

Práctica 1. Hadoop y Spark

Blanca Cano Camarero y Iker Villegas Labairu

Madrid 8 de octubre del 2022

Índice

Práctica 1 Hadoop y Spark	5
1. Instalación de Hadoop	5
¿En qué directorio del HDFS se copian los ficheros?	8
¿Qué ocurre si no hubiéramos creado el directorio?	8
Ejercicio 1.1	8
¿Qué ficheros ha modificado para activar la configuración del HDFS? ¿Qué líneas ha sido necesario modificar?	8
Ejercicio 1.2	9
2. Ejecución de la máquina de ejemplo	9
3. Programación de aplicaciones Hadoop con java	10
1. Modificar el ejemplo de WordCount que hemos tomado como partida, para que no tenga en cuenta signos de puntuación ni las mayúsculas ni las minúsculas volver a ejecutar la aplicación	12
2. Comparar resultados de la aplicación desarrollada con la que se puede ejecutar directamente en los ejemplos hadoop map-reduce	13
3.1 ¿Dónde se crea hdfs? ¿Cómo se puede decidir su localización?	13
3.2 ¿Cómo se puede borrar todo el contenido del HDFS, incluido su estructura?	13
3.3 Si estás utilizando hdfs, ¿Cómo puedes volver a ejecutar WordCount como si fuese single.node?	14
3.4 ¿Cuáles son las 10 palabras más utilizadas?	14
3.5 ¿Cuántas veces aparece el artículo “el” y la palabra “dijo”?	14
3.6 ¿El resultado coincide utilizando la aplicación wordcount que se da en los ejemplos? Justifique la respuesta	14
4. Modificación de parámetros <i>mapreduce</i>	14
4.1 Usar el tamaño de bloque por defecto de HDFS	14
4.2 Indicar el tamaño de bloque en la línea de comandos al escribir el fichero	15
4.3 Editar el fichero de configuración <code>hdfs-site.xml</code> y modificar el tamaño de bloque con el parámetro	15
4.4 Comprobar el efecto del tamaño de bloques en el funcionamiento de la aplicación WordCount. ¿Cuántos procesos Maps se lanzan en cada caso? Indique como lo ha comprobado	16
Parte 2: PySpark en Google Colab	16
Pregunta TS1.1 ¿Cómo hacer para obtener una lista de los elementos al cuadrado?	16
Pregunta TS1.2 ¿Cómo filtrar los impares?	16

Pregunta TS1.3 ¿Tiene sentido cambiar la suma por una diferencia? ¿Si se repite se obtiene siempre el mismo resultado?	17
Respuesta TS1.3	17
Solución	18
Pregunta TS1.4 ¿Cómo lo ordenarías para que primero aparezcan los impares y luego los pares?	18
Pregunta TS1.5 ¿Cuántos elementos tiene cada rdd? ¿Cuál tiene más?	18
Respuesta TS1.5	19
Pregunta TS1.6 ¿De qué tipo son los elementos del rdd palabras_map? ¿Por qué palabras_map tiene el primer elemento vacío?	19
Respuesta TS1.6	19
Pregunta TS1.7. Prueba la transformación distinct si lo aplicamos a cadenas.	19
Respuesta TS1.7	20
Pregunta TS1.8 ¿Cómo se podría obtener la misma salida pero utilizando una sola transformación y sin realizar la unión?	20
Respuesta TS1.8	20
Diferencias entre hacer map y reduce con números y cadenas	20
Pregunta TS1.9 ¿Cómo explica el funcionamiento de las celdas anteriores?	22
TS1.10 Responda a las preguntas planteadas al hacer los cambios sugeridos en las siguientes celdas	22
Respuestas TS1.10	23
Simular un GroupBy con un reduceByKey y un map	23
Pregunta TS1.11 Borra la salida y cambia las particiones en parallelize ¿Qué sucede?	24
Respuesta TS1.11	25
Procesando el QUIJOTE	25
Pregunta TS2.1 Explica la utilidad de cada transformación y detalle para cada una de ellas si cambia el número de elementos en el RDD resultante. Es decir si el RDD de partida tiene N elementos, y el de salida M elementos, indica si $N > M$, $N = M$ o $N < M$	26
Respuesta TS2.1	27
Map	27
Flatmap	27
Filter	28
Distinct	28
Sample	29
Union	30
Pregunta TS2.2 Explica el funcionamiento de cada acción anterior	30

Pregunta: Implementa la opción count de otra manera:	31
Respuesta	31
Utilizando solo reduce	31
Pregunta TS2.3 Explica el proposito de cada una de las operaciones anteriores	31
Pregunta TS2.4 ¿Cómo puede implementarse la frecuencia con groupByKey y transformaciones?	32
Optimizaciones	32
Pregunta TS2.5 ¿Cuál de las dos siguientes celdas es más eficiente? Justifique la respuesta. .	32
Pregunta TS2.6 Antes de guardar el fichero, utilice coalesce con diferente valores ¿Cuál es la diferencia?	34
Parte3: Práctica opcional Hadoop y Spark	35
Dataset 1: Análisis del comportamiento local de los diferentes equipos que han jugado en la Premier League inglesa	35
Descripción del dataset	35
Objetivo de la aplicación	36
Estructura de la solución propuesta	36
Subida de datos	36
La siguiente cuestión es si existe un beneficio a jugar en local	40
Descripción del tratamiento de los datos	40
Diseño del tratamiento de los datos	42
Experimento 2 beneficio de ganar en local más para todos los equipos	43
Diseño tratamiento de los datos	43
Conclusiones	44
Conclusiones sobre Spark	55

Práctica 1 Hadoop y Spark

1. Instalación de Hadoop

Para la instalación se ha seguido estrictamente el tutorial de clase, además para agilizar el proceso y puesto que en ambiente actual es interesante el uso de contenedores, hemos realizado un script de bash, para automatizar el proceso de instalación hasta la parte de ssh en un contenedor de docker con CentOS:7:

```
1 echo "Task 1: Updating system"
2 yum -y update && yum clean all
3 echo "Task 2: Installing packages"
4 yum -y install wget
5 yum -y install which
6 yum -y install rsync
7 yum -y install java-1.7.0-openjdk
8 echo "Task 3: Hadoop installation (may take a while)"
9 wget https://archive.apache.org/dist/hadoop/common/hadoop-2.8.1/hadoop-2.8.1.tar.gz
10 tar xvzf hadoop-2.8.1.tar.gz
11 mv hadoop-2.8.1 opt/
12 cd opt/ && ln -s hadoop-2.8.1 hadoop
13 export JAVA_HOME=/usr/lib/jvm/jre-1.7.0-openjdk
14 echo 'export JAVA_HOME=/usr/lib/jvm/jre-1.7.0-openjdk' >> ./hadoop/etc/hadoop/hadoop-env.sh
15 # Comando de prueba
16 #./opt/hadoop/bin/hadoop
17
18 yum -y install openssh-server openssh-clients
```

El respectivo Dockerfile sería el siguiente:

```
1 FROM centos:7
2 COPY ./initial.sh /
3 COPY ./material /
4 RUN bash ./initial.sh
5 EXPOSE 50070
```

Donde la carpeta material contiene:

```
1 (main)> tree material/
2 material/
3 |--WordCount_final.java
4 |-- quijote.txt
```

Para llevar a cabo la instalación de hadoop se han llevado a cabo los siguientes requisitos:

- En primer lugar, hemos instalado la versión 1.7 de java en OpenJDK la cual ya viene con CentOS.

```
yum -y install java-1.7.0-openjdk
```

- Por otro lado, también se ha instalado el paquete `rsync`, el cual está incluido en Linux. `yum -y install rsync`
- Por último, hemos obtenido hadoop a través del siguiente modo: `wget apache.rediris.es/hadoop/common/hadoop-2.8.1/hadoop-2.8.1.tar.gz`

Una vez llevado a cabo estos puntos hemos descomprimido el archivo `hadoop-2.8.1.tar.gz` y lo hemos movido a la carpeta `/opt`. Para finalizar se ha llevado una modificación en el fichero `/opt/hadoop/etc/hadoop/hadoop-env.sh` a través del comando `vim`, donde el código `export JAVA_HOME=${JAVA_HOME}` ha pasado a ser `export JAVA_HOME=/usr/lib/jvm/jre-1.7.0-openjdk`.

A continuación, hemos clonado la máquina virtual para llevar a cabo dicha instalación de dos formas distintas: Standalone y pseudo-distributed. Comenzamos con el modo standalone. Para ello veamos el siguiente ejemplo. Desde la carpeta `/opt/hadoop`, creamos un directorio al que llamaremos `prueba` y lo rellenamos con los datos.

```
cp etc/hadoop/*.xml prueba
```

Y ejecutamos el comando siguiente

```
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.8.1.jar -input prueba -output salida -mapper cat -reducer wc
```

Obteniendo así la siguiente salida: 757 2803 27305.

Pasamos ahora al caso donde la instalación se ha llevado a cabo en modo pseudo-distributed. Aquí, hemos comenzado configurando el `ssh` de la siguiente forma para que funcione sin contraseña con conexiones en localhost:

```
1 ssh-keygen
2 ssh-copy-id localhost
```

A continuación, hemos modificado los siguientes ficheros a través del comando `vim`:

- `/opt/hadoop/etc/hadoop/core-site.xml`, donde se ha llevado a cabo la modificación

```
1 <configuration>
2     <property>
3         <name>fs.defaultFS</name>
4         <value>hdfs://localhost:9000</value>
5     </property>
6 </configuration>
```

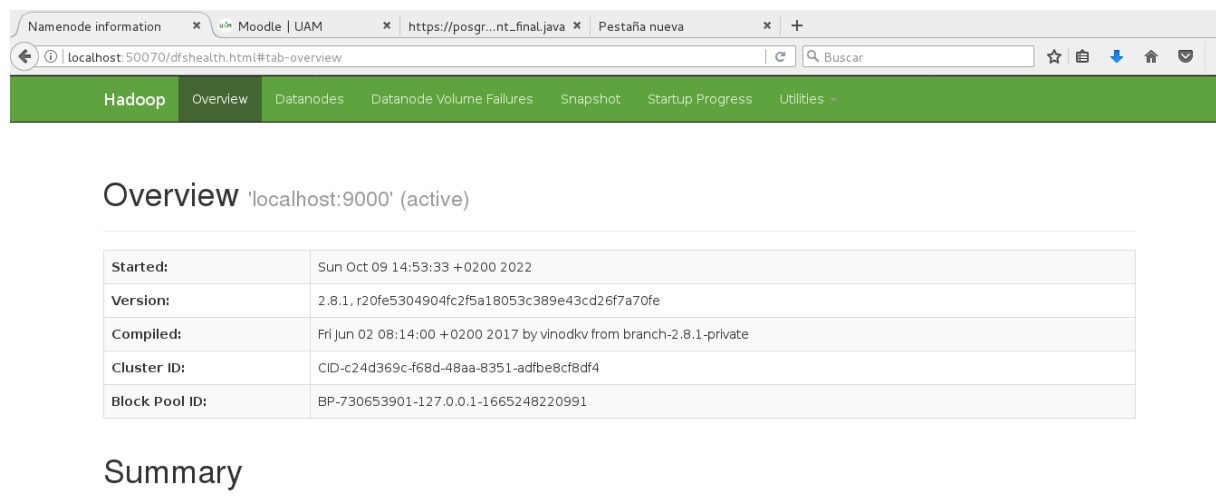
- `/opt/hadoop/etc/hadoop/core-site.xml`, en el cual hemos cambiado lo siguiente:

```
1 <configuration>
2     <property>
3         <name>dfs.replication</name>
4         <value>1</value>
5     </property>
6 </configuration>
```

Para finalizar hemos formateado el sistema de ficheros y hemos iniciado el NameNode y DataNode de la forma siguiente:

```
1 /opt/hadoop/bin/hdfs namenode -format
2 /opt/hadoop/sbin/start-dfs.sh
```

Comprobamos que podemos acceder a través de la web al NameNode con dirección: <http://localhost:50070>.



