Universidade de Brasília

DEPARTAMENTO DE ESTATÍSTICA

18 de janeiro de 2023

Resolução - Marcos Augusto D. Barbosa (220006024)

Lista 3: Manipulação e modelagem de dados com Spark

Computação em Estatística para dados e cálculos massivos

Tópicos especiais em Estatística 2

Prof. Guilherme Rodrigues

César Augusto Fernandes Galvão (aluno colaborador)

Gabriel Jose dos Reis Carvalho (aluno colaborador)

1. **As questões deverão ser respondidas em um único relatório *PDF* ou *html*, produzido usando as funcionalidades do *Rmarkdown* ou outra ferramenta equivalente.**
2. **O aluno poderá consultar materiais relevantes disponíveis na internet, tais como livros, *blogs* e artigos.**
3. **O trabalho é individual. Suspeitas de plágio e compartilhamento de soluções serão tratadas com rigor.**
4. **Os códigos *R* utilizados devem ser disponibilizados na integra, seja no corpo do texto ou como anexo.**
5. **O aluno deverá enviar o trabalho até a data especificada na plataforma Microsoft Teams.**
6. **O trabalho será avaliado considerando o nível de qualidade do relatório, o que inclui a precisão das respostas, a pertinência das soluções encontradas, a formatação adotada, dentre outros aspectos correlatos.**
7. **Escreva seu código com esmero, evitando operações redundantes, visando eficiência computacional, otimizando o uso de memória, comentando os resultados e usando as melhores práticas em programação.**

Questão 1: Criando o cluster spark.

a) Crie uma pasta (chamada datasus) em seu computador e faça o download dos arquivos referentes ao Sistema de informação de Nascidos Vivos (SINASC), os quais estão disponíveis em https://datasus.saude.gov.br/transferencia-de-arquivos/.

Atenção: Considere apenas os Nascidos Vivos no Brasil (sigla DN) entre 1996 e 2020, incluindo os dados estaduais e excluindo os arquivos referentes ao Brasil (sigla BR). Use wi-fi para fazer os downloads!

Dica: O endereço ftp://ftp.datasus.gov.br/dissemin/publicos/SINASC/1996_/Dados/DNRES/ permite a imediata identificação dos endereços e arquivos a serem baixados.

Solução:

Primeiro, importamos todos os pacotes necessários.

```
if (!require("pacman")) install.packages("pacman")
p_load(data.table,
       dplyr,
       geobr,
       sparklyr,
       purrr,
       stringr,
       lubridate,
       arrow,
       read.dbc,
       microbenchmark)
```

Segundo, criamos a pasta datasus/, na qual iremos salvar vários arquivos .dbc, por meio de um processo iterativo da unidade federativa e do ano.

```
dir.create('datasus')

years = 1996:2020 %>% as.character()
states = read_state() %>%
  as_tibble() %>%
  select(abbrev_state) %>%
  as.vector() %>%
  .[[1]]

origin = 'ftp://ftp.datasus.gov.br/dissemin/publicos/SINASC/'

for (i in seq_along(states)){

  for (j in seq_along(years)){
    download_link = paste0(origin, "1996_/Dados/DNRES/DN",
                           states[i], years[j], '.dbc')

    download.file(url=download_link,
                  destfile=paste0('datasus/', states[i], years[j], '.dbc'),
                  mode = 'wb')

  }

}
```

b) Usando a função `p_load` (do pacote `pacman`), carregue os pacotes `arrow` e `read.dbc` e converta os arquivos baixados no item a) para formato o *.parquet*. Em seguida, converta para *.csv* apenas os arquivos referentes aos estados GO, MS e ES. Considerando apenas os referidos estados, compare o tamanho ocupado pelos arquivos nos formatos *.parquet* e *.csv* (use a função `file.size`).

