Tutorial E1 - RevoScaler

Leonardo Sangali Barone

April 19, 2017

Tutorial E1 - RevoScaleR

Neste tutorial vamos ver um breve introdução ao pacote *RevoScaleR*. Infelizmente, *RevoScaleR* não é um pacote disponível para a distribuiçao open-source do R e para utilizá-la devemos instalar o Microsoft R Client, disponível apenas para Windows.

Nos laboratórios do curso não há instalação do Microsoft R Cliente e faremos apenas uma demonstração de seu uso. Você pode ler este tutorial para aprender um pouco mais sobre *RevoScaleR*, o formato de dados que utiliza e as funções e métodos disponíveis e, se tiver interesse, acompanhá-lo quando tiver uma versão do Microsoft R Client instalada.

Porque usar o *RevoScaleR*?

Há duas razões pelas quais o *RevoScaleR* é atrativo no contexto do curso. A primeira delas é, se seu uso do R for estritamente local, o pacote oferece uma estratégia eficiente de superar o problema de dados que extrapolam a memória RAM do computador. Essa limitação é superada pelo uso de um formato próprio de dados cuja extensão é .xdf (External Data Frame), que é armazenado no disco e não na memória RAM de seu computador. Basicamente, *RevoScaleR* é um pacote que oferece métodos para arquivos em formato .xdf, que vão desde a manipulação básica de dados e apresentação gráfica, até os principais algoritmos de aprendizado de máquina.

A segunda razão é que o Microsoft R Client oferece integração com o SGBD Teradata. A opção da empresa Teradata foi descontinuar a integração de seus produtos com a versão open-source do R e focar no desenvolvimento integrado com o produto da Microsoft. Você pode ler mais sobre o tema e aprender sobre essa integração em RevoScaleR Teradata Getting Started Guide. Você também encontra informações adicionais para instalação aqui. Para usufruir da integração é preciso aprender a utilizar o pacote *RevoScaleR*, o que faremos no presente tutorial.

Mais uma gramática para dados: *RevoScaleR*

RevoScaleR tem a sua própria gramática de manipulação, apresentação e análise de dados. Essa gramática é exclusiva de dados em formato .xdf.

```
library(RevoScaleR)
```

O primeiro passo para lidar com dados com a gramática do *RevoScaleR* é transformá-los em formato .xdf. Vamos fazer um exemplo com dados em formato .csv, mas você pode importar dados via ODBC (veja este guia RevoScaleR ODBC Data Import). Neste outro guia Data Sources você encontra mais informações sobre os métodos disponíveis para importação de dados, que inclui SPSS e Teradata.

Faremos um exemplo simples com dados gerados com simulação apra fins didáticos ("fake_data"), disponíveis aqui. Vamos começar criando dois vetores atômicos de texto (vulgo, um objeto que só contém um texto), um com o endereço do arquivo .csv existente e outro com o endereço deseja para o futuro arquivo .xdf:

```
download.file("https://raw.githubusercontent.com/leobarone/FLS6397/master/data/fake_data.csv",
              "fake_data.csv")
```

```
arquivo_csv <- "fake_data.csv"
arquivo_xdf <- "fake_data.xdf"
```

A seguir, vamos utilizar a função *rxImport* para gerar o arquivo .xdf. Apesar do nome, a função não carrega os dados na memória RAM. Há diversos argumentos para a função, mas, basicamente, precisamos apenas informar os dados que serão transformados e o arquivo .xdf que será gerado:

```
rxImport(inData = arquivo_csv,
         outFile = arquivo_xdf)
```

```
## Rows Read: 30, Total Rows Processed: 30, Total Chunk Time: 0.045 seconds
```

Se o resultado de rxImport for atribuído a um objeto, como no exemplo abaixo, teremos uma representação do data frame externo em nosso workspace (algo semelhante ao que fizemos com dados de SGBD como o MySQL, mas tendo como local dos dados o disco):

```
fake_data <- rxImport(inData = arquivo_csv,
                      outFile = arquivo_xdf,
                      overwrite = T)
```

```
## Rows Read: 30, Total Rows Processed: 30, Total Chunk Time: 0.052 seconds
```

