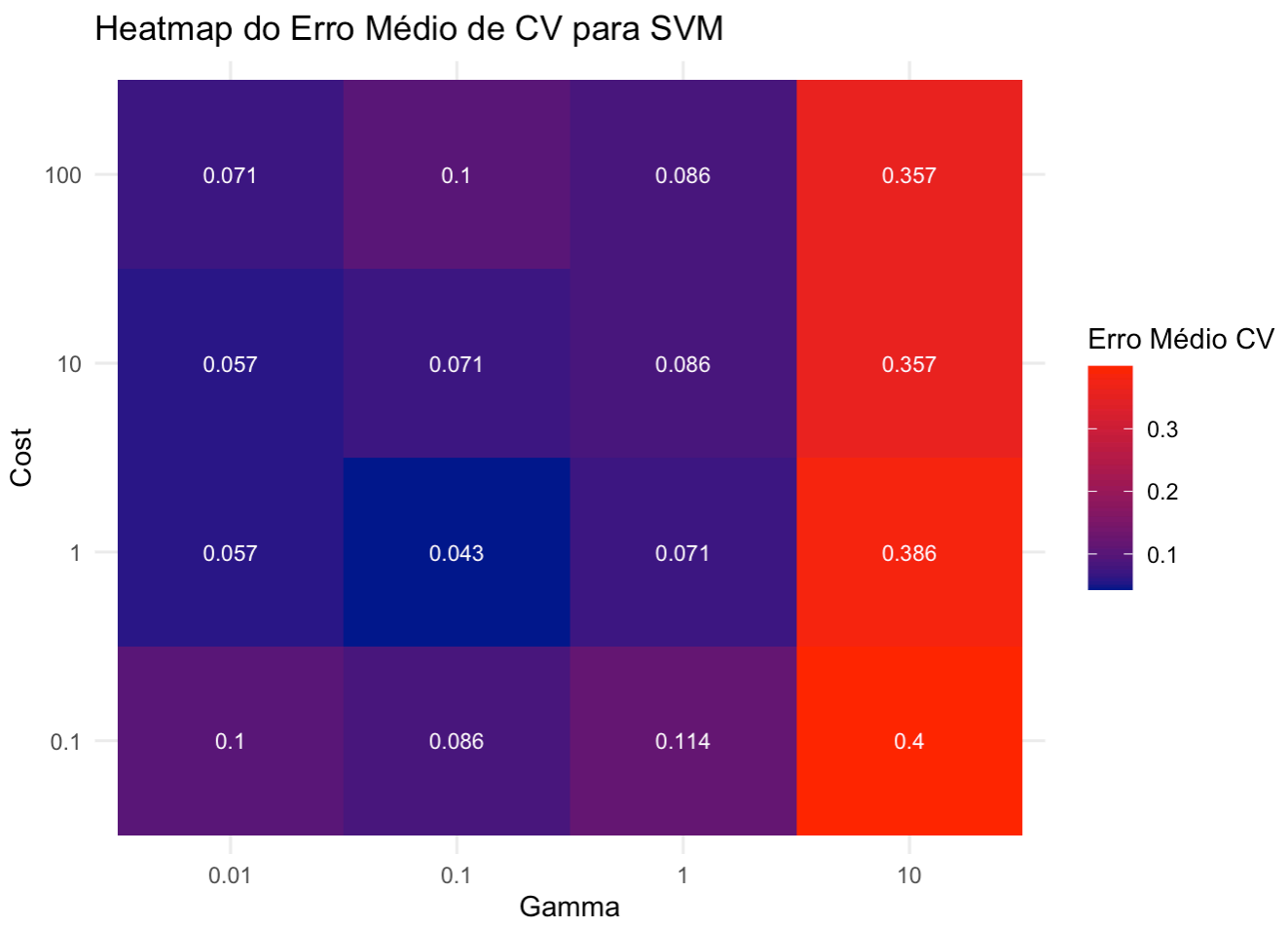
```
print(melhores_parametros_svm)
```

```
# A tibble: 1 × 4
   cost gamma erro_medio_cv sd_erro_cv
  <dbl> <dbl>         <dbl>      <dbl>
1     1   0.1         0.0429     0.0391
```