The screenshot shows a web browser window with the URL `localhost:50070/dfshealth.html#tab-overview`. The page title is "Overview 'localhost:9000' (active)". Below the title is a table with the following information:

Started:	Sun Oct 09 14:53:33 +0200 2022
Version:	2.8.1, r20fe5304904fc2f5a18053c389e43cd26f7a70fe
Compiled:	Fri Jun 02 08:14:00 +0200 2017 by vinodkv from branch-2.8.1-private
Cluster ID:	CID-c24d369c-f68d-48aa-8351-adf8e8cf8df4
Block Pool ID:	BP-730653901-127.0.0.1-1665248220991

Below the table is a "Summary" section.

Figura 1: Visualización de webnode desde navegador

Para corroborar su funcionamiento hemos creado los siguientes directorios en HDFS desde la carpeta `hadoop`:

```
1 bin/hdfs dfs -mkdir /user
2 bin/hdfs dfs -mkdir /user/root
3 bin/hdfs dfs -mkdir /user/bigdata
4 bin/hdfs dfs -mkdir /user/bigdata/prueba
```

Y hemos ejecutado el siguiente código para copiar los archivos de extensión `.xml` situados en la carpeta `etc/hadoop` al directorio `/user/bigdata`

```
1 bin/hdfs dfs -put etc/hadoop/*.xml /user/bigdata
```

Se plantean las siguientes cuestiones:

¿En qué directorio del HDFS se copian los ficheros?

Los ficheros se copian en el directorio `/user/bigdata`, tal y como se había indicado en el comando anterior.

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	supergroup	4.83 KB	Oct 08 19:00	1	128 MB	capacity-scheduler.xml
-rw-r--r--	root	supergroup	908 B	Oct 08 19:00	1	128 MB	core-site.xml
-rw-r--r--	root	supergroup	9.46 KB	Oct 08 19:00	1	128 MB	hadoop-policy.xml
-rw-r--r--	root	supergroup	891 B	Oct 08 19:00	1	128 MB	hdfs-site.xml
-rw-r--r--	root	supergroup	620 B	Oct 08 19:00	1	128 MB	https-site.xml
-rw-r--r--	root	supergroup	3.44 KB	Oct 08 19:00	1	128 MB	kms-acls.xml
-rw-r--r--	root	supergroup	5.42 KB	Oct 08 19:00	1	128 MB	kms-site.xml
-rw-r--r--	root	supergroup	690 B	Oct 08 19:00	1	128 MB	yarn-site.xml

Figura 2: Directorio donde se copian ficheros HDFS

¿Qué ocurre si no hubiéramos creado el directorio?

Si no hubiéramos creado el directorio `/user/bigdata` del HDFS no se llevaría a cabo la copia de ficheros y saltaría el siguiente mensaje de error:

```
1 put: `/user/bigdata': No such file or directory
```

Procedemos a responder los siguientes ejercicios

Ejercicio 1.1

¿Qué ficheros ha modificado para activar la configuración del HDFS? ¿Qué líneas ha sido necesario modificar?

Para activar la configuración HDFS se han modificado los archivos `hdfs-site.xml` y `core-site.xml`, de la forma que se ha explicado anteriormente. Es necesario además ejecutar el comando `bin/dfs namenode -format` que afecta a ciertos ficheros así que estos también se ven modificados.

Ejercicio 1.2

Para pasar a la ejecución de Hadoop sin HDFS ¿es suficiente con parar el servicio con `stop-dfs.sh`? ¿Cómo se consigue? No, pese a que con el comando `sbin/stop-dfs.sh` se detienen todos los en particular la ejecución de Hadoop con HDFS; esto no es suficiente. Deberíamos eliminar la propiedades:

- `dfs.replication` de `hdfs-site.xml`.
- `fs.defaultFS` de `core-site.xml`.

2. Ejecución de la máquina de ejemplo

Comenzaremos para el caso donde la instalación se ha llevado del modo standalone. Simplemente ejecutaremos un wordcount desde hadoop

```
1 bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.1.  
  jar wordcount prueba salida1
```

Obteniendo así el archivo `/opt/hadoop/salida1/part-r-000000` con el resultado siguiente, donde mostramos las primeras líneas del archivo:

```
1 "*" 18  
2 "AS 8  
3 "License"); 8  
4 "alice,bob 18  
5 "clumping" 1  
6 "kerberos";. 1  
7 "simple"; 1  
8 'HTTP/' 1  
9 'none' 1  
10 'random' 1  
11 'sasl' 1  
12 'string' 1  
13 'zookeeper' 2  
14 'zookeeper'. 1  
15 (ASF) 1  
16 (Kerberos). 1  
17 (default) 1  
18 (default), 1
```

Tal y como vemos el programa realiza un conteo de la aparición de las cadenas de caracteres que aparecen sin ningún tipo de filtro.

Para la ejecución de la aplicación de ejemplo desde el modo hdfs, descargamos del mismo modo el fichero `quijote.txt` y lo guardamos en una carpeta local siguiendo la siguiente ruta `/home/bigdata`

/quijote.txt. A continuación, lo pasamos a un directorio de HDFS, desde la carpeta hadoop, del siguiente modo:

```
1 bin/hadoop dfs -copyFromLocal /home/bigdata/quijote.txt /user/bigdata/prueba
```

Vemos que aparece el fichero quijote.txt en el nuevo directorio

Y llevamos a cabo el proceso del wordcount

```
1 bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.1.jar wordcount /user/bigdata/prueba/quijote.txt /user/bigdata/prueba/salida
```

Obteniendo de nuevo un archivo /user/bigdata/prueba/salida/part-r-000000 cuyas primeras líneas son:

```
1 "Apenas 1
2 "Caballero 4
3 "Conde 1
4 "Ea, 1
5 "Miau", 1
6 "Rastrea 1
7 "Ricamonte", 1
8 "Tablante", 1
9 "dichosa 1
10 "el 8
11 "y 1
```

Como se puede observar, cuenta de manera diferente palabras que van precedidas o seguidas de signos de puntuación, así como, los casos en los que su primera letra es mayúscula. Este problema no nos permite obtener un análisis muy fino del número de veces que aparece cada palabra en el fragmento estudiado.

3.Programación de aplicaciones Hadoop con java

Para ambos casos, a parte de los requisitos expuestos anteriormente, necesitaremos instalar el entorno de desarrollo java, de la siguiente forma

```
1 sudo yum install java-1.7.0-openjdk-devel.x86_64
```

Empezaremos para el caso en el que no vamos a usar HDFS. A partir de ahora necesitaremos ser super usuario, así que aplicamos el comando su. Crearemos un nuevo directorio desde la carpeta opt al que llamaremos work, y dentro del mismo, otro que denominaremos WordCount. Además, descargaremos y guardaremos los ficheros quijote.txt y WordCount.java en la ruta /opt/work. A continuación, compilaremos el archivo de java del siguiente modo:

```
1 javac -classpath /opt/hadoop/share/hadoop/common/*:/opt/hadoop/share/hadoop/mapreduce/* -d WordCount WordCount.java
```

Y creamos el archive .jar

```
1 jar -cvf WordCount.jar -C WordCount/ .
```

Ahora que ya tenemos todo listo, ejecutaremos el WordCount desde /opt/hadoop

```
1 bin/hadoop jar /opt/work/WordCount.jar uam.WordCount /opt/work/quijote.txt /opt/work/salida1
```

Obteniendo así un nuevo archivo con el conteo de todas las veces que aparece cada palabra en dicho fragmento del Quijote.

Ahora tomamos el caso donde utilizamos HDFS. Tomaremos el fichero `WordCount.java` anteriormente descargado y guardado en el directorio `/opt/work` el cual contiene las clases mapper y reducer, así como el método main ya completos. Además, crearemos el siguiente script, al que llamaremos `compilar.bash`:

```
1 #!/bin/bash
2
3 file=$1
4
5 HADOOP_CLASSPATH=/opt/hadoop/share/hadoop/common/*:/opt/hadoop/share/hadoop/mapreduce/*
6 echo $HADOOP_CLASSPATH
7
8 rm -rf ${file}
9 mkdir -p ${file}
10
11 javac -classpath $HADOOP_CLASSPATH -d ${file} ${file}.java
12 jar -cvf ${file}.jar -C ${file} .
```

Para mejorarlo, y que podamos elegir el nombre que le damos al archivo .jar creado, añadiremos una segunda entrada, quedando el código ahora de la siguiente forma:

```
1 #!/bin/bash
2
3 file=$1
4 name=$2
5
6 HADOOP_CLASSPATH=/opt/hadoop/share/hadoop/common/*:/opt/hadoop/share/hadoop/mapreduce/*
7 echo $HADOOP_CLASSPATH
8
9 rm -rf ${file}
10 mkdir -p ${file}
11
```

```
12 javac -classpath $HADOOP_CLASSPATH -d ${file} ${file}.java
13 jar -cvf ${name}.jar -C ${file} .
```

A continuación, compilamos el archivo como se sigue

```
1 ./compilar.bash WordCount result
```

Obteniendo así el archivo `result.jar`, que lanzaremos invocando la aplicación del siguiente modo

```
1 bin/hadoop jar /opt/work/result.jar uam.WordCount /user/bigdata/prueba/
  quijote.txt /user/bigdata/prueba/salida_128MB
```

Ahora disponemos de las siguientes cuestiones:

1. Modificar el ejemplo de WordCount que hemos tomado como partida, para que no tenga en cuenta signos de puntuación ni las mayúsculas ni las minúsculas volver a ejecutar la aplicación

Para conseguir esto hemos modificado el método `map` de la siguiente manera

```
1 public void map(Object key, Text value, Context context) throws
  IOException, InterruptedException {
2     String minimized_value = value.toString().toLowerCase();
3     String processed_value = minimized_value.replaceAll("[^áéíóúüñá-
  z0-9]+", "");
4     StringTokenizer itr = new StringTokenizer(processed_value);
5     while (itr.hasMoreTokens()) {
6         word.set(itr.nextToken());
7         context.write(word, one);
8     }
9 }
```

Tras compilar el archivo y ejecutarlo, podemos observar que ya no se tienen en cuenta, ni mayúsculas ni signos de puntuación. Aquí una muestra de las primeras líneas:

```
1 a 1428
2 aa 1
3 abadejo 2
4 abades 1
5 abajo 11
6 abalánzase 1
7 abejas 1
8 abencerraje 1
9 abiertas 1
10 abierto 3
11 abiertos 2
12 abindarráez 3
13 abismo 2
14 ablande 1
```

```
15 aborrascadas 1
16 aborrece 1
17 aborrecido 3
18 aborrecimiento 2
```

2. Comparar resultados de la aplicación desarrollada con la que se puede ejecutar directamente en los ejemplos hadoop map-reduce

Observamos claramente que el conteo de palabras en este último caso se ha realizado de una manera muy fina, dónde se han ignorado caracteres de puntuación y mayúsculas. De este modo, se muestra de una mejor forma el número real de veces que aparece cada cadena de caracteres.

