ISEL – Instituto Superior de Engenharia de Lisboa

DEETC - Departamento de Engenharia de Eletrónica e Telecomunicações e Computadores

MEIC

Mestrado em Engenharia Informática e Computadores

Mineração de Dados em Larga Escala

Aula de Laboratório #1

(Fevereiro 2024)

Configuração da linguagem R e do ambiente de trabalho

Nuno Gomes 18364
Ricardo Ramos 46638
Rafael Carvalho 47663

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

1. Introdução

A análise de grandes conjuntos de dados para extrair informações valiosas e padrões significativos é o foco da disciplina de Big Data Mining. Com o aumento exponencial do volume de dados na era digital, tornou-se essencial contar com ferramentas e tecnologias especializadas para lidar eficientemente com esses vastos conjuntos de informações.

O Apache Spark destaca-se como uma ferramenta poderosa no contexto do Big Data, proporcionando um ambiente de computação em cluster rápido e versátil para o processamento de dados. A linguagem de programação R é amplamente reconhecida no âmbito da estatística e análise de dados, sendo uma escolha popular devido à sua sintaxe amigável e à extensa coleção de pacotes, especialmente para quem trabalha com dados em grande escala.

Neste trabalho introdutório à unidade curricular, pretende-se testar, ensaiar e familiarizar quer com o ambiente Spark quer com a linguagem de programação R, constituindo estes o alicerce para os demais trabalhos e desenvolvimento das aprendizagens da unidade curricular. Em síntese, o relatório subdivide-se em dois temas: Instruções na linguagem R, onde é abordado a prática de implementação de instruções R básicas, essenciais, e configuração do Spark, onde, sucintamente são descritos os resultados da aplicação do código.

2. Instruções na linguagem R

2.1 Escalares e Operadores

Na linguagem R, um escalar representa um valor único, podendo ser numérico ou um caráter. Os operadores podem ser categorizados como aritméticos, exemplificados por '+', '-', '*' e '/', destinados a realizar operações de soma, subtração, multiplicação e divisão, respetivamente. Existem também operadores de comparação, tais como '<', '>', '==' e '!=', utilizados para avaliar se um valor é maior, menor, igual ou diferente de outro.

Segue-se uma tabela com as instruções a serem executadas, acompanhada de uma explicação dos resultados obtidos para cada uma.

1. var1 <− 3	Atribui o valor 3 à variável var1
2. show(var1)	Mostra o valor da variável var1
3. var2 <− var1 ∗ var1	Atribui à variável var2 o quadrado do valor de var1
4. var3 <− var1 ∗∗ 2	Atribui à variável var3 o valor de var1 elevado ao quadrado
5. var4 <− var1 ^ 2	(^ operador de potência) Atribui à variável var4 o valor de var1 ao quadrado
6. var1 < var1	Retorna o resultado booleano da comparação do valor de var1 com o valor de var1
7. var3 != var4	Verifica se var3 é diferente de var4 e retorna TRUE ou FALSE
8. var2 == var4	Verifica se var2 é igual a var4 e retorna TRUE ou FALSE
9. var2 <− var2 − var2	Subtrai o valor atual de var2 do próprio var2, resultando em 0 e guarda em var2
10. var5 <− var3 / var2	Divide var3 pelo valor de var2 e atribui o resultado à variável var5
11. var5 + 1	Adiciona 1 ao valor de var5 e retorna o resultado

Tabela 1 - Escalares e Operadores

2.2 Manipulação de Vetores

O vetor é uma estrutura de dados que pode conter múltiplos escalares do mesmo tipo, e é criado através da função c(). Os vetores podem ter uma dimensão, ou várias, criando matrizes.

