UNIVERSIDAD DE JAÉN

Computación Distribuida para la gestión de datos a gran escala

Práctica 2.2. Spark

Sergio Perea De La Casa

Máster Universitario en Ingeniería Informática

Ejercicio 1

Transforma el RDD de las instancias en un RDD numérico de forma que cada línea sea un array de valores numéricos. Para ello se ha de eliminar el último atributo que es la clase y de tipo nominal.

Unset

```
scala> val rdd = sc.textFile("../iris.dat")
rdd: org.apache.spark.rdd.RDD[String] = ../iris.dat
MapPartitionsRDD[1] at textFile at <console>:23

scala> val datos = rdd.filter(rdd =>
!(rdd.contains("@")))
datos: org.apache.spark.rdd.RDD[String] =
MapPartitionsRDD[2] at filter at <console>:23


scala> val instancia = datos.map(line =>
line.split(", "))
instancia: org.apache.spark.rdd.RDD[Array[String]] =
MapPartitionsRDD[3] at map at <console>:23


scala> val nueva_instancia = instancia.map(line =>
line.slice(0, 4))
nueva_instancia:
org.apache.spark.rdd.RDD[Array[String]] =
MapPartitionsRDD[4] at map at <console>:2


scala> val resultado_1 = nueva_instancia.map(line =>
line.map(_.toDouble))
resultado_1: org.apache.spark.rdd.RDD[Array[Double]]
= MapPartitionsRDD[5] at map at <console>:23
```


```
scala> val rdd = sc.textFile("../iris.dat")
rdd: org.apache.spark.rdd.RDD[String] = ../iris.dat MapPartitionsRDD[1] at textFile at <console>:23
```

```
scala> val datos = rdd.filter(rdd => !(rdd.contains("@")))
datos: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at <console>:23
```

```
scala> val instancia = datos.map(line => line.split(", "))
instancia: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[3] at map at <console>:23
```

```
scala> val nueva_instancia = instancia.map(line => line.slice(0, 4))
nueva_instancia: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[4] at map at <console>:23
```

```
scala> val resultado_1 = nueva_instancia.map(line => line.map(_.toDouble))
resultado_1: org.apache.spark.rdd.RDD[Array[Double]] = MapPartitionsRDD[5] at map at <console>:23
```

```
scala> resultado_1.collect()
res0: Array[Array[Double]] = Array(Array(5.1, 3.5, 1.4, 0.2), Array(4.9, 3.0, 1.4, 0.2), Array(4.6, 3.1, 1.5, 0.2), Arra
y(5.0, 3.6, 1.4, 0.2), Array(5.4, 3.9, 1.7, 0.4), Array(4.6, 3.4, 1.4, 0.3), Array(5.0, 3.4, 1.5, 0.2), Array(4.4, 2.9,
1.4, 0.2), Array(5.4, 3.7, 1.5, 0.2), Array(4.8, 3.4, 1.6, 0.2), Array(4.8, 3.0, 1.4, 0.1), Array(4.3, 3.0, 1.1, 0.1), A
rray(5.7, 4.4, 1.5, 0.4), Array(5.4, 3.9, 1.3, 0.4), Array(5.1, 3.5, 1.4, 0.3), Array(5.7, 3.8, 1.7, 0.3), Array(5.1, 3.
8, 1.5, 0.3), Array(5.4, 3.4, 1.7, 0.2), Array(5.1, 3.7, 1.5, 0.4), Array(4.6, 3.6, 1.0, 0.2), Array(5.1, 3.3, 1.7, 0.5)
, Array(4.8, 3.4, 1.9, 0.2), Array(5.0, 3.0, 1.6, 0.2), Array(5.0, 3.4, 1.6, 0.4), Array(5.2, 3.5, 1.5, 0.2), Array(5.2,
 3.4, 1.4, 0.2), Array(4.7, 3.2, 1.6, 0.2), Array(4.8, 3.1, 1.6, 0.2), Array(...
```

Ejercicio 2

Calcula la suma de todos los valores de todos los atributos

Unset

```
scala> val sum = resultado_1.map(line => line.sum).sum
sum: Double = 2078.2000000000003
```


```
scala> val sum = resultado_1.map(line => line.sum).sum
sum: Double = 2078.2000000000003
```


Ejercicio 3

Calcula la suma de todos los valores de cada instancia (aunque esto no tendría sentido para el aprendizaje del modelo).