Para obter informações rápidas sobre o data frame externo usamos a função *rxGetInfo*

```
rxGetInfo(fake_data, getVarInfo = TRUE)
```

```
## File name: C:\Users\Nathan\Desktop\mq_bsb_17\mq_bsb_17\mq_bsb_17\tutoriais\fake_data.xdf
## Number of observations: 30
## Number of variables: 13
## Number of blocks: 1
## Compression type: zlib
## Variable information:
## Var 1: age, Type: integer, Low/High: (23, 43)
## Var 2: sex, Type: character
## Var 3: educ, Type: character
## Var 4: income, Type: numeric, Storage: float32, Low/High: (43.6993, 8213.8105)
## Var 5: savings, Type: numeric, Storage: float32, Low/High: (4203.0215, 45351.4102)
## Var 6: marriage, Type: character
## Var 7: kids, Type: character
## Var 8: party, Type: character
## Var 9: turnout, Type: character
## Var 10: vote_history, Type: character
## Var 11: economy, Type: character
## Var 12: incumbent, Type: character
## Var 13: candidate, Type: character
```

Se adicionarmos o argumento "numRows", também teremos uma visão rápida das primeiras linhas:

```
rxGetInfo(fake_data, getVarInfo = TRUE, numRows = 6)
```

```
## File name: C:\Users\Nathan\Desktop\mq_bsb_17\mq_bsb_17\mq_bsb_17\tutoriais\fake_data.xdf
## Number of observations: 30
## Number of variables: 13
## Number of blocks: 1
## Compression type: zlib
## Variable information:
## Var 1: age, Type: integer, Low/High: (23, 43)
## Var 2: sex, Type: character
```

```
## Var 3: educ, Type: character
## Var 4: income, Type: numeric, Storage: float32, Low/High: (43.6993, 8213.8105)
## Var 5: savings, Type: numeric, Storage: float32, Low/High: (4203.0215, 45351.4102)
## Var 6: marriage, Type: character
## Var 7: kids, Type: character
## Var 8: party, Type: character
## Var 9: turnout, Type: character
## Var 10: vote_history, Type: character
## Var 11: economy, Type: character
## Var 12: incumbent, Type: character
## Var 13: candidate, Type: character
## Data (6 rows starting with row 1):
##   age    sex                   educ   income   savings marriage kids
## 1  37 Female     College Incomplete 2595.252 14842.203      Yes    0
## 2  26 Female  No High School Degree 2166.740 24903.906      Yes    0
## 3  35   Male  No High School Degree 8213.811  5944.118      Yes    1
## 4  43   Male     High School Degree 4619.973 14576.176       No    0
## 5  36 Female College Degree or more 4072.067 25398.732       No    1
## 6  38   Male     College Incomplete 6463.019 14363.654       No    0
##             party turnout vote_history       economy     incumbent
## 1     Independent      No            4 About average          Good
## 2     Independent     Yes            0           Bad    Don't Know
## 3 Socialist Party      No            0           Bad About average
## 4     Independent     Yes            4           Bad           Bad
## 5 Socialist Party      No            2          Good      Very Bad
## 6     Independent      No            1           Bad About average
##   candidate
## 1     Other
## 2    Trampi
## 3    Trampi
## 4     Other
## 5    Rilari
## 6      None
```

Outra função útil para conhecer os dados, semelhante à função *summary* da biblioteca básica do R, é *rxSummary* (você já deve ter notado que todas as funções do *RevoScaleR* começam com "rx"). O detalhe, comum às demais funções do pacote, é que precisamos especificar uma fórmula (mesmo não havendo nenhum modelo sendo contruído). As fórmulas funcionam da mesma maneira que nos demais pacotes de R e têm a seguinte estrutura: "varivel_dependente ~ variavel_independente1 + variavel_independente2 + ...". Quando não há "variável dependente", como é o caso de um sumário de dados, basta omitir o lado esquerdo da fórmula e começar a partir do símbolo "~"

```
rxSummary(formula = ~ income + savings + age,
          data = fake_data)
```

```
## Rows Read: 30, Total Rows Processed: 30, Total Chunk Time: 0.007 seconds
## Computation time: 0.015 seconds.
```

