Universidade de Brasília

DEPARTAMENTO DE ESTATÍSTICA

02 de agosto de 2022

Lista 3: Manipulação e modelagem de dados com Spark

Computação em Estatística para dados e cálculos massivos

Tópicos especiais em Estatística 1

Prof. Guilherme Rodrigues

César Augusto Fernandes Galvão (aluno colaborador)

Gabriel Jose dos Reis Carvalho (aluno colaborador)

José Vítor Barreto Porfírio 190089971 (**autor das soluções**)

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 1994 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: Utilizando o endereço do servidor `ftp` disponibilizado e também identificável por inspeção da página do Datasus, uma estratégia viável é fazer a requisição ao servidor e extrair os arquivos referentes a cada estado e ano por meio de um *loop* em conjunto com a função `urlretrieve` da biblioteca nativa `urllib` do Python, assim como no bloco de código que se segue

```
from urllib.request import urlretrieve
import os


def main():
    estados = [
        "AC", "AL", "AM", "AP", "BA", "CE",
        "DF", "ES", "GO", "MA", "MG", "MS",
        "MT", "PA", "PB", "PE", "PI", "PR",
        "RJ", "RN", "RO", "RR", "RS", "SC",
        "SE", "SP", "TO"
    ]
    anos = range(1996, 2021)

    db = "ftp://ftp.datasus.gov.br/dissemin/publicos/SINASC/1996_/Dados/DNRES"

    path = "./Lista_3/datasus"
    if not os.path.exists(path):
        os.mkdir(path)

    for estado in estados:
        print(estado)
        for ano in anos:
            print(ano)
            file = f"DN{estado}{ano}.dbc"
            urlretrieve(f"{db}/{file}", f"{path}/{file}")


if __name__ == "__main__":
    main()
```

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: O pacote `arrow` dispõe das funções `write_parquet` e `write_csv_arrow` para escrever arquivos `.parquet` e `.csv`, respectivamente. A estratégia escolhida foi ler os bancos baixados com o pacote `read.dbc` e em seguida fazer a "conversão" para os formatos desejados. Como há muitos arquivos, o procedimento é um pouco demorado e optou-se por agilizar as coisas com o pacote `furrr`

```r
pacman::p_load(
  "arrow", "read.dbc", "glue",
  "dplyr", "stringr", "furrr"
)


conversor_dbc <- function(files, origin_path, oformat) {
  "Recebe os arquivos dbc, seu caminho de origem e o formato de saída
    e abre um plano de multisessões do future para fazer a conversão"
  future::plan(future::multisession, workers = 7)
  if (oformat == "parquet") convert <- write_parquet
  if (oformat == "csv") convert <- write_csv_arrow

  # Gambiarra do future
  origin_path <- origin_path

  future_walk(
    files,
    ~ read.dbc(glue("{origin_path}/{.x}")) %>%
      convert(sink = glue(
        "./Lista_3/{oformat}s/{
                    stringr::str_replace(.x, 'dbc', oformat)
                }"
      ))
  )
}
```

Primeiro seguiu-se convertendo apenas os arquivos relativos ao estados GO, ES e MS, não só no caso dos `.csvs`, mas também para `.parquet`

```r
path <- "./Lista_3/datasus"
files <- list.files(path)
GO_ES_MS <- str_detect(files, "GO|MS|ES")

conversor_dbc(
  files = files[GO_ES_MS],
  origin_path = path,
  oformat = "parquet"
)
conversor_dbc(
  files = files[GO_ES_MS],
  origin_path = path,
  oformat = "csv"
)

parquets <- list.files("./Lista_3/parquets/", full.names = TRUE)
csvs <- list.files("./Lista_3/csvs/", full.names = TRUE)
file.size(parquets) %>% sum() / 1024^2 #> 103,31 Mb
file.size(csvs) %>% sum() / 1024^2 #> 968,21 Mb
```

concluindo o raciocínio que os aquivos em `.parquet` estão ocupando cerca de 100 *Megabytes*, por volta de 9 vezes menos memória que os arquivos em `.csv`, ocupando na casa dos 900 *Megabytes.