Unset

```
scala> val sum = resultado_1.map(line => line.sum)
sum: org.apache.spark.rdd.RDD[Double] = MapPartitionsRDD[7] at
map at <console>:23
```

```
scala> val sum = resultado_1.map(line => line.sum)
sum: org.apache.spark.rdd.RDD[Double] = MapPartitionsRDD[7] at map at <console>:23
```

```
scala> sum.collect()
res4: Array[Double] = Array(10.2, 9.5, 9.399999999999999, 10.2, 11.4, 9.700000000000001, 10.1, 8.9, 10.8, 9.999999999999998, 9.29
99999999999999, 8.5, 12.000000000000002, 11.000000000000002, 10.3, 11.5, 10.7, 10.7, 10.700000000000001, 9.399999999999999, 10.599
999999999998, 10.299999999999999, 9.799999999999999, 10.4, 10.399999999999999, 10.2, 9.7, 9.7, 10.700000000000001, 10.9, 11.29999
9999999999, 9.6, 9.599999999999998, 10.5, 9.6, 8.9, 10.2, 10.100000000000001, 8.4, 11.2, 9.5, 10.699999999999998, 9.3999999999999
99, 10.7, 9.9, 16.299999999999997, 15.600000000000001, 16.4, 13.100000000000001, 14.3, 15.9, 11.600000000000001, 13.2000000000000
01, 11.5, 14.600000000000001, 13.2, 15.1, 13.4, 15.600000000000001, 14.6, 13.6, 14.4, 13.1, 15.700000000000003, 14.2, 15.20000000
0000001, 14.79999999999...
```

Ejercicio 4

Comprueba que la suma calculada en el apartado 2 coincide con la que se haría sumando todos los resultados obtenidos en el apartado 3.

Unset

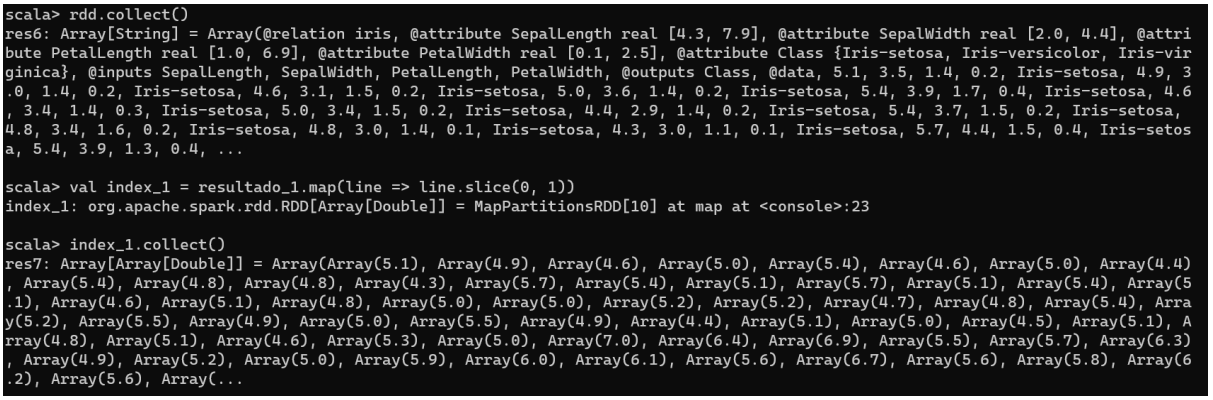
```
//Para esto, lo unico que tenemos que hacer es sumar cada uno
de los valores de las instancias en la suma por instancias.

scala> val sum_instancias_total = sum.sum
sum_instancias_total: Double = 2078.2000000000003

scala> val sum = resultado_1.map(line => line.sum).sum
sum: Double = 2078.2000000000003
```


```
scala> val sum_instancias_total = sum.sum
sum_instancias_total: Double = 2078.2000000000003

scala> val sum = resultado_1.map(line => line.sum).sum
sum: Double = 2078.2000000000003
```

