Lab 1: BDA1 - Spark - Exercises

Anubhav Dikshit(anudi287) and Sae Won Jun (saeju204)

2019-04-25

Contents

Question 1:

- What are the lowest and highest temperatures measured each year for the period 1950-2014. Provide the lists sorted in the descending order with respect to the maximum temperature. In this exercise you will use the `temperature-readings.csv` file.
- Non-parallelized program in Python to find the maximum temperatures for each year without using Spark. In this case you will run the program using: `python script.py`

Python Code (Non-parallelized):

```
import time, argparse

parser = argparse.ArgumentParser(description='MinMaxTempExtractor')
parser.add_argument('-o', '--output', help='Output file', default='out_temp.csv')
requiredNamed = parser.add_argument_group('Required arguments')
requiredNamed.add_argument('-t', '--time', help='Time interval in years ex. 2005:2010',
                           required=True)
requiredNamed.add_argument('-i', '--input', help='Input file', required=True)
args = parser.parse_args()
iFile = args.input
oFile = args.output
years =  args.time.split(":")
fromYear = int(years[0])
```

```
toYear = int(years[1])

start = time.time()
print('Running Python MinMaxTempExtractor:\nFrom %s To %s\nInput file: %s\nOutput file: %s'
      % (fromYear, toYear, iFile, oFile))
temp_dict = dict()
with open(iFile) as f:
    for l in f:
        line = l.split(";")
        year = int(line[1].split("-")[0])
        if year >= fromYear and year <= toYear:
            temp = temp_dict.get(year)
            station = line[0];
            curr_temp = float(line[3]);
            if not temp:
                #1st list for min temp and 2nd list for max temp
                temp_dict[year] = [[station, curr_temp], [station, curr_temp]]
            else:
                min = float(temp[0][1])
                max = float(temp[1][1])
                if curr_temp < min:
                    temp[0][0] = station
                    temp[0][1] = curr_temp
                if curr_temp > max:
                    temp[1][0] = station
                    temp[1][1] = curr_temp
#close the file after reading the lines.
f.close()

#sort temperatures descending by max temp
sorted_temp = temp_dict.items()
sorted_temp.sort(key=lambda x: x[1][1][1], reverse=True)

#write the output to file.
with open(oFile,'wb+') as f:
    for i in sorted_temp:
        #python will convert \n to os.linesep
        f.write('%s,%s,%s,%s,%s\n' % (i[0], i[1][0][0], i[1][0][1], i[1][1][0],
                                      i[1][1][1]))
#close the file after writting the lines.
f.close()
end = time.time()
print('Done in %s seconds' % (end - start))
```

Copying code from local

```
scp .\serial_code.py x_anudi@heffa.nsc.liu.se:/nfshome/x_anudi/
```

Running the python code

```
[x_anudi@heffa1 ~]$ python serial_code.py -t 1950:2014 -i  Data/temperature-readings.csv -o Data/min_ma
Running Python MinMaxTempExtractor:
From 1950 To 2014
Input file: Data/temperature-readings.csv
Output file: Data/min_max_report
Done in 302.203989983 seconds
```

Code (Spark):

```
from pyspark import SparkContext

iFile = 'Data/temperature-readings.csv'
oFile = 'Data/min_max_temperature'
fromYear = 1950
toYear = 2014

sc = SparkContext(appName="MinMaxTempExtractorSparkJob")

lines = sc.textFile(iFile)

lines = lines.map(lambda a: a.split(";"))

lines = lines.filter(lambda x: int(x[1][0:4]) >= 1950 and int(x[1][0:4]) <= 2014)

temperatures = lines.map(lambda x: (x[1][0:4], (x[0], float(x[3]))))

################## Custom MIN/MAX Function ############################

min = (lambda x, y: x if x[1] < y[1] else y)
max = (lambda x, y: x if x[1] > y[1] else y)

minTemperatures = temperatures.reduceByKey(min)

maxTemperatures = temperatures.reduceByKey(max)

minMaxTemp = minMaxTemp = minTemperatures.union(maxTemperatures). \
reduceByKey(lambda x,y: (x[0],x[1],y[0],y[1]))

sortedMinMaxTemp = minMaxTemp.sortBy(ascending=False, keyfunc=lambda a: a[1][3])

sortedMinMaxTempCsv = sortedMinMaxTemp.map(lambda a: '%s,%s,%s,%s,%s' \
% (a[0], a[1][0], a[1][1], a[1][2], a[1][3]))

sortedMinMaxTempCsv.coalesce(1).saveAsTextFile(oFile)
```

