

Sistemas de Gestión de Datos y de la Información
Máster en Ingeniería Informática, 2020-21
Práctica MapReduce - Apache Spark

Fecha de entrega: domingo 29 de noviembre de 2020, 23:55h

Entrega de la práctica

La práctica se entregará en un único fichero **ZIP** mediante el Campus Virtual de la asignatura. El fichero estará organizado en carpetas por cada apartado. Dentro de estas carpetas habrá un fichero `.py` por cada tarea resuelta.

Lenguaje de programación

3.6 o superior

Declaración de autoría e integridad

Todos los ficheros entregados contendrán una cabecera en la que se indique la asignatura, la práctica, el grupo y los autores. Esta cabecera también contendrá la siguiente declaración de integridad:

Yo, *Nombre Alumno*, declaro que esta solución es fruto exclusivamente de mi trabajo personal. No he sido ayudado por ninguna otra persona ni he obtenido la solución de fuentes externas, y tampoco he compartido mi solución con nadie. Declaro además que no he realizado de manera deshonesto ninguna otra actividad que pueda mejorar mis resultados ni perjudicar los resultados de los demás.

No se corregirá ningún fichero que no venga acompañado de dicha cabecera.


```
>>> counts = pairs.reduceByKey(lambda x,y: x+y)
>>> counts.collect()
[('hola', 1), ('que', 1), ('al', 1), ('sol', 3), ('el', 2), ('pega', 2),
 ('haces', 1), ('me', 1), ('voy', 1)]
```

Una vez hayáis hecho las pruebas pertinentes en `pyspark`, debéis escribir el programa Python completo, en este caso `word_count_spark.py`. Este se lanza desde el **terminal** con el comando:

```
$ spark-submit word_count_spark.py texto.txt
```

Los parámetros detrás del nombre del programa Python, si los hay, se pasarán como argumentos del programa a través de `sys.argv`. Al igual que con *mrjob*, la salida del programa principal se mostrará por la salida estándar y los mensajes informativos y de depuración se mostrarán por la salida de error. Para esta práctica, **los programas Spark deben mostrar el resultado del cómputo por la salida estándar y no escribirlos en disco.**

1) Palabras en sci.space [3.5pt]

En este apartado debes realizar una tarea MapReduce **usando mrjob** para procesar colecciones de mensajes procedentes de grupos de noticias. Concretamente, queremos conocer en qué documento se repite más veces cada palabra junto con el número de repeticiones en dicho documento. A la hora de detectar palabras no se debe hacer distinción entre mayúsculas y minúsculas, y las palabras no deben contener signos de puntuación. Es necesario que la tarea sea capaz de procesar la colección de mensajes sobre el espacio (**space.json**)² que se puede descargar del Campus Virtual. Este fichero contiene un documento JSON por cada línea, que representa un mensaje diferente en el foro. Cada JSON tiene dos campos: **filename** y **content**.

A cada palabra del listado resultante le debe acompañar una pareja <fichero, #repeticiones>, donde **fichero** es el nombre del fichero (únicamente el nombre, no es necesaria la ruta absoluta) y **#repeticiones** es el número de repeticiones de dicha palabra en ese fichero. La salida debe ser un listado con un aspecto como el que sigue (ten en cuenta que **space.json** contiene ficheros cuyo nombre es un número de 5 cifras):

```
...
surely      <59907, 1>
surface     <61435, 48>
surfaces    <60922, 2>
...
```

Nota: Se valorará el uso de un **combinador** para tratar de reducir el número de parejas emitidas.

²Extraído de <http://qwone.com/~jason/20Newsgroups/>

2) Felicidad de «Los Simpsons» [3.5pt]

El conjunto de datos que podéis encontrar en <https://www.kaggle.com/prashant111/the-simpsons-dataset> contiene mucha información sobre los episodios de «Los Simpsons». En este apartado nos centraremos en el fichero `simpsons_script_lines.csv` que contiene todas las frases que aparecen en cada uno de los capítulos (diálogos, carteles, descripciones, etc.). En base a las palabras que aparecen en estos capítulos, nos gustaría obtener la «alegría acumulada» de cada uno de ellos y mostrarlos de mayor a menor.

