xLaboratorio 7.
Spark RDDs

INSTRUCCIONES:

En esta guía se detallan las instrucciones para poder completar los ejercicios que complementan el contenido visto en la semana. El objetivo de la misma es que el estudiante se familiarice y conozca los aspectos internos de un RDD, que son la base del Core de Spark, para luego facilitar el aprendizaje de otros componentes.

Este es un laboratorio guiado. Eso quiere decir que la mayor parte de ejercicios consiste en ir replicando el código compartido en cada sección e ir interactuando paulatinamente con los datos y cómo estos RDDs van cambiando con las transformaciones que se van realizando. Sin embargo, hay preguntas marcadas en **negritas** que deberá realizar usted por su cuenta o bien discutir los resultados que va viendo. Esto también está asociado con su nota, para lo cual puede consultar la sección de Evaluación.

Realizar de forma **individual**, si surge alguna duda siéntase en la comodidad y libertad de hacer preguntas a su catedrático. Para este único laboratorio, deberá instalar **pyspark** (pyspark==3.5.1 o pyspark==3.4.1) a través de pip en su equipo local. Es importante que considere que esta versión de Spark es compatible desde la versión 11 hasta la 17 de Java.

PREPARACIÓN DEL AMBIENTE

Dentro del material proporcionado, copie el archivo **constitution.txt** dentro de una carpeta en su ambiente de trabajo (workspace) de su ambiente.

A partir de este momento nos referiremos a esta ubicación como ambiente de trabajo (**working_dir**). El working_dir justamente será la ruta de su sistema de archivos en donde tendrá ubicado el archivo de texto. Habrán algunas celdas

Se le adjuntan también los archivos necesarios para crear un ambiente de Docker por si gusta utilizarlo.

EJERCICIOS

CREACIÓN DE LA APLICACIÓN

Debe importar la biblioteca SparkSession de pyspark.sql y crear una aplicación.

```
%python

>>> spark = SparkSession.builder.appName("MiApp").
master("local[*]").getOrCreate()
```

EXPLORACION BÁSICA

Vamos a cargar un RDD a partir del archivo `constitution.txt` que se encuentra en nuestro ambiente de trabajo.

```
%python

>>> rdd = sc.textFile(working_dir + 'constitution.txt')
```

Analicemos el RDD que acabamos de crear. ¿Qué tipo de data almacena? Tomemos una muestra para conocer la forma que tiene cada elemento del RDD:

```
%python

>>> rdd.take(3)
```

Observemos que la función sc.textFile() nos carga a partir de un archivo de texto un RDD donde cada elemento representa una fila del mismo. Con sólo una línea de código responda **¿cuántas líneas hay en el documento que cargamos?**

Y si lo que queremos responder es ¿cuántas palabras hay?, o ¿cuáles son las palabras que más se repiten? Para esto último, haremos las secciones a continuación.

WORD COUNT

Para poder responder la última pregunta, realizaremos este ejercicio que se le conoce como “Word Count”. En este laboratorio vamos a programar uno.

Lo primero que queremos hacer con el RDD que tenemos cargado es cambiar su forma, transformarlo. En este momento cada elemento del RDD es una línea de texto. Necesitamos separar esa línea en palabras.

```
%python

>>> splitted_lines = rdd.map(lambda line : line.split(' '))
>>> splitted_lines.take(3)
```

La estructura que conseguimos con este código, **¿cumple con nuestro propósito?** Si corremos un `splitted_lines.count()`, **¿conoceremos cuántas palabras hay en el texto?**

Aún no tenemos la estructura que buscamos, ya que en lugar de que cada elemento de nuestro RDD sea una palabra lo que tenemos es que cada elemento es una lista de palabras, lo cual es distinto.

Spark nos provee una forma de resolver este problema utilizando `.flatMap()`:

```
%python

>>> words_rdd = rdd.flatMap(lambda line : line.split(' '))
>>> words_rdd.take(3)
```

Ahora sí, podemos saber cuántas palabras hay en el documento

```
%python

>>> words_rdd.count()
8435
```

OJO: parte del trabajo de un científico de datos es cuestionarse los resultados que va obteniendo, y esto no es la excepción. Realmente el count que nos muestra como resultado no es un reflejo real de la cantidad de palabras que tenemos. Esto es debido a cómo funciona el método `.split()` en python, que tienen un comportamiento distinto cuando el separador que colocamos como parámetro (" ") se encuentra al principio o al final del string. Veamos un ejemplo:

```
%python

>>> splitted = "Hola esta es una prueba! ".split(' ')
>>> splitted
```

```
['Hola,', 'esta', 'es', 'una', 'prueba!', '']
>>> len(splitted)
6
# Nos da 6 como resultado cuando realmente solo hay 5 palabras.
```

El comportamiento que buscamos que lo que queremos son palabras y a pesar que el string vacío `''` es un elemento, no queremos que cuente como palabra. Hagamos un ajuste:

```
%python

>>> words_rdd = rdd.flatMap(lambda line : line.strip().split(' '))
>>> words_rdd.count()
7763
```

¡Ahora sí!

Hemos usado la función `.count()` para conocer la cantidad de elementos de nuestro RDD. A diferencia de los momentos en los que usamos `.map()` y `.flatMap()`, esta sí ejecutó procesamiento en Spark debido a que es una acción y las anteriores son sólo transformaciones.