Running the spark code

```
./runYarn.sh spark_max_temp_prec_extractor.py
```

Final Output The lowest and highest temperatures:

```
year,station,min,station,max
1975,157860,-37.0,86200,36.1
1992,179960,-36.1,63600,35.4
1994,179960,-40.5,117160,34.7
2010,191910,-41.7,75250,34.4
2014,192840,-42.5,96560,34.4
1989,166870,-38.2,63050,33.9
1982,113410,-42.2,94050,33.8
1968,179950,-42.0,137100,33.7
1966,179950,-49.4,151640,33.5
1983,191900,-38.2,98210,33.3
```

- It's clearly appears that the Spark parallel version around 2 mins takes much less time than the serial one (302.203989983 seconds) as this the main benefits of multiprocessing environment.

Question 2:

- Count the number of readings for each month in the period of 1950-2014 which are higher than 10 degrees. Repeat the exercise, this time taking only distinct readings from each station. That is, if a station reported a reading above 10 degrees in some month, then it appears only once in the count for that month.

Code:

```
from pyspark import SparkContext

iFile = 'Data/temperature-readings.csv'
oFile = 'Data/over_ten_mth_temp_counts'
oFile2 = 'Data/over_ten_temp_distinct_counts'
fromYear = 1950
toYear = 2014
target_temp = 10

sc = SparkContext(appName="TempCounterSparkJob")

lines = sc.textFile(iFile)

lines = lines.map(lambda a: a.split(";"))

observations = lines.filter(lambda observation:
                            (int(observation[1][:4]) >= fromYear and
                             int(observation[1][:4]) <= toYear)) \
                    .cache()

"""
Q2a. Year-month, number
"""
temperatures = observations.map(lambda observation:
                                (observation[1][:7], (float(observation[3]), 1))) \
                           .filter(lambda (month, (temp, count)): temp > target_temp)

reading_counts = temperatures.reduceByKey(lambda (temp1, count1), (temp2, count2):
                                          (temp1, count1 + count2)) \
                             .map(lambda (month, (temp, count)):(month, count))

reading_counts.repartition(1).saveAsTextFile(oFile)

"""
Q2b. Year-month, distinct number
"""
station_temperatures = observations.map(lambda observation:
                                        (observation[1][:7],
                                         (observation[0], float(observation[3])))) \
                                   .filter(lambda (month, (station, temp)):
                                           temp > target_temp)

year_station = station_temperatures.map(lambda (month, (station, temp)):
```

```
                                            (month, (station, 1))).distinct()

reading_counts = year_station.reduceByKey(lambda (station1, count1), (station2, count2):
                                              (station1, count1 + count2)) \
                                  .map(lambda (month, (station, count)): (month, count))

reading_counts.repartition(1).saveAsTextFile(oFile2)
```

Running the code

```
# create a directory
hdfs dfs mkdir Data

# copy files to hdfs from local
hdfs dfs -copyFromLocal temperature-readings.csv Data/
hdfs dfs -copyFromLocal stations-Ostergotland.csv Data/


# check if the files are copied
hdfs dfs -ls /user/x_anudi/Data/

# run code
./runYarn.sh spark_temp_count_extractor.py

# copy output from hdfs to local
hdfs dfs -copyToLocal /user/x_anudi/Data/over_ten_mth_temp_counts
hdfs dfs -copyToLocal /user/x_anudi/Data/over_ten_temp_distinct_counts
```