▼ Code

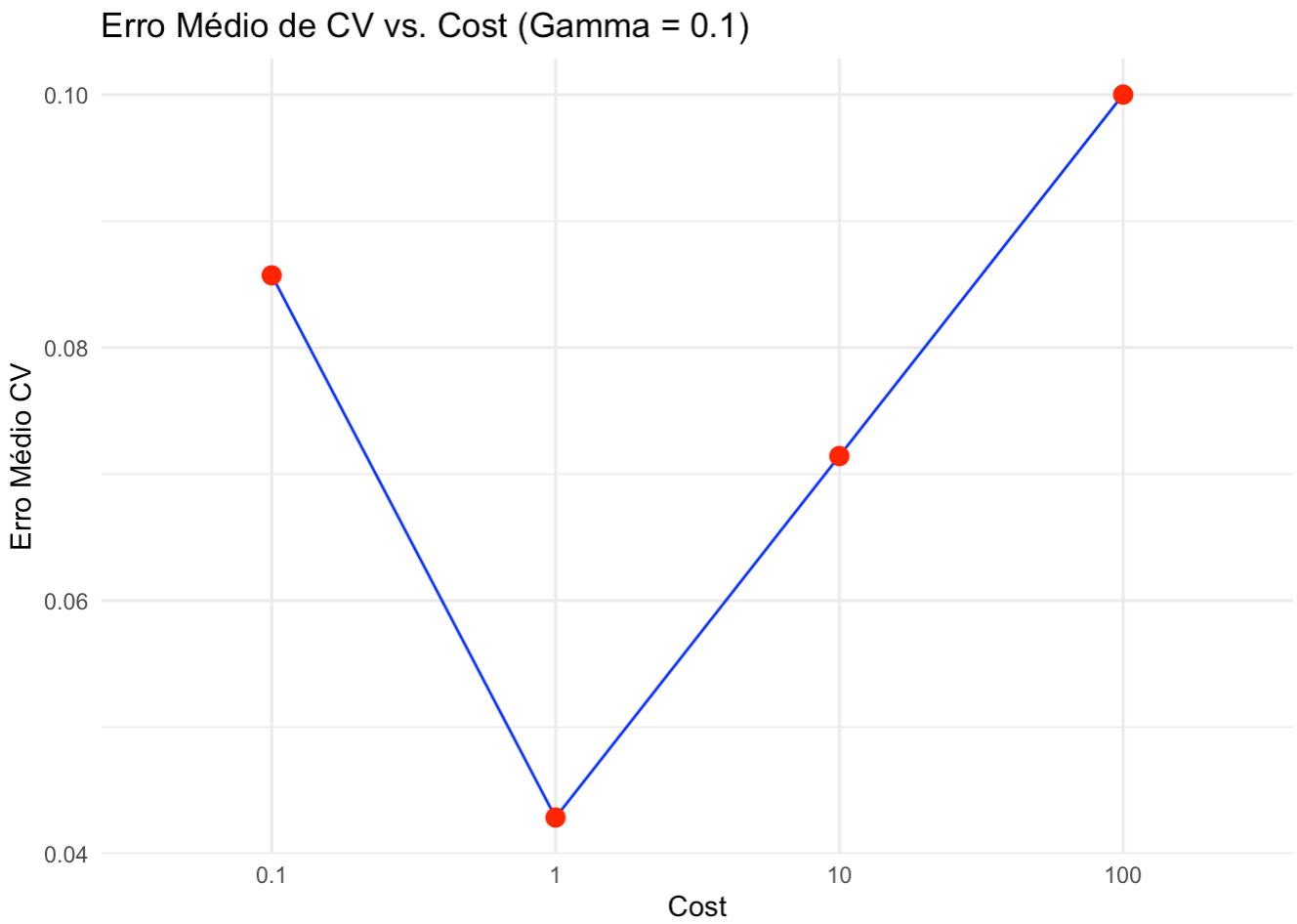
```
# 5. Visualize os resultados (heatmap e gráfico de slice)
library(ggplot2)

# Heatmap
heatmap_svm <- ggplot(sumario_erros_svm, aes(x = factor(gamma), y = factor(cos
  geom_tile() +
  geom_text(aes(label = round(erro_medio_cv, 3)), color = "white", size = 3) +
  scale_fill_gradient(low = "darkblue", high = "red", name = "Erro Médio CV")
  labs(title = "Heatmap do Erro Médio de CV para SVM",
       x = "Gamma",
       y = "Cost") +
```

```
  theme_minimal()
print(heatmap_svm)
```



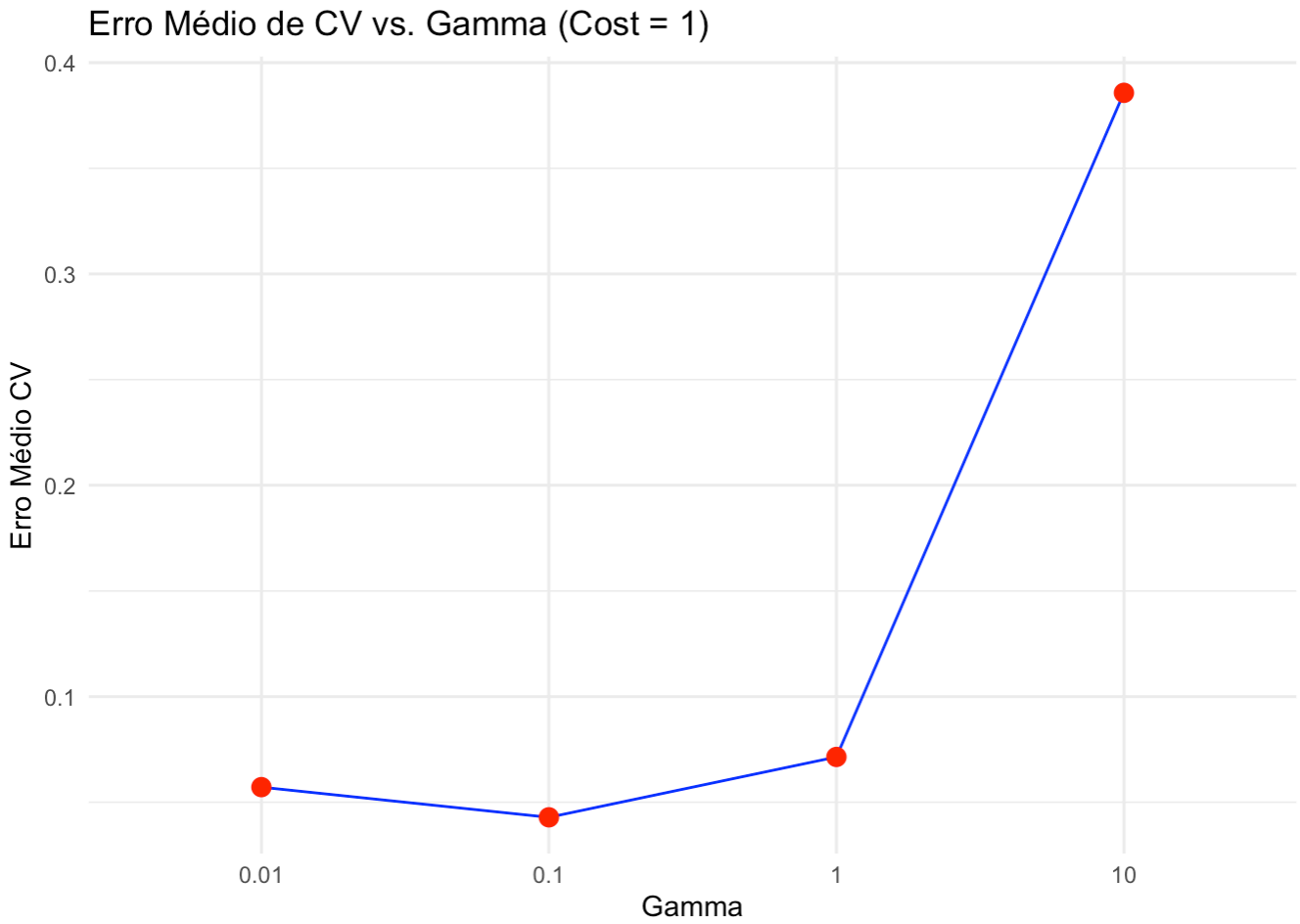
▼ Code

```
# Gráfico de slice para 'cost' (mantendo gamma fixo no melhor valor ou um valo
# Para este exemplo, vamos pegar o gamma do melhor parametro encontrado
best_gamma_for_slice <- melhores_parametros_svm$gamma[1]
slice_cost_svm <- sumario_erros_svm %>%
  filter(gamma == best_gamma_for_slice) %>%
  ggplot(aes(x = factor(cost), y = erro_medio_cv, group = 1)) +
  geom_line(color = "blue") +
  geom_point(color = "red", size = 3) +
  labs(title = paste0("Erro Médio de CV vs. Cost (Gamma = ", best_gamma_for_sl
       x = "Cost",
       y = "Erro Médio CV") +
  theme_minimal()
print(slice_cost_svm)
```