Para obtener la «alegría acumulada» seguiremos un enfoque bastante simple: usaremos el fichero `happiness.txt` del ejercicio del 12 de noviembre y por cada capítulo sumaremos la felicidad media (`happiness_average`) de todas sus palabras. Si alguna palabra no aparece en el fichero, no sumaremos nada. Recordad que el archivo `happiness.txt` está separado por **tabuladores**, donde cada línea tiene el siguiente formato:

```
word happiness_rank happiness_average happiness_standard_deviation twitter_rank
google_rank nyt_rank lyrics_rank
```

Antes de empezar, revisad con cuidado el fichero `simpsons_script_lines.csv` para entender bien qué hay en cada campo. El texto aparece varias veces, y en algunos campos es más fácil de procesar que en otros.

El resultado final que se debe mostrar por pantalla es un listado de identificador de episodio junto con su «felicidad acumulada», ordenados de mayor a menor felicidad. La salida debería parecerse a la siguiente (*los valores no son reales*):

```
234 5487.21
548 5208.99
33 5102.87
...
```

Nota: se valorará el uso acumuladores y variables *broadcast* si pueden mejorar la eficiencia del cómputo.

3) Éxito de los capítulos de «Los Simpsons» [3pt]

En este apartado seguiremos con el conjunto de datos del apartado anterior pero lo extendaremos a los guiones, ubicaciones y personajes. Queremos dar respuesta a varias preguntas sobre si existe alguna relación entre la puntuación IMDB de un determinado episodio y:

1. El número de **ubicaciones diferentes** que aparecen en el episodio
2. El número de **personajes femeninos diferentes** que aparecen en el episodio
3. La cantidad total de **palabras que aparecen en los diálogos** del episodio
4. La cantidad total de **líneas de diálogo** que hay en el episodio

Tened en cuenta que en los dos últimos puntos hablamos expresamente de *diálogos*, así que habría que excluir aquellas líneas que representan carteles, descripción de eventos, etc.

Desarrollaremos un programa Spark utilizando **DataFrames**³ en lugar de RDDs para resolver este problema. Los DataFrames son estructuras bidimensionales similares a tablas que tratan de acercar la programación en Spark al mundo *relacional*. Son secuencias de entradas, todas ellas con el mismo número y tipo de atributos. Además, la evaluación de DataFrames utiliza internamente un optimizador de consultas llamado *Catalyst*⁴ que busca el plan de ejecución más eficiente.

Notas:

- Este apartado requiere que seáis capaces de descubrir y entender el tipo de datos **DataFrame** disponible en Apache Spark. Para ello tendréis que consultar la documentación del sistema y la bibliografía propuesta en las transparencias.

Tareas a realizar y orden recomendado:

1. Crea un DataFrame **locations** con 3 columnas: identificador de episodio, puntuación IMDB y número de ubicaciones diferentes que aparecen en dicho episodio.
2. Genera un DataFrame **characters** con 3 columnas: identificador de episodio, puntuación IMDB y número de personajes femeninos diferentes que aparecen. Tened en cuenta que hay muchos personajes sin género consignado en nuestros datos de entrada. Estos personajes no se pueden contabilizar como femeninos ni masculinos, así que los ignoraremos.

³<http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame>

⁴Spark SQL: Relational Data Processing in Spark, SIGMOD'15. http://people.csail.mit.edu/matei/papers/2015/sigmod_spark_sql.pdf

3. Construye un DataFrame `script` con 4 columnas: identificador de episodio, puntuación IMDB, número total de palabras que aparecen en los diálogos del episodio y el número total de diálogos en el episodio. **Cuidado:** no hay que tener en cuenta las líneas del *script* que no son diálogos.
4. Finalmente, calcula el *Coeficiente de correlación de Pearson*⁵ (https://es.wikipedia.org/wiki/Coeficiente_de_correlaci%C3%B3n_de_Pearson) entre la valoración IMDB y cada una de las variables que queríamos comprobar. Puedes realizar este cálculo «a mano» sobre los DataFrames o utilizar funciones del API de Spark, que es mucho más sencillo y cómodo.

⁵Este coeficiente mide la relación lineal entre dos variables aleatorias, tomando valores en el rango $[-1,1]$: ≈ -1 si están inversamente relacionadas, ≈ 1 si están directamente relacionadas y ≈ 0 si no existe relación lineal.