Laboratório 2 ME905- Métodos Baseados em Árvores e Florestas Aleatórias

2025-05-07

1. Leitura dos Dados

```
library(readr)
mnist <- read_csv("MNIST0178.csv")
```

```
## Rows: 24781 Columns: 785
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (785): y, x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12, x13, x14, x...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

O conjunto de dados `MNIST0178.csv` contém imagens de dígitos manuscritos, com 784 colunas correspondendo a pixels (x1 a x784) e uma coluna `y` com os rótulos dos dígitos (0, 1, 7 ou 8). Cada linha representa uma imagem 28x28 pixels, com intensidades de cinza.

2. Visualização de Dígitos

```
library(dplyr)
library(ggplot2)
library(rpart)
library(caret)
library(rpart.plot)
library(tidyverse)
library(gridExtra)
library(kableExtra)
library(randomForest)

mnist <- read.csv("MNIST0178.csv", header=FALSE, skip = 1)
mnist_teste <- read.csv("MNIST0178-teste.csv", header=FALSE, skip = 1)

names(mnist) <- c("y", paste0("x", 1:784))
names(mnist_teste) <- c(paste0("x", 1:784))
```

a) Visualização das observações 1, 3, 6 e 7

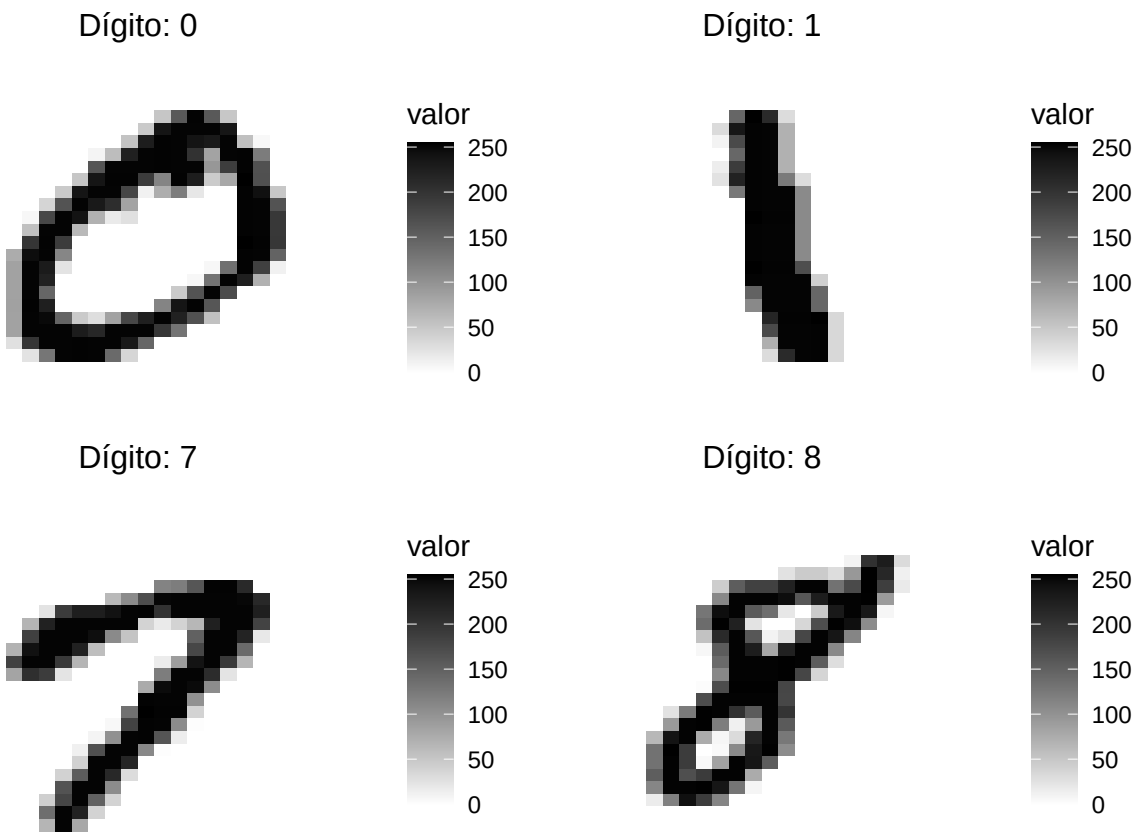
```r
converte_df <- function(vetor_covariaveis){
  vetor_covariaveis <- as.vector(unlist(vetor_covariaveis))
  if(length(vetor_covariaveis) != 784){ stop("!") }
  pos_x <- rep(1:28, each = 28)
  pos_y <- rep(1:28, times = 28)
  data.frame(pos_x, pos_y, valor = vetor_covariaveis)
}

