Laboratório 2 - Métodos Baseados em Árvores e Florestas Aleatórias

ME905

1. Leitura dos Dados

```
# lendo 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.
```

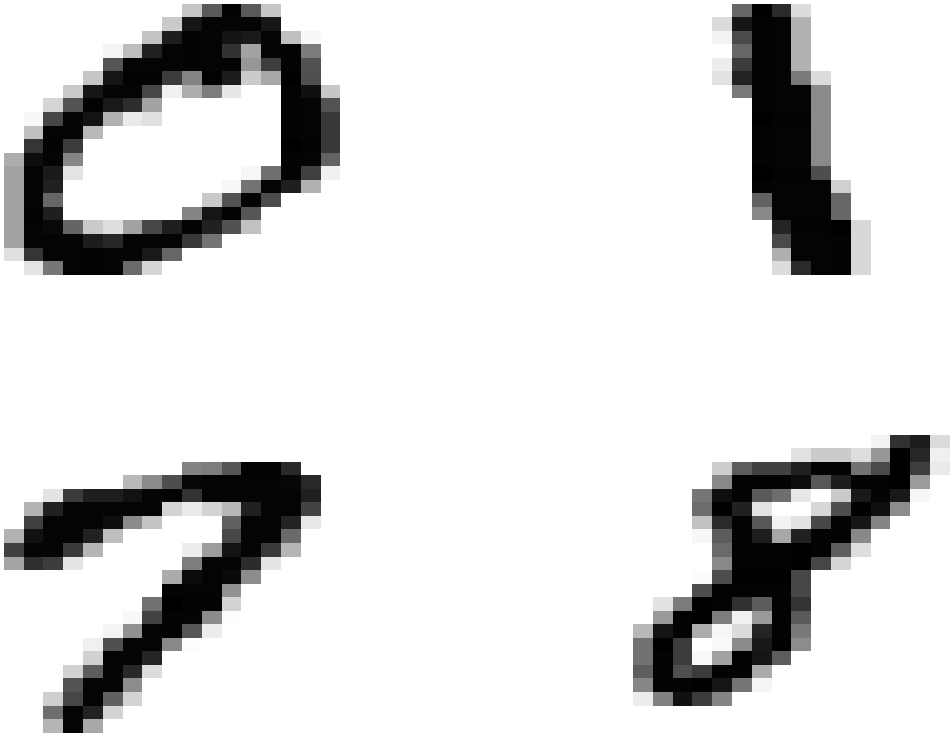
```
mnist$y <- as.factor(mnist$y)
```

2. Visualização de Dígitos

a)

```
# visualizando dados
converte_df <- function(vetor_covariaveis) {
  vetor_covariaveis <- as.vector(unlist(vetor_covariaveis))
  if(length(vetor_covariaveis) != 784){
    stop("Passe um vetor com 784 valores!")
  }

  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() +
    theme(legend.position = 'none')
}

# lendo números
n1 <- converte_df(mnist[1,-1])
```

```
n2 <- converte_df(mnist[3,-1])
n3 <- converte_df(mnist[6,-1])
n4 <- converte_df(mnist[7,-1])

(visnum(n1) + visnum(n2)) / (visnum(n3) + visnum(n4))
```



Vemos os números 0, 1, 7 e 8, respectivamente.

b)

Possívelmente, os números mais difíceis de serem distinguidos são os dígitos 1 e 7 pela sua semelhança e,analogamente, teremos menos dificuldade entre o 0 e o 8 com relação aos demais.

3. Árvore de Classificação com rpart

```
fit <- rpart(y ~ ., data = mnist)
pred <- predict(fit, type = 'class')

table('Predição' = pred, 'Valores Verdadeiros' = mnist$y)
```

```
##         Valores Verdadeiros
## Predição    0    1    7    8
##        0 5526   42   54  166
##        1    1 6118  100  271
##        7  222   79 5824  250
##        8  174  503  287 5164
```

```
sum(pred == mnist$y)/length(mnist$y)
```

```
## [1] 0.9132803
```

Houve uma acurácia de aproximadamente 90%, então a árvore foi eficaz. No entanto, nossa conjectura não se provou verdadeira, veja que o 1 e o 7 foram mais confundidos com o 8 e o 8 foi mais confundido com o 1. Contudo, como nossas suspeitas, o 0 foi mais confundido com o 8.

