Spark Practice 1: Simple Join khordad 99

Make sure first you were able to complete the "Setup PySpark on the Cloudera VM" tutorial .
In this practice you will implement in Spark the same code in the previous practice to perform a join of 2 different WordCount datasets.

Load datasets

First of all open the **pyspark** shell and load the datasets you created for the previous practice from HDFS:

```
fileA = sc.textFile("input/join1_FileA.txt")
```

Let's make sure the file content is correct:

```
fileA.collect()
```

should return:

```
Out[]: [u'able,991', u'about,11', u'burger,15', u'actor,22']
```

Then load the second dataset:

```
fileB = sc.textFile("input/join1_FileB.txt")
```

same verification:

```
fileB.collect()
Out[29]:
[u'Jan-01 able,5',
 u'Feb-02 about,3',
 u'Mar-03 about,8 ',
 u'Apr-04 able,13',
 u'Feb-22 actor,3',
 u'Feb-23 burger,5',
 u'Mar-08 burger,2',
 u'Dec-15 able,100']
```

Mapper for fileA

First you need to create a map function for fileA that takes a line, splits it on the comma and turns the count to an integer.
You need to copy paste the following function into the pyspark console, then edit the 2 <ENTER_CODE_HERE> lines to perform the necessary operations:

```
def split_fileA(line):
    # split the input line in word and count on the comma
    <ENTER_CODE_HERE>
    # turn the count to an integer
    <ENTER_CODE_HERE>
    return (word, count)
```

You can test your function by defining a test variable:

```
test_line = "able,991"
```

and make sure that:

```
split_fileA(test_line)
```

returns:

```
Out[]: ('able', 991)
```

Now we can proceed on running the map transformation to the fileA RDD:

```
fileA_data = fileA.map(split_fileA)
```

If the mapper is implemented correctly, you should get this result:

```
fileA_data.collect()
Out[]: [(u'able', 991), (u'about', 11), (u'burger', 15), (u'actor', 22)]
```

Make sure that the key of each pair is a string (i.e. is delimited by ' ‘) and the value is an integer.

Mapper for fileB

The mapper for fileB is more complex

```
def split_fileB(line):
    # split the input line into word, date and count_string
    <ENTER_CODE_HERE>
    <ENTER_CODE_HERE>
    return (word, date + " " + count_string)
```

running:

```
fileB_data = fileB.map(split_fileB)
```

and then gathering the output back to the pyspark Driver console:

```
fileB_data.collect()
```

should give the result:

```
Out[]:
[(u'able', u'Jan-01 5'),
 (u'about', u'Feb-02 3'),
 (u'about', u'Mar-03 8 '),
 (u'able', u'Apr-04 13'),
 (u'actor', u'Feb-22 3'),
 (u'burger', u'Feb-23 5'),
 (u'burger', u'Mar-08 2'),
 (u'able', u'Dec-15 100')]
```

Run join

The goal is to join the two datasets using the words as keys and print for each word the wordcount for a specific date and then the total output from A.

Basically for each word in fileB, we would like to print the date and count from fileB but also the total count from fileA.

Spark implements the join transformation that given a RDD of (K, V) pairs to be joined with another RDD of (K, W) pairs, returns a dataset that contains (K, (V, W)) pairs.

```
fileB_joined_fileA = fileB_data.join(fileA_data)
```

Verify the result

You can inspect the full result with:

```
fileB_joined_fileA.collect()
```

You should try to make sure that this result agrees with what you were expecting.

Submit one line for grading

Finally, you need to create a text file with just one line for submission.

From the Cloudera VM, open the text editor from Applications > Accessories > gedit Text Editor.

Paste 1 single line of code from your pyspark console, the line related to the word actor:

```
 (u'actor', ??????????)
```

do NOT copy the comma at the end, the line should start with open parenthesis (and end with closed parenthesis).Check your Results with your TA for grading.

Spark Practice 2: Advanced Join Week13: 9-15 Ordibehesht

Make sure first you were able to complete the "Setup PySpark on the Cloudera VM".

Verify input data

In this practice you will use the data you generated in the previous practice **(Practice 1-2 Map/Reduce join).**

Make sure the 6 files are available in the HDFS input/ folder by running this command in the terminal (not in PySpark!):

```
hdfs dfs -ls input/
```

Should contain the 6 files:

```
input/join2_genchanA.txt
input/join2_genchanB.txt
input/join2_genchanC.txt
input/join2_gennumA.txt
input/join2_gennumB.txt
input/join2_gennumC.txt
```

Goal of the programming Practice

The gennum files contain show names and their viewers, genchan files contain show names and their channel. **We want to find out the total number of viewers across all shows for the channel BAT.**

Read shows files

The gennum files contain show names and number of viewers. You can read them into Spark with a pattern matching, see the ? which will match either A, B or C:

```
show_views_file = sc.textFile("input/join2_gennum?.txt")
```

Remember you can check what Spark is doing by copying some elements of an RDD back to the driver:

```
show_views_file.take(2)
```

will return the first 2 elements of the dataset:

```
[u'Hourly_Sports,21', u'PostModern_Show,38']
```

Parse shows files

Next you need to write a function that splits and parses each line of the dataset.

```
def split_show_views(line):
    <INSERT_CODE_HERE>
    return (show, views)
```

Then you can use this function to transform the input RDD:

```
show_views = show_views_file.<INSERT_CODE_HERE>(split_show_views)
```

By now you should know how to check that the show_views RDD is how you expect.

Read channel files

The genchan files contain show names and channel. You can read into Spark all of them with a pattern matching, see the ? which will match either A, B or C:

```
show_channel_file = sc.textFile("input/join2_genchan?.txt")
```

Parse channel files

Write a function to parse each line of the dataset:

```
def split_show_channel(line):
    <INSERT_CODE_HERE>
    return (show, channel)
```

Use it to parse the channel files:

```
show_channel = show_channel_file.<INSERT_CODE_HERE>
```

Join the 2 datasets

At this point you should use the join transformation to join the 2 datasets using the show name as the key. You can join the datasets in any order, as long as you are consistent, both are fine.

```
joined_dataset = <INSERT_CODE_HERE>
```

Extract channel as key

You want to find the total viewers by channel, so you need to create an RDD with the channel as key and all the viewer counts, whichever is the show.

```
def extract_channel_views(show_views_channel):
    <INSERT_CODE_HERE>
    return (channel, views)
```

Now you can apply this function to the joined dataset to create an RDD of channel and views:

```
channel_views = joined_dataset.<INSERT_CODE_HERE>(extract_channel_views)
```

Sum across all channels

Finally, we need to sum all of the viewers for each channel:

```
def some_function(a, b):
    <INSERT_CODE_HERE>
    return some_result
```

This is the final stage of your analysis, so you can copy the results back to the Driver with collect:

```
channel_views.<INSERT_CODE_HERE>(some_function).collect()
```

Submit one line for grading

Finally, you need to create a text file with just one line for submission.

From the Cloudera VM, open the text editor from Applications > Accessories > gedit Text Editor.

Paste 1 single number into gedit, **the number of viewers for the BAT channel, with no punctuation, spaces, commas. Just the digits of the number.**

In gedit, click on the Save button and save it in the default folder (/home/cloudera) with the name **yourname_yourID_bat_viewers.txt.** **Finally, Check your Results with your TA for grading.**