Por otro lado, se plantean los siguientes ejercicios:

3.1 ¿Dónde se crea hdfs? ¿Cómo se puede decidir su localización?

La localización de los archivos hdfs viene determinada por la variable `dfs.datanode.data.dir`, cuyo valor por defecto es `file://${hadoop.tmp.dir}/dfs/data`. Dicha variable se puede modificar desde el archivo `/opt/hadoop/etc/hadoop/hdfs-site.xml` del siguiente modo:

```
1 <property>
2   <name> dfs.datanode.data.dir</name>
3   <value> file://${hadoop.tmp.dir}/dfs/data</value>
4 </property>
```

Además, cabe destacar, que la variable `hadoop.tmp.dir` podemos modificarla desde el archivo `/opt/hadoop/etc/hadoop/core-site.xml`, cuyo valor por defecto viene dado por `/tmp/hadoop-${user}.name`.

3.2 ¿Cómo se puede borrar todo el contenido del HDFS, incluido su estructura?

Todo el contenido de HDFS, incluyendo su estructura, podría borrarse formateando el Namenode con el comando `bin/hdfs namenode -format` una vez que se hayan parado todos los servicios del HDFS (`sbin/stop.dfs.sh`). De este modo, una vez que se proceda a iniciar el Namenode y el Datanode, todo el contenido, tanto ficheros, como estructura del HDFS, habrá desaparecido.

3.3 Si estás utilizando hdfs, ¿Cómo puedes volver a ejecutar WordCount como si fuese single.node?

Para volver a ejecutar WordCount como si fuese single.node, se deberán quitar los cambios que hemos realizado en los archivos `core-site.xml` y `hdfs-site.xml` contenidos en el directorio `/opt/hadoop/etc/hadoop`.

Respecto al fragmento del Quijote, y tomando el código java modificado,

3.4 ¿Cuáles son las 10 palabras más utilizadas?

Las 10 palabras más utilizadas son, en orden decreciente: que (3055), de (2816), y (2585), a (1428), la (1423), el (1232), en (1155), no (915), se (753) y los (696).

3.5 ¿Cuántas veces aparece el artículo “el” y la palabra “dijo”?

El artículo el aparece un total de 1232 veces, mientras que la palabra dijo 272.

3.6 ¿El resultado coincide utilizando la aplicación wordcount que se da en los ejemplos? Justifique la respuesta

Claramente no coincide. En este último caso el análisis se realiza de una forma más rápida, así como, hemos podido obviar la aparición de mayúsculas y signos de puntuación en las cadenas de caracteres que estorbaban la obtención del número real de veces que aparecía la palabra en el texto. Todo ello no se veía reflejado en la aplicación wordcount dada en los ejemplos.

4. Modificación de parámetros *mapreduce*

En primer lugar, vamos a comenzar creando un segundo archivo de quijote.txt, pero esta vez de mayor tamaño. Para ello concatenamos el fichero varias veces de la forma siguiente:

```
1 cat quijote.txt quijote.txt >> quijotex15.txt
```

Llevaremos a cabo los pasos citados a continuación:

4.1 Usar el tamaño de bloque por defecto de HDFS

Ya hemos situado anteriormente el archivo `quijote.txt` en el directorio de HDFS `/user/bigdata/prueba`. Cabe destacar que el tamaño del archivo es de 128 Mb.

4.2 Indicar el tamaño de bloque en la línea de comandos al escribir el fichero

Tomamos como parámetro de tamaño, `dfs.blocksize = 2097152` y pasamos este nuevo archivo `quijotex15.txt` directamente al directorio HDFS `/user/bigdata/prueba`.

```
1 sudo bin/hdfs dfs -D dfs.blocksize=2097152 -put /home/bigdata/
  quijotex15.txt /user/bigdata/prueba
```

Tal y como se muestra en la imagen en comparación con el `quijote.txt`, el tamaño de este archivo es de 2 Mb.

Browse Directory

Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	supergroup	310.17 KB	Oct 08 19:13	1	128 MB	quijote.txt
-rw-r--r--	root	supergroup	620.35 KB	Oct 08 19:30	1	2 MB	quijotex15.txt
drwxr-xr-x	root	supergroup	0 B	Oct 08 19:16	0	0 B	salida

Showing 1 to 3 of 3 entries

Previous 1 Next

Figura 3: image

4.3 Editar el fichero de configuración `hdfs-site.xml` y modificar el tamaño de bloque con el parámetro

Mediante el comando `vim` llevamos a cabo el siguiente cambio en el archivo `hdfs-site.xml`

```
1 <configuration>
2     <property>
3         <name>dfs.replication</name>
4         <value>1</value>
5     </property>
6     <property>
7         <name>dfs.block.size</name>
8         <value>2097152</value>
9     </property>
10 </configuration>
```

4.4 Comprobar el efecto del tamaño de bloques en el funcionamiento de la aplicación WordCount. ¿Cuántos procesos Maps se lanzan en cada caso? Indique como lo ha comprobado

Tras realizar una serie de lanzamientos a ambos archivos del quijote de distinto tamaño vemos lo siguiente: para el archivo de `quijote.txt` el cual poseía un tamaño de 128 Mb, el número de procesos Maps que se han lanzado a sido de 1, tal y como muestra el siguiente código de salida:

```
1  Shuffled Maps =1
2  Failed Shuffles=0
3  Merged Map outputs=1
```

Mientras que para el caso del `quijotex15.txt`, cuyo tamaño es de 2 Mb, el número de Maps es de 3:

```
1  Shuffled Maps =3
2  Failed Shuffles=0
3  Merged Map outputs=3
```

Por lo tanto, el tamaño de los bloques afecta al número de procesos de Maps, y por ende de procesos reduce, que se realizan, así como, al número de bloques creados.

Parte 2: PySpark en Google Colab

Pregunta TS1.1 ¿Cómo hacer para obtener una lista de los elementos al cuadrado?

```
1  #¿Cómo hacer para obtener una lista de los elementos al cuadrado?
2  numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])
3  rdd = numeros.map(lambda e: e**2)
4  print(rdd.collect())
```

```
1  [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Pregunta TS1.2 ¿Cómo filtrar los impares?

```
1  #¿Cómo filtrar los impares?
2  rddi = numeros.filter(lambda n: n%2 == 0) # Mostrará los números
    impares
3  print(rddi.collect())
```

```
1  [2, 4, 6, 8, 10]
```


Pregunta TS1.3 ¿Tiene sentido cambiar la suma por una diferencia? ¿Si se repite se obtiene siempre el mismo resultado?

```
1 #Tiene sentido la reducción(elem1-elem2)?
2 numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])
3
4 print (numeros.reduce(lambda elem1,elem2: elem1-elem2))
```

```
1 15
```

Respuesta TS1.3

Tal operación **no** tiene sentido ya que la operación obtenida no es la esperada.

Para entender qué es lo que está ocurriendo es necesario consultar la documentación oficial de spark última consulta 30 de septiembre del 2022

al realizar parallelize por defecto se hacen dos particiones de RDD, por tanto al hacer el reduce se calcularía la operación en las respectivas dos particiones y tras esto se haría la resta de los resultados de ambas.

Si quisiéramos que el resultado fuera el esperado deberíamos de especificar de que solo hubiera una partición (ver la celda siguiente)

```
1 # Celda de ejemplo de cómo habría que programarlo para que el resultado
   fuera el correcto:
2
3 #Tiene sentido la reducción(elem1-elem2)?
4 numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10],1)
5
6 print (numeros.reduce(lambda elem1,elem2: elem1-elem2))
```

```
1 -53
```

Como es de esperar si variamos el número de particiones la salida se verá afectada. Esto nos hace pensar que para poder paralelizar este tipo de operaciones es necesario que las operaciones sean **asociativas y conmutativas**.

```
1 #Tiene sentido la reducción(elem1-elem2) si la repetimos y los
   elementos están en otro orden?
2 numeros = sc.parallelize([2,1,4,3,5,6,7,8,9,10])
3
4 print (numeros.reduce(lambda elem1,elem2: elem1-elem2))
```

```
1 17
```

Como podemos ver no tiene sentido el resultado, ya que debería ser negativo.

```
1 #Tiene sentido la reducción(elem1-elem2)? ¿qué pasa si cambiamos la
  paralelización a 5? ¿y si es 2?
2 numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10],5)
3
4 print (numeros.reduce(lambda elem1,elem2: elem1-elem2))
```

```
1 3
```

Ya hemos respondido a estas preguntas en **Respuesta TS1.3**. #Acciones

```
1 numeros = sc.parallelize([5,3,1,2,4])
2
3 print (numeros.take(3))
4 #¿Qué sucede si ponemos 30 en vez de 3 elementos?
```

```
1 [5, 3, 1]
```

Solución

Tomará los 30 primeros, en nuestro caso, puesto que solo hay 5 tomará los 5 primeros.

```
1 numeros = sc.parallelize([3,2,1,4,5])
2 # (La segunda función se utiliza como índice de ordenación)
3 print (numeros.takeOrdered(3, lambda elem: -elem))
4 # La función lambda se está utilizando para crear el índice de la lista
  de ordenación
```

```
1 [5, 4, 3]
```

Pregunta TS1.4 ¿Cómo lo ordenarías para que primero aparezcan los impares y luego los pares?

```
1 #¿Cómo lo ordenarías para que primero aparezcan los impares y luego los
  pares?
2 numeros = sc.parallelize([3,2,1,4,5])
3 #print(numeros...)
4 print (numeros.takeOrdered(5, lambda elem: -(elem %2)*elem))
```

```
1 [5, 3, 1, 2, 4]
```

Pregunta TS1.5 ¿Cuántos elementos tiene cada rdd? ¿Cuál tiene más?

```
1 #¿Cuántos elementos tiene cada rdd? ¿Cuál tiene más?
2 print (palabras_flat.collect())
3 print (palabras_map.collect())
```

```
1 ['a', 'a', 'b', 'a', 'b', 'c']
2 [[], ['a'], ['a', 'b'], ['a', 'b', 'c']]
```

Respuesta TS1.5

El primero posee 6, mientras que el otro 4. Es evidente que el primero posee más.

Pregunta TS1.6 ¿De qué tipo son los elementos del rdd palabras_map? ¿Por qué palabras_map tiene el primer elemento vacío?

Respuesta TS1.6

Son listas.

Está vacía porque se generó a partir de un string sin ningún carácter ni espacio, que tras ser *separados* por la función `split` se traduce en la lista vacía.

