Práctica - Spark

Para esta práctica vamos a seguir utilizando la misma VM de la práctica anterior.

Spark

Para correr spark se puede entrar al **spark-shell** (el comando es exactamente ese) o se puede usar spark-shell -i <path>, para que ejecute las instrucciones en el ***path***. Recordar que la diferencia entre spark-shell y el sbt de scala es que el spark-shell tiene instanciados contextos de spark, por lo que se puede usar directamente el val **sc**.

1. Para comenzar y para no perder la costumbre vamos a contar las palabras de libros y sacar el top 20 de las palabras. Vamos a hacerlo paso a paso.
    a. Primero vamos a leer el archivo del libro:

```
val lines =
sc.textFile("file:/home/cloudera/datasets/datasets-small/books/dracula.
txt")
```

    b. Para obtener las palabras queremos primero llevar a minúsculas, tokenizar y eliminar todo lo que no sea letras.

```
val lower = lines.map(line => line.toLowerCase)
val words = lower.flatMap(line => line.split("\\s+")).filter(word =>
!word.isEmpty)
val cleanWords = words.map(word => word.replaceAll("[^a-zA-Z0-9]+", ""))
```

    c. Para realizar el conteo, hacemos el mapeo de cada palabra en la tupla palabra y 1 (la cantidad que fue visto por ahora) y luego agrupamos por palabra y sumamos el valor del conteo.

```
val counts = cleanWords.map(word => (word, 1))
val freq = counts.reduceByKey(_ + _)
```

    d. Para realizar el ordenamiento eficientemente usamos el método **tops** que resuelve un top local en cada partición y luego realiza el top general utilizando solo los candidatos de cada partición. Tops ordena por defecto por el primer elemento de la tupla, por lo cual primero se usa swap.

```
val invFreq = freq.map(_.swap)
val tops = invFreq.top(20)
```

    e. Solo resta mostrar los datos.

```
tops.foreach(println)
```

f. O escribirlos a archivos. Como el resultado top no es un RDD si no un Array por lo que no puede llamarse al *saveAsTextFile*.

```
invFreq.saveAsTextFile("hdfs:/results/spark/wordCount")
```

2. Una operación que puede resultar interesante es obtener el camino de dependencias de nuestro RDD:

```
invFreq.toDebugString
```

3. Sabiendo que algún RDD se puede llegar a reutilizar en diferentes operaciones se puede usar la operación cache (o persist) para que quede calculado en la memoria del cluster. Guardemos el RDD de palabras.

```
cleanWords.cache
```

4. Realice el cálculo de la cantidad de palabras, pero solo para aquellas palabras que comienzan con **'t'**. **Tip:** Scala provee la función startsWith para el tipo String.
5. Queremos saber si hay palabras que coinciden en la cantidad de veces que aparecen en el texto. Para lo cual calcule a partir de la frecuencia de las palabras que ya se cálculo (el word count), la frecuencia de palabras que cayeron en cada valor y detecte si hay cantidades que tienen más de una palabra.c
6. Realice los mismos cálculos realizados hasta este punto pero utilizando todos los libros del set **"dataset-med"**. **Tip:** sc.textFile, puede recibir un directorio y lee todos los archivos del mismo para formar el RDD.
7. A partir de la frecuencia ya calculada se quiere saber por cada frecuencia cuántas palabras hay. O sea si hay 2 palabras cuya frecuencia es 5 veremos una tupla (5, 2.
8. Habiendo obtenido ya el listado de palabras del dataset-med, obtener la frecuencia de palabras según su cantidad de caracteres. **Tip:** Scala provee la función t*oList* para los string que devuelve una lista con los elementos del texto o length para obtener la cantidad de caracteres
9. Obtener el tamaño total en caracteres del texto procesado.

SparkSQL

1. Primero vamos a leer el archivo de películas:

```
val sqlcontext = new org.apache.spark.sql.SQLContext(sc)
val all =
sqlcontext.read.json("file:/home/cloudera/datasets/datasets-small/imdb/imdb-40.jso
n")
```

2. Podemos ver el esquema y contenido de este dataframe.

```
all.printSchema //schema
all.show //content
```

3. Como este archivo contiene series y películas, solo queremos las películas.

```
val movies =  all.filter("Type = 'movie'")
//alternativas que devuelven lo mismo
// val movies =  all.where("Type" === "movie")
// val movies =  all.where(all("Type") === "movie")
```

4. Quiero ver el título de las películas que tengan más de 50 en Metacritic.

```
movies.filter("Metascore > 50").select("Title").show()
```

   En caso de usar las alternativas tipo filter(movies(“Metascore”) > 50), hay que tomar en cuenta que todos los campos son string, asi que habria que cambiar el esquema de alguna manera.

5. Para hacer la misma consulta de manera más familiar, transformemos el Dataframe una tabla para realizar la query.

```
movies.registerTempTable("movies")
sqlcontext.sql("select title from movies where  Metascore > 50").show()
```

6. Listar los actores de las películas.
7. Como pueden ver todos los actores están en la misma columna de la tabla movies. Como nos interesa poder hacer consultas por los actores individualmente vamos a separarlos.

```
val collaboration = movies.select(movies("Title"), movies("Actors"),
movies("Director")).flatMap((tupla) =>
tupla.getAs[String]("Actors").split(',').map(actor =>
(tupla.getAs[String]("Title"), tupla.getAs[String]("Director"),
actor.trim))).distinct

collaboration.take(10).foreach(print)
```

8. El map en el dataframe nos lo transformó en un rdd (escribir `collaboration` y enter para que muestre el tipo), así que vamos a usar una case class para transformar este rdd en un dataframe y registrar la tabla.

```
case class Collaboration(title: String, director: String, actor:String)

val collaborationDF = collaboration.map{case (title, director, actor)=>
Collaboration(title, director, actor)}.toDF

collaborationDF.registerTempTable("collaboration")

sqlContext.sql("select  * from collaboration ").show()
```

   La respuesta del método sql es otro DataFrame, en el caso del ejemplo anterior el contenido del dataframe `collaborationDF`.
9. Teniendo ambas tablas se pueden combinar con sql:

```
sqlContext.sql("select * from movies m join collaboration c on c.title =
m.title").show()
```

10. Para poder exportar el contenido de un dataframe ejecutar:

```
collaborationDF.write.format("json").save("hdfs:/results/spark/collaborations"
                                          )
```

11. Con las tablas creadas obtener:
    a. Quienes son los directores que tienen mejor promedio de metascore
    b. Quienes son los actores más solicitados (que actuaron en más películas)
    c. Quienes son los pares actor, director que más colaboraron juntos.
    d. Quienes son los actores cuyas películas sumaron más en el tomatoMeter.
12. Pensar los mismos ejercicios usando Spark Core (RDD) en vez de SparkSQL.