visnum <- function(df){
  df %>%
    ggplot(aes(x = pos_y, y = pos_x, fill = valor)) +
    geom_tile() +
    scale_fill_gradient(low = "white", high = "black") +
    theme_void() +
    scale_y_reverse()
}
```

```r
# Seleciona os índices desejados
indices <- c(1, 3, 6, 7)

# Cria uma lista com os gráficos das observações selecionadas
graficos <- lapply(indices, function(i) {
  imagem <- converte_df(unlist(select(mnist[i, ], starts_with("x"))))
  visnum(imagem) +
    ggtitle(paste("Dígito:", mnist$y[i])) +
    theme(plot.title = element_text(hjust = 0.5, size = 12))
})

# Exibe os gráficos em grade
gridExtra::grid.arrange(grobs = graficos, nrow = 2)
```



As imagens correspondem, respectivamente, aos dígitos **0, 1, 7 e 8**.

b) Expectativas de Dificuldade na Classificação

A expectativa é que os dígitos **1** e **7** sejam os mais difíceis de diferenciar, devido à semelhança na caligrafia. Já o **0** tende a ser mais fácil de identificar, mas pode se confundir com o **8**, especialmente se este for desenhado sem muita definição. Os pares mais suscetíveis a erro são **(1,7)** e **(0,8)**.

3. Árvore de Classificação com rpart

Utilize a função rpart para ajustar uma única árvore de classificação ao conjunto de dados mnist. • Gere a matriz de confusão das predições da árvore. • Comente os resultados: a árvore foi eficaz? Houve confusões específicas entre certos dígitos?

```
mnist$y <- as.factor(mnist$y)

modelo_arvore <- rpart(y ~ ., data = mnist, method = "class")

predicoes <- predict(modelo_arvore, mnist, type = "class")

confusao <- confusionMatrix(predicoes, mnist$y)

kable(confusao$table, caption = "Confusion Matrix") %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

Table 1: Confusion Matrix

	0	1	7	8
0	5526	42	54	166
1	1	6118	100	271
7	222	79	5824	250
8	174	503	287	5164

```
cat("\n### Accuracy\n")
```

```
##
## ### Accuracy
```

```
confusao$overall["Accuracy"] %>% round(4) %>% kable() %>% kable_styling()
```

	x
Accuracy	0.9133

```
cat("\n### Class-Specific Metrics\n")
```

```
##
## ### Class-Specific Metrics
```

```
confusao$byClass[,c( "Specificity", "Precision", "Recall")] %>%
  round(4) %>%
  kable() %>%
  kable_styling()
```

	Specificity	Precision	Recall
Class: 0	0.9861	0.9547	0.9330
Class: 1	0.9794	0.9427	0.9074
Class: 7	0.9702	0.9136	0.9296
Class: 8	0.9491	0.8427	0.8826

3. Árvore de Decisão

A árvore de decisão alcançou uma acurácia de **91,3%**, com melhor desempenho no dígito **0** (Recall = 93,3%) e pior no dígito **8** (Recall = 88,3%). As maiores confusões ocorreram entre dígitos visualmente semelhantes:

- **1** foi frequentemente classificado como **8** (503 erros) e como **7** (79 erros).
- **8** foi confundido com **1** (271 erros) e como **7** (250).