Solução:

Primeiro, armazenamos todos os nomes dos arquivos da pasta datasus/ e armazenamos também, em um outro vetor, apenas os arquivos dos estados de GO, MS e ES.

```
dbc_files = list.files(path='datasus/',
                       pattern='.dbc',
                       full.names=TRUE)
dbc_ESGOMS_files = list.files(path='datasus/',
                              pattern="GO|MS|ES",
                              full.names=FALSE)
```

Segundo, criamos uma função que lê o arquivo .dbc, seleciona colunas se é necessário, converte todas as colunas para tipo character()-essa conversão é crucial, para evitar problemas na hora de ler múltiplos arquivos que eventualmente possuam colunas de diferentes tipoS, e.g, em um arquivo a coluna de sexo possui os valores textuais 'M' ou 'F' e em outro, os valores numéricos 0 ou 1. No final da função, é salvo um arquivo transformado em .csv em uma pasta de destino desejada.

Após a definição da função, esta é executada para todos os arquivos dos estados de ES, GO e MS.

```
dbc_to_csv = function(file, currfolder, destfolder, cols){

  path = paste0(currfolder, file)

  if (missing(cols)){
    df = read.dbc(path)
  }

  else {
    df = read.dbc(path) %>% select(all_of(cols))
  }

  df = df %>% mutate_all(as.character)
  csv_path = paste0(destfolder, gsub('.dbc', '.csv', file))
  write.csv(df, file = csv_path)
}

dir.create('ESGOMS_csv/')
walk(.x=dbc_ESGOMS_files,
     .y=dbc_to_csv,
     currfolder='datasus/',
     destfolder='ESGOMS_csv/')
```

Terceiro, criamos uma função que lê o arquivo .dbc, seleciona colunas se é necessário, converte todas as colunas para tipo character(). No final da função, é salvo um arquivo transformado em .parquet em uma pasta de destino desejada.

Após a definição da função, esta é executada para todos os arquivos dos estados de ES, GO e MS.

```
dbc_to_parquet = function(file, currfolder, destfolder, cols){

  path = paste0(currfolder, file)

  if (missing(cols)){
    df = read.dbc(path)
  }

  else {
    df = read.dbc(path) %>% select(all_of(cols))
```

```
  }

  df = df %>% mutate_all(as.character)
  parquet_path = paste0(destfolder, gsub('.dbc', '.parquet', file))
  write_parquet(df, sink = parquet_path)
}
dir.create('ESGOMS_parquet')
walk(.x=dbc_ESGOMS_files,
     .y=dbc_to_parquet,
     currfolder='datasus/',
     destfolder='ESGOMS_parquet/')
```

```
parquet_ESGOMS_files = list.files(path='ESGOMS_parquet/', full.names=TRUE)
csv_ESGOMS_files = list.files(path='ESGOMS_csv/', full.names=TRUE)

sum(sapply(parquet_ESGOMS_files, file.size))/10**6
```

```
## [1] 36.66042
```

```
sum(sapply(csv_ESGOMS_files, file.size))/10**6
```

```
## [1] 516.663
```

Logo, nota-se que a pasta com os arquivos .parquet possui um tamanho bastante pequeno quando comparada com a pasta dos arquivos .dbc.

c) Crie uma conexão `Spark`, carregue para ele os dados em formato *.parquet* e *.csv* e compare os respectivos tempos computacionais. Se desejar, importe apenas as colunas necessárias para realizar a Questão 2.

OBS: Lembre-se de que quando indicamos uma pasta na conexão, as colunas escolhidas para a análise precisam existir em todos os arquivos.

Solução:

Primeiro, armazenamos os nomes das colunas presentes no ano de 1996, mas retiramos duas colunas, pois são problemática e não possuem muita relevância. Em seguida, nas respectivas pastas dos estados ES, GO e MS, salvamos novamente arquivos .csv e .parquet, mas agora selecionamos colunas específicas.

```
cols = read.dbc('datasus/ES1996.dbc') %>%
  select(-c('contador', 'CODOCUPMAE')) %>%
  colnames()

walk(.x=dbc_ESGOMS_files,
     .y=dbc_to_csv,
     currfolder='datasus/',
     destfolder='ESGOMS_csv/',
     cols=cols)
walk(.x=dbc_ESGOMS_files,
     .y=dbc_to_parquet,
     currfolder='datasus/',
     destfolder='ESGOMS_parquet/',
     cols=cols)
```