Count of the readings above 10 degrees for each month:

```
print reading_counts.take(10)
[(u'1967-07', 53813),
(u'1974-07', 66277),
(u'2003-05', 48264),
(u'1978-03', 306),
(u'1981-10', 9882),
(u'1983-09', 38692),
(u'1987-05', 17191),
(u'1979-04', 1684),
(u'2009-07', 133008),
(u'1986-11', 1198)]
```

Distinct count of the readings above 10 degrees for each month:

```
print reading_counts.take(10)
[(u'1997-04', 190),
(u'1974-07', 362),
(u'2003-05', 321),
(u'1981-10', 325),
```

```
(u'1983-09', 332),
(u'1987-05', 320),
(u'1979-04', 227),
(u'2009-07', 312),
(u'1986-11', 138),
(u'1966-08', 359)]
```

Question 3:

- Find the average monthly temperature for each available station in Sweden. Your result should include average temperature for each station for each month in the period of 1960-2014. Bear in mind that not every station has the readings for each month in this timeframe.

Code:

```
from pyspark import SparkContext

iFile = 'Data/temperature-readings.csv'
oFile = 'Data/station_avg_mth_temp'
fromYear = 1960
toYear = 2014

sc = SparkContext(appName="AvgTempSparkJob")

lines = sc.textFile(iFile)

lines = lines.map(lambda a: a.split(";"))

observations = lines.filter(lambda observation:
                                    (int(observation[1][:4]) >= fromYear and
                                     int(observation[1][:4]) <= toYear))

stationDailyTemps = observations.map(lambda observation:
                                            ((observation[1], observation[0]),
                                             (float(observation[3]),
                                              float(observation[3]))))

stationDailyMinMaxTemps = stationDailyTemps.reduceByKey(lambda
                                                    (mintemp1, maxtemp1),
                                                    (mintemp2, maxtemp2):
                                                    (min(mintemp1, mintemp2),
                                                    max(maxtemp1, maxtemp2)))

stationMonthlyAvgTemps = stationDailyMinMaxTemps.map(lambda ((day, station),
                                                          (mintemp, maxtemp)):
                                                       ((day[:7], station),
                                                        (sum((mintemp, maxtemp)), 2))) \
                                                .reduceByKey(lambda (temp1, count1),
                                                             (temp2, count2):
                                                             (temp1 + temp2,
                                                              count1 + count2)) \
                                                .map(lambda ((month, station),
                                                             (temp, count)):
                                                             ((month, station),
                                                             temp / float(count)))

stationMonthlyAvgTemps.repartition(1).saveAsTextFile(oFile)
```

Running the code

```
# run the code
./runYarn.sh spark_temp_avg_extractor.py

# copying the result
hdfs dfs -copyToLocal /user/x_anudi/Data/station_avg_mth_temp
```

Station average monthly temperature:

```
print stationMonthlyAvgTemps.take(10)
[((u'2001-01', u'63280'), 0.0032258064516128633),
((u'1969-03', u'83620'), -4.780645161290321),
((u'1978-09', u'156730'), 5.5566666666666675),
((u'1991-11', u'106500'), -0.051666666666666645),
((u'1995-06', u'112080'), 12.001666666666669),
((u'1998-05', u'172770'), 5.282258064516128),
((u'2003-06', u'177930'), 8.711666666666668),
((u'1976-10', u'89240'), 6.138709677419355),
((u'2001-02', u'159770'), -14.35),
((u'1963-07', u'53560'), 17.738709677419358)]
```

Question 4:

- Provide a list of stations with their associated maximum measured temperatures and maximum measured daily precipitation. Show only those stations where the maximum temperature is between 25 and 30 degrees and maximum daily precipitation is between 100 mm and 200 mm.