Para converter o restante dos arquivos para `.parquet` basta tomar a conversão dos arquivos complementares tal qual no próximo bloco de código, todavia, isso só será feito após as comparações de tempo computacional do item **c)** para evitar complexidades a mais

```
conversor_dbc(
  files = files[!GO_ES_MS],
  origin_path = path,
  oformat = "parquet"
)
```

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: Utilizando as funções `spark_read_parquet` e `spark_read_csv` é possível ler os arquivos dispostos nos respectivos formatos, é possível até mesmo informar uma pasta de arquivos para serem lidos (com ou sem ajuda de um *wildcard* "*" no caminho). Dessa forma obteve-se a tabela 1, da qual é percepitível uma vitória avassaladora dos arquivos `.parquet`, cerca de 10 vezes mais rápido.

```
sc <- spark_connect(master = "local", version = "2.4.3")

microbenchmark(
  "parquet" = spark_read_parquet(sc, path = "Lista_3/parquets/*"),
  "csv" = spark_read_csv(sc, path = "Lista_3/csvs/*"),
  times = 3L, unit = "s"
)
```

Tabela 1: Comparação entre o tempo coputacional, em segundos, para ler os aquivos .parquet e .csv do SINASC, nos estados ES, GO e MS

Solução	Mínimo	Média	Máximo
parquet	0.6	0.7	0.8
csv	4.9	5.1	5.3

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: O primeiro passo é carregar todos arquivos para o Spark, em seguida as variáveis foram transformadas para receberem seus valores como variáveis numéricas para possibilitar as análises subsequentes, especificamente a data de nascimento foi transformada pelas funções do Hive (uma dependência do Spark) para o dia da semana códificado de forma que a segunda-feira é representada por "1", terça-feira por "2" e assim sucessivamente.

```
sinasc <- spark_read_parquet(
  sc,
  path = "Lista_3/parquets/*"
) %>%
  select(-c(
    contador, LOCNASC, CODMUNNASC,
```

```
    CODMUNRES, APGAR1, APGAR5,
    RACACOR, CODANOMAL
  )) %>%
  mutate(DTNASC = from_unixtime(unix_timestamp(DTNASC, "ddMMyyyy"), "u")) %>%
  mutate(CODOCUPMAE = as.numeric(CODOCUPMAE)) %>%
  mutate(CONSULTAS = as.numeric(CONSULTAS)) %>%
  mutate(DTNASC = as.numeric(DTNASC)) %>%
  mutate(ESCMAE = as.numeric(ESCMAE)) %>%
  mutate(ESTCIVMAE = as.numeric(ESTCIVMAE)) %>%
  mutate(GESTACAO = as.numeric(GESTACAO)) %>%
  mutate(GRAVIDEZ = as.numeric(GRAVIDEZ)) %>%
  mutate(IDADEMAE = as.numeric(IDADEMAE)) %>%
  mutate(PARTO = as.numeric(PARTO)) %>%
  mutate(PESO = as.numeric(PESO)) %>%
  mutate(QTDFILMORT = as.numeric(QTDFILMORT)) %>%
  mutate(QTDFILVIVO = as.numeric(QTDFILVIVO)) %>%
  mutate(SEXO = as.numeric(SEXO))
```

As funções `sdf_dim` e `glimpse` permitem começar a análise exploratória dando uma olhada nas dimensões da tabela carregada e também em como as variáveis foram guardadas, como o *output* da função `glimpse` é muito grande, recomenda-se a consulta ao arquivo pdf destinado à documentação no próprio site do Datasus. Em particular, os dados são armazenados de forma que as variáveis qualitativas são indicadas por números inteiros de 1 em diante indo no máximo até o 9 (que representa "não informado").

```
sdf_dim(sinasc)
glimpse(sinasc)
```

A função `sdf_describe` do `sparklyr` permite a extração imediata de medidas descritivas a partir das colunas do banco de dados, logo, seguiu-se tomando as variáveis quantitativas de interesse como no seguitne bloco de código e obtendo as medidas apresentadas na tabela 2

```
sinasc %>%
  select(IDADEMAE, QTDFILVIVO, QTDFILMORT) %>%
  sdf_describe()
```

Da tabela 2 já é possível perceber uma discrepância na qualidade dos dados informados em cada variável, o número de observações válidas (não `NA` e numérica) para a idade da mãe evidencia que há uma melhor coleta dessa informação em comparação com o número de filhos mortos, portanto decidiu-se estudar o número de observações faltantes dessa e das demais variáveis e descartar das análises aquelas que possuíssem mais de 10% de dados faltantes, pela falta de qualidade. Ainda sobre a qualidade dos dados, nota-se que estranhamente que há mulheres dando a luz aos 99 anos de idade, o que pode indicar que houve um equívoco nos registros.

Tabela 2: Medidas descritivas das variáveis quantitivas do SINASC.