1. a <- c(1,2,5.3,6,-2, 4, 3.14159265359)	A função c() cria um vetor, afeta a variável 'a' com o vetor
2. a	Mostra o vetor 'a'
3. b <- c("1", "2", "3")	Cria um vetor de caracteres 'b'
4. "2" %in% b	Verifica se "2" está presente no vetor 'b'
5. "5" %in% b	Verifica se "5" está presente no vetor 'b'
6. c <- c(TRUE, TRUE, FALSE, TRUE)	Cria um vetor de booleanos 'c'
7. a <- c	Afeta a variável 'a' com os valores de 'c'
8. a[0]	Acede ao elemento na posição 0 de 'a'. De notar que a indexação dos elementos começa em 1
9. a[1]	Acede ao primeiro elemento de 'a'
10. a[-1]	Apresenta todos os valores exceto o primeiro elemento de 'a'
11. a[8]	Acede ao oitavo elemento de 'a' (se existir)
12. a[c(1, 3)]	Seleciona os elementos nas posições 1 e 3 de 'a'
13. a[c(3, 1)]	Seleciona os elementos nas posições 3 e 1 de 'a'
14. a[a>2]	Seleciona apenas os elementos maiores que 2 do vetor 'a'

Tabela 2 - Manipulação de vetores

É de referir que a operação 14 pode ser separada em dois passos:

1) Compara todos os elementos de ‘a’ com 2:

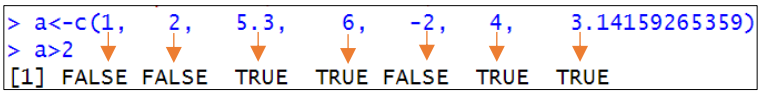

Figura 1 - Comparação dos elementos do vetor 'a' com 2

2) Indexa o vetor ‘a’ e retorna todos os elementos onde o vetor de a > 2 é TRUE:

```
> a
[1]  1.000000  2.000000  5.300000  6.000000 -2.000000  4.000000  3.141593
> a[c(FALSE, FALSE, TRUE, TRUE, FALSE, TRUE, TRUE)]
[1] 5.300000 6.000000 4.000000 3.141593
> a[a>2]
[1] 5.300000 6.000000 4.000000 3.141593
```

Figura 2 - Retorno de a[a>2]

2.3 Manipulação de matrizes

1. m <- matrix(1:6, nrow=3, ncol=2)	Cria uma matriz 3x2 com valores de 1 a 6
2. show(m)	Mostra a matriz ‘m’
3. n <- matrix(2:7, nrow=2, ncol=3)	Cria uma matriz 2x3 com valores de 2 a 7
4. n	Mostra a matriz ‘n’
5. m[, 2]	Seleciona a segunda coluna da matriz ‘m’
6. n[1,]	Seleciona a primeira linha da matriz ‘n’
7. m[2:3, 1:2]	Seleciona as linhas 2 e 3 e as colunas 1 e 2 da matriz ‘m’
8. n %*% m	Multiplica as matrizes ‘n’ e ‘m’
9. m %*% n	Multiplica as matrizes ‘m’ e ‘n’
10. n %*% n	Multiplica a matriz ‘n’ por ela mesma
11. n^2	Eleva cada elemento da matriz ‘n’ ao quadrado
12. sqrt(n)	Calcula a raiz quadrada de cada elemento da matriz ‘n’

Tabela 3 - Manipulação de matrizes

Para a matriz construída ‘n’, de 2 linhas e 3 colunas, a operação nº. 10 não é matematicamente possível. Este tipo de operações apenas é possível em situações onde a matriz é quadrada.

```
> n
     [,1] [,2] [,3]
[1,]    2    4    6
[2,]    3    5    7
> n%*%n
Error in n %*% n : non-conformable arguments
```

Figura 3 - Demonstração da operação nº. 10 na matriz 'n'

2.4 Manipulação de *data frames*

Um *data frame* é uma estrutura de dados tabular assemelhando-se a uma tabela, onde cada coluna pode conter valores escalares de diferentes tipos de dados. No exemplo seguinte, cria-se um *data frame* inicialmente com três colunas, incluindo um identificador numérico, uma string e um valor booleano. As colunas são nomeadas como “ID”, “Name” e “Passed”

utilizando a função names(). Posteriormente, é adicionada uma nova coluna chamada “Failed” através da função cbind() (Column Bind), seguida da inclusão de uma nova entrada (linha) utilizando a função rbind() (Row Bind).