Segundo, criamos a conexão Spark

```
config = spark_config()
config$spark.executor.cores = 4
config$spark.executor.memory = "8G"
sc = spark_connect(master="local", config=config)
spark_version(sc)
```

```
## [1] '3.3.1'
```

Terceiro, avaliamos os tempos computacionais de leitura das duas estratégias.

```
microbenchmark(
  parquet_option = spark_read_parquet(sc=sc,
                                      path='ESGOMS_parquet/',
                                      header=TRUE,
                                      memory=FALSE),
  csv_option = spark_read_csv(sc=sc,
                              name='sinasc',
                              path='ESGOMS_csv/',
                              header=TRUE,
                              delimiter=',',
                              charset='latin1',
                              infer_schema=FALSE,
                              memory=FALSE),
  times=5)
```

```
## Unit: milliseconds
##            expr       min        lq      mean    median        uq       max
##  parquet_option  405.3589  442.1613  589.3819  459.5523  480.5221  1159.315
##      csv_option 1057.0976 1113.2674 3016.0491 1124.6409 1181.4870 10603.753
##  neval
##      5
##      5
```

Questão 2: Preparando e modelando os dados.

Atenção: **Elabore seus comandos dando preferência as funcionalidades do pacote sparklyr**.

a) Faça uma breve análise exploratória dos dados (tabelas e gráficos) com base somente nas colunas existente nos arquivos de 1996. O dicionário das variaveis encontra-se no mesmo site do item a), na parte de documentação. Corrija eventuais erros encontrados; por exemplo, na variavel sexo são apresentados rótulos distintos para um mesmo significado.

Solução:

Primeiro, convertemos, selecionamos colunas de todos os arquivos .dbc e salvamos em .parquet em uma pasta de destino específica.

```
dir.create('filtered_parquet')

dbc_all_files = list.files(path='datasus/')
walk(.x=dbc_all_files,
     .y=dbc_to_parquet,
     currfolder='datasus/',
     destfolder='filtered_parquet/',
     cols=cols)
```

Segundo, lemos, por Spark, os dados da pasta.

```
spark_read_parquet(sc=sc,
                   name='sinasc',
                   path='filtered_parquet/',
                   header=TRUE,
                   memory=FALSE)
```

```
## # Source: spark<sinasc> [?? x 19]
##    LOCNASC CODMUNNASC IDADEMAE ESTCIVMAE ESCMAE QTDFIL~1 QTDFI~2 CODMU~3 GESTA~4
##    <chr>   <chr>      <chr>    <chr>     <chr>  <chr>    <chr>   <chr>   <chr>
##  1 1       1100205    15       5         2      <NA>     <NA>    3548906 4
##  2 1       1100205    22       2         3      01       <NA>    3524501 4
##  3 1       1600303    20       <NA>      <NA>   <NA>     <NA>    3510500 5
##  4 1       1716109    13       1         <NA>   <NA>     <NA>    3535705 5
##  5 1       1708205    22       5         4      00       00      3550308 5
##  6 1       1721000    <NA>     1         2      01       02      3535705 5
##  7 1       1702109    25       2         5      01       00      3516200 5
##  8 1       1702109    23       5         2      02       00      3542800 5
##  9 1       1702109    26       <NA>      <NA>   01       00      3543907 5
## 10 1       1709500    20       <NA>      <NA>   01       <NA>    3557105 5
## # ... with more rows, 10 more variables: GRAVIDEZ <chr>, PARTO <chr>,
## #   CONSULTAS <chr>, DTNASC <chr>, SEXO <chr>, APGAR1 <chr>, APGAR5 <chr>,
## #   RACACOR <chr>, PESO <chr>, CODANOMAL <chr>, and abbreviated variable names
## #   1: QTDFILVIVO, 2: QTDFILMORT, 3: CODMUNRES, 4: GESTACAO
```