Ejercicio 5

Calcula la suma de todos los valores para el atributo SepalLength, su media, valor máximo y mínimo. Se puede comprobar que estos dos últimos valores coinciden con los valores de rango indicados para el atributo en la cabecera.

```
Unset
scala> val index_1 = resultado_1.map(line => line.slice(0, 1))
index_1: org.apache.spark.rdd.RDD[Array[Double]] =
MapPartitionsRDD[10] at map at <console>:23

scala> val resultado_2 = index_1.map(line => line.sum).sum
resultado_2: Double = 876.4999999999998

scala> val num_items = index_1.count
num_items: Long = 150

scala> val suma_1 = resultado_2 / num_items
suma_1: Double = 5.843333333333332

scala> val max_index_1 = index_1.sortBy(-_(0)).first.seq(0)
max_index_1: Double = 7.9

scala> val max_index_1 =
index_1.sortBy(-_(0),false).first.seq(0)
max_index_1: Double = 4.3
```



```
scala> val resultado_2 = index_1.map(line => line.sum).sum
resultado_2: Double = 876.4999999999998
```

```
scala> val num_items = index_1.count
num_items: Long = 150
```

```
scala> val suma_1 = resultado_2 / num_items
suma_1: Double = 5.843333333333332
```

```
scala> val max_index_1 = index_1.sortBy(-_(0)).first.seq(0)
max_index_1: Double = 7.9
```

```
scala> val max_index_1 = index_1.sortBy(-_(0),false).first.seq(0)
max_index_1: Double = 4.3
```

Ejercicio 6

Calcula lo mismo que en el anterior, la suma, media, valor máximo y mínimo, pero para todos los atributos de entrada que hubiese

@attribute SepalWidth real [2.0, 4.4]

Unset

```
scala> val index_2 = resultado_1.map(line => line.slice(1, 2))
index_2: org.apache.spark.rdd.RDD[Array[Double]] =
MapPartitionsRDD[45] at map at <console>:23


scala> index_2.map(line => line.sum).sum
res11: Double = 458.09999999999997

scala> val num_items_2 = index_2.count
num_items_2: Long = 150
```

```
scala> val suma_2 = index_2.map(line => line.sum).sum
suma_2: Double = 458.09999999999997

scala> val media_2 = suma_2 / num_items_2
media_2: Double = 3.054


scala> val max_index_2 = index_2.sortBy(-_(0)).first.seq(0)
max_index_2: Double = 4.4

scala> val min_index_2 =
index_2.sortBy(-_(0),false).first.seq(0)
min_index_2: Double = 2.0
```

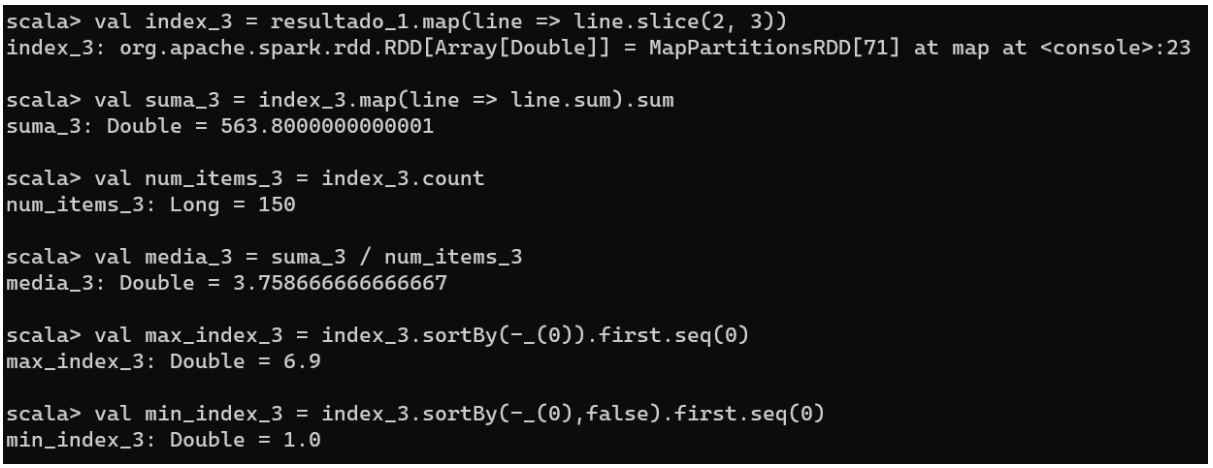
```
scala> val index_2 = resultado_1.map(line => line.slice(1, 2))
index_2: org.apache.spark.rdd.RDD[Array[Double]] = MapPartitionsRDD[4
5] at map at <console>:23
```

```
scala> index_2.map(line => line.sum).sum
res11: Double = 458.09999999999997

scala> val num_items_2 = index_2.count
num_items_2: Long = 150

scala>

scala> val suma_2 = index_2.map(line => line.sum).sum
suma_2: Double = 458.09999999999997

scala> val media_2 = suma_2 / num_items_2
media_2: Double = 3.054
```

```
scala> val max_index_2 = index_2.sortBy(-_(0)).first.seq(0)
max_index_2: Double = 4.4

scala> val min_index_2 = index_2.sortBy(-_(0),false).first.seq(0)
min_index_2: Double = 2.0
```