1. d <- c(1, 2, 3, 4)	Cria um vetor ‘d’ com valores de 1 a 4
2. e <- c("Bob", "Alice", NA, "Joe")	Cria um vetor ‘e’ de strings
3. f <- c(TRUE, TRUE, FALSE, TRUE)	Cria um vetor lógico ‘f’
4. my.data <- data.frame(d, e, f, stringsAsFactors=FALSE)	Cria um data frame chamado 'my.data'
5. show(my.data)	Mostra o conteúdo do data frame 'my.data'
6. names(my.data) <- c("ID", "Name", "Passed")	Renomeia as colunas do data frame 'my.data'
7. View(my.data)	Visualização da matriz do data frame 'my.data'. A visualização é apresentada na Figura 7
8. my.data$Name	Seleciona a coluna 'Name' do data frame 'my.data'
9. my.data <- cbind(my.data, Failed=!f)	Adiciona uma nova coluna 'Failed' ao data frame 'my.data'. Esta nova coluna contém os valores resultantes da negação da variável “f”, na Figura 6
10. View(my.data)	Mostra o data frame 'my.data' após a alteração. O data frame é mostrado na Figura 5
11. my.data <- rbind(my.data, c(5, "Carol", FALSE, TRUE))	Adiciona uma nova linha ao data frame 'my.data'. Esta nova linha contém os valores 5, “Carol”, FALSE, TRUE.
12. View(my.data)	Mostra o data frame 'my.data' após a alteração. O data frame é mostrada na Figura 4
13. ?data.frame	Abre a documentação para a função data.frame

Tabela 4 - Manipulação de Data Frames

	ID	Name	Passed
1	1	Bob	TRUE
2	2	Alice	TRUE
3	3	*NA*	FALSE
4	4	Joe	TRUE

Figura 7 - Vista de my.data na operação nº. 7

```
> !f
[1] FALSE FALSE  TRUE FALSE
```

Figura 6 - Negação da variável "f"

	ID	Name	Passed	Failed
1	1	Bob	TRUE	FALSE
2	2	Alice	TRUE	FALSE
3	3	*NA*	FALSE	TRUE
4	4	Joe	TRUE	FALSE

Figura 5 - Vista de my.data na operação nº. 10

	ID	Name	Passed	Failed
1	1	Bob	TRUE	FALSE
2	2	Alice	TRUE	FALSE
3	3	*NA*	FALSE	TRUE
4	4	Joe	TRUE	FALSE
5	5	Carol	FALSE	TRUE

Figura 4 - Vista de my.data na operação 12

2.5 Manipulação de fatores

Os fatores são um tipo de dados utilizados para classificar categoricamente diferentes variáveis com um conjunto fixo, e bem determinado, de valores. Os valores destas variáveis podem ser textuais, no entanto são representados de forma numérica nos fatores. Podem ser usados para distinguir, por exemplo, duas cores, como demonstrado no exemplo abaixo, onde é definido um vetor com a função rep(), que replica um determinado valor passado como argumento, neste caso "red" 20 vezes, e "blue" 30 vezes.

Ao aplicar a função factor() sobre este vetor, passamos a ter 2 níveis: "Red" e "Blue", que depois permite inferir resumos estatísticos deste vetor com a função summary().

1. colour <- c(rep("red", 20), rep("blue", 30))	Cria um vetor chamado 'colour' com repetição de "red" 20 vezes e "blue" 30 vezes
2. colour <- factor(colour)	Converte 'colour' num fator apresentado todos os valores diferentes presentes
3. summary(colour)	Apresenta um resumo estatístico do fator 'colour'.
4. dimensions <- c("large", "medium", "small")	Cria um vetor de strings chamado 'dimensions'
5. show(dimensions)	Mostra o vetor 'dimensions'
6. dimensions <- ordered(dimensions)	Converte 'dimensions' num fator ordenado, a ordenação é feita alfabeticamente
7. show(dimensions)	Mostra o vetor 'dimensions' após a alteração