```
1 palabras_flat.take(1)
```

```
1 ['a']
```

```
1 palabras_map.take(1)
```

```
1 [[]]
```

Pregunta TS1.7. Prueba la transformación `distinct` si lo aplicamos a cadenas.

```
1 # Prueba la transformación distinct si lo aplicamos a cadenas. ¿Que
   realiza?
2 log = sc.parallelize(['E: e21', 'I: i11', 'W: w12', 'I: i11', 'W: w13',
   'E: e45'])
3 print( log.distinct().collect() )
```

```
1 ['I: i11', 'W: w12', 'W: w13', 'E: e45', 'E: e21']
```

Respuesta TS1.7

Podemos observar que muestra los elementos de la lista sin repeticiones.

Pregunta TS1.8 ¿Cómo se podría obtener la misma salida pero utilizando una sola transformación y sin realizar la unión?

```
1 log = sc.parallelize(['E: e21', 'I: i11', 'W: w12', 'I: i11', 'W: w13',  
    'E: e45'])  
2  
3 infos = log.filter(lambda elemento: elemento[0]=='I')  
4 errors = log.filter(lambda elemento: elemento[0]=='E')  
5  
6 inferr = infos.union(errors)  
7  
8 print (inferr.collect())
```

```
1 ['I: i11', 'I: i11', 'E: e21', 'E: e45']
```

Respuesta TS1.8

Para ello vamos a realizar directamente el filtro donde se admitan ambas condiciones.

```
1 #¿Cómo se podría obtener la misma salida pero utilizando una sola  
    transformación y sin realizar la unión?  
2 log = sc.parallelize(['E: e21', 'I: i11', 'W: w12', 'I: i11', 'W: w13',  
    'E: e45'])  
3 result = log.filter(lambda elemento: elemento[0]=='I' or elemento[0]=='  
    E')  
4 print(result.collect())
```

```
1 ['E: e21', 'I: i11', 'I: i11', 'E: e45']
```

Diferencias entre hacer map y reduce con números y cadenas

```
1 numeros = sc.parallelize([1,2,3,4,5])  
2  
3 print (numeros.reduce(lambda elem1,elem2: elem2+elem1))
```

```
1 15
```

```

1 #Tiene sentido esta operación?¿Cómo sale el resultado? pruebe los
  resultados
2 # empezando con 2 elementos e ir incrementando hasta tener 5
3
4 numeros = sc.parallelize([1,2,3,4,5])
5
6 print (numeros.reduce(lambda elem1,elem2: elem2-elem1))

```

```

1 3

```

No tiene sentido porque el resultado no es correcto.

```

1 for i in range(2,6):
2     print(f"Para {i} elementos, la lista que se es {list(range(1,i+1))}"
3         )
4     numeros = sc.parallelize(list(range(1,i+1)))
5     print (f"Para {i} elementos", numeros.reduce(lambda elem1,elem2:
6         elem2-elem1))

```

```

1 Para 2 elementos, la lista que se es [1, 2]
2 Para 2 elementos 1
3 Para 3 elementos, la lista que se es [1, 2, 3]
4 Para 3 elementos 0
5 Para 4 elementos, la lista que se es [1, 2, 3, 4]
6 Para 4 elementos 0
7 Para 5 elementos, la lista que se es [1, 2, 3, 4, 5]
8 Para 5 elementos 3

```

```

1 palabras = sc.parallelize(['HOLA', 'Que', 'TAL', 'Bien'])
2
3 pal_minus = palabras.map(lambda elemento: elemento.lower())
4
5 print (pal_minus.reduce(lambda elem1,elem2: elem1+"-"+elem2))
6 #y esta tiene sentido esta operación?
7 #Sí tiene
8 # Qué pasa si ponemos elem2+"-"+elem1
9 #

```

```

1 hola-que-tal-bien

```

```

1 print (pal_minus.reduce(lambda elem1,elem2: elem2+"-"+elem1))

```

```

1 bien-tal-que-hola

```

En este ejemplo se pone de manifiesto claramente lo que se comentó en la respuesta Respuesta TS1.3.

```

1 r = sc.parallelize([('A', 1),('C', 4),('A', 1),('B', 1),('B', 4)])
2 rr = r.reduceByKey(lambda v1,v2:v1+v2)

```

```
3 print (rr.collect())
```

```
1 [('C', 4), ('A', 2), ('B', 5)]
```

```
1 r = sc.parallelize([('A', 1), ('C', 4), ('A', 1), ('B', 1), ('B', 4)])
2 rr1 = r.reduceByKey(lambda v1,v2:v1+v2)
3 print (rr1.collect())
4 rr2 = rr1.reduceByKey(lambda v1,v2:v1)
5 print (rr2.collect())
```

```
1 [('C', 4), ('A', 2), ('B', 5)]
2 [('C', 4), ('A', 2), ('B', 5)]
```

Pregunta TS1.9 ¿Cómo explica el funcionamiento de las celdas anteriores?

```
1 #palabras = sc.parallelize(['HOLA', 'Que', 'TAL', 'Bien'])
2 #print (pal_minus.reduce(lambda elem1,elem2: elem1+"-"+elem2))
3 #y esta tiene sentido esta operación?
4 # Qué pasa si ponemos elem2+"-"+elem1
```

```
1 r = sc.parallelize([('A', 1), ('C', 4), ('A', 1), ('B', 1), ('B', 4)])
2 rr = r.reduceByKey(lambda v1,v2:v1+v2)
3 print (rr.collect())
```

```
1 [('C', 4), ('A', 2), ('B', 5)]
```

De cada dupla suma el valor de los elementos que comparten la misma clave.

```
1 r = sc.parallelize([('A', 1), ('C', 4), ('A', 1), ('B', 1), ('B', 4)])
2 rr1 = r.reduceByKey(lambda v1,v2:v1+v2)
3 print (rr1.collect())
4 rr2 = rr1.reduceByKey(lambda v1,v2:v1)
5 print (rr2.collect())
```

```
1 [('C', 4), ('A', 2), ('B', 5)]
2 [('C', 4), ('A', 2), ('B', 5)]
```

TS1.10 Responda a las preguntas planteadas al hacer los cambios sugeridos en las siguiente celdas

```
1 r = sc.parallelize([('A', 1), ('C', 4), ('A', 1), ('B', 1), ('B', 4)])
2 rr1 = r.reduceByKey(lambda v1,v2:'hola')
3 print (rr1.collect())
```

```
4 rr2 = rr1.reduceByKey(lambda v1,v2: 'hola')
5 print (rr2.collect())
```

```
1 [('C', 4), ('A', 'hola'), ('B', 'hola')]
2 [('C', 4), ('A', 'hola'), ('B', 'hola')]
```

```
1 r = sc.parallelize([('A', 1), ('C', 2), ('A', 3), ('B', 4), ('B', 5)])
2 rr = r.groupByKey()
3 res= rr.collect()
4 for k,v in res:
5     print (k, list(v))
6 # Que operación se puede realizar al RDD rr para que la operacion sea
  como un reduceByKey
7 #¿Y simular un group con un reduceByKey y un map?
```

```
1 C [2]
2 A [1, 3]
3 B [4, 5]
```

Respuestas TS1.10

Transformar groupByKey para que se transforme como un reduce

```
1 from functools import reduce
2 r = sc.parallelize([('A', 1), ('C', 2), ('A', 3), ('B', 4), ('B', 5)])
3 # Transformación con sentencias de python
4 rr = (r.groupByKey()).collect()
5 res = list(map(lambda v: (v[0], reduce(lambda x,y: "hola", list(v[1]))), rr))
6 print(res)
```

```
1 [('C', 2), ('A', 'hola'), ('B', 'hola')]
```

Simular un GroupBy con un reduceByKey y un map

De cada par clave valor la idea es introducir cada valor a una lista y despues realizar `reduceByKey` donde la operación de *reduce* es concatenar las listas.

```
1 r = sc.parallelize([('A', 1), ('C', 4), ('A', 1), ('B', 1), ('B', 4)])
2 r_map = r.map(lambda x: (x[0], [x[1]]))
3 rr1 = r_map.reduceByKey(lambda v1,v2: v1+v2)
4 print (rr1.collect())
```

```
1 [('C', [4]), ('A', [1, 1]), ('B', [1, 4])]
```

```
1 rdd1 = sc.parallelize([('A',1), ('B',2), ('C',3)])
2 rdd2 = sc.parallelize([('A',4), ('B',5), ('C',6)])
```

```

3
4 rddjoin = rdd1.join(rdd2)
5
6 print (rddjoin.collect())
7 # Prueba a cambiar las claves del rdd1 y rdd2 para ver cuántos
  elementos se crean

```

```

1 [('A', (1, 4)), ('B', (2, 5)), ('C', (3, 6))]

```

```

1 rdd1 = sc.parallelize([('A',1),('B',2),('C',3)])
2 rdd2 = sc.parallelize([('A',4),('A',5),('B',6),('D',7)])
3
4 rddjoin = rdd1.join(rdd2)
5
6 print (rddjoin.collect())
7 #Modifica join por leftOuterJoin, rightOuterJoin y fullOuterJoin ¿Qué
  sucede?

```

```

1 [('A', (1, 4)), ('A', (1, 5)), ('B', (2, 6))]

```

```

1 rdd = sc.parallelize([('A',1),('B',2),('C',3),('A',4),('A',5),('B',6)])
2
3 res = rdd.sortByKey(False)
4
5 print (res.collect())

```

```

1 [('C', 3), ('B', 2), ('B', 6), ('A', 1), ('A', 4), ('A', 5)]

```

```

1
2
3 ```python
4 # Recupera el fichero guardado y realiza la suma
5 n2 = sc.textFile('salida').map(lambda a:int(a))
6 print(n2.reduce(lambda v1,v2: v1 + v2))
7
8 # Prueba este código y mira qué genera?
9 # Borra la salida y cambia las particiones en parallelize ¿Qué sucede?
10 # (pe c.parallelize(xrange(0,1000),8))

```

```

1 499500

```

Pregunta TS1.11 Borra la salida y cambia las particiones en parallelize ¿Qué sucede?

(pe c.parallelize(xrange(0,1000),8))

```

1 # Recupera el fichero guardado y realiza la suma
2 n2 = sc.textFile('salida').map(lambda a:int(a))

```



```

3 print(n2.reduce(lambda v1,v2: v1 + v2))
4
5 # Prueba este código y mira qué genera?
6 # Borra la salida y cambia las particiones en parallelize ¿Qué sucede?
7 sc.parallelize(range(0,1000),8)
8 numeros.saveAsTextFile('salida3')

```

```

1 499500

```

```

1 # Prueba este código y mira qué genera?
2 # Borra la salida y cambia las particiones en parallelize ¿Qué sucede?
3 %rm -rf salida3
4 n = sc.parallelize(range(0,17),8)
5 n.saveAsTextFile('salida3')
6 %cat salida3/part-000000

```

```

1 0
2 1

```

Respuesta TS1.11

Podemos observar que cada proceso en paralelo sobrescribe el fichero y por tanto solo se refleja la última que ha sido guardada.