Terceiro, fazemos algumas transformações e corrigimos algumas colunas que tinham inconsistências, que foram percebidas, após uma breve análise exploratória que não será exposta aqui.

```
NUMERIC_COLUMNS = c('IDADEMAE','QTDFILVIVO','QTDFILMORT',
                    'PESO', 'PARTO', 'CONSULTAS')

sinasc = tbl(sc, "sinasc") %>%
  mutate_at(NUMERIC_COLUMNS, as.double) %>%
  mutate(SEXO = case_when(SEXO=='M' ~ '1',
                          SEXO=='F' ~ '2',
                          SEXO=='I' ~ '0',
                          SEXO=='9' ~ '0',
                          TRUE ~ SEXO),
         RACACOR = case_when(RACACOR=='9' ~ NA,
                             RACACOR=='0' ~ NA,
                             TRUE ~ RACACOR),
         PARTO = case_when(PARTO==1 ~ 0,
                           PARTO==2 ~ 1,
                           TRUE ~ NA),
         QTDFILVIVO = na_if(QTDFILVIVO, 99),
         QTDFILMORT = na_if(QTDFILMORT, 99),
         APGAR1 = na_if(APGAR1, 99),
         APGAR5 = na_if(APGAR5, 99),
         DTNASC = to_date(DTNASC, "ddMMyyyy"),
         DAYWEEK = date_format(DTNASC, "E"),
         PESO = na_if(PESO, 9999),
  ) %>%
  filter(LOCNASC!='5', IDADEMAE>0, PESO>0, !is.na(PARTO), !is.na(CONSULTAS))
```

b) Ultilizando as funções do **sparklyr**, preencha os dados faltantes na idade da mãe com base na mediana. Se necessário, faça imputação de dados também nas demais váriaveis.

Solução:

Iremos, a partir deste item, fazer o uso de pipeline. Neste item específico, fazemos imputação por mediana das variáveis numéricas.

```
pipeline =  ml_pipeline(sc) %>%
  ft_imputer(input_col='IDADEMAE', output_col='IDADEMAE_IMPUTED',
             strategy='median') %>%
  ft_imputer(input_col='PESO', output_col='PESO_IMPUTED',
             strategy='median') %>%
  ft_imputer(input_col='QTDFILVIVO', output_col='QTDFILVIVO_IMPUTED',
             strategy='median') %>%
  ft_imputer(input_col='QTDFILMORT', output_col='QTDFILMORT_IMPUTED',
             strategy='median')
```

c) Novamente, utilizando as funções do **sparklyr**, normalize (retire a média e divida pelo desvio padrão) as variáveis quantitativas do banco.

Solução:

Fazemos a normalização das variáveis numéricas. No final, todas as 4 variáveis numéricas consideradas ficam dentro de uma única nova coluna denominada numerical_features_scaled.

```
pipeline = pipeline %>%
  ft_vector_assembler(
    input_col = c('IDADEMAE_IMPUTED', 'PESO_IMPUTED',
                  'QTDFILVIVO_IMPUTED', 'QTDFILMORT_IMPUTED'),
    output_col = "numerical_features"
  ) %>%
  ft_standard_scaler(input_col="numerical_features",
                     output_col="numerical_features_scaled",
                     with_mean = TRUE)
```

d) Crie variáveis dummy (*one-hot-encoding*) que conjuntamente indiquem o dia da semana do nascimento (SEG, TER, ...). Em seguida, *binarize* o número de consultas pré-natais de modo que "0" represente "até 5 consultas" e "1" indique "6 ou mais consultas". (Utilize as funções **ft_**)

Solução:

Mais transformações e, no final, todas as features que serão consideradas no ajuste do modelo ficam dentro de uma única nova coluna denominada final_features.

```
pipeline = pipeline %>%
  ft_string_indexer(input_col = 'DAYWEEK', output_col='DAYWEEK_indexed') %>%
  ft_one_hot_encoder(
    input_cols = c('DAYWEEK_indexed'),
    output_cols = c('DAYWEEK_encoded')
  ) %>%
  ft_binarizer(input_col = "CONSULTAS",
               output_col = "CONSULTAS_binarized",
               threshold = 3.9) %>%
  ft_vector_assembler(
    input_col = c('IDADEMAE_IMPUTED', 'PESO_IMPUTED',
                  'QTDFILVIVO_IMPUTED', 'QTDFILMORT_IMPUTED',
                  'DAYWEEK_encoded', 'CONSULTAS_binarized',
                  'numerical_features_scaled'),
    output_col = "final_features"
  )
```

e) Particione os dados aleatoriamente em bases de treinamento e teste. Ajuste, sobre a base de treinamento, um modelo de regressão logistica em que a variável resposta (y), indica se o parto foi ou não

cesáreo. Analise o desempenho preditivo do modelo com base na matrix de confusão obtida no conjunto de teste.

Solução:

Primeiro, particionamos o conjunto de dados em treino e em teste. Após a operação, fazemos o cache de cada um.

```
partition_sample = sinasc %>%
  sdf_random_split(training=0.80, test=0.20, seed=47)

# register the resulting Spark SQL in Spark
sdf_register(partition_sample$test, "test")
```

```
## # Source: spark<test> [?? x 20]
##    LOCNASC CODMUNNASC IDADEMAE ESTCIVMAE ESCMAE QTDFIL~1 QTDFI~2 CODMU~3 GESTA~4
##    <chr>   <chr>         <dbl> <chr>     <chr>     <dbl>   <dbl> <chr>   <chr>
##  1 1       110002           31 2         5             1       0 352590  5
##  2 1       1100106          24 1         4             1      NA 3532801 5
##  3 1       1100122          24 2         3             0       0 3550209 5
##  4 1       1100122          28 <NA>      6             5       0 3507506 8
##  5 1       1100189          26 <NA>      6             2       0 3518800 5
##  6 1       110020           19 1         4             0       0 353440  5
##  7 1       1100205          15 5         2            NA      NA 3548906 4
##  8 1       1100304          27 1         5             1       0 3525904 5
##  9 1       1100601          21 2         <NA>         NA      NA 3512001 5
## 10 1       1200401          15 5         2            NA      NA 3509502 5
## # ... with more rows, 11 more variables: GRAVIDEZ <chr>, PARTO <dbl>,
## #   CONSULTAS <dbl>, DTNASC <date>, SEXO <chr>, APGAR1 <chr>, APGAR5 <chr>,
## #   RACACOR <chr>, PESO <dbl>, CODANOMAL <chr>, DAYWEEK <chr>, and abbreviated
## #   variable names 1: QTDFILVIVO, 2: QTDFILMORT, 3: CODMUNRES, 4: GESTACAO
```

```
tbl_cache(sc, "test")
```

```
sdf_register(partition_sample$train, "train")
tbl_cache(sc, "train")
```

Este é o pipeline final. O ajuste do modelo, por meio de ml_fit() já foi executado e salvo previamente, pois exige muito esforço computacional.

```
pipeline = pipeline %>%
  ml_logistic_regression(features_col="final_features",
                         label_col="PARTO")

pipeline_model = ml_fit(pipeline, tbl(sc, 'train'))
#ml_save(x=pipeline_model, path='sinasc_model', overwrite=FALSE)
```

Sendo assim, carregamos o modelo já treinado.

```
reloaded_model = ml_load(sc, "sinasc_model")
```

Finalmente, computamos a matriz de confusão, no conjunto de dados de teste.

```
predictions = ml_transform(x=reloaded_model, dataset=tbl(sc, 'test'))
predictions %>% count(PARTO, prediction)
```

```
## # Source: spark<?> [?? x 3]
## # Groups: PARTO
##   PARTO prediction       n
##   <dbl>      <dbl>   <int>
## 1     1          0 2664954
## 2     0          1 2301010
## 3     0          0 5223097
## 4     1          1 4288394
```

É obtida uma acurácia de, aproximadamente, 66%.

```
spark_disconnect(sc)
```