Medida	Idade da Mãe	Nº de Filhos vivos	Nº de Filhos mortos	Peso
Dados válidos (Milhões)	74,22	66,94	60,89	71,11
Média	25.68	1.88	1.59	3193.11
Desvio-padrão	7.24	8.28	11.71	601.37
Mínimo	0.00	0.00	0.00	0.00
Máximo	99.00	99.00	99.00	9999.00

Seguindo com a análise sobre dados faltantes, seguiu-se escolhendo remover as variáveis Estado Civil da Mãe, Código de ocupação da Mãe e Nº de Filhos vivos, uma vez que essas variáveis apresentaram baixa qualidade dos dados, como indicado pela tabela 3.

```
### Pega a frequência relativa de NAs em cada variável
### e decide sobre manter ou não com base em um threshold de 10%
linhas <- sdf_dim(sinasc)[1]
faltante <- sinasc %>%
  summarise_all(~ sum(as.integer(is.na(.)) / linhas)) %>%
  collect()

sinasc <- sinasc %>%
  select(-colnames(faltante[which(faltante > 0.10)]))
```

Tabela 3: Frequência relativa do número de observações faltantes por variável do SINASC.

Idade da Mãe	Estado Civil da Mãe	Escolaridade da Mãe	Código de ocupação da Mãe
0,2%	15.0%	2.5%	33.4%
Nº de Filhos vivos	Nº de Filhos mortos	Gestação	Gravidez
10.0%	18.1%	2.2%	1.02%
Consultas Pré-natais	Data de Nascimento	Sexo	Peso
1.9%	0.8%	4.0%	0.4%

Visando escolher as variáveis mais tarde se adequam ao propósito de fazer um bom ajuste para previsão do tipo de parto, é importante analisar quais variáveis aparentam ter algum efeito sobre a decisão do tipo de parto, para tanto seguiu-se com uma função customizada que agrupa os dados por cada variável explicativa conjuntamente com o tipo de parto e retorna a frequência relativa de cada grupo da variável explicativa, como a seguir

```
covariancia_freq <- function(covar) {
  sinasc %>%
    group_by_at(c(covar, "PARTO")) %>%
    tally() %>%
    mutate(frac = round(n / sum(n), 2)) %>%
    arrange_at(c(covar, "PARTO")) %>%
    collect() %>%
    as.data.frame()
}

walk(colnames(sinasc), ~ print(covariancia_freq(.x)))
```

Dos resultados apresentados ao executar a função `walk`, do bloco anterior, pode-se observar que de fato algumas variáveis ajudam a explicar o tipo de parto mais do que outras. Como os resultados são tabelas muito extensas, optou-se por utilizar a função `ggplot` (disposto no próximo bloco de código) e apresentar as figuras 1 e 3, que ilustram como a idade da mãe e o tipo de gestação são variáveis que pesam a favor de um parto cesáreo.

As figuras 1 e 3 permitem intuir que a regressão logística não fará um bom ajuste dos dados, o que acontece é que em ambos os casos a proporção de partos cesáreos para cada grupo da variável, em geral, é maior que a de partos não cesáreos. Analisando as outras variáveis explicativas, há alguns casos em que ocorre uma divisão próxima ao empate para o tipo de parto, não obstante, isso também não é suficiente para que um modelo de regressão logística performe bem. Tradicionalmente, espera-se que haja uma concentração de cada um dos 2 possíveis resultados da variável resposta em uma parte do domínio dos dados, por exemplo: suponha tentar predizer se um rato está obeso baseando-se em seu peso, intui-se que ratos muito pesados tem uma alta probabilidade de estarem obsos e que ratos muito leves tem uma baixa probabilidade de estarem obesos, se a intuição se satisfizer ao analisar os dados, então a regressão logística fará um bom ajuste.

```r
covariancia_freq("IDADEMAE") %>%
  filter(!(IDADEMAE %in% c(0, 99))) %>%
  ggplot(aes(x = IDADEMAE, y = frac)) +
  geom_histogram(aes(fill = PARTO), stat = "identity", position = "stack")

covariancia_freq("GESTACAO") %>%
  ggplot(aes(x = GESTACAO, y = frac)) +
  geom_histogram(aes(fill = PARTO), stat = "identity", position = "stack")
```

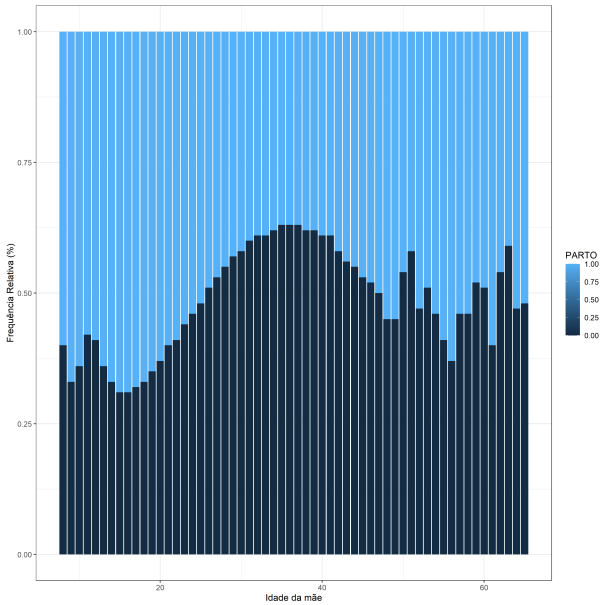

Figura 1: Histograma da frequência relativa de partos cesáreos pela idade da mãe.