Tabela 5 - Manipulação de Fatores

```
> colour <- c(rep("red", 20), rep("blue", 30))
> summary(colour)
   Length     Class      Mode
       50 character character
> colour = factor(colour)
> summary(colour)
blue  red
  30   20
```

Figura 8 - Transformação do vetor 'colour' em fatores, e sumário descritivo

2.6 Algumas funções úteis

O R disponibiliza também outras funções que podem por vezes ser úteis, como por exemplo a função length(), que retorna a dimensão de um determinado vetor, class(), que retorna a classe do objeto passado como argumento, nrow() e ncol() para obter a informação de linhas e colunas de um data frame. Para obter informação acerca da sessão atual, usa-se a função sessionInfo(); a listagem dos objetos criados é feita com ls() e podem ser removidos com rm(), passando como argumento o objeto a apagar. Para situações onde a data frame é muito extensa, a função glimpse() é útil uma vez que apresenta os dados de todas as colunas na vertical, e os dados de cada, na horizontal, como demonstrado abaixo:

1. length(colour)	Retorna o comprimento (número de elementos) do vetor 'colour'
2. length(a)	Retorna o comprimento do vetor 'a'
3. class(colour)	Retorna a classe do objeto 'colour'
4. class(dimensions)	Retorna a classe do objeto 'dimensions'
5. class(a)	Retorna a classe do objeto 'a'
6. class(my.data)	Retorna a classe do objeto 'my.data'
7. nrow(my.data)	Retorna o número de linhas do data frame 'my.data'
8. ncol(my.data)	Retorna o número de colunas do data frame 'my.data'
9. str(my.data)	Apresenta a estrutura do data frame 'my.data'
10. sessionInfo()	Mostra informações sobre a sessão do R
11. ls()	Lista os objetos no ambiente de trabalho
12. rm(a)	Remove o objeto 'a' do ambiente de trabalho
13. glimpse(my.data)	Apresenta uma visão resumida do data frame 'my.data'

Tabela 6 - Manipulação de Fatores

```
> glimpse(data)
Rows: 4
Columns: 4
$ ID      <dbl> 1, 2, 3, 4
$ Name    <chr> "John", "Alice", "Bob", "Emma"
$ Age     <dbl> 25, 30, 28, 22
$ Married <lgl> TRUE, FALSE, TRUE, FALSE
>
```

Figura 9 - Exemplo de utilização da função glimpse()

2.7 Pacotes

O R permite adicionar novos pacotes com mais funcionalidades. Estes pacotes podem ser listados com a função library(), e quando passado um pacote como argumento, por exemplo "MASS", carrega o mesmo. Para confirmar que foi carregado, utiliza-se a função search(). A instalação de novos pacotes é feita com a função install.packages().

Em situações onde são necessários carregar múltiplos pacotes de uma só vez, pode ser útil definir um vetor com o nome de cada um, como mostrado no exemplo 9 deste subcapítulo, e utilizar a função lapply() passando o vetor e a função a ser aplicada ao mesmo, neste caso, aplica-se a função require() ao vetor 'x', como demonstrado em 10.

1. library()	Mostra os pacotes atualmente carregados
2. search()	Mostra a lista de pacotes disponíveis
3. library("MASS")	Carrega o pacote "MASS"
4. search()	Mostra a lista de pacotes após a carga do "MASS"
5. .libPaths()	Mostra os caminhos do diretório onde os pacotes são instalados
6. install.packages("e1071")	Instala o pacote "e1071"
7. install.packages("funModeling")	Instala o pacote "funModeling"
9. x <- c("MASS", "dplyr", "e1071")	Cria um vetor 'x' com os nomes dos pacotes
10. lapply(x, require, character.only=TRUE)	Carrega os pacotes usando *lapply*
12. require(funModeling)	Carrega o pacote "funModeling"

Tabela 7 - Manipulação de Pacotes

2.8 Estatísticas descritivas básicas e outros

1. iris	Retorna o conjunto de dados 'iris'
2. summary(iris)	Apresenta um resumo estatístico do conjunto de dados 'iris'

