s299165_lab3

January 21, 2024

Laboratory 3 - Spark SQL

In this lab, we will analyze historical data about theusage of the bike sharing system of Barcelona. We will consider the occupancy of the stations where users can pick up or drop offbikes. Your task consists in identifying the most “critical” timeslots (day of the week,hour) for each station.

1 Input Files

The analysis is based on 2 files available in the HFDS shared folder of theBigData@Polito cluster

1.1 register.csv

This contains the historical information about the number of used and free slots for~3000 stations from May 2008 to September 2008.

1.1.1 Questions

- Question 1.1.1. How many rows of data we obtain before and after the data cleaning above?
- Question 2.1. Computes the criticality value C(Si, Tj) for each pair (Si, Tj)
- Question 2.2. Selects only the critical pairs (Si, Tj) having a criticality value C(Si, Tj) greater than a minimum threshold. The minimum criticality (threshold is a float between 0 and 1 passed as an argument of the application.
- Question 2.3. Order the results by increasing criticality. If there are two or more records characterized by the same criticality value, consider the station id value (in ascending order). If also the station is the same, consider the day of the week(ascending from Monday to Sunday) and finally the hour (ascending from 0 to 23).
- Question 2.4. Store the sorted critical pairs in the output folder (also an argumentof the application), by using a csv files (with header) , where columns areseparated by “tab”. Store exactly the following attributes separated by a “tab”
- Question 2.5. How many critical pairs do you obtain? Report also the complete output resultof the applications.

Answer: (Sloution_Version 1) The code below shows loading a file from the CSV files, filtering out rows with “used_slots = 0” and “free_slots = 0,” and then printing the number of data before and after the cleaning process. The provided outputs are: 25319028 before cleaning and 25104121 after cleaning. The questions are answered in the codes below (creating pairs, sorting them, filtering outputs, creat new file to save with headings and tab separation) and 5 stations are the results as critical stations.

[1]:
```python
from datetime import datetime

#Address for input & output
inputPath_register = "/data/students/bigdata_internet/lab3/register.csv"
inputPath_stations = "/data/students/bigdata_internet/lab3/stations.csv"
outputaddress ="res_out_Lab3_1/"
```

[2]:
```python
#Read the register file (RDD Method)
readFile_register = sc.textFile(inputPath_register)
```

[3]:
```python
#Remove the header
header_register = readFile_register.first()
RDD_readFile_register = readFile_register.filter(lambda x: x != header_register)
```

[4]:
```python
#Number of data before cleaning
RDD_readFile_register.count()
```

[4]:
```
25319028
```

[5]:
```python
#Considering the filter for "used_slots = 0" and "free_slots = 0"
RDD_filtered = RDD_readFile_register.map(lambda line: line.split('\t')).
 ↪filter(lambda line: \
                                          (line[2] != '0') or (line[3] != '0')).
 ↪filter(lambda line: not (line[2] == '0' and line[3] == '0'))
```

[6]:
```python
#Number of data after cleaning
RDD_filtered.count()
```

[6]:
```
25104121
```

[7]:
```python
#A function to extract date and make tuple (day, hour)
def extract_station_day_hour(line):
    timestamp = line[1]
    station = line[0]
    day = datetime.strptime(timestamp, '%Y-%m-%d %H:%M:%S').strftime('%A')
    hour = datetime.strptime(timestamp, '%Y-%m-%d %H:%M:%S').hour
    return (station, (day, hour))
```

[8]:
```python
#extract timestamp for all recorded data & output is like ('1', ('Thursday',␣
 ↪12))
```

```
RDD_station_timeslots = RDD_filtered.map(lambda line:␣
↪extract_station_day_hour(line))
```

[9]:
```
#Assign a number to each line of (station , (day, hour)) for ALL RECORDS &␣
↪output is like (('1', ('Thursday', 12)), 1)
RDD_data_record_St = RDD_station_timeslots.map(lambda line: (line, 1))
```