Ejecutemos ahora una acción distinta, como buscar cuál es la palabra más larga en el texto. Ya que no existe una acción *prefabricada* como el `.count()`, nos tocará hacer nuestro propio `.reduce()`:

```
%python

>>> words_rdd.reduce(lambda a, b : a if len(a) > len(b) else b)
u'Representatives,'
```

La función anónima que pasamos como parámetro del método `.reduce()`, al igual que las que hemos pasado en los métodos `.map()`, `.flatMap()` o `.filter()`, es una función que el framework de Spark aplicará a los elementos del RDD para lograr el objetivo del método. En esta ocasión debe recibir dos argumentos y ser conmutativa y asociativa para poder garantizar su correcta ejecución en paralelo.

Como pudimos ver en el output, la palabra más larga de nuestro texto tiene 16 caracteres, pero ¿ha notado algo extraño?

Lo que comparamos es el tamaño de cada string y en ningún momento le decimos a nuestro programa que las palabras están compuestas sólo de caracteres alfanuméricos y que la coma (‘,’) o

cualquier otro caracter que sea un signo de puntuación no es parte de una palabra, aún cuando no hay espacios de por medio.

Utilizando el método `.isalnum()` de los strings, haga una modificación en el código de manera que el RDD `words_rdd` tenga palabras formadas exclusivamente por caracteres alfanuméricos.

KEY-VALUE RDDs

A pesar que la mayoría de operaciones de Spark funcionan sobre RDDs que contienen cualquier tipo de dato, existen algunas operaciones que se pueden aplicar únicamente a un tipo determinado de RDDs. Estos son los que tienen la forma key-value, y algunas operaciones especiales son por ejemplo `.reduceByKey()` ó `.groupByKey()`.

Si en la sección anterior también quisiéramos conocer cuáles son las palabras más repetidas del texto nos veríamos en un problema si no contáramos con estas operaciones Key-Value, ya que necesitamos ejecutar un conteo de manera individual para cada palabra, en lugar de uno general que logramos con el `.count()`. Podemos comenzar convirtiendo nuestro RDD en un RDD especial de Key-Values:

```
%python

>>> keyval_rdd = words_rdd.map(lambda word : (word, 1))
```

Veamos algunos elementos de nuestro RDD con `.take(3)`. Ahora cada elemento de nuestro RDD es una Tupla. Sin embargo, el framework de Spark ya lo reconoce como una pareja Key-Value y "desbloquea" varias operaciones específicas para este tipo de RDDs como lo es el `.reduceByKey()` que vamos a utilizar de la misma manera que utilizamos antes un `.reduce()` pero este sólo juntará los elementos que tengan la misma llave (key):

```
%python

>>> wordcount = keyval_rdd.reduceByKey(lambda a, b : a + b)
```

Ahora ya tenemos una tupla para cada palabra con su conteo. Note como a diferencia del `.reduce()`, el `.reduceByKey()` no es una acción sino una transformación. Esto es debido a que lo que genera es un nuevo RDD, y con este todavía se pueden hacer operaciones.

De este nuevo RDD, ¿cómo conocemos cuáles son las 5 palabras más repetidas? Podríamos tomarlas todos con un `.collect()` y usar operaciones de Python sobre la lista de tuplas para conocer la palabra con el conteo máximo, pero eso no es viable cuando se trabaja con datasets más grandes.

Lo que haremos será usar la operación `.sortByKey()`, sin embargo no queremos ordenar las palabras como tal (A-Z) por lo que primero vamos a invertir las tuplas para que lo que se ordene sean los conteos. Ahora los counts serán nuestra *key* y la palabra nuestro *value*:

```
%python

>>> sorted = (wordcount
...   .map(lambda x: (x[1], x[0]))
...   .sortByKey(ascending=False)
... )


>>> sorted.take(5)
[(662, u'the'), (494, u'of'), (306, u'shall'), (258, u'and'), (184, u'to')]
```

Con esto, tenemos el resultado que buscábamos: podemos conocer cuáles son las 5 palabras que más se repiten en nuestro texto y cuáles son sus respectivos conteos. De una manera similar, usando las funciones de Spark, podríamos ejecutar casi cualquier análisis que se nos ocurra sobre un dataset de cualquier tamaño.

Finalmente, como se pudo dar cuenta, el resultado tiene un RDD que cuenta con múltiples stop words. **Aplique los conocimientos que adquirió en minería de textos y muestre las 5 palabras más repetidas excluyendo las stop words de su RDD resultante**.

EVALUACIÓN

La evaluación depende de la ejecución y replicación de las celdas aquí descritas y aquellas que debió de realizar y explicar usted mismo. En caso esté incompleto, obtendrá la mitad de la nota.

- **Entrega completa: 100 puntos** (incluye código guiado y discusión propia)
- **Entrega parcial: 50 puntos** (incluye el código guiado y algunas partes de la discusión propia)
- **Sin entrega: 0 puntos** (omite entregar o bien, omite algunas partes del código guiado)

MATERIAL A ENTREGAR

- Script de Python (.ipynb) que utilizó con la disucsión generada utilizando markdown.

FECHAS DE ENTREGA

- **DOCUMENTO FINAL COMPLETO:** 25 de septiembre de 2025 a las 17:20.