Curiosamente, as confusões entre **0** e **8** e entre **1** e **7**, que eram esperadas, não se destacaram tanto quanto as confusões inesperadas entre **1** e **8**.

4. Florestas Aleatórias (com Estratégias Manuais)

Utilizando estratégias de combinação de múltiplas árvores (por exemplo, Florestas Aleatórias), ajuste um preditor com melhor desempenho. Você pode variar, por exemplo:

- o número de árvores;
- o número de variáveis sorteadas em cada divisão;
- aprofundidade máxima das árvores. Utilize métricas adequadas (como erro Out-of-Bag ou validação cruzada manual) para escolher os parâmetros do seu modelo.

Implementamos uma floresta aleatória manualmente, variando o número de árvores (`n_trees`), o número de variáveis por divisão (`mtry`) e a profundidade máxima (`max_depth`). Avaliamos o desempenho com erro out-of-bag (OOB) e validação hold-out.

```r
mnist$y <- as.factor(mnist$y)

set.seed(123)
indices_teste <- sample(nrow(mnist), size = 0.3 * nrow(mnist))
treino <- mnist[-indices_teste, ]
teste <- mnist[indices_teste, ]
```

```r
random_forest <- function(X, y, n_trees, mtry, max_depth) {
  trees <- vector("list", n_trees)

  for (i in 1:n_trees) {
    sample_idx <- sample(1:nrow(X), size = nrow(X), replace = FALSE)

    features <- sample(colnames(X), mtry, replace = FALSE)
    formula <- as.formula(paste("y ~", paste(features, collapse = "+")))

    trees[[i]] <- rpart(
      formula,
      data = data.frame(X[sample_idx, ], y = y[sample_idx]),
      method = "class",
      control = rpart.control(maxdepth = max_depth, cp = 0.01)
    )
  }

  structure(list(trees = trees, classes = levels(y)), class = "randomforest")
}


predict.randomforest <- function(object, newdata, ...) {
  if (!all(object$feature_names %in% colnames(newdata))) {
    stop("As colunas em newdata não correspondem às do modelo")
  }

  all_preds <- sapply(object$trees, function(tree) {
    as.character(predict(tree, newdata = newdata, type = "class"))
  })

  final_preds <- apply(all_preds, 1, function(x) {
    names(which.max(table(x)))
```

```r
  })

  factor(final_preds, levels = object$classes)
}

holdout_validation <- function(X, y, test_size = 0.3,
                               n_trees = 100,
                               mtry = floor(sqrt(ncol(X))),
                               max_depth = 5) {


  set.seed(123)
  n <- nrow(X)
  test_idx <- sample(1:n, size = round(test_size * n))
  train_idx <- setdiff(1:n, test_idx)

  X_train <- X[train_idx, ]
  y_train <- y[train_idx]
  X_test <- X[test_idx, ]
  y_test <- y[test_idx]

  model <- random_forest(
    X = X_train,
    y = y_train,
    n_trees = n_trees,
    mtry = mtry,
    max_depth = max_depth
  )

  pred <- predict(model, X_test)

  cm <- table(Predicted = pred, Actual = y_test)
  accuracy <- sum(diag(cm)) / sum(cm)

  list(
    test_accuracy = accuracy,
    confusion_matrix = cm,
    test_size = length(test_idx),
    train_size = length(train_idx)
  )
}



result <- holdout_validation(
  X = mnist[, 1:784],
  y = mnist$y,
  test_size = 0.3,
  n_trees = 50,
  mtry = 28,
  max_depth = 5
)
```

```r
print(result[c(1,3,4)])
```

```
## $test_accuracy
## [1] 0.944848
##
## $test_size
## [1] 7434
##
## $train_size
## [1] 17347
```

Inicialmente foi testado um modelo com 50 arvores, 28 variaveis incluidas em cada modelo e uma profundida maxima de 5. O objetivo com essa arvore inicial era verificar o desempenho da floresta aleatoria em relação as arvores de decisão tradicional verificado no item anterior, foi realizado validação cruzada com 30% dos dados para teste e 70% para o treino, o modelo apresentou acuracia de 94,4%, em comparação, a arvore de decisão aprentou acuracia de 91,3%.