[10]:
```
#Sum the total number of records for each station, at each hour & output is␣
↪like (('1', ('Thursday', 14)), 508)
RDD_totalStationHourlyRecords = RDD_data_record_St.reduceByKey(lambda x, y: x+y)
```

[11]:
```
#A function to extract stations with "0" free slots (critical stations)
def extract_station_freeSlot(line):
    freeSlot = line[3]
    if freeSlot == '0':
        return line
    else:
        return None
```

[12]:
```
#extract stations with 0 free slots & output is like ['1', '2008-05-17 14:56:
↪00', '20', '0']
criticalRecords = RDD_filtered.map(extract_station_freeSlot).filter(lambda x: x␣
↪is not None)
```

[13]:
```
#extract timestamp for critical stations & output is like ('1', ('Saturday',␣
↪14))
RDD_CRITICAL_station_timeslots = criticalRecords.map(lambda line:␣
↪extract_station_day_hour(line))
```

[14]:
```
#Assign a number to each line of (station , (day, hour)) for CRITICAL STATIONS␣
↪& output is like (('1', ('Thursday', 12)), 1)
RDD_CRITICAL_station_St = RDD_CRITICAL_station_timeslots.map(lambda line:␣
↪(line, 1))
```

[15]:
```
#Sum the total number of records for each CRITICAL STATION, at each hour &␣
↪output is like (('1', ('Sunday', 22)), 43)
RDD_totalCRITICALStationHourlyRecords = RDD_CRITICAL_station_St.
↪reduceByKey(lambda x, y: x+y)
```

[16]:
```
#Join the data of stations and data of CRITICAL STATIONs, to achieve the␣
↪criticality factor & output is like (('55', ('Thursday', 21)), (585, 55))
RDD_station_and_critical_join = RDD_totalStationHourlyRecords.
↪join(RDD_totalCRITICALStationHourlyRecords)
```

[17]:
```
#Computing the Criticality Factor Function
def criticality_factor(data):
    Cf = data[1][1] / data[1][0]
    return (data[0], round(Cf, 3))
```

[18]:
```
#Computing the Criticality Factor for stations at each hour & output is like␣
 ↪(('263', ('Thursday', 16)), 0.12)
stations_criticality = RDD_station_and_critical_join.map(criticality_factor)
```

[19]:
```
#Subtract the data of stations and data of CRITICAL STATIONs, to achieve the␣
 ↪stations with criticality equal to '0'
RDD_station_not_critical = RDD_totalStationHourlyRecords.
 ↪subtractByKey(RDD_totalCRITICALStationHourlyRecords)
```

[20]:
```
#assigning 0 to stations that are not in critical situation
station_with_Zerocriticality = RDD_station_not_critical.map(lambda x: (x[0], 0.
 ↪000))
```

[21]:
```
#Use union method, to have all critical and normal stations as the result & the␣
 ↪output is like (('263', ('Thursday', 16)), 0.12)
result_register = station_with_Zerocriticality.union(stations_criticality)
```

[22]:
```
#Filtering the stations with minimum threshold of criticality equal to 0.6
station_critical_filter = result_register.filter(lambda x: x[1] >= 0.6)
```

[23]:
```
# Sort the RDD by increasing criticality, station ID, and day of the week
sorted_station_critical_filter = station_critical_filter.sortBy(lambda x:␣
 ↪(x[1], x[0][0], x[0][1][0]))
```

[24]:
```
#Count the number of critical stations
sorted_station_critical_filter.count()
```

[24]:
```
5
```

[27]:
```
#Show the critical stations
sorted_station_critical_filter.collect()
```

[27]:
```
[(('9', ('Friday', 10)), 0.613),
 (('10', ('Saturday', 0)), 0.622),
 (('58', ('Monday', 1)), 0.624),
 (('9', ('Friday', 22)), 0.626),
 (('58', ('Monday', 0)), 0.632)]
```