3. fivenum(iris$Sepal.Length)	*fivenum* é utilizada para calcular resumos estatísticos chamados de "Five-Number Summary" (Resumo de Cinco Números). Esses cinco números são o mínimo, o primeiro quartil (Q1), a mediana (Q2), o terceiro quartil (Q3) e o máximo. Atribuir rótulos a estes valores, pode-se fazer algo como:
	`> nomes <- c("Mínimo", "Q1", "Mediana", "Q3", "Máximo")`
	`> resumo_rotulado <- setNames(fivenum(iris$Sepal.Length), nomes)`
	`> print (resumo_rotulado)`
	`Mínimo Q1 Mediana Q3 Máximo`
	`4.3 5.1 5.8 6.4 7.9`

4. status(iris)	Chama a função status com o conjunto de dados 'iris' (caso dê erro, executar primeiro *require(funModeling)*

Tabela 8 - Análise do conjunto de dados 'iris'

(c) A explicação que se segue é feita com base nos exercícios propostos para o ficheiro anexo ao trabalho, "ControlFlow.R".

ii. Explique o propósito dos três exemplos chamados EX.1, EX.2 e EX.3.

O primeiro exemplo apresenta um bloco 'for', com uma variável 'i' iniciada a 0, e terminada em 10, com iterações unitárias, definido pela função seq(). Por cada iteração, a variável 'x', posteriormente iniciada com o valor 0, é atualizada com a soma do seu valor, e do valor de 'i'. Desta forma, pode-se concluir que está a ser feito o somatório de 0 a 10, e que 'x' termina com o valor 55.

```
#### Ex1 ####
x<-0
for (i in seq(0,10,1))
{
  x<-x+i
}
```

Figura 10 - EX.1

No segundo exercício, novamente com um bloco 'for', a variável de incremento 'i' toma o valor 0 durante 10 iterações, definida pela função rep(), em que o primeiro argumento é o valor a replicar, e o segundo o número de vezes a ser replicado. Desta forma, em cada iteração, é mostrado na consola (print()), o valor de 'i', que é sempre 0.

```
#### Ex2 ####
for (i in rep(0,10))
{
  print(i)
}
```

Figura 12 - EX.2

```
[1] 0
[1] 0
[1] 0
[1] 0
[1] 0
[1] 0
[1] 0
[1] 0
[1] 0
[1] 0
```

Figura 11 - Output do exercício 2

No terceiro exercício, inicia-se um vetor 'v', e, num bloco '*while*', verifica-se o vetor tem dimensão positiva. Por cada iteração, o vetor 'v' é atualizado, removendo o primeiro elemento. De seguida, é mostrado na consola o novo vetor. O bloco '*if*' verifica se o valor 6 ainda está presente em 'v', e, caso não se encontre, termina o ciclo. Desta forma, o exercício remove todos os elementos de um vetor 'v', até que o número 6 não esteja presente.

```
#### Ex3 ####
v<-c(1,2,3,4,5,6,7,8,9)
while(length(v)>0)
{
  v<-v[-1]
  print(v)
  if( (6 %in% v) == FALSE)
    break
}
```

Figura 13 - EX.3

(d) Implemente e mostre um código que obtém as duas primeiras colunas de my.data, usando a indexação do operador de intervalo.

O operador de intervalo no R é a função range(). Desta forma, uma vez que pretendemos obter todas as linhas das duas primeiras colunas, podemos utilizar o comando abaixo:

$$my.data[, range(1,2)]$$

Como não foi introduzido nenhum valor para a indexação das linhas são mostradas todas as linhas disponíveis no *data frame*. Para a indexação das colunas é utilizada a função range(1,2). Esta função cria um vetor que contém os valores mínimo (1) e máximo (2), por omissão o valor de incremento é 1 e por isso obtém-se o seguinte vetor.

```
> range(1,2)
[1] 1 2
```

Figura 14 - Vetor criado pela função range()

```
> my.data[,range(1,2)]
  ID  Name
1  1   Bob
2  2 Alice
3  3  <NA>
4  4   Joe
5  5 Carol
```

Figura 15 - Indexação com operador de intervalo range()

(e) Implemente e mostre um código que obtém as duas primeiras colunas de my.data, usando a indexação vetorial.