@attribute PetalLength real [1.0, 6.9]

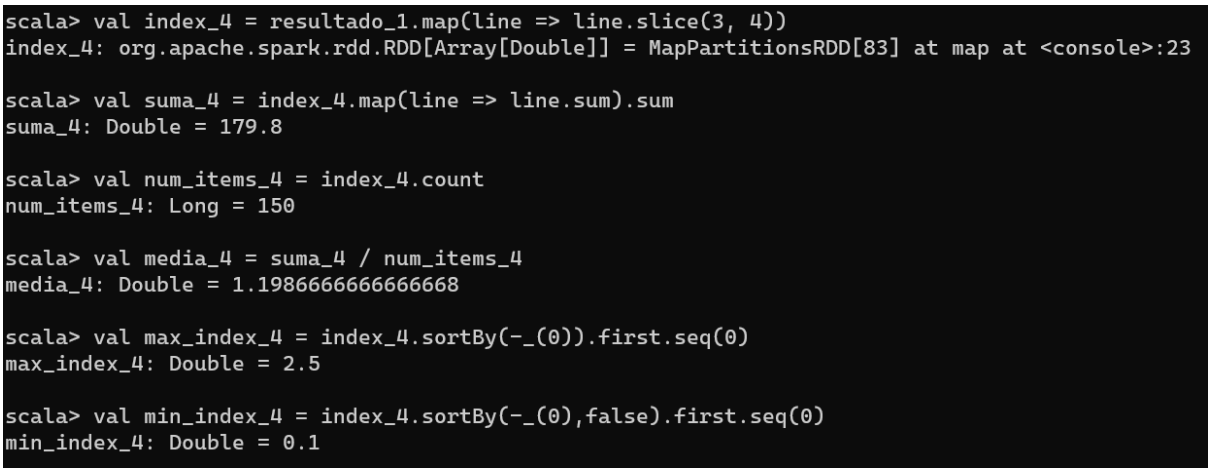
```
Unset
scala> val index_3 = resultado_1.map(line => line.slice(2, 3))
index_3: org.apache.spark.rdd.RDD[Array[Double]] =
MapPartitionsRDD[71] at map at <console>:23

scala> val suma_3 = index_3.map(line => line.sum).sum
suma_3: Double = 563.8000000000001

scala> val num_items_3 = index_3.count
num_items_3: Long = 150

scala> val media_3 = suma_3 / num_items_3
media_3: Double = 3.758666666666667

scala> val max_index_3 = index_3.sortBy(-_(0)).first.seq(0)
max_index_3: Double = 6.9

scala> val min_index_3 =
index_3.sortBy(-_(0),false).first.seq(0)
min_index_3: Double = 1.0
```



@attribute PetalWidth real [0.1, 2.5]

Unset

```
scala> val index_4 = resultado_1.map(line => line.slice(3, 4))
index_4: org.apache.spark.rdd.RDD[Array[Double]] =
MapPartitionsRDD[83] at map at <console>:23

scala> val suma_4 = index_4.map(line => line.sum).sum
suma_4: Double = 179.8

scala> val num_items_4 = index_4.count
num_items_4: Long = 150

scala> val media_4 = suma_4 / num_items_4
media_4: Double = 1.1986666666666668

scala> val max_index_4 = index_4.sortBy(-_(0)).first.seq(0)
max_index_4: Double = 2.5

scala> val min_index_4 =
index_4.sortBy(-_(0),false).first.seq(0)
min_index_4: Double = 0.1
```



Ejercicio 7

Captura el DAG de alguna de las operaciones que has realizado.