```

1 %cat salida3/part-000000

```

```

1 0
2 1
3
4
5 '/content'

```

Procesando el QUIJOTE

Transformaciones

```

1 charsPerLine = quijote.map(lambda s: len(s))
2 allWords = quijote.flatMap(lambda s: s.split())
3 allWordsNoArticles = allWords.filter(lambda a: a.lower() not in ["el",
4                                     "la"])
5 allWordsUnique = allWords.map(lambda s: s.lower()).distinct()
6 sampleWords = allWords.sample(withReplacement=True, fraction=0.2, seed
7                               =666)
8 weirdSampling = sampleWords.union(allWordsNoArticles.sample(False,
9                               fraction=0.3))
10 # cómo funciona cada transformación

```

- `charsPerLine`: Caracteres por líneas.
- `allWords`: Contiene un RDD cuyos elementos son variables de tipo `String` con las palabras del quijote.
- `allWordsNoArticles`: Minimiza las palabras del RDD `allWords` y devuelve todas menos los artículos `el` y `la`.
- `allWordsUnique`: Minimiza las palabras del RDD `allWords` y deja solo un representante.
- `sampleWords`: Toma una muestra aleatoria de tamaño aproximadamente 20 % del RDD de `allWords` con remplazamiento y semilla 666.
- `weirdSampling`: Devuelve un RDD que contiene los elementos de `sampleWords` y una muestra aleatoria de `allWordsNoArticles` sin remplazamiento y de tamaño alrededor del 30 %.

Se detallarán las funciones en la respuesta a TS2.1.

```
1 allWordsUnique.take(10)
```

```
1 ['el',  
2  'hidalgo',  
3  'don',  
4  'mancha',  
5  'saavedra',  
6  'que',  
7  'condición',  
8  'y',  
9  'del',  
10 'd.']
```

Pregunta TS2.1 Explica la utilidad de cada transformación y detalle para cada una de ellas si cambia el número de elementos en el RDD resultante. Es decir si el RDD de partida tiene N elementos, y el de salida M elementos, indica si $N > M$, $N = M$ o $N < M$.

- `map`
- `flatMap`
- `filter`
- `distinct`
- `sample`
- `union`

Respuesta TS2.1

Map

Los argumentos de `map` son `map(f)` donde `f` a cuyo dominio pertenecen los elementos del RDD.

El RDD resultante son las respectivas imágenes de aplicar los elementos del RDD de entrada a `f`.

Es por ello que $N = M$.

Ejemplo:

```
1 numeros = sc.parallelize(list(range(10)))
2 rdd = numeros.map(lambda x: x*10+x)
3 print(rdd.collect())
```

```
1 [0, 11, 22, 33, 44, 55, 66, 77, 88, 99]
```

Flatmap

Funciona como un `map` el cual generaría un RDD cuyos elementos deben ser iterables, a éstos se le aplica la operación de *flattening*, es decir *saca* los elementos que estén en una lista (solo actúa a un nivel, una lista de lista devolvería una lista).

No se puede establecer que ninguna relación entre N y M ya que si hay listas vacías entonces pudiera darse el caso que $N > M$ (ver ejemplo 2 que se mostrará a continuación). Aunque si imponemos que no hay listas vacías entonces $N \leq M$.

```
1 # Ejemplo 1: N < M
2 numeros = sc.parallelize(list(range(10)))
3 rdd = numeros.flatMap(lambda x: [x,x*10+x])
4 print(rdd.collect())
5
6 # Ejemplo 2: N > M
7 def lista_recurativa (n):
8     if n <= 1:
9         return []
10    else:
11        return [lista_recurativa(n-1)]
12 numeros = sc.parallelize(list(range(1,4)))
13 print("Ejemplo 2: si se aplicara un map solo solo:\n",
14       numeros.map(lambda x: lista_recurativa(x)).collect())
15 print("Ejemplo 2: de qué habría en un flatMap:\n",
16       numeros.flatMap(lambda x: lista_recurativa(x)).collect())
17
18 # Ejemplo 3: N = M
19 def lista_recurativa_2 (n):
```

```

20  if n <= 1:
21      return [n]
22  else:
23      return [lista_rekursiva_2(n-1)]
24
25  numeros = sc.parallelize(list(range(3)))
26  print("Ejemplo 3: si se aplicara un map solo:\n",
27        numeros.map(lambda x: lista_rekursiva_2(x)).collect())
28
29  print("Ejemplo 3: si se aplicara un flapMap:\n",
30        numeros.flatMap(lambda x: lista_rekursiva_2(x)).collect())

```

```

1  [0, 0, 1, 11, 2, 22, 3, 33, 4, 44, 5, 55, 6, 66, 7, 77, 8, 88, 9, 99]
2  Ejemplo 2: si se aplicara un map solo solo:
3  [[], [[]], [[]]]
4  Ejemplo 2: de qué habría en un flapMap:
5  [[], [[]]]
6  Ejemplo 3: si se aplicara un map solo:
7  [[0], [1], [[1]]]
8  Ejemplo 3: si se aplicara un flapMap:
9  [0, 1, [1]]

```

Filter

Recibe una función con dominio en los elemntos del RDD de partida y salida un booleano. Devuelve un RDD con los elementos cuya imagen sea `True`. En este caso $N \geq M$, donde la igualdad se dará si todos los elementos tienen imagen `True`.

```

1  numeros = sc.parallelize([0,1,2,3,4])
2  rddi = numeros.filter(lambda x: x < 2) # Mostrará los números menores
   de dos
3  print(rddi.collect())

```

```

1  [0, 1]

```

Distinct

Elimina los elementos repetidos dejando solo una incidencia de ellos. Por tanto $N \geq M$ donde la igualdad se da si no hay ningún elemento repetido.

```

1  # Ejemplo sencillo
2  n = sc.parallelize([0,0,1,1,0,2])
3  rdd = n.distinct()
4  print(rdd.collect())
5

```

```
6 # Ejemplo complejo, compara elemento a elemento
7 n = sc.parallelize([('a',2),('a',2), ('a',1), ('b',1),])
8 rdd = n.distinct()
9 print(rdd.collect())
```

```
1 [0, 2, 1]
2 [('a', 1), ('a', 2), ('b', 1)]
```

Sample

Es un mecanismo para extraer muestras aleatorias, sus argumentos son:

`sample(withReplacement, fraction, seed=None)`

- `withReplacement`: booleano que indica si hay reemplazamiento.
- `fraction`: decimal en $[0, 1]$ fracción de la muestra que se desea. **No se garantiza que la muestra siempre de esa fracción, puede ser menor o mayor.**
- `seed` semilla.

Por la observación de que no se garantice la fracción no se puede establecer ninguna relación de orden entre N y M , mostramos tales situaciones a continuación.

```
1 # Ejemplo 1: N > M
2 print(f'RDD de partida {list(range(5))} N = 5')
3 print('Ejemplo 1: N > M')
4 numeros = sc.parallelize(list(range(5)))
5 rdd = numeros.sample(True, 0.5, 0)
6 print(rdd.collect())
7 # Ejemplo 2: N < M
8 print('Ejemplo 2: N < M')
9 numeros = sc.parallelize(list(range(5)))
10 rdd = numeros.sample(True, 1,1)
11 print(rdd.collect())
12 # Ejemplo 3: N = M
13 print('Ejemplo 3: N = M')
14 numeros = sc.parallelize(list(range(5)))
15 rdd = numeros.sample(False, 1,1)
16 print(rdd.collect())
```

```
1 RDD de partida [0, 1, 2, 3, 4] N = 5
2 Ejemplo 1: N > M
3 [3, 4]
4 Ejemplo 2: N < M
5 [0, 0, 1, 1, 2, 3, 3, 4, 4, 4, 4]
```

```
6 Ejemplo 3: N = M
7 [0, 1, 2, 3, 4]
```

Union

Dados dos RDD devuelve la unión de los dos, es decir, uno RDD con todos los elementos de ambos (cabe destacar que en caso de estar repetidos los elementos los volvería a repetir). Para este si N_1, N_2 son los respectivos tamaños de los RDD de entrada y M el de salida se cumple que $N_1 + N_2 = M$.

```
1 # Ejemplos 1:
2 n1 = sc.parallelize([1,2,3])
3 n2 = sc.parallelize([1,3,5])
4 m = n1.union(n2)
5 print (m.collect())
6
7 # Ejemplo 2: Une elementos de distinto tipo
8 n1 = sc.parallelize([1,2,3])
9 n2 = sc.parallelize(['1','3','5'])
10 m = n1.union(n2)
11 print (m.collect())
```

```
1 [1, 2, 3, 1, 3, 5]
2 [1, 2, 3, '1', '3', '5']
```

Pregunta TS2.2 Explica el funcionamiento de cada acción anterior

- Tengamos presente que `quijote` es un RDD donde cada elemento es una línea del quijote. Por lo que `quijote.count()` dirá el número de líneas del quijote.
- La variable `charsPerLine` contiene el número de caracteres que hay en cada línea, luego `numChars = charsPerLine.reduce(lambda a,b: a+b)` # also `charsPerLine.sum()` devuelve el número de caracteres total.
- La variable `allWordsNoArticles` contineee las palabras minimizadas con un único representante. por lo que

```
1 sortedWordsByLength = allWordsNoArticles.takeOrdered(20, key=lambda x:
    -len(x))
```

Toma las 20 primeras ordenadas de mayor tamaño a menor.

Pregunta: Implementa la opción count de otra manera:

- utilizando transformaciones map y reduce
- utilizando solo reduce en caso de que sea posible.

Respuesta

Utilizando transformaciones map y reduce Cada línea se mapeará por un 1 y con reduce se sumarán.

```
1 lineas = quijote.map(lambda linea : 1)
2 suma_lineas = lineas.reduce(lambda x,y: x+y)
3 print(suma_lineas)
```

```
1 5534
```

Utilizando solo reduce

La idea clave está en la función del reduce, que debe de ser capaz de tener como entrada cadenas de caracteres y números.

Si se encuentra con una cadena de caracter es porque se trata de una línea que debe de ser contada, por tanto la cambia a 1 y la suma.

Si se encuentra un número es de la cuenta anteriores y por tanto lo suma tal cual.