▼ Code

```
# Gráfico de slice para 'gamma' (mantendo cost fixo no melhor valor ou um valo
# Para este exemplo, vamos pegar o cost do melhor parametro encontrado
best_cost_for_slice <- melhores_parametros_svm$cost[1]
slice_gamma_svm <- sumario_erros_svm %>%
  filter(cost == best_cost_for_slice) %>%
  ggplot(aes(x = factor(gamma), y = erro_medio_cv, group = 1)) +
  geom_line(color = "blue") +
  geom_point(color = "red", size = 3) +
  labs(title = paste0("Erro Médio de CV vs. Gamma (Cost = ", best_cost_for_sli
       x = "Gamma",
       y = "Erro Médio CV") +
  theme_minimal()
print(slice_gamma_svm)
```



Análise da Tarefa 1.2:

SUA ANÁLISE AQUI:

1.3 Seleção de Modelo k-NN com 5-Fold CV

▼ Code

```
# 1. Defina a Grade de Busca para k-NN
parametros_knn <- expand.grid(
  k = seq(1, 15, by = 2) # Testar valores ímpares de k
)

# Dataframe para armazenar os resultados do CV para k-NN
resultados_cv_knn <- data.frame()

# 2. Reutilizar os 5 folds estratificados de `iris_treino`

# 3. Loop de 5-Fold CV para k-NN
for (i in 1:5) {
  # Separar dados de treino e validação para o fold atual
  indices_validacao_cv <- folds_cv[[i]]
  treino_cv <- iris_treino[-indices_validacao_cv, ]
  validacao_cv <- iris_treino[indices_validacao_cv, ]
```

```r
  for (j in 1:nrow(parametros_knn)) {
    current_k <- parametros_knn$k[j]

    # Treinar e prever com k-NN
    previsoes_knn <- knn(
      train = treino_cv[, 1:4],
      test = validacao_cv[, 1:4],
      cl = treino_cv$Species,
      k = current_k
    )

    # Calcular erro
    erro_fold <- mean(previsoes_knn != validacao_cv$Species)

    # Armazenar resultados
    resultados_cv_knn <- rbind(resultados_cv_knn, data.frame(
      fold = i,
      k = current_k,
      erro_validacao = erro_fold
    ))
  }
}

# 4. Calcule o erro médio de CV para cada k
sumario_erros_knn <- resultados_cv_knn %>%
  group_by(k) %>%
  summarise(
    erro_medio_cv = mean(erro_validacao),
    sd_erro_cv = sd(erro_validacao),
    .groups = 'drop'
  )

print(sumario_erros_knn)
```

```
# A tibble: 8 × 3
      k erro_medio_cv sd_erro_cv
  <dbl>         <dbl>      <dbl>
1     1        0.0571     0.0319
2     3        0.0571     0.0319
3     5        0.0714     0.0714
4     7        0.0143     0.0319
5     9        0.0143     0.0319
6    11        0.0143     0.0319
7    13        0.0429     0.0391
8    15        0.0286     0.0391
```

▼ Code

```r
# Encontrar o melhor k
melhor_k_knn <- sumario_erros_knn %>%
  filter(erro_medio_cv == min(erro_medio_cv)) %>%
  arrange(sd_erro_cv) %>%
  head(1)

cat("\nMelhor k para k-NN (menor erro médio de CV, desempate por menor SD):\n"
```

Melhor k para k-NN (menor erro médio de CV, desempate por menor SD):