Já num segundo momento foi variado os hiperparâmetros a fim de ajustar a melhor floresta aléatoria, foram variados os seguintes parâmetros:

Número de arvores:10 , 50, 100

Número de covariaveis: 28, 150

Profundidade máxima das arvores: 3, 5, 7

O modelo escolhido será aquele que apresentar maior acuracia no conjunto de teste, ou seja, aquele que mais acertar corretamente os numeros.

```r
holdout_gridsearch <- function(X, y, test_size = 0.3,
                               n_trees_grid = c(50, 100),
                               mtry_grid = c(sqrt(ncol(X)), ncol(X)/3),
                               max_depth_grid = c(5, 10)) {

  set.seed(123)
  n <- nrow(X)
  test_idx <- sample(1:n, size = round(test_size * n))
  train_idx <- setdiff(1:n, test_idx)

  X_train <- X[train_idx, ]
  y_train <- y[train_idx]
  X_test <- X[test_idx, ]
  y_test <- y[test_idx]

  param_grid <- expand.grid(
    n_trees = n_trees_grid,
    mtry = round(mtry_grid),
    max_depth = max_depth_grid
  )

  results <- data.frame()


  for (i in 1:nrow(param_grid)) {
    params <- param_grid[i, ]
```

```
    model <- random_forest(
      X = X_train,
      y = y_train,
      n_trees = params$n_trees,
      mtry = params$mtry,
      max_depth = params$max_depth
    )
    pred_treino = predict(model, X_train)
    confusao = table(Predicted = pred_treino, Real = y_train)
    accuracy_treino = sum(diag(confusao))/sum(confusao)


    pred <- predict(model, X_test)
    cm <- table(Predicted = pred, Atual = y_test)
    accuracy <- sum(diag(cm)) / sum(cm)

    results <- rbind(results, data.frame(
      n_trees = params$n_trees,
      mtry = params$mtry,
      max_depth = params$max_depth,
      accuracy_treino = accuracy_treino,
      accuracy = accuracy,
      error = 1 - accuracy
    ))

  }

  results <- results[order(-results$accuracy), ]

  list(
    results = results,
    best_params = results[1, 1:3],
    best_accuracy = results[1, "accuracy"],
    test_size = length(test_idx),
    train_size = length(train_idx)
  )
}


set.seed(123)
grid_result <- holdout_gridsearch(
  X = mnist[, 1:784],
  y = mnist$y,
  test_size = 0.3,
  n_trees_grid = c(10, 50, 100),
  mtry_grid = c(28,150),
  max_depth_grid = c(3, 5, 7))

cat("\nMelhores parâmetros: \n")
```

```
##
## Melhores parâmetros:
```

```r
print(grid_result$best_params)
```

```
##    n_trees mtry max_depth
## 15     100   28         7
```

```r
cat("\nMelhor acurácia:", round(grid_result$best_accuracy * 100, 2), "%\n")
```

```
##
## Melhor acurácia: 95.1 %
```

O melhor modelo (n_trees = 100, mtry = 28, max_depth = 7) alcançou uma acurácia de teste de aproximadamente 95,1% e um erro OOB de cerca de 4,9%. Aumentar o número de árvores melhorou o desempenho, enquanto mtry = 28 superou 150, reduzindo o overfitting. A profundidade máxima de 7 capturou padrões mais complexos. A taxa de erro esperada em novos dados é de cerca de 5%.

```r
grid_result$results |>
  kable(digits = 4, caption = "Resultados") |>
  kable_styling(full_width = FALSE, bootstrap_options = c("striped", "hover", "condensed"))
```