```
1 def reduce_suma_lineas(x,y):
2     if type(x) == str:
3         x = 1
4     if type(y) == str:
5         y = 1
6     return x+y
7
8 suma_lineas = quijote.reduce(reduce_suma_lineas)
9 print(suma_lineas)
```

```
1 5534
```

Pregunta TS2.3 Explica el proposito de cada una de las operaciones anteriores

Las operaciones en cadena son:

1. `. joinFreq.map(lambda e: (e[0], (e[1][0] - e[1][1])/(e[1][0] + e[1][1]))`
) Devuelve una tupla con la palabra y la diferencia entre el número de operaciones en la primera parte del Quijote menos las apariciones en la segunda parte entre la suma total de las apariciones en los dos libros.

La idea que subyace en el cálculo es ver si la frecuencia de aparición de palabras es la misma. Será próxima a cero si es así, cercana a menos uno si aparece mayoritariamente en el segundo y cercana a uno si lo es en primero.

2. `takeOrdered(10, lambda v: -v[1])` Ordena de mayor a menor las frecuencias obtenidas.
3. `joinFreq.map(lambda e: (e[0], (e[1][0] - e[1][1])/(e[1][0] + e[1][1]))`

Repite el paso 1.

4. `takeOrdered(10, lambda v: +v[1])` Ahora ordena de menor a mayor las frecuencias.

Pregunta TS2.4 ¿Cómo puede implementarse la frecuencia con `groupByKey` y transformaciones?

```
1 frequencies_group = words.groupByKey().map(lambda e: (e[0], sum(list(e
  [1])))
2 frequencies_group.take(10)
```

```
1 [('e', 1232),
2  ('hidalgo', 14),
3  ('don', 370),
4  ('mancha', 26),
5  ('saavedra', 1),
6  ('que', 3032),
7  ('condición', 16),
8  ('y', 2573),
9  ('del', 415),
10 ('d', 14)]
```

Optimizaciones

Pregunta TS2.5 ¿Cuál de las dos siguientes celdas es más eficiente? Justifique la respuesta.

La más eficiente es la segunda, puesto que se está guardando en caché los datos y la operación de cálculo de apariciones no tiene que repetirse.


```

1 joinFreq.map(lambda e: (e[0], (e[1][0] - e[1][1])/(e[1][0] + e[1][1])))
  .takeOrdered(10, lambda v: -v[1]), joinFreq.map(lambda e: (e[0], (e
    [1][0] - e[1][1])/(e[1][0] + e[1][1])))
  .takeOrdered(10, lambda v: +v
    [1])

```

```

1 ([('pieza', 0.8),
2  ('corral', 0.8),
3  ('rodelá', 0.7777777777777778),
4  ('curar', 0.75),
5  ('valle', 0.75),
6  ('entierro', 0.75),
7  ('oh', 0.7142857142857143),
8  ('licor', 0.7142857142857143),
9  ('difunto', 0.7142857142857143),
10 ('pago', 0.6666666666666666)],
11 [('teresa', -0.9767441860465116),
12  ('roque', -0.96),
13  ('paje', -0.9565217391304348),
14  ('duque', -0.9565217391304348),
15  ('blanca', -0.9565217391304348),
16  ('gobernador', -0.9503105590062112),
17  ('diego', -0.9459459459459459),
18  ('tarde', -0.9428571428571428),
19  ('mesmo', -0.9381443298969072),
20  ('letras', -0.9354838709677419)])

```

```

1 result = joinFreq.map(lambda e: (e[0], (e[1][0] - e[1][1])/(e[1][0] + e
  [1][1])))
2 result.cache()
3 result.takeOrdered(10, lambda v: -v[1]), result.takeOrdered(10, lambda
  v: +v[1])

```

```

1 ([('pieza', 0.8),
2  ('corral', 0.8),
3  ('rodelá', 0.7777777777777778),
4  ('curar', 0.75),
5  ('valle', 0.75),
6  ('entierro', 0.75),
7  ('oh', 0.7142857142857143),
8  ('licor', 0.7142857142857143),
9  ('difunto', 0.7142857142857143),
10 ('pago', 0.6666666666666666)],
11 [('teresa', -0.9767441860465116),
12  ('roque', -0.96),
13  ('paje', -0.9565217391304348),
14  ('duque', -0.9565217391304348),
15  ('blanca', -0.9565217391304348),
16  ('gobernador', -0.9503105590062112),
17  ('diego', -0.9459459459459459),
18  ('tarde', -0.9428571428571428),

```

```
19 ('mesmo', -0.9381443298969072),
20 ('letras', -0.9354838709677419)])
```

```
1 result.coalesce(numPartitions=2) # Avoids the data movement, so it
   tries to balance inside each machine
2 result.repartition(numPartitions=2) # We don't care about data movement
   , this balance the whole thing to ensure all machines are used
```

```
1 MapPartitionsRDD[633] at coalesce at NativeMethodAccessorImpl.java:0
```

```
1 result.take(10)
2 allWords.cache() # allWords RDD must stay in memory after computation,
   we made a checkpoint (well, it's a best effort, so must might be
   too strong)
3 result.take(10)
```

```
1 [('el', -0.5620334162815499),
2  ('hidalgo', -0.5),
3  ('don', -0.6255060728744939),
4  ('mancha', -0.5905511811023622),
5  ('saavedra', 0.0),
6  ('que', -0.5361077111383109),
7  ('y', -0.5789904278818621),
8  ('del', -0.5281409891984082),
9  ('en', -0.5704722945332837),
10 ('cuyo', -0.5217391304347826)]
```

Pregunta TS2.6 Antes de guardar el fichero, utilice coalesce con diferente valores ¿Cuál es la diferencia?

De acorde a la documentación de coalescence

```
1 allWords2.getNumPartitions()
```

```
1 2
```

```
1 for j in range(1,10):
2     print("numero de particiones = {}".format(j))
3     !rm -rf words_coalesce2
4     allWords = allWords2.coalesce(numPartitions=j).saveAsTextFile("
   words_coalesce2")
5     !ls words_coalesce2
```

```
1 numero de particiones = 1
2 part-000000 _SUCCESS
3 numero de particiones = 2
4 part-000000 part-000001 _SUCCESS
```

```
5 numero de particiones = 3
6 part-000000 part-000001 _SUCCESS
7 numero de particiones = 4
8 part-000000 part-000001 _SUCCESS
9 numero de particiones = 5
10 part-000000 part-000001 _SUCCESS
11 numero de particiones = 6
12 part-000000 part-000001 _SUCCESS
13 numero de particiones = 7
14 part-000000 part-000001 _SUCCESS
15 numero de particiones = 8
16 part-000000 part-000001 _SUCCESS
17 numero de particiones = 9
18 part-000000 part-000001 _SUCCESS
```

Se observa que el número de particiones de disco se estanca cuando el número de particiones llega a 2, dejando así de incrementar su valor para las sucesivas. Cabe destacar, que sería equivalente el usar la función `repartition`, ya que no se tiene en cuenta que los datos se muevan en el disco para relizar dichas particiones.

Parte3: Práctica opcional Hadoop y Spark

Dataset 1: Análisis del comportamiento local de los diferentes equipos que han jugado en la Premier League inglesa

Descripción del dataset

Puede encontrar los datos en [aquí](#), nos vamos a centrar en el fichero `results.csv` que cuenta con las siguientes 6 columnas:

- `home_team`: String, nombre del equipo que juega en casa, hay 39 valores únicos.
- `home_goals`: Number, número de goles marcados por el equipo local.
- `away_goals`: Número de goles marcados por el equipo visitante.
- `results`: String que puede ser `H`, `A`, `D` he indica al ganador. `H` para el local, `A` para el visitante, `D` si hay empate.

En total hay 4560 datos donde todos los atributos son correctos (ie no contiene valores perdidos o mal clasificados).

Objetivo de la aplicación

Se pretende saber cómo se comportan en local los equipos, es decir responder a las siguientes preguntas:

1. ¿Cómo juega cada equipo en local en cada temporada?
2. ¿Existe una tendencia a ganar más jugando como local?

Estructura de la solución propuesta

Los pasos a seguir son:

- Se cargarán los datos desde google drive.
- No se realizará un preprocesamiento ya que todos los datos son correctos.
- Con pyspark se realizará el tratamiento de los datos necesario para extraer la información.

Subida de datos

Se ha añadido a google drive el fichero de [resultados](#) (puede encontrarlo en la carpeta).

y ahora vamos a proceder a leerlo instalando antes todos los paquetes necesarios como ya hicimos en la práctica de Spark.

```
1 # instalar Java
2 !apt-get install openjdk-8-jdk-headless -qq > /dev/null
3 # Descargar la ultima versión de java ( comprobar que existen los path
  de descarga)
4 !wget -q https://downloads.apache.org/spark/spark-3.3.0/spark-3.3.0-bin
  -hadoop3.tgz
5 !tar xf spark-3.3.0-bin-hadoop3.tgz
6 # instalar pyspark
7 !pip install -q pyspark
8 # variables de entorno
9 import os # libreria de manejo del sistema operativo
10 os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
11 os.environ["SPARK_HOME"] = "/content/spark-3.3.0-bin-hadoop3"
12 from pyspark.sql import SparkSession
13 # Iniciamos sesión con spark
14 APP_NAME = "PDGE-AnálisisFútbol"
15 SPARK_URL = "local[*]"
16 spark = SparkSession.builder.appName(APP_NAME).master(SPARK_URL).
  getOrCreate()
17 spark
```

```
1 <div>
2   <p><b>SparkSession - in-memory</b></p>
```

```
1 <p><b>SparkContext</b></p>
2
3 <p><a href="http://82af4c36730e:4040">Spark UI</a></p>
4
5 <dl>
6   <dt>Version</dt>
7   <dd><code>v3.3.0</code></dd>
8   <dt>Master</dt>
9   <dd><code>local[*]</code></dd>
10  <dt>AppName</dt>
11  <dd><code>PDGE-AnalisisFutbol</code></dd>
12 </dl>
```

```
1 </div>
```

```
1 # Cargamos contexto de spark
2 sc = spark.sparkContext
3 #obtener el contexto de ejecución de Spark del Driver.
4 sc
```

```
1 <p><b>SparkContext</b></p>
2
3 <p><a href="http://82af4c36730e:4040">Spark UI</a></p>
4
5 <dl>
6   <dt>Version</dt>
7   <dd><code>v3.3.0</code></dd>
8   <dt>Master</dt>
9   <dd><code>local[*]</code></dd>
10  <dt>AppName</dt>
11  <dd><code>PDGE-AnalisisFutbol</code></dd>
12 </dl>
```

```
1 # Lectura del fichero desde google drive
2 from google.colab import drive
3 #drive.flush_and_unmount()
4 drive.mount('/content/gdrive')
```

```
1 Drive already mounted at /content/gdrive; to attempt to forcibly
  remount, call drive.mount("/content/gdrive", force_remount=True).
```

```
1 # Nota: He tenido que añadir la ruta esta para concreta para que lo lea
  en mi drive
2 data = sc.textFile("gdrive/MyDrive/Colab\ Notebooks/results.csv")
```

```
1 data.take(10)
```

```
1 ['home_team,away_team,home_goals,away_goals,result,season',
2  'Sheffield United,Liverpool,1.0,1.0,D,2006-2007',
3  'Arsenal,Aston Villa,1.0,1.0,D,2006-2007',
4  'Everton,Watford,2.0,1.0,H,2006-2007',
5  'Newcastle United,Wigan Athletic,2.0,1.0,H,2006-2007',
6  'Portsmouth,Blackburn Rovers,3.0,0.0,H,2006-2007',
7  'Reading,Middlesbrough,3.0,2.0,H,2006-2007',
8  'West Ham United,Charlton Athletic,3.0,1.0,H,2006-2007',
9  'Bolton Wanderers,Tottenham Hotspur,2.0,0.0,H,2006-2007',
10 'Manchester United,Fulham,5.0,1.0,H,2006-2007']
```

Vamos a procesar los datos para que podamos trabajar con ellos.

Para ello:

1. Eliminamos primera fila
2. Realizamos un split elemento a elemento. Esto nos dará una lista de listas, donde para cada sublista los elementos serán:

Atributo	Índice
home_team	0
away_team	1
home_goals	2
away_goals	3
result	4
season	5

```
1 data_without_head = data.filter(lambda x: "-" in x)
2 splited_data = data_without_head.map(lambda d: tuple(d.split(',')))
3 splited_data.take(5)
```

```
1 [('Sheffield United', 'Liverpool', '1.0', '1.0', 'D', '2006-2007'),
2  ('Arsenal', 'Aston Villa', '1.0', '1.0', 'D', '2006-2007'),
3  ('Everton', 'Watford', '2.0', '1.0', 'H', '2006-2007'),
4  ('Newcastle United', 'Wigan Athletic', '2.0', '1.0', 'H', '2006-2007')
5  ,
6  ('Portsmouth', 'Blackburn Rovers', '3.0', '0.0', 'H', '2006-2007')]
```

Con esto ya estamos preparados para realizar los primeros cálculos.

Los equipos que más victorias tenga en local por años, para ello primero sacaremos los años distintos que hay.