Code:

```
from pyspark import SparkContext

iFile = 'Data/temperature-readings.csv'
iFile2 = 'Data/precipitation-readings.csv'
oFile = 'Data/max_temperature_precipitation'

sc = SparkContext(appName="MaxTempPrecExtractorSparkJob")

######### Max Temperatures #########
temperatures = sc.textFile(iFile)
temperatures = temperatures.map(lambda a: a.split(";"))
temperatures = temperatures.map(lambda x: (x[0], float(x[3])))
maxTemperatures = temperatures.reduceByKey(max)
maxTemperatures = maxTemperatures.filter(lambda a: a[1] > 25 and a[1] < 30)

######### Max Precipitations #########
precipitations = sc.textFile(iFile2)
precipitations = precipitations.map(lambda a: a.split(";"))
precipitations = precipitations.map(lambda x: (x[0]+','+x[1], float(x[3])))
```

```
maxPrecipitations = precipitations.reduceByKey(lambda v1, v2: v1+v2)
maxPrecipitations = maxPrecipitations.filter(lambda a: a[1] >= 100 and a[1] <= 200) \
.map(lambda x: (x[0].split(",")[0], x[1])).reduceByKey(max)

######### Merged Max Temperatures/Precipitations #########
maxTempPrec = maxTemperatures.leftOuterJoin(maxPrecipitations)

maxTempPrecCsv = maxTempPrec.map(lambda a: '%s,%s,%s' % (a[0], a[1][0], a[1][1]))

maxTempPrec.coalesce(1).saveAsTextFile(oFile)
```

Running the code

```
# running the code
./runYarn.sh spark_max_temp_prec_extractor.py

# copying the file to local
hdfs dfs -copyToLocal /user/x_anudi/Data/max_temperature_precipitation
```

stations maximum measured daily temperatures/precipitation:

```
station,maxTemp,maxPrec
128510,29.5,None
192830,29.5,None
84660,27.6,None
139110,29.0,None
161670,25.7,None
166940,27.9,None
77180,29.3,None
180740,29.0,None
72340,29.8,None
147560,29.9,None
```

Question 5:

- Calculate the average monthly precipitation for the Ostergotland region (list of stations is provided in the separate file). In order to do this, you will first need to calculate the total monthly precipitation for each station before calculating the monthly average (by averaging over stations).

Code:

```
import pyspark
path = '/user/x_anudi/data/'
sc = pyspark.SparkContext(appName="assignment05_anudi")

# Loading the file.
preci_data = sc.textFile(path + "precipitation-readings.csv")
```

```
stations = sc.textFile(path + "stations-Ostergotland.csv")
stations = stations.map(lambda x: x.split(";"))
stations = stations.map(lambda line: line[0])
stations = stations.coalesce(1)
stations = stations.collect()
stations = sc.broadcast(stations)

preci = preci_data.map(lambda x: x.split(";"))
preci = preci.map(lambda x: ((x[0], x[1][0:4], x[1][5:7]), (float(x[3]))))
preci = preci.filter(lambda x: x[0][0] in stations.value)

preci_2016 = preci.filter(lambda x: int(x[0][1]) >= 1993 and int(x[0][1]) <= 2016)
preci_2016 = preci_2016.reduceByKey(lambda x, y: x + y)
preci_2016 = preci_2016.map(lambda line: ((line[0][1], line[0][2]), (line[1], 1)))
preci_2016 = preci_2016.reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))
preci_2016 = preci_2016.map(lambda line: (line[0][0], line[0][1], (line[1][0] / line[1][1])))
preci_2016 = preci_2016.coalesce(1)
preci_2016.saveAsTextFile("res/assignment05_anudi")
```

Running the code

```
# running the code
./runYarn.sh spark_ostergot_avg_prec_extractor.py

# copying data from hdfs
hdfs dfs -copyToLocal /user/x_anudi/Data/OstergotlandAveMonthlyPrec
```

