Práctica - Spark - Apache Zeppelin

Para esta práctica vamos a utilizar una nueva VM que tiene instalados spark y apache zeppelin. El mismo está instalado como un servicio web dentro de un debian server, por lo que al iniciar no se verá un entorno visual, se verá simplemente una consola negra. Está configurado para que una vez iniciado se pueda utilizar desde la máquina host (las laptops con windows) accediendo mediante a cualquier browser (chrome, firefox) a la url **http://localhost:8080/.**

Puede ser que al pasar el mouse sobre la consola la misma “capture” el mouse, para salir apretar una vez el botón **ctrl** derecho del teclado.

Para apagar la máquina virtual, al finalizar la práctica, clikear sobre la consola y apretar enter. Loguearse usando el usuario y password vagrant, y una vez que hay promp escribir:

sudo shutdown now.

Y presionar el botón enter

Dataset

Para evitar problemas de conectividad la virtual ya contienen una serie de archivos que se pueden utilizar para la práctica, los mismos se encuentran en las siguientes direcciones:

```
/home/vagrant/data/bank.csv
/home/vagrant/data/datasets
/home/vagrant/data/datasets/datasets-small
/home/vagrant/data/datasets/datasets-small/temp-hourly
/home/vagrant/data/datasets/datasets-small/temp-hourly/199701hourly.txt
/home/vagrant/data/datasets/datasets-small/books
/home/vagrant/data/datasets/datasets-small/books/dracula.txt
/home/vagrant/data/datasets/datasets-small/book-output-count
/home/vagrant/data/datasets/datasets-small/book-output-count/part-r-00000
/home/vagrant/data/datasets/datasets-small/book-output-count/.part-r-00000.crc
/home/vagrant/data/datasets/datasets-small/imdb
/home/vagrant/data/datasets/datasets-small/imdb/imdb-40.json
/home/vagrant/data/datasets/datasets-med
/home/vagrant/data/datasets/datasets-med/temp-hourly
/home/vagrant/data/datasets/datasets-med/temp-hourly/199701hourly.txt
/home/vagrant/data/datasets/datasets-med/temp-hourly/199609hourly.txt
/home/vagrant/data/datasets/datasets-med/temp-hourly/199607hourly.txt
/home/vagrant/data/datasets/datasets-med/temp-hourly/199612hourly.txt
/home/vagrant/data/datasets/datasets-med/temp-hourly/199611hourly.txt
/home/vagrant/data/datasets/datasets-med/temp-hourly/199608hourly.txt
/home/vagrant/data/datasets/datasets-med/temp-hourly/199610hourly.txt
/home/vagrant/data/datasets/datasets-med/books
/home/vagrant/data/datasets/datasets-med/books/2000010.txt
/home/vagrant/data/datasets/datasets-med/books/dracula.txt
/home/vagrant/data/datasets/datasets-med/books/mobydick.txt
```

/home/vagrant/data/datasets/datasets-med/books/longfellow-village-211.txt
/home/vagrant/data/datasets/datasets-med/books/2city10.txt
/home/vagrant/data/datasets/datasets-med/imdb
/home/vagrant/data/datasets/datasets-med/imdb/imdb-20K.json
/home/vagrant/data/imdb-40.json

Spark-Core

Recordar que en caso de que no tengamos Zeppelin y solo tengamos spark (como tenemos en la virtual de cloudera utilizada anteriormente). Se puede correr las pruebas usando spark-shell y escribiendo el código interactivamente en el shell o alternativamente escribir un archivo con el código y pedirle a spark-shell que lo ejecute invocando:
spark-shell -i <archivo>
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. Pero para hacerlo más cómodo o simple usamos Zeppelin:
    a. Acceder al Zeppelin ingresando **localhost:8080** en un browser,
    b. Buscar la notebook Word Count analizar cada párrafo antes de correrlo. Calcula el top 5 de las frecuencias utilizando los siguientes párrafos:
        i. Presentación.
        ii. Uno que lee las líneas de un libro.
        iii. El que calcula la frecuencia (word count).
        iv. El que saca el top 5
        v. Opcional uno que muestra cómo guardar a archivo y otras cosas.