```
1 get_season = splited_data.map(lambda x: x[5]).distinct()
2 season_list = get_season.collect()
3 season_list.sort()
4 print(season_list)
```

```
1 ['2006-2007', '2007-2008', '2008-2009', '2009-2010', '2010-2011', '
   2011-2012', '2012-2013', '2013-2014', '2014-2015', '2015-2016', '
   2016-2017', '2017-2018']
```

```
1 results = {}
2 # Solo si se ha ganado contará como una victoria en local
3 letter_to_victory = lambda x : x == 'H' if 1 else 0
4 for season in season_list:
5     # 1. Filtramos los datos de cada temporada
6     # 2. Mapeamos cada tupla para que solo quede (clave: equipo local,
7         valor: 1 si ganó 0 en caso contrario)
8     filtered_season = splited_data.filter(lambda x: x[5]== season).map(
9         lambda x : (x[0],letter_to_victory(x[4])))
10    # 3. Sumamos las victorias de cada equipo
11    reduced = filtered_season.reduceByKey(lambda a,b: a+b)
12    results[season] = reduced.collect()
13    results[season].sort(key= lambda x: -x[1]) # Los mejores resultados
14    antes
```

Muestra de cada victoria Como resultado de la operación tenemos un diccionario con cada una de las victorias en local ordenadas

```
1 top = 3 # número de mejores jugadores
2 for season in [season_list[2],season_list[5], season_list[-3],
3     season_list[-1]]:
4     print(f'\nLos {top} mejores equipos en {season} fueron:')
5     equipos = results[season][:top]
6     [print(f"{i+1}. {equipos[i][0]} con {equipos[i][1]} victorias locales
7         ") for i in range(0,top)]
```

```
1 Los 3 mejores equipos en 2008-2009 fueron:
2 1. Manchester United con 16 victorias locales
3 2. Manchester City con 13 victorias locales
4 3. Liverpool con 12 victorias locales
5
6 Los 3 mejores equipos en 2011-2012 fueron:
7 1. Manchester City con 18 victorias locales
8 2. Manchester United con 15 victorias locales
9 3. Tottenham Hotspur con 13 victorias locales
10
```

```
11 Los 3 mejores equipos en 2015-2016 fueron:
12 1. Manchester United con 12 victorias locales
13 2. Manchester City con 12 victorias locales
14 3. Leicester City con 12 victorias locales
15
16 Los 3 mejores equipos en 2017-2018 fueron:
17 1. Manchester City con 16 victorias locales
18 2. Manchester United con 15 victorias locales
19 3. Arsenal con 15 victorias locales
```

```
1 # Vamos a mostrar los resultados totales
2 #!pip install -q tabulate # para imprimir tablas bonitas
3 from tabulate import tabulate
4
5 cabecera = ['Posición', 'Equipo', 'Victorias locales']
6 for season in season_list:
7     equipos = results[season]
8     tabla = [ [i+1, e[0], e[1]] for i,e in enumerate(equipos)]
9     print(f"\n\n#### Resultados temporada {season} \n")
10    print(tabulate(tabla, cabecera, tablefmt="github"))
```

La siguiente cuestión es si existe un beneficio a jugar en local

Para ello analizaremos el comportamiento de un equipo (introducir su nombre en la variable `equipo`) y después el de todos los equipos.

Formulación del experimento:

Realizaremos un Test de los rangos con signos de Wilcoxon ya que es el que más se adecua a la situación, donde

$$Z = \text{victorias locales} - \text{victorias como visitante}$$

para cada temporada.

- La hipótesis nula H_0 : Z sigue una distribución de media 0.
- La hipótesis alternativa H_1 : Z **no** sigue una distribución de media 0.

Vamos a proceder a calcular la variable Z para un equipo concreto:

Descripción del tratamiento de los datos

Se pretende calcular la variable z , esto es la diferencia entre victorias como local y como visitante para un cierto equipo en una cierta temporada, para ello:

1. Se filtran los datos para quedarnos solo con los concernientes al equipo buscado.

2. Se transformarán los datos obtenidos en (1) en par (`temporada`, `valor`) la temporada es un `str` y `valor` es un entero $-1, 1, 0$. que indica respectivamente si ganó como visitante, como local o ninguno de esos casos.
3. Se suman todos los datos de la misma temporada y el resultado es una lista con z por temporada.

```
1 equipo = 'Manchester United'
2
3 def traductor(x:tuple, equipo:str):
4     '''Devuelve:
5     1 si el equipo gana como local
6     -1 si el equipo gana como visitante
7     0 en caso contrario
8     '''
9     es_local = x[0] == equipo
10    salida = 0
11    # Si gana como local
12    if es_local and x[4] == 'H':
13        salida = 1
14    # Si gana como visitante
15    if not es_local and x[4] == 'A':
16        salida = -1
17    return salida
18
19 def victorias_locales_menos_visitante_por_equipo(equipo):
20     # Nos quedamos solo donde juegue el equipo
21     partidos_relevantes = splited_data.filter(lambda x: x[0]== equipo or
22                                                x[1] == equipo)
23     # nos quedamos con las victorias locales por sesión
24     victorias_locales = partidos_relevantes.map(lambda x:(x[5], traductor(
25         x, equipo)))
26     Z = victorias_locales.reduceByKey(lambda x,y: x+y).map(lambda x: x
27         [1])
28     z_list = Z.collect()
29     return z_list
30
31 print(f"La diferencia entre las victorias locales y como visitante del
32       {equipo} son: ")
33 print(victorias_locales_menos_visitante_por_equipo(equipo))
```

```
1 La diferencia entre las victorias locales y como visitante del
2   Manchester United son:
3 [8, -2, 5, 2, 7, 4, 5, 13, 2, 4, -1, 5]
```

Vamos a proceder a calcular el test, para ello utilizaremos la biblioteca de scipy

```
1 from scipy.stats import wilcoxon
2
3 def test_wilcoxon(equipo, a=0.05):
4     res = wilcoxon(victorias_locales_menos_visitante_por_equipo(equipo))
5     if res.pvalue < a :
```

```

6     print(f"Se ha rechazado la hipótesis nula con un p-valor de {res.
          pvalue} para el equipo {equipo}")
7     else:
8         print(f"No se ha podido rechazar la hipótesis nula ya que el p-
          valor es de {res.pvalue} para el equipo {equipo}")

```

```

1 # Vamos a mostrar algunos ejemplos concretos
2 equipos = ['Arsenal', 'Liverpool', 'Portsmouth', 'Manchester United']
3 for equipo in equipos:
4     test_wilcoxon(equipo)

```

```

1 Se ha rechazado la hipótesis nula con un p-valor de 0.00048828125 para
  el equipo Arsenal
2
3
4 /usr/local/lib/python3.7/dist-packages/scipy/stats/morestats.py:3141:
  UserWarning: Exact p-value calculation does not work if there are
  ties. Switching to normal approximation.
5     warnings.warn("Exact p-value calculation does not work if there are "
6
7
8 Se ha rechazado la hipótesis nula con un p-valor de
  0.010799590882009212 para el equipo Liverpool
9 No se ha podido rechazar la hipótesis nula ya que el p-valor es de 0.25
  para el equipo Portsmouth
10 Se ha rechazado la hipótesis nula con un p-valor de 0.00341796875 para
    el equipo Manchester United

```

Cabe destacar que el equipo Portsmouth descendió de división y que por tanto se tienen poco datos para su análisis, este puede ser el causante de que se pueda rechazar la hipótesis nula.

Diseño del tratamiento de los datos

Vamos a ver cuáles son los equipos y el número de temporadas en las que estuvieron, para ellos:

1. Transformamos cada tupla a un par (equipo, temporada).
2. Se eliminan las filas redundantes.
3. Sumamos cada aparición de cada equipo.

```

1 team_and_season = splited_data.map(lambda x: (x[0],x[5])).distinct()
2 team_appears = team_and_season.map(lambda x:(x[0],1)).reduceByKey(
    lambda x,y:x+y)
3 teams = team_appears.collect()
4 teams.sort(key=lambda x: -x[1])
5 ## Veamos cuales son los equipos que han jugado un mínimo de temporadas
6 temporadas_jugadas_minimas = 11
7 equipos_persistentes = team_appears.filter(lambda x: x[1] >=
    temporadas_jugadas_minimas).map(lambda x: x[0]).collect()

```

Vamos a realizar el experimento para todos los equipos que han jugado más de `temporadas_jugadas_minimas`.