Average monthly precipitation:

```
year,month,avgPrec
(u'2012', u'09', 72.75)
(u'1995', u'05', 26.00000000000002)
(u'2015', u'04', 15.337499999999997)
(u'2007', u'04', 21.249999999999996)
(u'2007', u'06', 108.95)
(u'2011', u'06', 88.35000000000001)
(u'2011', u'10', 43.75000000000001)
(u'2014', u'10', 72.13749999999999)
(u'1996', u'09', 57.466666666666676)
(u'1995', u'07', 43.6)
(u'2002', u'05', 72.13333333333334)
```

Question 6:

- Compare the average monthly temperature (find the difference) in the period 1950-2014 for all stations in Ostergotland with long-term monthly averages in the period of 1950-1980.

Code:

```
from pyspark import SparkContext

iFile = 'Data/stations-Ostergotland.csv'
iFile2 = 'Data/temperature-readings.csv'
oFile = 'Data/OstergotlandAvgMonthlyDiffTemp'
fromYear = 1960
toYear = 2014

sc = SparkContext(appName="OstergotlandAvgMonthlyTempDiffSparkJob")

ostergotlandStations = sc.textFile(iFile)

ostergotlandStations = ostergotlandStations.map(lambda line: line.split(";")). \
                                            map(lambda x: int(x[0])).distinct().collect()

isOstergotlandStation = (lambda s: s in ostergotlandStations)

temperatures = sc.textFile(iFile2).map(lambda line: line.split(";")). \
                                   filter(lambda x: isOstergotlandStation(int(x[0]))
                                          and int(x[1][0:4]) >= fromYear and int(x[1][0:4]) <= toYear)

monthlyAvgTemps = temperatures.map(lambda obs:
                                           ((obs[1], int(obs[0])),
                                                  (float(obs[3]), float(obs[3])))) \
                                           .reduceByKey(lambda (mint1, maxt1),
                                                               (mint2, maxt2):
                                                               (min(mint1, mint2),
                                                                max(maxt1, maxt2))) \
                                           .map(lambda ((day, station), (mint, maxt)):
                                                (day[:7], (mint + maxt, 2))) \
                                           .reduceByKey(lambda (temp1, count1),
                                                               (temp2, count2):
                                                               (temp1 + temp2,
                                                                count1 + count2)) \
                                           .map(lambda (month, (temp, count)):
                                                       (month, temp / float(count)))

monthlyLongtermAvgTemps = monthlyAvgTemps.filter(lambda (month, temp):
                                                        int(month[:4]) <= 1980) \
                                         .map(lambda (month, temp):
                                              (month[-2:], (temp, 1))) \
                                         .reduceByKey(lambda (temp1, count1),
                                                             (temp2, count2):
                                                             (temp1 + temp2,
                                                              count1 + count2)) \
```

```
                                                .map(lambda (month, (temp, count)):
                                                            (month, temp / float(count)))

month_temp = {month: temp for month, temp in monthlyLongtermAvgTemps.collect()}

monthlyAvgTemps = monthlyAvgTemps.map(lambda (month, temp):
                                            (month, (temp) - (month_temp[month[-2:]]))) \
                                    .sortBy(ascending=True, keyfunc=lambda (month, temp): month)

monthlyAvgTempsCsv = monthlyAvgTemps.map(lambda a: '%s,%s,%s' %
                                            (a[0].split('-')[0], a[0].split('-')[1], a[1]))

monthlyAvgTempsCsv.repartition(1).saveAsTextFile(oFile)
```