Utilizando a indexação vetorial é possível obter as duas primeiras colunas através da linha seguinte:

$$my.data[, 1:2]$$

O operador ":" é muito semelhante à função range() mas apenas funciona com incrementos ou decrementos de 1 em 1.

```
> 1:2
[1] 1 2
```

Figura 18 - Vetor criado pelo operador ":"

```
> my.data[,1:2]
  ID  Name
1  1   Bob
2  2 Alice
3  3  <NA>
4  4   Joe
5  5 Carol
```

Figura 17 - Indexação vetorial

	ID	Name
1	1	Bob
2	2	Alice
3	3	*NA*
4	4	Joe
5	5	Carol

Figura 16 - Representação (View()) das primeiras duas colunas de my.data

(f) Instalação do pacote Hmisc e utilização da função describe na variável my.data.

A função describe() é utilizada para gerar estatísticas descritivas para um conjunto de dados. Estas estatísticas incluem medidas de tendência central, dispersão, assimetria, curtose (descreve a forma da distribuição de um conjunto de dados em relação à sua "forma de pico" e à "largura das caudas" em comparação com a distribuição normal padrão e informações sobre a presença de dados ausentes. O resultado incluirá estatísticas descritivas para cada variável no conjunto de dados, como média, desvio padrão, valores mínimo e máximo, quartis, entre outros. Essas informações são úteis para entender a distribuição e as propriedades básicas das variáveis no conjunto de dados.

```
> describe(my.data)
my.data
 4  variables      5  Observations
---------------------------------------------------------------------------------------------------------------------------
ID
       n  missing distinct
       5        0        5

Value        1   2   3   4   5
Frequency    1   1   1   1   1
Proportion 0.2 0.2 0.2 0.2 0.2
---------------------------------------------------------------------------------------------------------------------------
Name
       n  missing distinct
       4        1        4

Value      Alice   Bob Carol   Joe
Frequency      1     1     1     1
Proportion  0.25  0.25  0.25  0.25
---------------------------------------------------------------------------------------------------------------------------
Passed
       n  missing distinct
       5        0        2

Value      FALSE  TRUE
Frequency      2     3
Proportion   0.4   0.6
---------------------------------------------------------------------------------------------------------------------------
Failed
       n  missing distinct
       5        0        2

Value      FALSE  TRUE
Frequency      3     2
Proportion   0.6   0.4
---------------------------------------------------------------------------------------------------------------------------
```

Figura 19 - Resultado da função describe()

3. Introdução ao Spark

(a) Verifique programaticamente se o pacote sparklyr está carregado. Em seguida, conecte-se ao Spark usando:

$$ss <- spark\_connect('local', version = '3.4.2', hadoop\_version = '3', config = list())$$

Consultando a documentação da linguagem R, podemos verificar programaticamente se o pacote está carregado através do exemplo:

```
if (!requireNamespace("sparklyr", quietly = TRUE)) {
  # Caso não esteja carregado, instala-se
  install.packages("sparklyr")
  # E então carrega-se
  library(sparklyr)
}
```

Figura 20 - Verificação programática que o pacote foi carregado

(b) Veja o conteúdo da variável ss.

ss	list [13] (S3: spark_connection, sp	List of length 13
master	character [1]	'local'
method	character [1]	'shell'
app_name	character [1]	'sparklyr'
config	list [2]	List of length 2
state	environment [11]	<environment: 0x565558e82d80>
extensions	list [6]	List of length 6
spark_home	character [1]	'/home/Grupo03/spark/spark-3.4.2-bin-hadoop3'
backend	integer [1] (S3: sockconn, connec	4
monitoring	integer [1] (S3: sockconn, connec	5
gateway	integer [1] (S3: sockconn, connec	3
output_file	character [1]	'/tmp/RtmpkRlxzo/file2b14e8936f6d_spark.log'
sessionId	double [1]	7462
home_version	character [1]	'3.4.2'