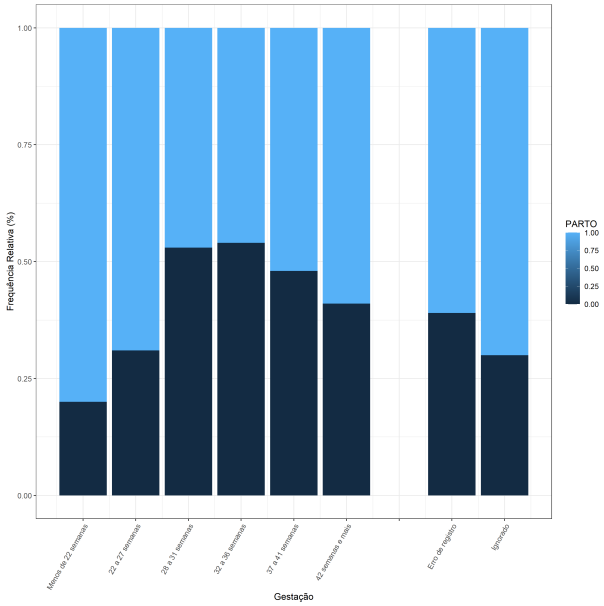

Figura 2: Histograma da frequência relativa de partos cesáreos pelo tipo de gestação.

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: Decidiu-se fazer a imputação dos dados para as variáveis quantitativas mantidas no banco utilizando a mediana, que com a função `ft_imputer` pode ser feita facilmente especificando as colunas a serem tratadas e o parâmetro `strategy = "median"`. Além disso, vale notar que talvez não faça muito sentido imputar através da mediana os dados faltantes em variáveis qualitativas (especialmente as não ordinais), portanto as observações faltantes foram trocadas por "9", código padrão do SINASC para "Ignorado", como no bloco de código a seguir

```
qnt_vars <- c("IDADEMAE", "QTDFILVIVO", "PESO")

sinasc <- sinasc %>%
  ft_imputer(
    input_cols = qnt_vars,
    output_cols = qnt_vars,
    strategy = "median"
  ) %>%
  filter(!is.na(DTNASC)) %>%
  mutate(CONSULTAS = if_else(is.na(CONSULTAS), 9, CONSULTAS)) %>%
  mutate(GRAVIDEZ = if_else(is.na(GRAVIDEZ), 9, GRAVIDEZ)) %>%
  mutate(GESTACAO = if_else(is.na(GESTACAO), 9, GESTACAO)) %>%
  mutate(ESCMAE = if_else(is.na(ESCMAE), 9, ESCMAE))
```

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

Solução: O Spark exige que a transformação de *features* por normalização seja feita a partir de um vetor de *features*, logo, foram utilizadas as funções `ft_vector_assembler` para criar uma coluna contendo um vetor das variáveis quantitativas e em seguida a função `ft_standard_scaler` para cumprir a normalização no vetor, isto é,

```
sinasc <- sinasc %>%
  ft_vector_assembler(
    input_cols = qnt_vars,
    output_col = "qnt_features"
  ) %>%
  ft_standard_scaler(
    input_col = "qnt_features",
    output_col = "standard_qnt"
  )
```

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 6 consultas" e "1" indique "7 ou mais consultas". (Ultilize as funções ft_)

Solução: Devido à utilização das funções do Hive para a leitura das datas de nascimento, elas já estão representando o dia da semana de forma codificada em índices, então o procedimento foi seguir com a função `ft_one_hot_encoder` na coluna `DTNASC`. Em seguida, a função `ft_binarizer` foi aplicada para binarizar o número de consultas pré-natais, de modo que "0" representa as consultas marcadas como {1,2,3} (até 6 consultas) e "1" representa {4,9} (mais de 6 consultas ou ignorado).

```
sinasc <- sinasc %>%
  ft_one_hot_encoder(
    input_cols = "DTNASC",
    output_cols = "DTNASC_dummy"
  ) %>%
  ft_binarizer(
    input_col = "CONSULTAS",
    output_col = "CONSULTAS_bin",
```

```
    threshold = 3
  )
```