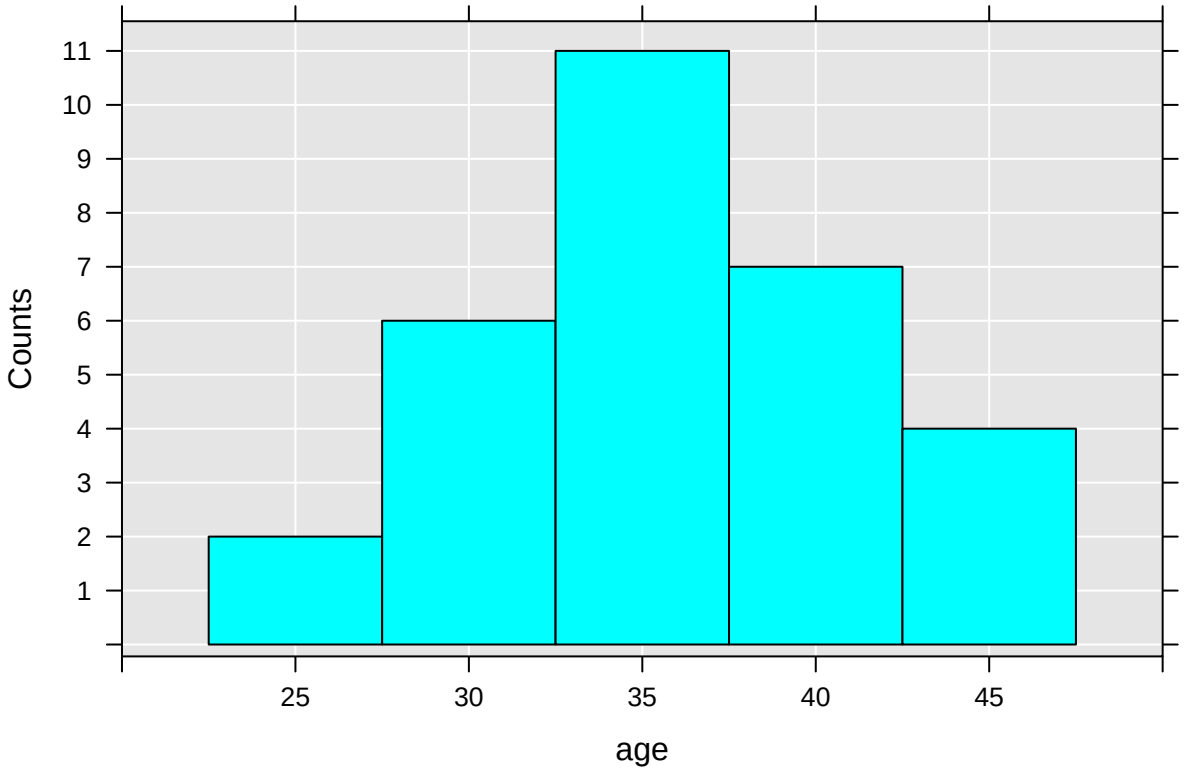
```
## Call:
## rxSummary(formula = ~income + savings + age, data = fake_data)
##
## Summary Statistics Results for: ~income + savings + age
## Data: fake_data (RxXdfData Data Source)
## File name: fake_data.xdf
## Number of valid observations: 30
##
```

```
##  Name     Mean         StdDev       Min         Max       ValidObs MissingObs
##  income    3038.17854  2171.555729   43.69933  8213.811 30       0
##  savings 13605.01431 10820.407133 4203.02148 45351.410 30       0
##  age        34.06667     5.205656   23.00000    43.000 30       0
```

Podemos produzir um histograma com *rxHistogram*, que utiliza os mesmos argumentos de *rxSummary*:

```
rxHistogram(formula = ~ age,
            data = fake_data)
```

```
## Rows Read: 30, Total Rows Processed: 30, Total Chunk Time: 0.007 seconds
## Computation time: 0.023 seconds.
```



A transformação de variáveis com *RevoScaleR* é feita basicamente utilizando a função *RxDataStep*. Esta função guarda bastante semelhança com *rxImport* por ter como argumentos principas "inData" e "outFile". Este último, porém, não precisa ser especificado se o objetivo for sobrescrever os dados originais com os dados tranformados (que não é o nosso caso).