Figura 21 - Conteúdo da variável 'ss' da ligação Spark

(d) Mostre alguns dados amostrais do conjunto de dados carregado.

```
> show(df)
# Source: spark<iris> [?? x 5]
   Sepal_Length Sepal_Width Petal_Length Petal_Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <chr>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa
 3          4.7         3.2          1.3         0.2 setosa
 4          4.6         3.1          1.5         0.2 setosa
 5          5           3.6          1.4         0.2 setosa
 6          5.4         3.9          1.7         0.4 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          5           3.4          1.5         0.2 setosa
 9          4.4         2.9          1.4         0.2 setosa
10          4.9         3.1          1.5         0.1 setosa
```

Figura 22 - Dados amostrais do conjunto de dados 'iris' carregados para o Spark

1. $head(select(df, Petal.Width, Species))$ 2. $head(filter(df, Petal.Width > 0{,}3))$

```
> head(select(df , Petal_Width , Species))
# Source: spark<?> [?? x 2]
  Petal_Width Species
        <dbl> <chr>
1         0.2 setosa
2         0.2 setosa
3         0.2 setosa
4         0.2 setosa
5         0.2 setosa
6         0.4 setosa
```

Figura 24 - Seleção das 'n' primeiras linhas de 'df', colunas 'Petal_Width' e 'Species'

```
> head(filter(df , Petal_Width > 0.3))
# Source: spark<?> [?? x 5]
  Sepal_Length Sepal_Width Petal_Length Petal_Width Species
         <dbl>       <dbl>        <dbl>       <dbl> <chr>
1          5.4         3.9          1.7         0.4 setosa
2          5.7         4.4          1.5         0.4 setosa
3          5.4         3.9          1.3         0.4 setosa
4          5.1         3.7          1.5         0.4 setosa
5          5.1         3.3          1.7         0.5 setosa
6          5           3.4          1.6         0.4 setosa
```

Figura 23 - Seleção das linhas onde 'Petal_Width' é > a 0.3, no conjunto de dados 'df'

3. $df \%>\% head$

```
> df %>% head
# Source: spark<?> [?? x 5]
  Sepal_Length Sepal_Width Petal_Length Petal_Width Species
         <dbl>       <dbl>        <dbl>       <dbl> <chr>
1          5.1         3.5          1.4         0.2 setosa
2          4.9         3            1.4         0.2 setosa
3          4.7         3.2          1.3         0.2 setosa
4          4.6         3.1          1.5         0.2 setosa
5          5           3.6          1.4         0.2 setosa
6          5.4         3.9          1.7         0.4 setosa
```

Figura 25 - Seleção das primeiras 6 linhas de 'df'

Para os exemplos anteriores, a função select() permite selecionar determinadas colunas, neste caso 'Petal_Width' e 'Species' do conjunto de dados 'df'. Para a função filter(), seleciona apenas as colunas que cumprem o predicado, neste caso 'Petal_Width' > 0.3.

Finalmente, com 'df%>%head', passamos o conjunto de dados 'df' à função head(), que retorna as primeiras 'n' linhas do conjunto de dados, onde, por padrão, n = 6.

(e) Utilizando SQL

2. $df\_sql <- dbGetQuery(ss, "SELECT * FROM iris WHERE Petal\_Width > 0.3 LIMIT 5")$
3. $show(df\_sql)$

```
> df_sql <- dbGetQuery(ss , "SELECT * FROM iris WHERE Petal_Width > 0.3 LIMIT 5")
> show(df_sql)
  Sepal_Length Sepal_Width Petal_Length Petal_Width Species
1          5.4         3.9          1.7         0.4  setosa
2          5.7         4.4          1.5         0.4  setosa
3          5.4         3.9          1.3         0.4  setosa
4          5.1         3.7          1.5         0.4  setosa
5          5.1         3.3          1.7         0.5  setosa
```

Figura 26 - Dados obtidos através de query SQL

(f) Obtenha dados dos nós Spark

```
> show(local_df)
# A tibble: 150 x 5
   Sepal_Length Sepal_Width Petal_Length Petal_Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <chr>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa
 3          4.7         3.2          1.3         0.2 setosa
 4          4.6         3.1          1.5         0.2 setosa
 5          5           3.6          1.4         0.2 setosa
 6          5.4         3.9          1.7         0.4 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          5           3.4          1.5         0.2 setosa
 9          4.4         2.9          1.4         0.2 setosa
10          4.9         3.1          1.5         0.1 setosa
```

Figura 27 - Dados do nó Spark

(g) Desconecte-se do Spark utilizando $spark\_disconnect(ss)$. Verifique programaticamente que a ligação foi fechada.

