UNIVERSIDAD DE JAÉN

Computación Distribuida para la gestión de datos a gran escala

Práctica 2.1. Spark

Sergio Perea De La Casa

Máster Universitario en Ingeniería Informática

Ejercicio 1 **3**

Crea un RDD con los datos de la cabecera y otro con los datos de las instancias. Muestra el número de líneas de cabecera que hay. 3

Ejercicio 2 **4**

Muestra el nombre del dataset. 4

Ejercicio 3 **5**

Cuenta el número de atributos de entrada. 5

Ejercicio 4 **6**

Obtén los nombres de los atributos de entrada a partir de las líneas que empiezan por @atribute y también a partir de la línea que empieza por @inputs. Comprueba que coindicen. 6

Ejercicio 5 **7**

Cuenta el número de instancias (puedes comprobar mirando en Keel que el número es correcto). 7

Ejercicio 6 **8**

Igual que antes, pero considerando que puede haber líneas en blanco entre los ejemplos que no deben ser contadas. 8

Ejercicio 1

Crea un RDD con los datos de la cabecera y otro con los datos de las instancias. Muestra el número de líneas de cabecera que hay.

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

//Creamos el RDD con solo la cabecera
scala> val cabecera = dataset.filter(dataset => dataset.contains("@"))
cabecera: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at <console>:23

//Creamos el RDD con el resto
scala> val datos = dataset.filter(dataset => !(dataset.contains("@")))
datos: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[4] at filter at <console>:23

//Muestra el número de líneas de cabecera que hay
scala> cabecera.count
res0: Long = 9
```

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

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

```
scala> val datos = dataset.filter(dataset => !(dataset.contains("@")))
datos: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[3] at filter at <console>:23
```

```
scala> cabecera.count
res0: Long = 9
```

Ejercicio 2

Muestra el nombre del dataset.

scala> val nombre_dataset = cabecera.filter(cabecera =>
cabecera.contains("@relation"))
nombre_dataset: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[5] at filter at <console>:23

//Aplicamos el collect() para que realice
scala> nombre_dataset.collect()
res1: Array[String] = Array(@relation iris)

// Como collect() devuelve un ARRAY, indicamos que imprima cada uno de sus elementos
scala> nombre_dataset.collect().foreach(println)
@relation iris

```
scala> val nombre_dataset = cabecera.filter(cabecera => cabecera.contains("@relation"))
nombre_dataset: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[5] at filter at <console>:23
```

```
scala> nombre_dataset.collect()
res1: Array[String] = Array(@relation iris)
```

```
scala> nombre_dataset.collect().foreach(println)
@relation iris
```

Ejercicio 3

Cuenta el número de atributos de entrada.

//Buscamos la entrada en la cabecera
scala> val atributosentrada = cabecera.filter(cabecera => cabecera.contains("@inputs"))
atributosentrada: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[6] at filter at <console>:23

//Filtramos el resultado a partir de un Split de las líneas de los atributos de entrada y después otro Split para que reconozca cada palabra separada por comillas. Posteriormente, hacemos un filtrado para que la palabra @inputs para que no entre como un atributo.
scala> val resultado = atributosentrada.flatMap(line => line.split(" ")).flatMap(line => line.split(",")).filter(line => !(line.contains("@inputs")))
resultado: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[9] at filter at <console>:23

//Sacamos el conteo
scala> resultado.count
res8: Long = 4

```
scala> val atributosentrada = cabecera.filter(cabecera => cabecera.contains("@inputs"))
atributosentrada: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[6] at filter at <console>:23

scala> val resultado = atributosentrada.flatMap(line => line.split(" ")).flatMap(line => line.split(",")).filter(line => !(line.conta
ins("@inputs")))
resultado: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[9] at filter at <console>:23

scala> resultado.count
res8: Long = 4
```

Ejercicio 4

Obtén los nombres de los atributos de entrada a partir de las líneas que empiezan por @atribute y también a partir de la línea que empieza por @inputs. Comprueba que coindicen.

//Se crea una variable atributos que recibe un filtrado de todas las líneas que contienen la palabra @attribute
scala> val atributos = cabecera.filter(line => line.contains("@attribute"))
atributos: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[14] at filter at <console>:23

//Se crea una variable que recibe un Split por espacio del filtrado anterior y que hace un recorte solo entre la posición 1 y 2 del array resultante que es donde estas los atributos
scala> val resultado_2 = atributos.flatMap(line => line.split(" ").slice(1, 2))
resultado_2: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[15] at flatMap at <console>:23

//Se comprueba que coinciden
scala> resultado.take(10)
res19: Array[String] = Array(SepalLength, SepalWidth, PetalLength, PetalWidth)

scala> resultado_2.take(10)
res20: Array[String] = Array(SepalLength, SepalWidth, PetalLength, PetalWidth, Class)

```
scala> val atributos = cabecera.filter(line => line.contains("@attribute"))
atributos: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[14] at filter at <console>:23


scala> val resultado_2 = atributos.flatMap(line => line.split(" ").slice(1, 2))
resultado_2: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[15] at flatMap at <console>:23
```

```
scala> resultado.take(10)
res19: Array[String] = Array(SepalLength, SepalWidth, PetalLength, PetalWidth)

scala> resultado_2.take(10)
res20: Array[String] = Array(SepalLength, SepalWidth, PetalLength, PetalWidth, Class)
```

Ejercicio 5

Cuenta el número de instancias (puedes comprobar mirando en Keel que el número es correcto).

```
//Se cuenta el número de instancias con la variable creada anteriormente
scala> datos.count
res21: Long = 150
```

Ejercicio 6

Igual que antes, pero considerando que puede haber líneas en blanco entre los ejemplos que no deben ser contadas.

```
//Se cuenta el número de instancias con la variable creada anteriormente
scala> evita_vacios.count
res24: Long = 150
```

```
scala> val evita_vacios = datos.filter(line => !(line.mkString("").isEmpty))
evita_vacios: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[18] at filter at <console>:23

scala> evita_vacios.count
res24: Long = 150
```