[28]:
```
#Read the station file
readFile_station = sc.textFile(inputPath_stations)
```

[29]:
```
#Remove the header
header_station = readFile_station.first()
RDD_readFile_station = readFile_station.filter(lambda x: x != header_station)
```

[30]:
```
#map the data using split function
RDD_station = RDD_readFile_station.map(lambda line: line.split('\t'))
```

[31]:
```
# Extract station ID from result_register and use it as a key & output is like␣
 ↪('211', (('211', ('Thursday', 23)), 0.0))
result_register_key = sorted_station_critical_filter.map(lambda x: (x[0][0], x))
```

[32]:
```
# Extract station ID from stations and use it as a key & output is like ('1',␣
 ↪['1', '2.180019', '41.397978', 'Gran Via Corts Catalanes'])
RDD_station_key = RDD_station.map(lambda x: (x[0], x))
```

[33]:
```
# Join the two RDDs based on the station ID & output is like ('1', ((('1',␣
 ↪('Thursday', 12)), 0.0),['1', '2.180019', '41.397978', 'Gran Via Corts␣
 ↪Catalanes']))
join_register_station = result_register_key.join(RDD_station_key)
```

[34]:
```
# Transform the result to the required format (Station ID, Longitude, ,␣
 ↪Latitude, Day, Criticality)
output_RDD = join_register_station.map(lambda x: "\t".join(map(str,␣
 ↪[int(x[0]),float(x[1][1][1]), float(x[1][1][2]), x[1][0][0][1][0],␣
 ↪x[1][0][1]])))
```

[35]:
```
#Add Header, The Header should be parallelized here!!
New_header = "Station_ID\tLongitude\tLatitude\tDay\tHour\tCriticality"
output_with_header = sc.parallelize([New_header]).union(output_RDD)
```

[36]:
```
output_with_header.collect()
```

[36]:
```
['Station_ID\tLongitude\tLatitude\tDay\tHour\tCriticality',
 '9\t2.185294\t41.385006\tFriday\t0.613',
 '9\t2.185294\t41.385006\tFriday\t0.626',
 '58\t2.170736\t41.377536\tMonday\t0.632',
 '58\t2.170736\t41.377536\tMonday\t0.624',
 '10\t2.185206\t41.384875\tSaturday\t0.622']
```

[37]:
```
#Save the file
#output_with_header.saveAsTextFile(outputaddress)
```

Answer: (Sloution_Version 2) The code below shows loading a DataFrame from the CSV files, filtering out rows with "used_slots = 0" and "free_slots = 0," and then printing the number of data before and after the cleaning process. The provided outputs are: 25319028 before cleaning and 25104121 after cleaning. The questions are answered in the codes below (creating pairs, sorting them, filtering outputs, creat new file to save with headings and tab separation) and 5 stations are the results as critical stations. (Exactly the same results are gained)

[38]:
```
from pyspark.sql.functions import to_timestamp , hour , dayofmonth , dayofweek␣
 ↪, date_format , col , round

#Address for input & output
inputPath_register = "/data/students/bigdata_internet/lab3/register.csv"
inputPath_stations = "/data/students/bigdata_internet/lab3/stations.csv"
outputaddress ="res_out_Lab3_1/"
```

[39]:
```
#Read the Register File
dfProfiles_register = spark.read.load(inputPath_register, format="csv" ,sep =␣
 ↪"\t", header = True, inferSchema=True)
```

[40]:
```
#Number of data before cleaning
dfProfiles_register.count()
```

[40]:
```
25319028
```

[41]:
```
#Considering the filter for "used_slots = 0" and "free_slots = 0"
filterProfile_register = dfProfiles_register.filter("used_slots = 0").
 ↪filter("free_slots = 0")
```

[42]:
```
#Clean the data by applying the filter using subtract method
dfProfiles_filtered = dfProfiles_register.subtract(filterProfile_register)
```