Para confirmar que a ligação foi fechada, utiliza-se o método $spark\_connection\_is\_open(ss)$

```
> spark_connection_is_open(ss)
[1] TRUE
> spark_disconnect(ss)
> spark_connection_is_open(ss)
[1] FALSE
```

Figura 28 - Verificação programática de conexão ao Spark

Este troço de código pode ser colocado num script R, para garantir que antes do mesmo terminar, a ligação ao Spark é fechada.

(h) Indique, um algoritmo de aprendizagem supervisionada implementado no Spark e disponível no pacote "sparklyr".

O algoritmo *Random Forest* é uma técnica de machine learning que pertence à categoria de aprendizagem supervisionada. Durante o treino de cada árvore, o *Random Forest* utiliza uma abordagem de amostragem aleatória, o que aumenta a diversidade entre as árvores e melhora a generalização do modelo. A previsão feita pelo *Random Forest* é realizada através da soma das previsões de todas as árvores, por fim é realizado um voto de maioria absoluta para a previsão final.

(i) Indique, um algoritmo de aprendizagem não supervisionada implementado no Spark e disponível no pacote "sparklyr".

O algoritmo *K-Means Clustering* é um método de aprendizagem não supervisionada utilizado para agrupar dados sem rótulos em clusters distintos. O algoritmo começa por agrupar vários conjuntos de dados em grupos tendo por base a proximidade das suas coordenadas. De seguida, são criados pontos, com coordenadas aleatórias, que funcionam como centroides para os grupos de dados. As coordenadas dos centroides são então recalculadas com base nos pontos atribuídos a cada cluster, o processo é repetido até que as coordenadas deixem de mudar significativamente.

(j) Indique de que forma o Spark pode apoiar o *pipeline* de processamento de megadados.

O Spark torna-se uma ferramenta muito poderosa e essencial no processamento de megadados num *pipeline* ao garantir que consegue lidar com grandes volumes de dados (megadados) de forma distribuída (paralela) em vários nós. O Spark Core possui APIs para diversas linguagens de programação que garantem um conjunto de operações de tratamento dos dados, como a filtragem e mapeamento. Em simultâneo, disponibiliza o serviço de *Streaming*, que permite ingerir e analisar dados em tempo real. Para consulta de dados, o Spark SQL oferece suporte para JDBC, ODBC, JSON, entre outras [1] [2].

O Spark possui também uma biblioteca de *Machine Learning*, com diversos algoritmos, oferecendo *pipelines* que permitem construir e implantar modelos de forma eficiente, e em grande escala [3].

4. Conclusão

Durante a elaboração do primeiro laboratório da unidade curricular de Mineração de Dados em Larga Escala, o grupo explorou as capacidades básicas da linguagem R, e a funcionalidade do Apache Spark, e de que forma permite desenvolver aplicações de *machine learning* em vários nós e *clusters*.

Foram estudados os elementos mais básicos, como escalares, vetores e matrizes, bem como os *data frames*, e de que forma podem ser utilizados para carregar nós no Spark.

Compreendeu-se que a simplicidade da linguagem R complementada pela capacidade de processamento distribuído do Spark, proporciona uma poderosa solução para explorar padrões complexos e extrair insights valiosos em ambientes de Big Data, contribuindo para uma análise robusta e informada.

5. Referências Bibliográficas

5.1 Webgrafia

[1] https://aws.amazon.com/pt/what-is/apache-spark/
[2] https://www.ksolves.com/blog/big-data/apache-spark-kafka-your-big-data-pipeline
[3] https://www.altexsoft.com/blog/apache-spark-pros-cons/