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: A variável `PARTO` ainda não havia sido tratada e conta com observações faltantes, optou-se então por fazer uma binarização da mesma, de forma que parto cesáreo é indicado por "1" e do contrário ou faltante é indicado por "0". Subsequentemente, A função `sdf_random_split` lida com com particionamento do banco em treino e teste

Nota: No *script* na integra a variável `PARTO` foi corrigida mais cedo para facilitar o processo de testagem do algoritmo.

```
particoes <- sinasc %>%
  mutate(PARTO = if_else(PARTO != 2, 1, 0, 0)) %>%
  sdf_random_split(training = 0.8, test = 0.2, seed = 2022)

treino <- particoes$training
teste <- particoes$test
```

Vale lembrar que algumas variáveis do banco estão repetidas, sendo elas: `IDADEMAE, QTDFILVIVO, PESO` e `qnt_features` (que estão dispostas na coluna `standard_qnt` devido ao item **c)**), e as colunas `DTNASC` e `CONSULTAS` (que estão dispostas nas colunas `DTNASC_dummy` e `CONSULTAS_bin`, respectivamente, devido ao item **d)**). Essas colunas repetidas não foram incluidas no modelo.

Tomada a decisão de incluir todas as demais colunas, que não a da variável resposta, para atuarem como covariáveis explicativas no modelo, seguiu-se informando essa decisão para o modelo no formato "PARTO ~ <demais variáveis>" por meio função `ml_logistic_regression`

```
label_out <- "PARTO"
features <- paste(
  colnames(select(
    sinasc,
    -c(qnt_vars, "qnt_features", "DTNASC", "CONSULTAS", "PARTO")
  )),
  collapse = " + "
)
formula <- (glue("{label_out} ~ {features}"))
lr_model <- ml_logistic_regression(treino, formula)
```

Com o modelo treino, pode-se seguir para a avaliação do banco de teste com a função `ml_evaluate` e seguida avaliando alguns métodos que envolvem a curva ROC (*Receiver Operating Characteristic*), que é obtida pela razão entre as razões de verdadeiros positivos em relação aos positivos totais e de positivos falsos em relação aos negativos totais, caracterizando os elementos da matriz de confusão

Uma vez feito esse procedimento não se pode coletar de fato as informações trazidas pela curva ROC, portanto foi criado um *script* alternativo que utiliza apenas os dados de 2016 a 2020 para fazer a classificação, dos quais obteve-se uma precisão próxima aos 70%.

Nota: Outros modelos também foram treinados, tanto mudando as variáveis escolhidas, acrescentando variáveis que foram deixadas para trás originalmente (como o estado civil da mãe), retirando variáveis que haviam sido colocadas, utilizando somente os dados de 1996 dos estados AC e AM (apenas 2 arquivos) e para todos o resultado foi muito parecido variando no intervalo de 65% a 70%.

```
validation_summary <- ml_evaluate(lr_model, teste)

validation_summary$area_under_roc() #> 69,86%

roc <- validation_summary$roc() %>%
  collect()

ggplot(roc, aes(x = FPR, y = TPR)) +
  geom_line() +
  geom_abline(lty = "dashed")
```

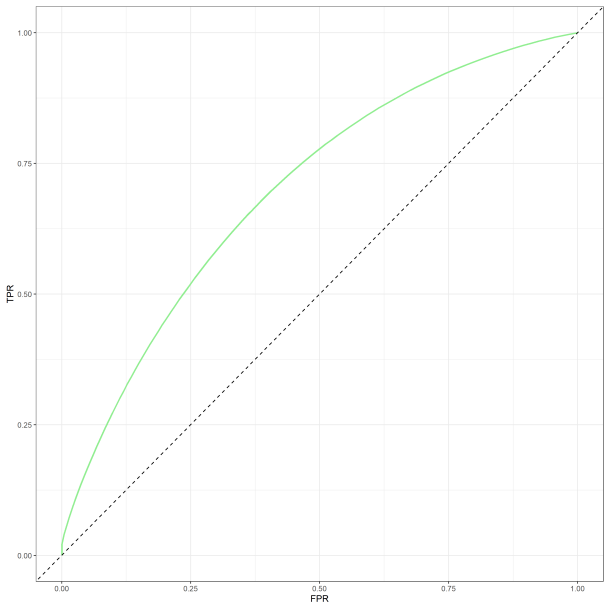

Figura 3: Curva ROC do modelo ajustado e reta identidade.

Repositório Contendo o Código Fonte

https://github.com/Voz-bonita/Big-Data-Compute