Três argumentos das funções permitem as transformações essenciais dos dados. "varsToKeep", se preenchido com um vetor de nome das variáveis, seleciona as variáveis que permanecerão no data frame resultantes. "rowSelection" permite fazer a seleção de linhas. Finalmente, "transforms" recebe uma lista de operações de variáveis a serem realizadas.

No exemplo abaixo, manteremos apenas as variáveis "age", "income" e "savings", selecionaremos as linhas que têm valor maior ou igual a 25 na variável "age" e produziremos três transformações a partir das variáveis originais:

```
arquivo2_xdf <- "~/mq_bsb_17/dados/fake_data2.xdf"
rxDataStep(inData = arquivo_xdf,
```

```
        varsToKeep = c( "age", "income", "savings"),
        rowSelection = age > 25,
          transforms = list(prop_income_sav = income / savings,
                            savings_age = savings/ age,
                            age_months = age))
```

```
## Rows Read: 30, Total Rows Processed: 30, Total Chunk Time: 0.015 seconds
```

```
##    age     income   savings prop_income_sav savings_age age_months
## 1   37 2595.25244 14842.203      0.17485628    401.1406         37
## 2   26 2166.74048 24903.906      0.08700404    957.8425         26
## 3   35 8213.81055  5944.118      1.38183837    169.8319         35
## 4   43 4619.97314 14576.176      0.31695372    338.9808         43
## 5   36 4072.06738 25398.732      0.16032561    705.5203         36
## 6   38 6463.01904 14363.654      0.44995646    377.9909         38
## 7   29 2219.01416  5118.181      0.43355526    176.4890         29
## 8   33 2826.06152 14922.285      0.18938530    452.1905         33
## 9   42  448.05939 14147.852      0.03166978    336.8536         42
## 10  41 6540.19678  5419.379      1.20681656    132.1800         41
## 11  35   43.69933  4203.021      0.01039712    120.0863         35
## 12  35  266.92899  5162.188      0.05170849    147.4911         35
## 13  28 6696.28955 45351.410      0.14765339   1619.6932         28
## 14  32  524.24487  4639.687      0.11299144    144.9902         32
## 15  40 2686.48047 34981.559      0.07679705    874.5390         40
## 16  37 4180.46631 15083.150      0.27716135    407.6527         37
## 17  34 4537.20117 35374.273      0.12826274   1040.4198         34
## 18  33 2202.34497  4867.248      0.45248263    147.4924         33
## 19  29  652.70715  4718.129      0.13834024    162.6941         29
## 20  43 2300.42407 14801.091      0.15542260    344.2114         43
## 21  35 2929.64185 15011.512      0.19515968    428.9003         35
## 22  30 6404.45068  4896.841      1.30787398    163.2280         30
## 23  29 3017.94946  5103.641      0.59133267    175.9876         29
## 24  32  874.76080  4884.785      0.17907866    152.6495         32
## 25  38 2879.75732  4785.094      0.60181831    125.9235         38
## 26  35 4035.14038 24970.193      0.16159828    713.4341         35
## 27  37  508.59933  4935.762      0.10304374    133.3990         37
## 28  32 2108.53662  4872.189      0.43276988    152.2559         32
```

Para situações em que precisamos combinar mais de um data frame, podemos usar *rxMerge*. Esta função recebe dois data frames externos ("inData1" e "inData2") e cria uma combinação segundo um "type" – que pode ser "inner", "full", "left" e "right" – a partir de uma ou mais chaves ("matchVars") e criando um novo arquivo .xdf ("outFile").