[43]:
```
#Number of data after cleaning
dfProfiles_filtered.count()
```

[43]:
```
25104121
```

[44]:
```
#Creat the dataframe & transforming days using timestamp function, add 2 colomn␣
 ↪for hour & day - Output : |station| timestamp|used_slots|free_slots|␣
 ↪day|hour|

df_register_filtered = dfProfiles_filtered.select('*', \
                                  date_format(filterProfile_register.timestamp,␣
 ↪'EEEE').alias("day"), hour(filterProfile_register.timestamp).alias("hour"))
```

[45]:
```
#Extract and count the zero values of free slots as critical situation for each␣
↪station
critical_station = df_register_filtered.filter(df_register_filtered.free_slots␣
↪== 0).groupBy("station", "hour", "day").count()
```

[46]:
```
#Rename the column for zero value counted in stations to "zero_value_count" -␣
↪Output : |station|hour| day|zero_value_count|
critical_station_headings = critical_station.withColumnRenamed("count",␣
↪"zero_value_count")
```

[47]:
```
#Counting the total records of each station at specific hour and rename the␣
↪column to"value_count"- Output :  |station|hour| day|value_count
total_records = df_register_filtered.groupBy("station", "hour", "day").count().
↪withColumnRenamed("count", "value_count")
```

[49]:
```
#Join critical stations and total records as first step op criticality␣
↪computation - Output : |station|hour| day|value_count|zero_value_count|
Stations_Criticals_join = total_records.join(critical_station_headings,␣
↪on=['station', 'hour', 'day'],how='left_outer')
```

[50]:
```
#Calculating the Criticality Ratio - Output : |station|hour|␣
↪day|value_count|zero_value_count|Criticality_Ratio|
result_df = (Stations_Criticals_join.withColumn("Criticality_Ratio",␣
↪round(col("zero_value_count") / col("value_count"), 3)))
#Replace Null values with 0.00 value as the Criticality Ratio
results_df_final = result_df.fillna(0.00, subset=["Criticality_Ratio"])
```

[51]:
```
#Obtaining the final number of critical stations by applying the filter of␣
↪threshold, and sorting by criticality, station, and day
final_critical_stations = results_df_final.filter(col("Criticality_Ratio") > 0.
↪6).orderBy(col("Criticality_Ratio").asc(),col("station").asc(),col("day").
↪asc())
```

[52]:
```
#show the result
final_critical_stations.show()
```

```
[Stage 34:=====================================================>(199 + 1) / 200]

+-------+----+------+-----------+----------------+-----------------+
|station|hour|   day|value_count|zero_value_count|Criticality_Ratio|
+-------+----+------+-----------+----------------+-----------------+
|      9|   8|Friday|        589|             361|            0.613|
|     10|  22|Friday|        389|             242|            0.622|
|     58|  23|Sunday|        359|             224|            0.624|
|      9|  20|Friday|        596|             373|            0.626|
|     58|  22|Sunday|        359|             227|            0.632|
+-------+----+------+-----------+----------------+-----------------+
```

[53]:
```
#Count the number of critical stations
final_critical_stations.count()
```

[53]:
```
5
```

[54]:
```
#Read the station File
dfProfiles_station = spark.read.load(inputPath_stations, format="csv" ,sep =␣
↪"\t", header = True, inferSchema=True)
```

[55]:
```
#Join station data, with results from register data - Output : | id|longitude|␣
↪latitude| name|station|hour|␣
↪day|value_count|zero_value_count|Criticality_Ratio|
df_join_register_station = dfProfiles_station.join(final_critical_stations,␣
↪col("station") == col("id"), "inner")
```

[56]:
```
#Extracting the required columns for the joind dataframes - Output :␣
↪|station|longitude| latitude|hour|   day|Criticality_Ratio|
final_result = df_join_register_station.select("station", "longitude",␣
↪"latitude" , "hour", "day", "Criticality_Ratio")
```