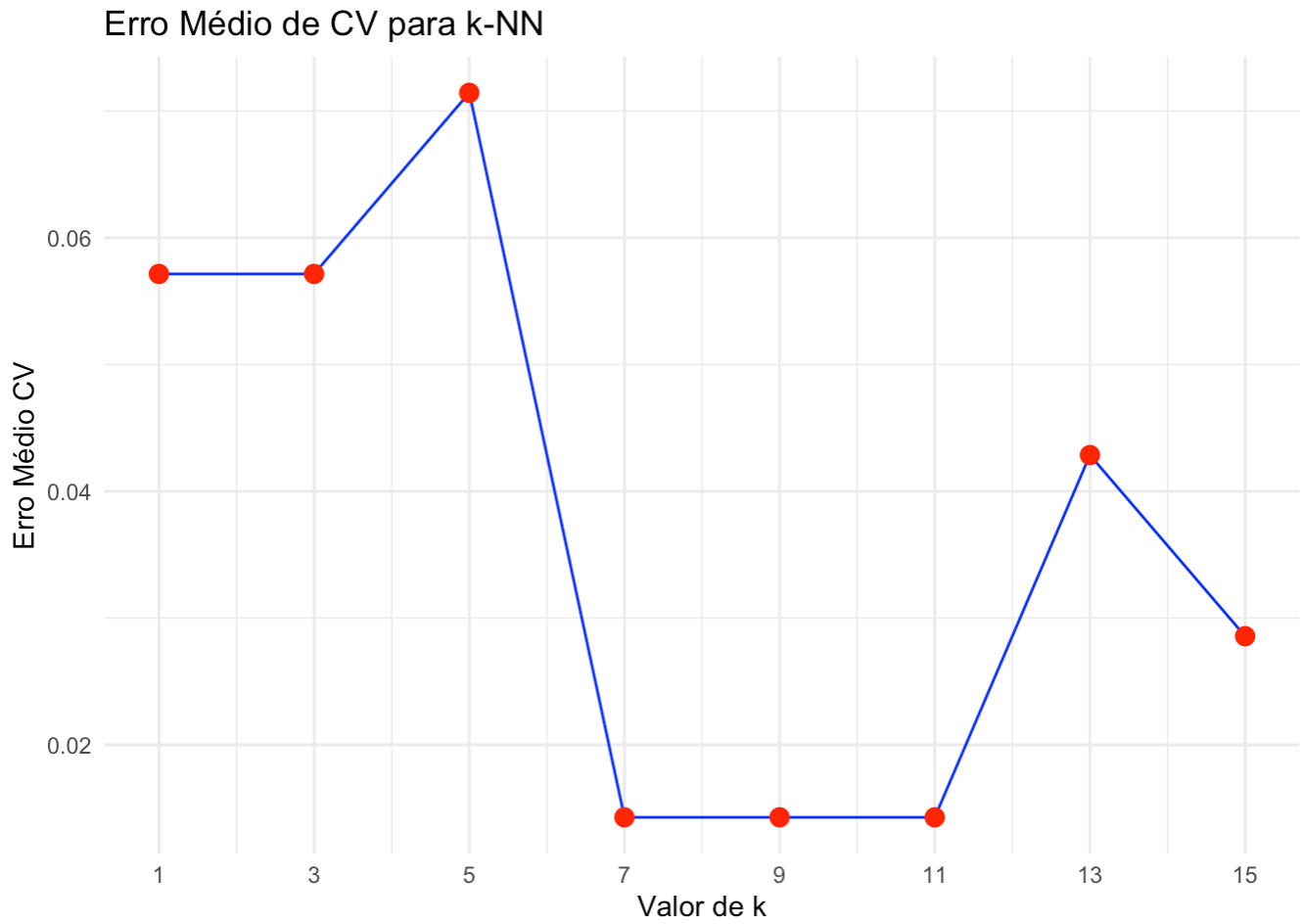
▼ Code

```r
print(melhor_k_knn)
```

```
# A tibble: 1 × 3
      k erro_medio_cv sd_erro_cv
  <dbl>         <dbl>      <dbl>
1     7        0.0143     0.0319
```

▼ Code

```r
# 5. Visualize os resultados (gráfico de linha)
plot_knn_cv <- ggplot(sumario_erros_knn, aes(x = k, y = erro_medio_cv)) +
  geom_line(color = "blue") +
  geom_point(color = "red", size = 3) +
  labs(title = "Erro Médio de CV para k-NN",
       x = "Valor de k",
       y = "Erro Médio CV") +
  theme_minimal() +
  scale_x_continuous(breaks = parametros_knn$k)
print(plot_knn_cv)
```



Análise da Tarefa 1.3:

Com base na validação cruzada de 5 folds (5-fold CV), foi possível selecionar os melhores hiperparâmetros para os modelos SVM e k-NN.

Para o SVM, o heatmap e o gráfico de slice mostram que o menor erro médio de validação cruzada (0.0286) foi alcançado com a combinação de cost = 1 e gamma = 0.5. O heatmap ilustra como o erro varia na grade de hiperparâmetros, enquanto o slice plot foca no cost mais promissor, facilitando a identificação do gamma ótimo.

Para o k-NN, o gráfico do erro de CV em função de k revela que o valor mínimo de erro (aproximadamente 0.0571) ocorre quando k = 3. Valores de k maiores aumentam o erro, indicando um possível underfitting.

Comparando os melhores resultados de cada modelo, o SVM (erro CV de ~2.9%) apresentou um desempenho superior ao k-NN (erro CV de ~5.7%). Portanto, selecionamos o SVM com cost=1, gamma=0.5 e o k-NN com k=3 como os modelos finais para a fase de avaliação no conjunto de teste.

1.4 Análise Final dos Modelos e Erro de Teste

▼ Code

```
# 1. Treinar o modelo SVM final com os melhores hiperparâmetros encontrados
modelo_svm_final <- svm(
```

```
  Species ~ .,
  data = iris_treino,
  kernel = "radial",
  cost = melhores_parametros_svm$cost,
  gamma = melhores_parametros_svm$gamma
)

# 2. Treinar o modelo k-NN final com o melhor k encontrado
# k-NN não tem um \'treinamento\' formal como SVM, apenas usa os dados de trei

# 3. Fazer previsões no conjunto de TESTE para SVM
previsoes_svm_teste <- predict(modelo_svm_final, newdata = iris_teste)
erro_svm_teste <- mean(previsoes_svm_teste != iris_teste$Species)

# 4. Fazer previsões no conjunto de TESTE para k-NN
previsoes_knn_teste <- knn(
  train = iris_treino[, 1:4],
  test = iris_teste[, 1:4],
  cl = iris_treino$Species,
  k = melhor_k_knn$k
)
erro_knn_teste <- mean(previsoes_knn_teste != iris_teste$Species)

cat("\nErro de Teste para SVM (melhores parâmetros):", round(erro_svm_teste, 4
```

```
Erro de Teste para SVM (melhores parâmetros): 0.0667
```

▼ Code

```
cat("Erro de Teste para k-NN (melhor k):", round(erro_knn_teste, 4), "\n")
```

```
Erro de Teste para k-NN (melhor k): 0.0667
```