```
1 print(f'Los equipos que han jugado más de {temporadas_jugadas_minimas}
   son: \n{equipos_persistentes}')
2 print('Veamos si estos equipos tienen más a ganar en local que en
   visitante')
3 for equipo in equipos_persistentes:
4     test_wilcoxon(equipo)
```

```
1 Los equipos que han jugado más de 11 son:
2 ['Manchester United', 'Chelsea', 'Manchester City', 'Arsenal', 'Everton
   ', 'West Ham United', 'Tottenham Hotspur', 'Liverpool']
3 Veamos si estos equipos tienen más a ganar en local que en visitante
4 Se ha rechazado la hipótesis nula con un p-valor de 0.00341796875 para
   el equipo Manchester United
5 Se ha rechazado la hipótesis nula con un p-valor de 0.04046183578416871
   para el equipo Chelsea
6 Se ha rechazado la hipótesis nula con un p-valor de
   0.007466958429676204 para el equipo Manchester City
7 Se ha rechazado la hipótesis nula con un p-valor de 0.00048828125 para
   el equipo Arsenal
8 Se ha rechazado la hipótesis nula con un p-valor de 0.00146484375 para
   el equipo Everton
9 Se ha rechazado la hipótesis nula con un p-valor de 0.0009765625 para
   el equipo West Ham United
10 Se ha rechazado la hipótesis nula con un p-valor de 0.00048828125 para
   el equipo Tottenham Hotspur
11 Se ha rechazado la hipótesis nula con un p-valor de
   0.010799590882009212 para el equipo Liverpool
```

Como vemos todos los equipos que han jugado más de 11 temporadas tienden a ganar con más partidos como locales que como visitantes.

Experimento 2 beneficio de ganar en local más para todos los equipos

Diseño tratamiento de los datos

Se busca como objetivo calcular la diferencia a ganar como local o como visitante de los equipos independientemente de la temporada.

1. Filtramos los empates que en nuestro caso no aportan información.
2. Transformamos los datos resultante en pares (`equipo ganados`, `-1` o `1`) donde `-1` corresponde si gana como visitante y `1` si lo hace como local.

3. Se suman todos los datos de un mismo equipo.

```

1 def traductor_dos(x):
2     '''
3     La clave es el nombre del equipo y el año.
4     '''
5     es_local = x[4] == 'H'
6     if es_local:
7         return x[0]+x[5], 1 # si gana como local sumamos 1 al equipo ese año
8     else:
9         return x[1]+x[5], -1 # si gana como visitante restamos uno al
10        equipo ese año
11 def victorias_locales_menos_visitante():
12     # Nos quedamos solo donde se gane a se pierda
13     partidos_relevantes = splited_data.filter(lambda x: x[4] != 'D')
14     victorias_locales = partidos_relevantes.map(traductor_dos)
15     Z = victorias_locales.reduceByKey(lambda x,y: x+y).map(lambda x: x
16        [1])
17     z_list = Z.collect()
18     return z_list
19 a = 0.05
20 res = wilcoxon(victorias_locales_menos_visitante())
21 if res.pvalue < a :
22     print(f"Se ha rechazado la hipótesis nula con un p-valor de {res.
23        pvalue}.")
24 else:
25     print(f"No se ha podido rechazar la hipótesis nula ya que el p-
26        valor es de {res.pvalue}.")

```

```

1 Se ha rechazado la hipótesis nula con un p-valor de 2.0695824579590253e
2 -34.

```

Conclusiones

Hemos extraído las victorias para cada equipo en cada temporada, que han sido:

Resultados temporada 2006-2007

Posición	Equipo	Victorias locales
1	Manchester United	15
2	Liverpool	14

Posición	Equipo	Victorias locales
3	Chelsea	12
4	Arsenal	12
5	Tottenham Hotspur	12
6	Reading	11
7	Everton	11
8	Portsmouth	11
9	Middlesbrough	10
10	Bolton Wanderers	9
11	Blackburn Rovers	9
12	West Ham United	8
13	Newcastle United	7
14	Aston Villa	7
15	Fulham	7
16	Sheffield United	7
17	Charlton Athletic	7
18	Manchester City	5
19	Wigan Athletic	5
20	Watford	3

Resultados temporada 2007-2008

Posición	Equipo	Victorias locales
1	Manchester United	17
2	Arsenal	14
3	Chelsea	12
4	Liverpool	12
5	Manchester City	11
6	Everton	11

Posición	Equipo	Victorias locales
7	Aston Villa	10
8	Sunderland	9
9	Reading	8
10	Wigan Athletic	8
11	Newcastle United	8
12	Tottenham Hotspur	8
13	Blackburn Rovers	8
14	Bolton Wanderers	7
15	Middlesbrough	7
16	West Ham United	7
17	Portsmouth	7
18	Birmingham City	6
19	Fulham	5
20	Derby County	1

Resultados temporada 2008-2009

Posición	Equipo	Victorias locales
1	Manchester United	16
2	Manchester City	13
3	Liverpool	12
4	Chelsea	11
5	Fulham	11
6	Arsenal	11
7	Stoke City	10
8	Tottenham Hotspur	10
9	West Ham United	9
10	Wigan Athletic	8

Posición	Equipo	Victorias locales
11	Everton	8
12	Portsmouth	8
13	Bolton Wanderers	7
14	Aston Villa	7
15	West Bromwich Albion	7
16	Sunderland	6
17	Blackburn Rovers	6
18	Middlesbrough	5
19	Newcastle United	5
20	Hull City	3

Resultados temporada 2009-2010

Posición	Equipo	Victorias locales
1	Chelsea	17
2	Manchester United	16
3	Arsenal	15
4	Tottenham Hotspur	14
5	Liverpool	13
6	Manchester City	12
7	Fulham	11
8	Everton	11
9	Blackburn Rovers	10
10	Sunderland	9
11	Aston Villa	8
12	Birmingham City	8
13	Burnley	7
14	Stoke City	7

Posición	Equipo	Victorias locales
15	West Ham United	7
16	Bolton Wanderers	6
17	Wigan Athletic	6
18	Hull City	6
19	Wolverhampton Wanderers	5
20	Portsmouth	5

Resultados temporada 2010-2011

Posición	Equipo	Victorias locales
1	Manchester United	18
2	Chelsea	14
3	Manchester City	13
4	Liverpool	12
5	Arsenal	11
6	Bolton Wanderers	10
7	Stoke City	10
8	Tottenham Hotspur	9
9	Everton	9
10	Aston Villa	8
11	Wolverhampton Wanderers	8
12	Fulham	8
13	West Bromwich Albion	8
14	Blackburn Rovers	7
15	Sunderland	7
16	Newcastle United	6
17	Birmingham City	6
18	Wigan Athletic	5

Posición	Equipo	Victorias locales
19	Blackpool	5
20	West Ham United	5

Resultados temporada 2011-2012

Posición	Equipo	Victorias locales
1	Manchester City	18
2	Manchester United	15
3	Tottenham Hotspur	13
4	Chelsea	12
5	Arsenal	12
6	Newcastle United	11
7	Fulham	10
8	Everton	10
9	Swansea City	8
10	Queens Park Rangers	7
11	Stoke City	7
12	Sunderland	7
13	Norwich City	7
14	Blackburn Rovers	6
15	Liverpool	6
16	West Bromwich Albion	6
17	Wigan Athletic	5
18	Aston Villa	4
19	Bolton Wanderers	4
20	Wolverhampton Wanderers	3

Resultados temporada 2012-2013

Posición	Equipo	Victorias locales
1	Manchester United	16
2	Manchester City	14
3	Chelsea	12
4	Everton	12
5	Arsenal	11
6	Tottenham Hotspur	11
7	Newcastle United	9
8	West Bromwich Albion	9
9	West Ham United	9
10	Liverpool	9
11	Norwich City	8
12	Fulham	7
13	Stoke City	7
14	Southampton	6
15	Swansea City	6
16	Aston Villa	5
17	Sunderland	5
18	Reading	4
19	Wigan Athletic	4
20	Queens Park Rangers	2

Resultados temporada 2013-2014

Posición	Equipo	Victorias locales
1	Manchester City	17
2	Liverpool	16

Posición	Equipo	Victorias locales
3	Chelsea	15
4	Arsenal	13
5	Everton	13
6	Tottenham Hotspur	11
7	Stoke City	10
8	Manchester United	9
9	Newcastle United	8
10	Southampton	8
11	Crystal Palace	8
12	Hull City	7
13	West Ham United	7
14	Aston Villa	6
15	Norwich City	6
16	Swansea City	6
17	Fulham	5
18	Sunderland	5
19	Cardiff City	5
20	West Bromwich Albion	4

Resultados temporada 2014-2015

Posición	Equipo	Victorias locales
1	Chelsea	15
2	Manchester United	14
3	Manchester City	14
4	Arsenal	12
5	Southampton	11
6	Stoke City	10

Posición	Equipo	Victorias locales
7	Liverpool	10
8	Tottenham Hotspur	10
9	West Ham United	9
10	Swansea City	9
11	Newcastle United	7
12	Leicester City	7
13	West Bromwich Albion	7
14	Everton	7
15	Queens Park Rangers	6
16	Crystal Palace	6
17	Aston Villa	5
18	Hull City	5
19	Burnley	4
20	Sunderland	4

Resultados temporada 2015-2016

Posición	Equipo	Victorias locales
1	Manchester United	12
2	Manchester City	12
3	Leicester City	12
4	Arsenal	12
5	Southampton	11
6	Tottenham Hotspur	10
7	West Ham United	9
8	Stoke City	8
9	Swansea City	8
10	Liverpool	8

Posición	Equipo	Victorias locales
11	Newcastle United	7
12	Everton	6
13	Norwich City	6
14	West Bromwich Albion	6
15	Sunderland	6
16	Watford	6
17	Crystal Palace	6
18	Chelsea	5
19	AFC Bournemouth	5
20	Aston Villa	2

Resultados temporada 2016-2017

Posición	Equipo	Victorias locales
1	Chelsea	17
2	Tottenham Hotspur	17
3	Arsenal	14
4	Everton	13
5	Liverpool	12
6	Manchester City	11
7	Burnley	10
8	Leicester City	10
9	AFC Bournemouth	9
10	West Bromwich Albion	9
11	Hull City	8
12	Manchester United	8
13	Swansea City	8
14	Watford	8

Posición	Equipo	Victorias locales
15	Stoke City	7
16	West Ham United	7
17	Southampton	6
18	Crystal Palace	6
19	Middlesbrough	4
20	Sunderland	3

Resultados temporada 2017-2018

Posición	Equipo	Victorias locales
1	Manchester City	16
2	Manchester United	15
3	Arsenal	15
4	Tottenham Hotspur	13
5	Liverpool	12
6	Chelsea	11
7	Everton	10
8	Newcastle United	8
9	Burnley	7
10	Watford	7
11	Crystal Palace	7
12	Brighton and Hove Albion	7
13	AFC Bournemouth	7
14	Leicester City	7
15	West Ham United	7
16	Huddersfield Town	6
17	Swansea City	6
18	Stoke City	5

Posición	Equipo	Victorias locales
19	Southampton	4
20	West Bromwich Albion	3

Además para los equipos de los que se tienen datos suficientes se ha encontrado que con más de un 95 % de confianza estos equipos tienden a ganar más como locales que como visitantes.

Finalmente se ha comprobado que de manera global este resultado también se ve avalado.

Conclusiones sobre Spark

Nótese que para los distintos apartados de la práctica hemos debido de extraer los siguientes datos:

- Victorias como local por equipo.
- El número de temporadas que juega cada equipo.
- Los equipos que juegan más de cierta cantidad de temporadas.
- Diferencias de victorias como locales y como visitantes de cierto equipo por temporadas.
- Diferencias de victorias como locales y como visitantes de todo los equipos por temporadas.

A pesar de ser python un lenguaje orientado a objetos la redacción de éstas tareas se han hecho con herramientas funcionales como: `reduce`, `map`, `filter` y `groupByKey` y en muy pocas líneas, lo cual demuestra la expresividad de tan lenguaje.