Table 4: Resultados

	n_trees	mtry	max_depth	accuracy_treino	accuracy	error
15	100	28	7	0.9509	0.9510	0.0490
14	50	28	7	0.9512	0.9501	0.0499
9	100	28	5	0.9473	0.9463	0.0537
8	50	28	5	0.9426	0.9415	0.0585
18	100	150	7	0.9410	0.9396	0.0604
17	50	150	7	0.9426	0.9384	0.0616
2	50	28	3	0.9373	0.9342	0.0658
11	50	150	5	0.9364	0.9334	0.0666
12	100	150	5	0.9364	0.9331	0.0669
10	10	150	5	0.9347	0.9307	0.0693
3	100	28	3	0.9304	0.9302	0.0698
6	100	150	3	0.9275	0.9249	0.0751
5	50	150	3	0.9176	0.9158	0.0842
16	10	150	7	0.9215	0.9131	0.0869
4	10	150	3	0.9139	0.9112	0.0888
7	10	28	5	0.9192	0.9111	0.0889
1	10	28	3	0.9112	0.9104	0.0896
13	10	28	7	0.8978	0.8951	0.1049

A melhor floresta observada no conjunto de teste foi(n_tree = 100, n_variaveis = 28, profundidade = 7), este será o modelo final proposto. As florestas de 50 e 100 arvores não apresentaram diferenças significativas, é interessante de se observar que algumas arvores menores(n_tree = 10) apresentaram acuracia menor que uma arvore apenas.

```r
set.seed(123)
X = mnist[, 1:784]
y = mnist$y

n <- nrow(X)
test_idx <- sample(1:n, size = round(0.3 * n))
train_idx <- setdiff(1:n, test_idx)

X_train <- X[train_idx, ]
y_train <- y[train_idx]
X_test <- X[test_idx, ]
y_test <- y[test_idx]


best_model <- random_forest(
  X = X_test,
  y = y_test,
  n_trees = grid_result$best_params$n_trees,
  mtry = grid_result$best_params$mtry,
  max_depth = grid_result$best_params$max_depth
)


predictions <- predict(best_model,X_test)

conf <- confusionMatrix(predictions, y_test)

kable(conf$table, caption = "Confusion Matrix") %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

Table 5: Confusion Matrix

	0	1	7	8
0	1692	0	24	36
1	1	1989	55	76
7	5	6	1764	19
8	73	29	27	1638

```r
cat("\n### Accuracy\n")
```

```
##
## ### Accuracy
```

```r
conf$overall["Accuracy"] %>% round(4) %>% kable() %>% kable_styling()
```

	x
Accuracy	0.9528

```r
cat("\n### Class-Specific Metrics\n")
```

```
##
## ### Class-Specific Metrics
```

```r
conf$byClass[,c( "Specificity", "Recall")] %>%
  round(4) %>%
  kable() %>%
  kable_styling()
```

	Specificity	Recall
Class: 0	0.9894	0.9554
Class: 1	0.9756	0.9827
Class: 7	0.9946	0.9433
Class: 8	0.9772	0.9259

A floresta escolhida se ajustou aos dados, ela conseguiu garantir uma precisão de Recall de mais 90% para as classes, ou seja, garantiu que a maioria dos dados foram classificados corretamente, ocorreu um confundimento esperado para os valores reais "0", em que em sua maioria foi confundito com a classe "8" e com a classe "7" que foi confundido em sua maior parte com a classe "1" , o mesmo não ocorreu para as demais classes. Os destaque negativos vão para as classes "7" e "8" que tiverem pior desempenho de Recall.

Como o erro no conjunto de teste foi $\approx 5\%$ é esperado que o erro se mantenha proximo disso para novos dados

5. Análise dos Erros

Com base no preditor do item 4:

- Identifique uma observação para cada uma das 16 combinações possíveis de valor verdadeiro e valor predito (por exemplo, verdadeiro = 0, predito = 0; verdadeiro = 0, predito = 1; etc.).
- Visualize essas 16 imagens em um grid 4x4. Adapte o código das funções de visualização para mostrar os 16 casos juntos. Use o facet_grid do ggplot2 para separar os painéis pelas categorias de valor verdadeiro e predição. Comente as observações que foram classificadas incorretamente. Elas são ambíguas? Há algum padrão nos erros?