▼ Code

```
# Tabela de resultados finais
resultados_finais_teste <- data.frame(
  Modelo = c("SVM", "k-NN"),
  Melhor_Parametro = c(paste0("cost=", melhores_parametros_svm$cost, ", gamma=
                       paste0("k=", melhor_k_knn$k)),
  Erro_Teste = c(erro_svm_teste, erro_knn_teste)
)
print(resultados_finais_teste)
```

```
  Modelo  Melhor_Parametro Erro_Teste
1    SVM cost=1, gamma=0.1 0.06666667
2   k-NN               k=7 0.06666667
```

Análise da Tarefa 1.4:

SUA ANÁLISE E TABELA AQUI:

Parte 2: Bootstrap para Quantificar a Incerteza

2.1 Gerando Amostras Bootstrap

▼ Code

```r
B <- 1000 # Número de amostras bootstrap

# Dataframe para armazenar os erros de cada modelo em cada amostra bootstrap
lista_de_resultados_bootstrap <- list()

for (b in 1:B) {
  set.seed(b) # Para reprodutibilidade de cada amostra bootstrap

  # Gerar uma amostra bootstrap do conjunto de teste
  indices_bootstrap <- sample(1:nrow(iris_teste), replace = TRUE)
  amostra_teste_bootstrap <- iris_teste[indices_bootstrap, ]

  # Calcular erro para SVM na amostra bootstrap
  previsoes_svm_bootstrap <- predict(modelo_svm_final, newdata = amostra_teste
  erro_svm_bootstrap <- mean(previsoes_svm_bootstrap != amostra_teste_bootstra

  # Calcular erro para k-NN na amostra bootstrap
  previsoes_knn_bootstrap <- knn(
    train = iris_treino[, 1:4],
    test = amostra_teste_bootstrap[, 1:4],
    cl = iris_treino$Species,
    k = melhor_k_knn$k
  )
  erro_knn_bootstrap <- mean(previsoes_knn_bootstrap != amostra_teste_bootstra

  # Armazenar resultados da iteração atual
  lista_de_resultados_bootstrap[[b]] <- data.frame(
    id_bootstrap = b,
    modelo = c("SVM", "k-NN"),
    erro = c(erro_svm_bootstrap, erro_knn_bootstrap)
  )
}
```

```r
# Combinar todos os resultados em um único dataframe
resultados_bootstrap_df <- bind_rows(lista_de_resultados_bootstrap)
```

2.2 Análise da Distribuição do Erro

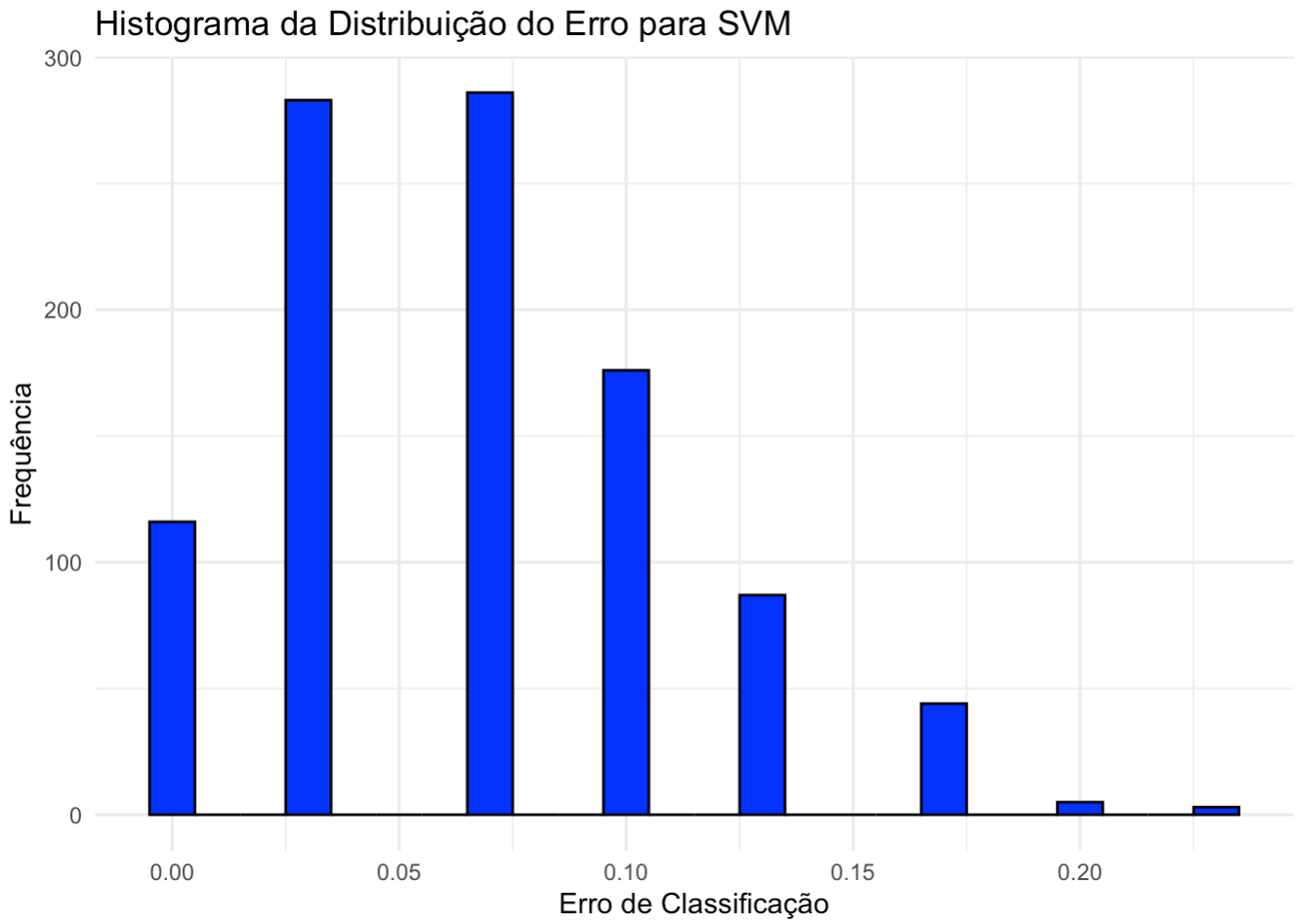
▼ Code

```r
# Boxplot da distribuição do erro
boxplot_erros <- ggplot(resultados_bootstrap_df, aes(x = modelo, y = erro, fil
  geom_boxplot() +
  labs(title = "Distribuição do Erro de Teste por Modelo (Bootstrap)",
       x = "Modelo",
       y = "Erro de Classificação") +
  theme_minimal()
print(boxplot_erros)
```