2. 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.
3. Modifique el cálculo para que solo cuente la frecuencia de las palabras que empiezan con la letra **'t'**. **Tip:** Scala provee la función *startsWith* para el tipo String.
4. 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.
5. 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

6. Obtener el tamaño total en caracteres del texto procesado.

SparkSQL

1. Abra la notebook Zeppelin Tutorial:
    a. analice cómo los párrafos nos permiten a partir de un csv con la información de los clientes de un banco poder consultas tipo sql sobre la información y cómo se pueden hacer consultas parametrizadas simples para uso de personas sin conocimiento de sql.
    b. Agregue un párrafo mostrando por cada nivel educativo la cantidad de cuentas que tienen balance negativo.
    c. Buscar a los clientes de entre 20 y 30 años que estén casados y indicar la suma de los balances de los mismos.
2. Volvamos a la notebook del Word Count. Queremos hacer más fácil consultas del tipo del ejercicio 4 utilizando SparkSQL
    a. Agregue un párrafo nuevo generando el dataframe con la estructura para la frecuencia:

```
//defino la clase
case class WordFrequency(word: String, count: Integer)

// genero el dataframe mediante a un map de la frecuencia
val freqDF = freq.map{case (word, count) => WordFrequency(word, count)}.toDF()
```

    b. En un nuevo párrafo registremos la tabla

```
freqDF.registerTempTable("word_frequency")
```

    c. Y ahora se puede generar párrafos sql:

```
%sql
select word, count from word_frequency order by count DESC LIMIT 10
```

    d. Realice entonces la consulta del ejercicio 4 de Spark Core, pero mostrando sólo la cantidad de palabras para las frecuencias mayores a 5 (va a haber muchas palabras que aparecen 1 o 2 veces, no nos interesan).
3. A continuación utilizaremos la notebook movies para hacer consultas sobre la información de algunas películas provistas por IMDB en formato JSON (o sea ya vienen con un esquema. La notebook ya viene con el código para hacer la lectura del archivo, el resto de los pasos se los iremos agregando en un párrafo por punto:
    a. Podemos (opcionalmente) ver el esquema que generó a partir del JSON y/o el contenido de este DF. Probar y eliminar este párrafo ya que no nos interesa conservarlo.

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

b. El archivo en realidad contienen información de películas y series, queremos filtrarlas para que solo queden 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")
```

c. Quiero ver el título de las películas que tengan más de 50 en Metacritic. Lo vamos a hacer usando los métodos de DataFrame, en vez de registrar la tabla (por ahora)

```
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.

d. 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()
```

e. O aprovechando que Zeppelin nos permite escribir sql directo.

```
%sql
select title from movies where  Metascore > 50
```

f. Generar una consulta que nos permita listar los actores de las películas.

g. 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 (por ejemplo veamos qué actores y directores colaboraron en que peliculas).

```
//selecciono de cada tupla solo el titulo, los actores, y el director.
val cast = movies.select(movies("Title"), movies("Actors"), movies("Director"))

//complicado pero aca trasnformamos la lista de (titulo, pelicula, actores),
// en muchos (titulo, pelicula, actor)
val colaborattion = cast.flatMap((tupla) => tupla.getAs[String]("Actors").split(',').map(actor =>
(tupla.getAs[String]("Title"), tupla.getAs[String]("Director"), actor.trim))).distinct
```

```
//imprimo para ver que quedo
collaboration.take(10).foreach(print)
```

h. El map en el dataframe nos lo transformó en un rdd nuevo, así que vamos a usar una" case class" para transformar este rdd en un dataframe y registrar la nueva 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`.

i. Teniendo ambas tablas se pueden combinar con sql:

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

o (por Zeppelin)

```
%sql
select * from movies m join collaboration c on c.title = m.title
```

j. Para poder exportar el contenido de un dataframe a hdfs se podría ejecutar (hdfs no está configurado en la virtual de Zeppelin):

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

k. o al fileSystem:

```
collaborationDF.write.format("json").save("file:/home/vagrant/collabora
                                tions")
```

l. Con las tablas creadas obtener:
   i. Quienes son los directores que tienen mejor promedio de metascore
   ii. Quienes son los actores más solicitados (que actuaron en más películas)
   iii. Quienes son los pares actor, director que más colaboraron juntos.

iv. Quienes son los actores cuyas películas sumaron más en el tomatoMeter.

m. Pensar los mismos ejercicios usando Spark Core (RDD) en vez de SparkSQL.