

Atividade em Laboratório Virtual 03

Disciplina	FAM – Fundamentos de Aprendizado de Máquina
-------------------	--

Objetivos

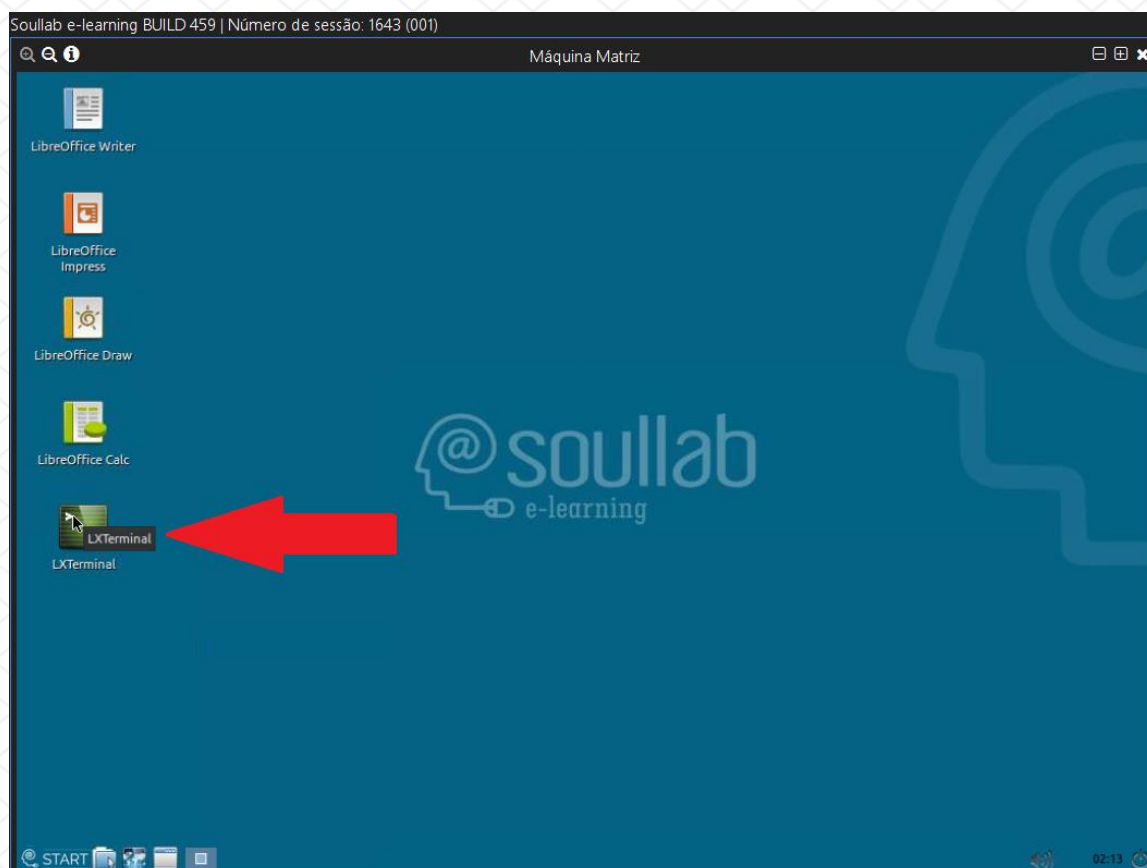
A terceira atividade prática em laboratório virtual possui como objetivo principal o uso do algoritmo de Árvore de Decisão por meio do framework Apache Spark e da biblioteca de Machine Learning MLlib.

Ao final desta atividade, o aluno deverá ser capaz de executar um experimento com o algoritmo de Árvore de Decisão utilizando o Apache Spark e a linguagem Scala.

Enunciado

No momento de iniciar esta atividade é esperado que o aluno já tenha recebido seus dados de login para acesso ao laboratório virtual.