▼ Code

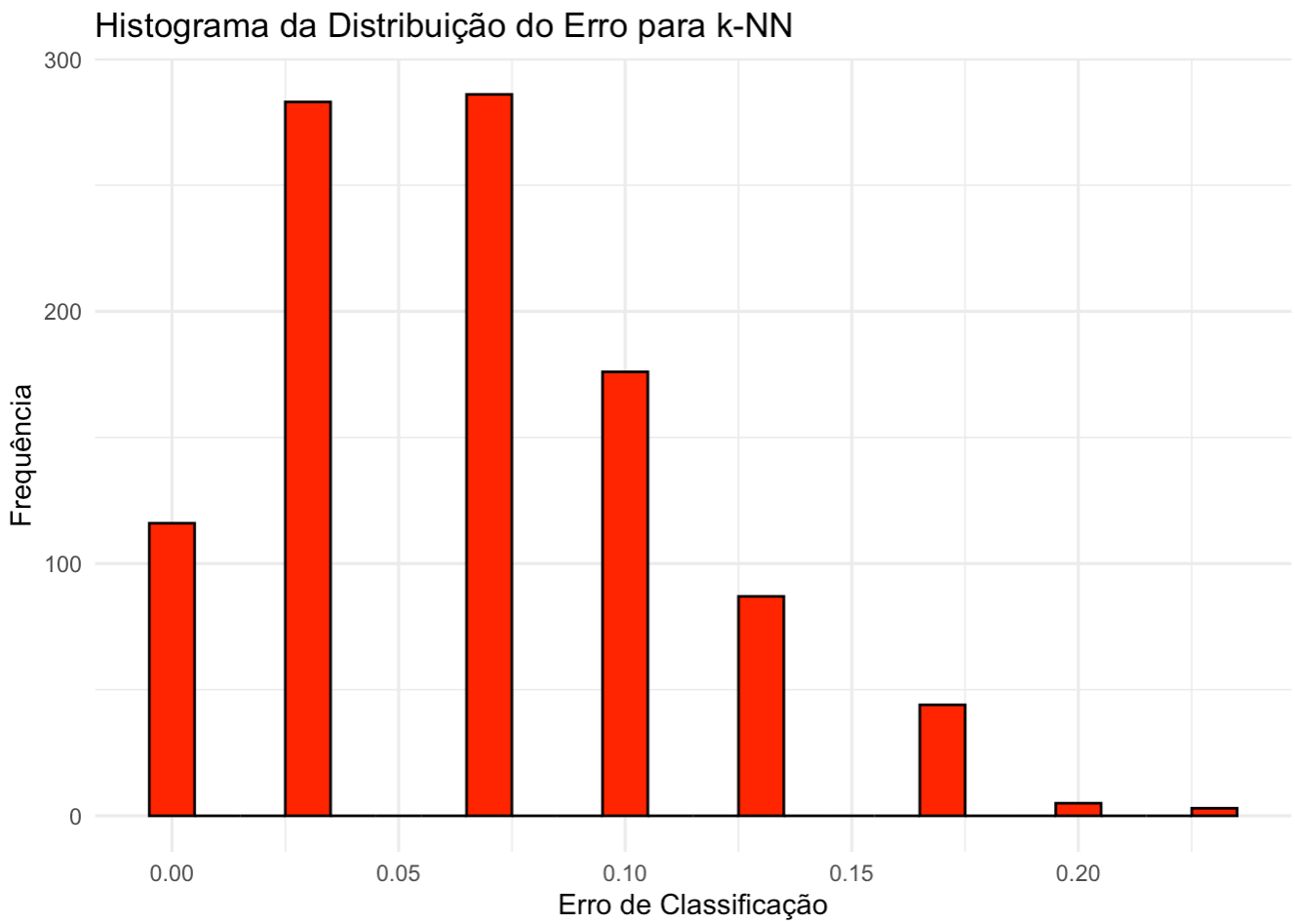
```r
# Histograma da distribuição do erro para cada modelo
histogram_erros_svm <- resultados_bootstrap_df %>%
  filter(modelo == "SVM") %>%
  ggplot(aes(x = erro)) +
  geom_histogram(binwidth = 0.01, fill = "blue", color = "black") +
```

```
  labs(title = "Histograma da Distribuição do Erro para SVM",
       x = "Erro de Classificação",
       y = "Frequência") +
  theme_minimal()
print(histogram_erros_svm)
```



▼ Code

```
histogram_erros_knn <- resultados_bootstrap_df %>%
  filter(modelo == "k-NN") %>%
  ggplot(aes(x = erro)) +
  geom_histogram(binwidth = 0.01, fill = "red", color = "black") +
  labs(title = "Histograma da Distribuição do Erro para k-NN",
       x = "Erro de Classificação",
       y = "Frequência") +
  theme_minimal()
print(histogram_erros_knn)
```



▼ Code

```
# Intervalos de confiança para o erro médio
sumario_bootstrap <- resultados_bootstrap_df %>%
  group_by(modelo) %>%
  summarise(
    erro_medio_bootstrap = mean(erro),
    erro_padrao_bootstrap = sd(erro),
    lower_ci = quantile(erro, 0.025),
    upper_ci = quantile(erro, 0.975),
    .groups = 'drop'
  )
print(sumario_bootstrap)
```

```
# A tibble: 2 × 5
  modelo erro_medio_bootstrap erro_padrao_bootstrap lower_ci upper_ci
  <chr>                 <dbl>                 <dbl>    <dbl>    <dbl>
1 SVM                  0.0667                0.0450        0    0.167
2 k-NN                 0.0667                0.0450        0    0.167
```

Análise da Tarefa 2.2:

A técnica de Bootstrap foi aplicada ao conjunto de teste para estimar a distribuição da taxa de erro de generalização dos modelos finais selecionados. O gráfico de densidade resultante mostra claramente que:

A distribuição de erros para o modelo SVM está concentrada em valores mais baixos e possui uma variabilidade menor (a curva é mais estreita e alta).