[57]:
```
#Show the final result
final_result.show()
```

```
+-------+---------+---------+----+------+-----------------+
|station|longitude| latitude|hour|   day|Criticality_Ratio|
+-------+---------+---------+----+------+-----------------+
|      9| 2.185294|41.385006|   8|Friday|            0.613|
|     10| 2.185206|41.384875|  22|Friday|            0.622|
|     58| 2.170736|41.377536|  23|Sunday|            0.624|
|      9| 2.185294|41.385006|  20|Friday|            0.626|
|     58| 2.170736|41.377536|  22|Sunday|            0.632|
+-------+---------+---------+----+------+-----------------+
```

2 BONUS TASK

Compute the distance between each station & the city center.

2.0.1 Questions

- Question 3.1. Are more used the stations closer to center or the ones farther? Report the values obtained of U1 and U2.

Answer: The code below calculate the distance of all stations to city center with the given coordinate, using Harversine formula, the results obtained, show that with a slight difference, stations close to citi center are more used according to the avarage slotes used per stations (required values are mentioned in code)

[58]:
```
#To calculate the haversine distance between two sets of latitude and longitude␣
↪we need to import some functions & creat a "def harvesine"
from pyspark.sql.functions import  lit, radians, asin, sin, cos, sqrt
from pyspark.sql import functions as F

def haversine(lat1, lon1, lat2, lon2):
    lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2]) # Convert␣
↪latitude and longitude from degrees to radians
    # Haversine formula
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    r = 6371.0             # Radius of Earth in kilometers
    distance_km = r * c    # Calculate the distance
    return round(distance_km,2)
```

[59]:
```
#Center Cordinates - output: | id|longitude| latitude|␣
↪name|distance_to_Center(km)|
center_lat = 41.386904
center_lon= 2.169989

# Add a new column in station dataframe with created haversine function
station_to_center = dfProfiles_station.withColumn("distance_to_Center(km)",␣
↪haversine(col("latitude"), col("longitude"), lit(center_lat),␣
↪lit(center_lon)))
```

[60]:
```
#find the avrage slot used from filtered register dataframe ('avg' and 'floor'␣
↪should be imported)
from pyspark.sql.functions import avg , floor , when
average_used_slots = df_register_filtered.groupBy("station").
↪agg(floor(avg("used_slots")).alias("avg_used_slots")) #use 'floor' to round␣
↪down the result
```

[61]:

```
#Join distance data, with avrage of used slots results from register data -␣
↪output:␣
↪|id|longitude|latitude|name|distance_to_Center(km)|station|avg_used_slots|
df_join_avg_distance = station_to_center.join(average_used_slots,␣
↪col("station") == col("id"), "left_outer")
```

[63]:
```
#As the number of id in station file is more than unique stations in register␣
↪file, some null values are appeared
df_join_avg_distance_filter = df_join_avg_distance.
↪filter(df_join_avg_distance['station'].isNotNull()) #select the rows without␣
↪null values
```

[64]:
```
#Specifying the station distance as a feature in new column
use_per_distance = df_join_avg_distance_filter.withColumn('distance_group',␣
↪when(col('distance_to_Center(km)') >= 1.5, 'd >= 1.5km').otherwise('d < 1.
↪5km'))
```

[65]:
```
# Group by the new column and calculate the average of 'avg_used_slots' for␣
↪each group
use_per_distance_df = use_per_distance.groupBy('distance_group').
↪agg(avg('avg_used_slots').alias('avg_used_slots'))
```

[66]:
```
# Show the resulting dataframe
use_per_distance_df.show()
```

```
+--------------+--------------+
|distance_group|avg_used_slots|
+--------------+--------------+
|      d < 1.5km|      7.671875|
|     d >= 1.5km|          7.35|
+--------------+--------------+
```