Após executar o login no laboratório virtual, inicie o Terminal do Linux, conforme a imagem abaixo:



Foi criado, no ambiente virtual, um usuário exclusivo para trabalhar com o Spark. O nome desse usuário é **spuser** e a sua senha é **spark**. Você deve fazer login com o usuário **spuser** e, para isso, execute o comando abaixo:

```
su spuser
```

Após digitar o comando acima, o sistema operacional irá solicitar a digitação da senha. Informe a senha “**spark**” (desconsidere as aspas).

A ferramenta Spark já foi previamente instalada no ambiente virtual e encontra-se no diretório `/usr/local/spark`.

Em seguida você deverá executar o Spark Shell. Para isso utilize o seguinte comando:

```
/usr/local/spark/bin/spark-shell
```

Nesse momento o Spark Shell será carregado e o seguinte resultado aparecerá na tela:

```
spuser@ip-172-31-26-173: /usr/local/spark/MPPApp
File Edit Tabs Help
r your platform... using builtin-java classes where applicable
18/03/31 23:56:10 WARN ObjectStore: Version information not found in metastore.
hive.metastore.schema verification is not enabled so recording the schema version
1.2.0
18/03/31 23:56:10 WARN ObjectStore: Failed to get database default, returning No
SuchObjectException
18/03/31 23:56:12 WARN ObjectStore: Failed to get database global_temp, returnin
g NoSuchObjectException
Spark context Web UI available at http://172.31.26.173:4040
Spark context available as 'sc' (master = local[*], app id = local-1522540559498
).
Spark session available as 'spark'.
Welcome to

      _/_/_/_/_/_/_/_/_/_/_/_/_/_/_/__
     /_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_\
    /_/_/_/_/_/_/_/_/_/_/_/_/_/_/_\_ \
   /_/_/_/_/_/_/_/_/_/_/_/_/_/_/_\_ \
  /_/_/_/_/_/_/_/_/_/_/_/_/_/_/_\_ \
 /_/_/_/_/_/_/_/_/_/_/_/_/_/_/_\_ \
/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_\_ \

version 2.1.0

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_111)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Os dados que serão utilizados no experimento já foram baixados e se encontram no diretório `/usr/local/exemplosml/Atividade3/Dados/arvore.txt`.

O objetivo desse experimento é analisar dados de automóveis e prever a quilometragem em 3 categorias: ruim (0), boa (1) e ok (2). Essa classificação será feita com base no número de cilindros (cylinders), deslocamento (displacement), potência (horsepower), peso (weight), tempo de aceleração até 60mph (acceleration), ano de fabricação (modelyear) e continente de origem (maker). Abaixo a tabela com alguns dados de exemplo:

MPG	CYLINDERS	DISPLACEMENT	HORSEPOWER	WEIGHT	ACCELERATION	MODELYEAR	MAKER
Bad	8	350	150	4699	14.5	74	America
Bad	8	400	170	4746	12	71	America
Bad	8	400	175	4385	12	72	America
Bad	6	250	72	3158	19.5	75	America
OK	4	121	98	2945	14.5	75	Asia
OK	6	232	90	3085	17.6	76	America
OK	4	120	97	2506	14.5	72	Europe

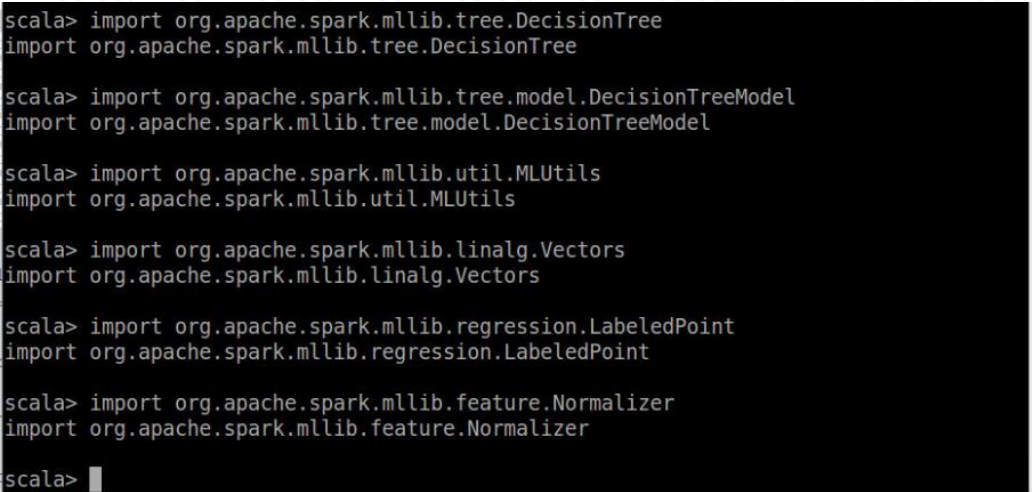
Todos os dados precisam ser transformados em um formato numérico e o arquivo que se encontra em `/usr/local/exemplosml/Atividade3/Dados/arvore.txt` já

sofreu essa transformação, além de receber mais alguns registros, em comparação com a tabela que foi apresentada acima.

Primeiramente, já dentro do Spark Shell, vamos incluir as bibliotecas necessárias para execução do experimento. São elas:

```
import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.mllib.tree.model.DecisionTreeModel
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.feature.Normalizer
```

Cada uma dessas linhas, que representam as bibliotecas do MLlib, deverá ser digitada no Spark Shell, conforme a figura abaixo:



```
scala> import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.mllib.tree.DecisionTree

scala> import org.apache.spark.mllib.tree.model.DecisionTreeModel
import org.apache.spark.mllib.tree.model.DecisionTreeModel

scala> import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.mllib.util.MLUtils

scala> import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Vectors

scala> import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LabeledPoint

scala> import org.apache.spark.mllib.feature.Normalizer
import org.apache.spark.mllib.feature.Normalizer

scala> 
```

Nesse momento nós vamos utilizar o método *textFile* para ler o arquivo *arvore.txt*, já utilizando uma função *Map* para dividir as linhas do arquivo em elementos do RDD. O comando se encontra abaixo:

```
val data = sc.textFile("/usr/local/exemplosml/Atividade3/Dados/arvore.txt").map(
  l => l.split("\\s+"))
```

Após a execução do comando acima, a seguinte tela será exibida. Aproveite e execute as ações *count* e *first* para verificar o conteúdo do RDD.

```
scala> val data = sc.textFile("/usr/local/exemplosml/Atividade3/Dados/arvore.txt").map(
  l => l.split("\\s+"))
data: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[65] at map at <console>:30

scala> data.count
res11: Long = 42

scala> data.first
res12: Array[String] = Array(0, 8, 350, 150, 4699, 14.5, 74, 0)
```

Em seguida vamos converter cada *String* do RDD “data” em *double*. Para isso, utilize o comando abaixo:

```
var dbl = data.map (m => m.map(l => l.toDouble))
```

Após a execução do comando, a seguinte tela será apresentada.

```
scala> var dbl = data.map (m => m.map(l => l.toDouble))
dbl: org.apache.spark.rdd.RDD[Array[Double]] = MapPartitionsRDD[66] at map at <console>:32

scala> dbl.count
res13: Long = 42

scala> dbl.first
res14: Array[Double] = Array(0.0, 8.0, 350.0, 150.0, 4699.0, 14.5, 74.0, 0.0)
```

Agora vamos criar um *Dense Vector* e um *LabeledPoint*. Um *Dense Vetor* é um tipo especial de vetor que não manipula valores em branco. O *LabeledPoint* é um objeto com rótulo (*label*) e características (*features*). O rótulo (*label*) é 0, 1 e 2 e as características são os outros dados: peso, cilindros, ano, etc.

O comando para criar o *Dense Vector* e o *LabeledPoint* está apresentado abaixo:

```
val parsedData = dbl.map { y =>
  val c = Vectors.dense(y(1), y(2), y(3), y(4), y(5), y(6), y(7))
  LabeledPoint(y(0), c)}.cache()
```

A figura abaixo apresenta o resultado:


```
scala> val parsedData = dbl.map { y =>
  | val c = Vectors.dense(y(1),y(2),y(3),y(4),y(5),y(6),y(7))
  | LabeledPoint(y(0), c)
  | }.cache()
parsedData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint]
= MapPartitionsRDD[67] at map at <console>:34

scala> parsedData.first
18/04/01 18:39:56 WARN Executor: 1 block locks were not released by TID = 35:
[rdd_67_0]
res15: org.apache.spark.mllib.regression.LabeledPoint = (0.0,[8.0,350.0,150.0,4699.0,14.5,74.0,0.0])
```

O número de classes, significa o número de classificações que o algoritmo poderá fazer. No nosso caso, serão 3 (0, 1 ou 2).

```
val numClasses = 3
```

```
scala> val numClasses = 3
numClasses: Int = 3
```

Nesse momento precisamos informar para o algoritmo que a sexta característica é categorizada com 3 possíveis valores (American, European ou Asian). O resto dos valores são contínuos. O código para criar essas categorias está disponível abaixo:

```
val categoricalFeaturesInfo = Map(6 -> 3)
val impurity = "gini"
val maxDepth = 9
val maxBins = 7
```

O resultado está apresentado na figura abaixo:

```
scala> val categoricalFeaturesInfo = Map(6 -> 3)
categoricalFeaturesInfo: scala.collection.immutable.Map[Int,Int] = Map(6 -> 3)

scala> val impurity = "gini"
impurity: String = gini

scala> val maxDepth = 9
maxDepth: Int = 9

scala> val maxBins = 7
maxBins: Int = 7
```

Agora o modelo será “alimentado” com os dados. O comando se encontra abaixo:

```
val model = DecisionTree.trainClassifier(parsedData, numClasses,
```

```
categoricalFeaturesInfo , impurity, maxDepth, maxBins)
```

Após a execução a seguinte tela será apresentada:

```
scala> val model = DecisionTree.trainClassifier(parsedData, numClasses, categoricalFeaturesInfo , impurity, maxDepth, maxBins)
18/04/01 18:40:32 WARN Executor: 1 block locks were not released by TID = 36: [rdd_67_0]
model: org.apache.spark.mllib.tree.model.DecisionTreeModel = DecisionTreeModel classifier of depth 5 with 21 nodes
```

Finalmente, podemos consultar o resultado que é a árvore de decisão. Para isso, digite o seguinte comando:

```
println("Learned classification tree model:\n" + model.toDebugString)
```

A figura abaixo apresenta a árvore que foi gerada:

```
scala> println("Learned classification tree model:\n" + model.toDebugString)
Learned classification tree model:
DecisionTreeModel classifier of depth 5 with 21 nodes
  If (feature 3 <= 2930.0)
    If (feature 1 <= 91.0)
      Predict: 2.0
    Else (feature 1 > 91.0)
      If (feature 5 <= 81.0)
        If (feature 2 <= 105.0)
          Predict: 1.0
        Else (feature 2 > 105.0)
          If (feature 0 <= 4.0)
            Predict: 0.0
          Else (feature 0 > 4.0)
            Predict: 1.0
        Else (feature 5 > 81.0)
          If (feature 2 <= 85.0)
            Predict: 0.0
          Else (feature 2 > 85.0)
            Predict: 2.0
      Else (feature 3 > 2930.0)
        If (feature 3 <= 3193.0)
          If (feature 2 <= 85.0)
            Predict: 0.0
```

Referências:

Rowe. Walker; Spark Decision Tree Classifier. Disponível em:

<https://www.bmc.com/blogs/spark-decision-tree-classifier/> Acesso em: 01/03/2018.