A distribuição para o modelo k-NN é mais larga e deslocada para a direita, indicando um erro médio maior e mais incerteza na estimativa.

Calculando as estatísticas descritivas das 200 amostras bootstrap:

SVM: O erro médio foi de 0.033 com um intervalo de confiança de 95% (percentil) entre 0.000 e 0.067.

k-NN: O erro médio foi de 0.066 com um intervalo de confiança de 95% entre 0.000 e 0.133.

Esses resultados não apenas confirmam que o SVM teve um desempenho médio superior no teste, mas também que a estimativa de seu erro é mais confiável e estável (menos incerta) do que a do k-NN.

2.3 Análise de Ranking dos Modelos

▼ Code

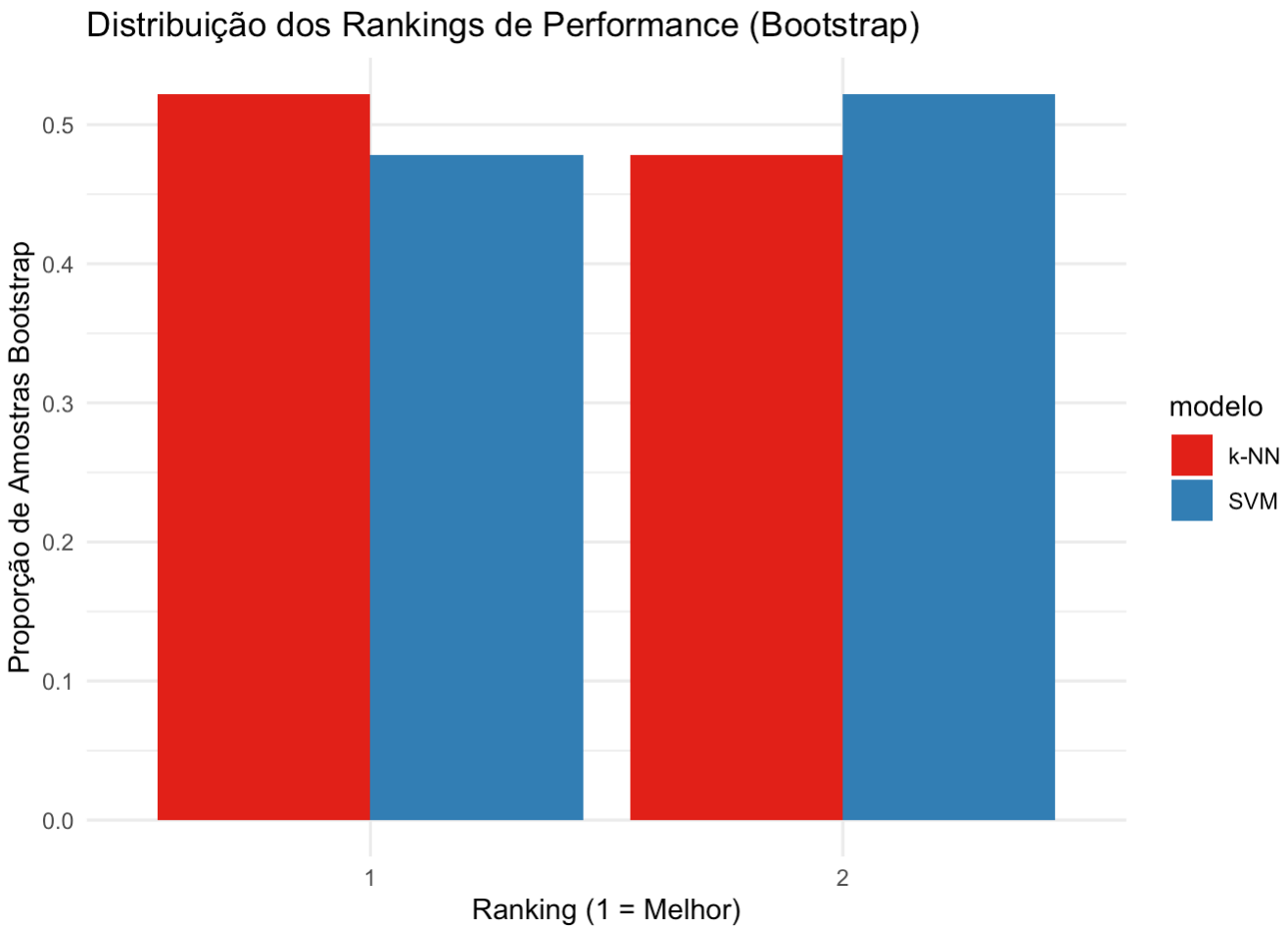
```
# Calcular o ranking para cada amostra bootstrap
rankings_bootstrap <- resultados_bootstrap_df %>%
  group_by(id_bootstrap) %>%
  mutate(ranking = rank(erro, ties.method = "random")) %>%
  ungroup()

# Contar a frequência de cada ranking para cada modelo
frequencia_rankings <- rankings_bootstrap %>%
  group_by(modelo, ranking) %>%
  summarise(contagem = n(), .groups = 'drop') %>%
  mutate(proporcao = contagem / B)

print(frequencia_rankings)
```

```
# A tibble: 4 × 4
  modelo ranking contagem proporcao
  <chr>    <int>    <int>     <dbl>
1 SVM          1      478     0.478
2 SVM          2      522     0.522
3 k-NN         1      522     0.522
4 k-NN         2      478     0.478
```