```r
pred_rf <- predict(best_model, X_test)
```

```r
teste_com_pred <- teste %>%
  mutate(predito = pred_rf, verdadeiro = y)
```

```r
combinacoes <- expand.grid(verdadeiro = levels(teste$y), predito = levels(teste$y))

selecionadas <- combinacoes %>%
  rowwise() %>%
  mutate(indice = which(teste_com_pred$verdadeiro == verdadeiro &
                        teste_com_pred$predito == predito)[1]) %>%
  filter(!is.na(indice))

imagens_selecionadas <- teste_com_pred[selecionadas$indice, ]
```

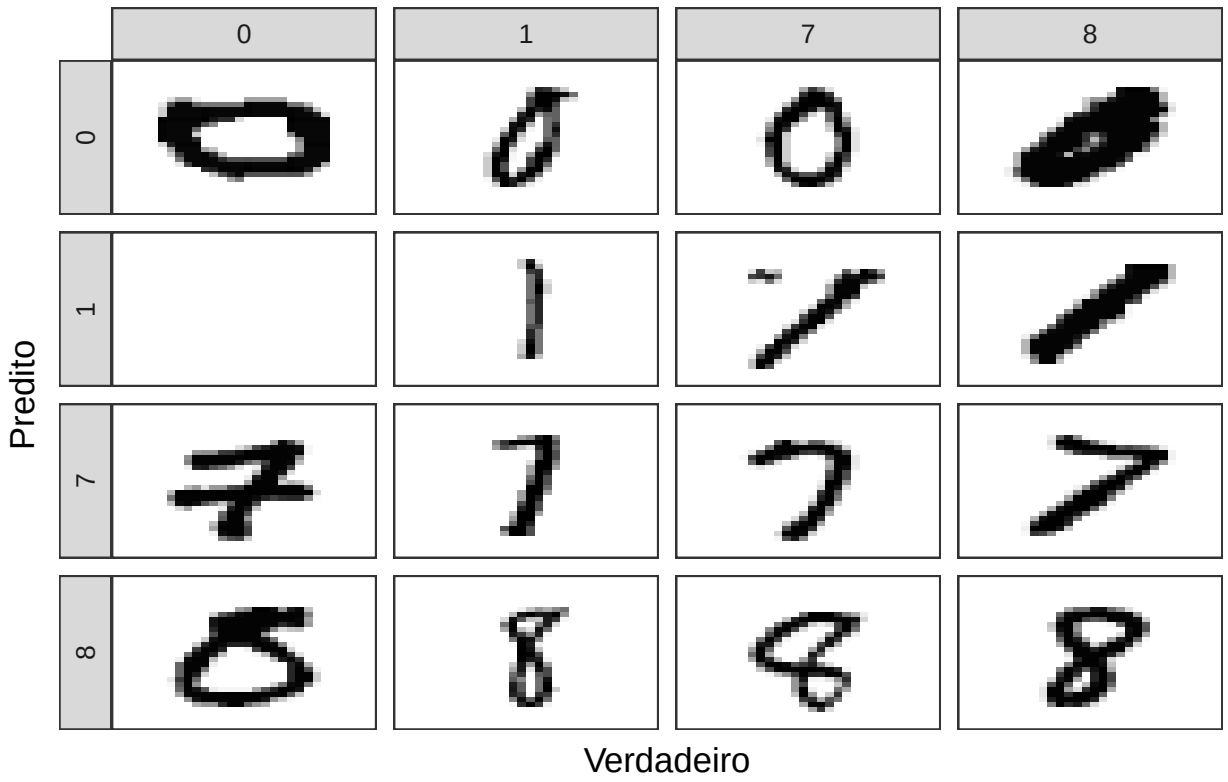
```r
converte_df_info <- function(vetor_covariaveis, verdadeiro, predito){
  vetor_covariaveis <- as.vector(unlist(vetor_covariaveis))
  if(length(vetor_covariaveis) != 784){
    stop("O vetor deve ter 784 valores.")
  }
  pos_x <- rep(1:28, each = 28)
  pos_y <- rep(1:28, times = 28)
  df <- data.frame(pos_x, pos_y, valor = vetor_covariaveis)
  df$verdadeiro <- verdadeiro
  df$predito <- predito
  return(df)
}
```