Vejamos como ficaria o exemplo de "inner join" que fizemos no Tutorial 3 se utilizarmos a função *rxMerge*:

```
download.file("https://raw.githubusercontent.com/leobarone/FLS6397/master/data/pagamentos11.csv",
              "pagamentos11.csv")
download.file("https://raw.githubusercontent.com/leobarone/FLS6397/master/data/pagamentos17.csv",
              "pagamentos17.csv")

pagamentos11 <- rxImport(inData = "pagamentos11.csv",
                         outFile = "pagamentos11.xdf")
```

```
## Rows Read: 39, Total Rows Processed: 39, Total Chunk Time: 0.048 seconds
```

```
pagamentos17 <- rxImport(inData = "pagamentos17.csv",
                         outFile = "pagamentos17.xdf")
```

```
## Rows Read: 48, Total Rows Processed: 48, Total Chunk Time: 0.071 seconds
```

```
rxMerge(inData1 = "pagamentos11.xdf",
        inData2 = "pagamentos17.xdf",
        outFile = "pagamentos.xdf",
        matchVars = "NIS Favorecido",
        type = "inner")
```

```
## Number of rows written to file: 39, Number of columns: 12, Total number of rows in file: 39
## Time to sort data file: 0.022 seconds
## Number of rows written to file: 48, Number of columns: 12, Total number of rows in file: 48
## Time to sort data file: 0.022 seconds
## Time to merge data file: 0.033 seconds
```

```
## RxXdfData Source
## "pagamentos.xdf"
## fileSystem:
##     fileSystemType: native
```

Há outras várias funções de manipulação de dados do *RevoScaleR*. Você encontrará uma lista de funções disponíveis no pacote *RevoScaleR* aqui

RevoScaleR para análise de dados e aprendizado de máquina

Ademais das funções de importação e manipulação de dados, o pacote *RevoScaleR* oferece diversas funções para análise e aprendizado de máquina.

Voltando a nossos dados originais, podemos realizar cruzamento de dados com *rxCrossTabs* e *rxCube*. Como essas funções dependem de variáveis do tipo "factor", vamos usar a função rxFactors para transformar "sex" e "educ" em factors e, a seguir, usar as funções que confeccionam tabela:

```
fake_data <- rxFactors(inData = fake_data, factorInfo = c("sex", "educ"))
```

```
## Rows Read: 30, Total Rows Processed: 30, Total Chunk Time: 0.007 seconds
##
Rows Processed: 30
## Time to read data file: 0.01 secs.
## Time to convert to data frame: less than .001 secs.
```

```
rxCrossTabs(formula = ~ sex + educ, data = fake_data)
```

```
## Warning in rxLinkFormulaComponents(formula = formula, fweights =
## fweights, : Please use ':' instead of '+' to separate the independent
## variables: '~sex + educ'
```

```
## Rows Read: 30, Total Rows Processed: 30, Total Chunk Time: 0.010 seconds
## Computation time: 0.015 seconds.
```

```
## Call:
## rxCrossTabs(formula = ~sex + educ, data = fake_data)
##
## Cross Tabulation Results for: ~sex + educ
## Data: fake_data
## Number of valid observations: 30
## Number of missing observations: 0
```

```
## Statistic: counts
##
## sex:educ (counts):
##         educ
## sex      College Incomplete No High School Degree High School Degree
##   Female                  4                     1                  8
##   Male                    2                     2                  6
##         educ
## sex      College Degree or more
##   Female                      2
##   Male                        5
```

```
rxCrossTabs(formula = income ~ sex + educ, data = fake_data)
```

```
## Warning in rxLinkFormulaComponents(formula = formula, fweights =
## fweights, : Please use ':' instead of '+' to separate the independent
## variables: 'income ~ sex + educ'
```

```
## Rows Read: 30, Total Rows Processed: 30, Total Chunk Time: 0.002 seconds
## Computation time: 0.008 seconds.
```

```
## Call:
## rxCrossTabs(formula = income ~ sex + educ, data = fake_data)
##
## Cross Tabulation Results for: income ~ sex + educ
## Data: fake_data
## Dependent variable(s): income
## Number of valid observations: 30
## Number of missing observations: 0
## Statistic: sums
##
## income (sums):
##         educ
## sex      College Incomplete No High School Degree High School Degree
##   Female           11146.99               2166.74           10725.03
##   Male              9149.50              10514.23           20290.63
##         educ
## sex      College Degree or more
##   Female               10476.52
##   Male                 16675.72
```

```
rxCube(formula = ~ sex + educ, data = fake_data)
```

```
## Warning in rxLinkFormulaComponents(formula = formula, fweights =
## fweights, : Please use ':' instead of '+' to separate the independent
## variables: '~sex + educ'
```

```
## Rows Read: 30, Total Rows Processed: 30, Total Chunk Time: 0.002 seconds
## Computation time: 0.006 seconds.
```

```
## Call:
## rxCube(formula = ~sex + educ, data = fake_data)
##
## Cube Results for: ~sex + educ
## Data: fake_data
## Number of valid observations: 30
## Number of missing observations: 0
```

```
##
##   sex    educ                    Counts
## 1 Female College Incomplete     4
## 2 Male   College Incomplete     2
## 3 Female No High School Degree  1
## 4 Male   No High School Degree  2
## 5 Female High School Degree     8
## 6 Male   High School Degree     6
## 7 Female College Degree or more 2
## 8 Male   College Degree or more 5
```

```
rxCube(formula = income ~ sex + educ, data = fake_data)
```

```
## Warning in rxLinkFormulaComponents(formula = formula, fweights =
## fweights, : Please use ':' instead of '+' to separate the independent
## variables: 'income ~ sex + educ'
```

```
## Rows Read: 30, Total Rows Processed: 30, Total Chunk Time: 0.002 seconds
## Computation time: 0.007 seconds.
```

```
## Call:
## rxCube(formula = income ~ sex + educ, data = fake_data)
##
## Cube Results for: income ~ sex + educ
## Data: fake_data
## Dependent variable(s): income
## Number of valid observations: 30
## Number of missing observations: 0
## Statistic: income means
##
##   sex    educ                    income   Counts
## 1 Female College Incomplete     2786.746 4
## 2 Male   College Incomplete     4574.750 2
## 3 Female No High School Degree  2166.740 1
## 4 Male   No High School Degree  5257.117 2
## 5 Female High School Degree     1340.629 8
## 6 Male   High School Degree     3381.772 6
## 7 Female College Degree or more 5238.259 2
## 8 Male   College Degree or more 3335.143 5
```