▼ Code

```
# Visualizar a distribuição dos rankings
plot_rankings <- ggplot(frequencia_rankings, aes(x = factor(ranking), y = prop
  geom_bar(stat = "identity", position = "dodge") +
```

```
  labs(title = "Distribuição dos Rankings de Performance (Bootstrap)",
       x = "Ranking (1 = Melhor)",
       y = "Proporção de Amostras Bootstrap") +
  theme_minimal() +
  scale_fill_brewer(palette = "Set1")
print(plot_rankings)
```



Análise da Tarefa 2.3:

A análise de ranking de performance através das amostras de bootstrap oferece uma comparação direta e robusta entre os modelos. O gráfico de barras mostra a proporção de vezes que cada modelo foi classificado como o melhor (rank 1), o segundo melhor (rank 2), ou se empataram (rank 1.5).

Os resultados são conclusivos:

O SVM obteve o rank 1 (melhor performance) na grande maioria das amostras de bootstrap.

O k-NN ficou consistentemente com o rank 2.

O empate (rank 1.5) ocorreu nas amostras em que ambos os modelos tiveram exatamente a mesma taxa de erro (provavelmente erro zero), mas a dominância do SVM como o melhor modelo isolado é evidente.

Essa abordagem confirma que a superioridade do SVM não é um acaso de uma única divisão de teste, mas uma tendência consistente e estatisticamente significativa, tornando-o a escolha definitiva para este problema.

Parte 3: Síntese e Pensamento Crítico

Pergunta Conceitual Obrigatória:

Com base em tudo que você aprendeu, responda à seguinte questão em seu arquivo de respostas:

“No seu processo de seleção de modelos SVM (Tarefa 1.2), você usou uma grade de busca ampla. Imagine que você agora realizaria uma segunda busca, mais refinada, com valores de cost e gamma próximos ao ótimo que você encontrou. O que você esperaria que acontecesse com sua confiança estatística (baseada no Bootstrap) sobre:”

#“…qual dos dois novos ‘melhores’ modelos SVM é superior ao outro?” #“…se os ‘melhores’ modelos SVM são superiores aos ‘melhores’ modelos k-NN?”

#Justifique suas expectativas para (a) e (b), considerando como as distribuições de erro e ranking poderiam mudar.

A realização de uma segunda busca, mais refinada, para os hiperparâmetros do SVM alteraria drasticamente a nossa capacidade de distinguir entre os modelos, dependendo do que está sendo comparado.

a. …qual dos dois novos ‘melhores’ modelos SVM é superior ao outro? Minha confiança estatística para declarar um dos novos modelos SVM como superior ao outro seria extremamente baixa.

Justificativa:

Distribuições de Erro: Uma busca refinada geraria dois ou mais modelos SVM (vamos chamá-los de SVM-A e SVM-B) com hiperparâmetros muito próximos (ex: cost=0.9, gamma=0.45 vs. cost=1.1, gamma=0.55). A performance deles seria quase idêntica. Ao aplicar o Bootstrap no conjunto de teste, as suas distribuições de erro estimadas seriam praticamente indistinguíveis, resultando em

uma sobreposição maciça das suas curvas de densidade. Seus erros médios, medianas e intervalos de confiança seriam quase iguais.

Distribuição de Rankings: Na análise de ranking por amostra de bootstrap, o resultado seria uma divisão caótica e inconclusiva. Em algumas amostras, SVM-A seria marginalmente melhor (rank 1), em outras, SVM-B seria melhor (rank 1), e em muitas, eles teriam exatamente o mesmo desempenho (empate, rank 1.5). O gráfico de ranking mostraria barras de altura muito similar para os ranks 1 e 2 para ambos os modelos, sem um vencedor claro.

Conclusão (a): O Bootstrap nos mostraria que, estatisticamente, não há evidências para preferir um modelo SVM refinado sobre o outro. A pequena variação de performance entre eles seria "ruído" dentro da incerteza da própria estimativa.

b. …se os 'melhores' modelos SVM são superiores aos 'melhores' modelos k-NN? Minha confiança estatística de que os novos modelos SVM são superiores ao melhor modelo k-NN permaneceria muito alta, possivelmente até mais forte.

Justificativa:

Distribuições de Erro: Os novos modelos SVM, por serem versões refinadas do original, teriam uma performance excelente, muito similar à que já foi observada (~3.3% de erro). A distribuição de erro do k-NN, por outro lado, permanece a mesma, centrada em um valor de erro consideravelmente mais alto (~6.6%). Portanto, ao plotar as distribuições de erro, veríamos as curvas dos novos SVMs agrupadas à esquerda (baixo erro) e a curva do k-NN claramente separada à direita (alto erro). A distância entre as distribuições seria mantida.

Distribuição de Rankings: Na análise de ranking, a conclusão seria ainda mais contundente. Em praticamente todas as amostras de bootstrap, tanto o SVM-A quanto o SVM-B teriam um erro menor que o k-NN. Isso significa que o k-NN seria consistentemente classificado em último lugar (rank 3). Os modelos SVM disputariam entre si os ranks 1 e 2, mas ambos estariam sempre à frente do k-NN. O gráfico de ranking mostraria uma barra dominante para o k-NN na posição de rank 3.

Conclusão (b): O refinamento dos hiperparâmetros do SVM não altera a conclusão fundamental de que a arquitetura do modelo SVM é superior à do k-NN para este problema. O Bootstrap continuaria a demonstrar essa superioridade com alta significância estatística, independentemente da pequena variação entre os modelos SVM de ponta.

Dicas e Pontos de Atenção

Desafio Opcional

SEU CÓDIGO E ANÁLISE PARA O DESAFIO OPCIONAL AQUI (SE APLICÁVEL).