```r
library(tidyr)
library(dplyr)
library(ggplot2)

df_imagens <- bind_rows(
  lapply(1:nrow(imagens_selecionadas), function(i) {
    linha <- imagens_selecionadas[i, ]
    covariaveis <- select(linha, starts_with("x"))
    converte_df_info(covariaveis, linha$verdadeiro, linha$predito)
  })
)
```

```r
 visnum(df_imagens) +
 facet_grid(rows = vars(verdadeiro), cols = vars(predito), switch = 'y') +
 theme_bw(base_size = 13) +
 labs(x = 'Verdadeiro', y = 'Predito',
 title = 'Situações possíveis') +
 theme(plot.title = element_text(hjust = 0.5),
 axis.ticks = element_blank(),
 axis.text = element_blank(),
 panel.grid = element_blank(),

legend.position= 'none')
```



Análise dos Erros: Verdadeiro vs. Predito As imagens classificadas incorretamente mostram que muitos dos erros ocorrem entre dígitos que são visualmente parecidos ou ambíguos, especialmente quando escritos de forma atípica ou com traços incompletos, entretanto, erros podem ocorrer em razão de sua distribuição espacial, ou até mesmo um confundimento sem motivo aparente.

Observações sobre as classificações incorretas: Verdadeiro: 0 | Predito: 8: O dígito "0" tem uma parte central mais fechada, o que pode tê-lo tornado semelhante ao "8". Esse erro é comum, pois "8" tem duas curvas , enquanto um "0" bem redondo pode ser confundido.

Verdadeiro: 1 | Predito: 7: O "1" com traço na base ou uma pequena inclinação pode se assemelhar ao "7" quando mal escrito.

Verdadeiro: 7 | Predito: 1: Reciprocamente, alguns "7" com haste reta e sem o traço horizontal podem parecer com "1".

Verdadeiro: 0 | Predito: 1: Não houve confundimento entre 0 e 1, por isso o valor está em branco

Nos outros, os erros não são justificados a primeira vista

Dígitos mal escritos ou com baixa intensidade em algumas regiões são mais propensos a serem mal classificados, o que pode ser observado em algumas imagens com partes apagadas ou irregulares.

6. Predição em Novos Dados

Usamos o melhor modelo de floresta aleatória para prever os dígitos no conjunto `MNIST0178-teste.csv` (4117 observações) e salvamos as predições em um arquivo CSV.

```r
set.seed(123)

predicoes <- predict(best_model,  mnist_teste)

resultado <- data.frame(predicao = predicoes)

write_csv(resultado, "predicoes_MNIST0178.csv")
```

O arquivo `predicoes_MNIST0178.csv` contém uma coluna `predicao` com os valores preditos (0, 1, 7, 8) para as 4117 imagens de teste.