4. Florestas Aleatórias (com Estratégias Manuais)

```r
# bootstrap ------------------------------------------------------------------
# data banco de dados com a primeira coluna sendo a resposta.

bs <- function(data) {
  return(sample(1:nrow(data), nrow(data), replace = T))
}


# random_forest --------------------------------------------------------------
# função que gera floresta aleatória.
# data banco de dados.
# n_tree número de árvores.
# ... parâmetros para o ajuste da função `rpart.

random_forest <- function(data, n_tree, p = NULL, ...) {
  controle <- rpart.control(...)  # controle de parâmetros do rpart

  # definindo objetos
  n <- nrow(data)  # número de observações
  p_tot <- ncol(data)
  classes <- c('0', '1', '7', '8')
  floresta <- vector(mode = 'list', length = n_tree)  # armazenar as árvores
  if (is.null(p)) p <- round(sqrt(ncol(data)))  # sugestão apresentada em aula
  oob_pred <- data.frame(matrix(NA, nrow=n, ncol=n_tree,  # guarda predição por árvore
                                dimnames = list(NULL, paste0(rep('tree'), 1:n_tree))))

  for (t in 1:n_tree) {
    linhas_sorteadas <- bs(data)
    oob_indices <- setdiff(1:n, unique(linhas_sorteadas)) # linhas não sorteadas

    colunas_selecionadas <- c(1, sample(2:p_tot, p))  # y e preditoras selecionadas
    db_bs <- data[linhas_sorteadas, colunas_selecionadas]

    floresta[[t]] <- rpart(y ~ ., db_bs, control = controle)  # ajustando arvore

    oob_data <- data[oob_indices, colunas_selecionadas]  # df de obs. não usadas
    pred <- predict(floresta[[t]], newdata = oob_data, type = 'class')
    oob_pred[oob_indices,t] <- as.character(pred)
  }

  pred <- apply(oob_pred, 1, \(x) {  # voto da maioria
    table <- table(x)
    if (length(table) == 0) NA else names(which.max(table))
  })

  oob <- mean(pred != data$y, na.rm = T)  # erro oob do modelo
  oob_class_error <- data.frame(          # erro out of bag por classe
```

```
    classe_0 = mean(data$y[which(data$y == 0)] == pred[which(data$y == 0)], na.rm=T),
    classe_1 = mean(data$y[which(data$y == 1)] == pred[which(data$y == 1)], na.rm=T),
    classe_7 = mean(data$y[which(data$y == 7)] == pred[which(data$y == 7)], na.rm=T),
    classe_8 = mean(data$y[which(data$y == 8)] == pred[which(data$y == 8)], na.rm=T)
  )

  return(list(
    forest = floresta,
    oob_class_error = oob_class_error,
    oob_accuracy = 1 - oob
  ))
}
```

Com a função definida vamos ajustar 6 modelos variando `n_tree`, `p` e `maxdepth`.

```
# ajsutando modelos
modelos <- list(
  modelo1 = random_forest(mnist, n_tree = 5 , p = 10, maxdepth = 5 ),
  modelo2 = random_forest(mnist, n_tree = 5 , p = 10, maxdepth = 20),
  modelo3 = random_forest(mnist, n_tree = 10, p = 28, maxdepth = 5 ),
  modelo4 = random_forest(mnist, n_tree = 10, p = 28, maxdepth = 20),
  modelo5 = random_forest(mnist, n_tree = 15, p = 50, maxdepth = 5 ),
  modelo6 = random_forest(mnist, n_tree = 15, p = 50, maxdepth = 20),
  modelo7 = random_forest(mnist, n_tree = 50, p = 50, maxdepth = 5 ),
  modelo8 = random_forest(mnist, n_tree = 50, p = 50, maxdepth = 20)
)

# resultados
sapply(modelos, \(x)
  round(unlist(c(oob_accuracy = x$oob_accuracy, x$oob_class_error)), 4),
  simplify='matrix') |>
  knitr::kable()
```