Calcular uma matriz de correlações, novamente usando o argumento "formula":

```
rxCor(formula = ~ age + income + savings,
      data = fake_data)
```

```
## Rows Read: 30, Total Rows Processed: 30, Total Chunk Time: Less than .001 seconds
## Computation time: 0.007 seconds.
```

```
##                 age    income     savings
## age      1.00000000 0.1137671 -0.01703835
## income   0.11376714 1.0000000  0.31922330
## savings -0.01703835 0.3192233  1.00000000
```

Ou produzir um modelo linear com *rxLinMod*:

```
modelo_linear <- rxLinMod(formula = income ~ savings,
         data = fake_data)
```

```
## Rows Read: 30, Total Rows Processed: 30, Total Chunk Time: 0.001 seconds
```

```
## Computation time: 0.853 seconds.
```

```
summary(modelo_linear)
```

```
## Call:
## rxLinMod(formula = income ~ savings, data = fake_data)
##
## Linear Regression Results for: income ~ savings
## Data: fake_data
## Dependent variable(s): income
## Total independent variables: 2
## Number of valid observations: 30
## Number of missing observations: 0
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.167e+03  6.208e+02   3.490  0.00162 **
## savings     6.407e-02  3.594e-02   1.782  0.08553 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2094 on 28 degrees of freedom
## Multiple R-squared: 0.1019
## Adjusted R-squared: 0.06983
## F-statistic: 3.177 on 1 and 28 DF,  p-value: 0.08553
## Condition number: 1
```

Para reproduzir integralmente os exemplos de aprendizado de máquina como o pacote *RevoScaleR* utilize as funções *rxPredict*, para obter valores previstos, *rxDTree*, para árvores de decisão e *rxKmeans* para Kmeans Cluster. Dê uma olhada novamente na lista de funções de Analysis, Learning, and Prediction Functions for Statistical Modeling do *RevoScaleR* para mais.

Onde aprender mais

Há um curso gratuito no site Datacamp criado pela prórpia Revolution Analytics, antes mesmo da empresa ter sido comprada pela Microsoft. Além disso, há uma introdução no site da Microsoft bastante útil. Finalmente, você pode ler o RevoScaleR User's Guider.

Finalmente, *RevoScaleR* tem suporte para computação em diversos contextos – Hadoop, Spark, Teradata e SQL, por exemplo. Para trabalhar localmente com mais de um "core" do processador, é possível utilizar a função *RxLocalParallel* e *rxSetComputeContext*. Para um guia completo sobre "parallel computaing" com *RevoScaleR* veja: RevoScaleR Distributed Computing Guide