	modelo1	modelo2	modelo3	modelo4	modelo5	modelo6	modelo7	modelo8
oob_accuracy	0.7013	0.6777	0.8526	0.8574	0.9079	0.9118	0.9459	0.9419
classe_0	0.7510	0.8209	0.9332	0.9286	0.9601	0.9521	0.9720	0.9624
classe_1	0.8304	0.9656	0.9582	0.9603	0.9448	0.9510	0.9709	0.9608
classe_7	0.5733	0.5288	0.8503	0.7963	0.8905	0.8717	0.9379	0.9216
classe_8	0.6399	0.3585	0.6519	0.7320	0.8312	0.8689	0.8992	0.9210

A tabela apresentada contém as informações da acurácia geral do modelo (primeira linha) e para cada dígito. Além disso, note que os melhores modelos foram o modelo 5 e 6, justamente os que apresentam o maior número de árvores e preditoras por árvore. Além disso, note como o modelo 1 e 2 foram excepcionalmente ruins e são justamente os que apresentam o menor número de árvores e preditoras. Então, os parâmetros que mais parecem importar são: número de árvores (`n_tree`) e quantidade de preditoras no treino (`p`).

Uma boa medida para a acurácia do modelo é a acurácia out-of-bag (primeira linha da tabela), ou seja, verifica-se o desempenho do modelo analisando a classificação das árvores para observações não utilziadas em seu treino.

Além disso, também testamos ooutros parâmetros, como `minbucket`, mas paraceu não surtir efeito no modelo para valores abaixo de 50, talvez pelo número de observações ser elevado; e `cp`, o qual, para valores grandes, como 1 e 0.5, apresentou um desempenho pior do que quando comparado com valores pequenos, como 0.01, 0.001 e 0.0001.

5. Análise dos Erros

```
# predict.forest ------------------------------------------------------------
# forest precisa ser um objeto da saída da função `random_forest`.
# data banco de dados para predição.

predict.forest <- function(forest, data) {
  pred_tree <- vector(mode='list', length(forest$forest))

  for (i in seq_along(forest$forest)) {
    pred_tree[[i]] <- unname(predict(forest$forest[[i]], type = 'class',
                                     newdata = data))
  }
  voto <- data.frame(tree = unname(do.call(cbind.data.frame, pred_tree)))
  apply(voto, 1, \(x) names(which.max(table(unlist(x)))))
}

# fazendo predição e modificando `converte_df` --------------------------------
pred_forest <- predict.forest(modelo1, mnist)
mnist_com_pred <- cbind('pred' = pred_forest, mnist)  # juntando dados com predição

# combinando casos
combinacoes <- data.frame()

for (y in c(0,1,7,8)) {
  for (pred in c(0,1,7,8)) {
    combinacoes <- bind_rows(combinacoes, data.frame(pred = pred,
            mnist[mnist_com_pred$y == y & mnist_com_pred$pred == pred,][1,]))
  }
}

# modificando converte_df
converte_df_mod <- function(data) {
  pos_x <- rep(1:28, each = 28)
  pos_y <- rep(1:28, times = 28)
  resultado <- data.frame()

  for (i in 1:16) {
    pred <- data$pred[i]
    y <- data$y[i]
    vetor_covariaveis <- as.vector(unlist(data[i,-c(1,2)]))
    resultado <- bind_rows(resultado, bind_cols(pred = pred, y = y,
                    data.frame(pos_x, pos_y, valor = vetor_covariaveis)))
  }

  return(resultado)
}

erro_long <- converte_df_mod(combinacoes)

# plot
visnum(erro_long) +
  facet_grid(rows = vars(pred), cols = vars(y), switch = 'y') +
  theme_bw(base_size = 15) +
```

```
  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')
```

6. Predição em Novos Dados

```
mnist_teste <- read.csv('MNIST0178-teste.csv')
```