Running the code

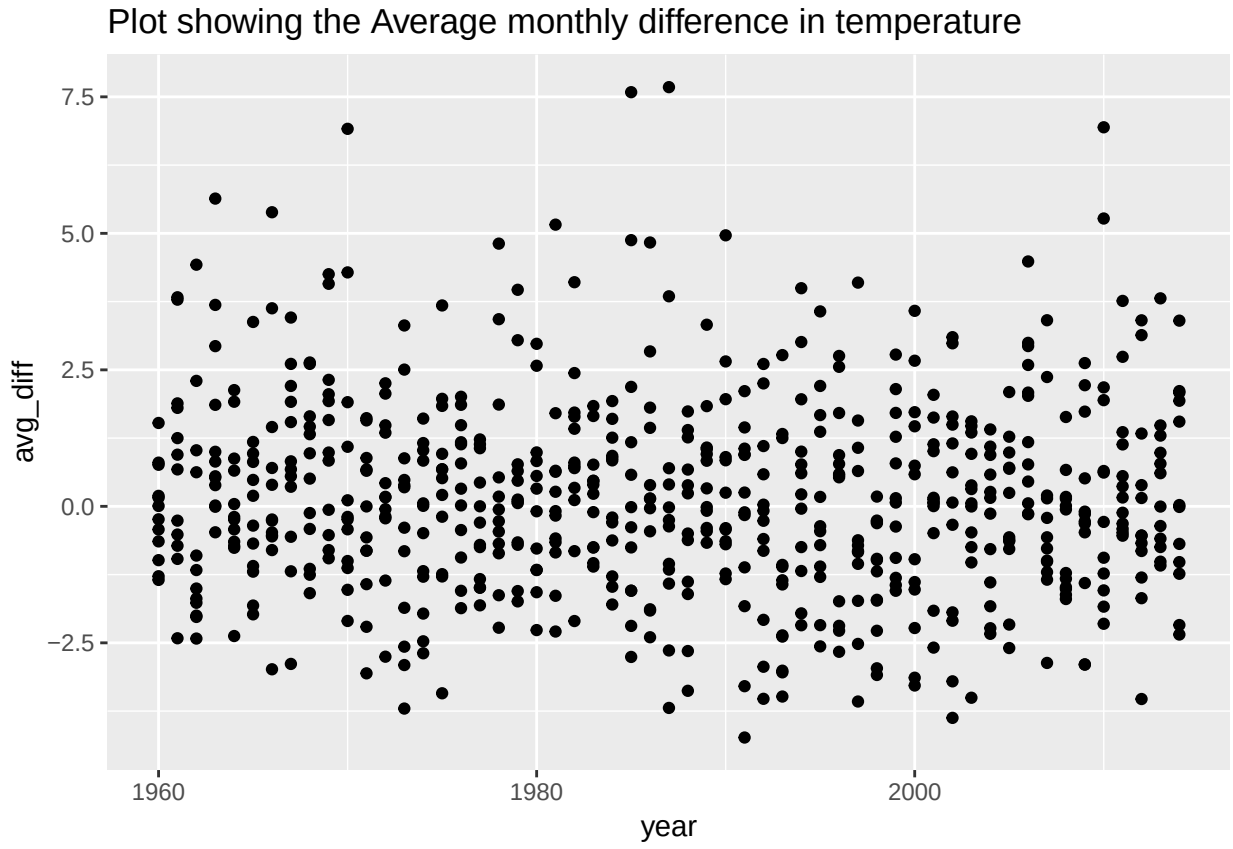
```
# running the code
./runYarn.sh spark_ostergot_avg_temp_diff_extractor.py

# copying results
hdfs dfs -copyToLocal /user/x_anudi/Data/OstergotlandAvgMonthlyDiffTemp
```

Average monthly temperature compared with long-term monthly averages:

```
print monthlyAvgTempsCsv.take(10)
[u'1960,01,0.761463766648',
u'1960,02,1.5272002457',
u'1960,03,0.192050448272',
u'1960,04,-0.642024384346',
u'1960,05,0.157772987381',
u'1960,06,-0.237311108204',
u'1960,07,-1.28047382725',
u'1960,08,-0.985763468482',
u'1960,09,0.00674101691958',
u'1960,10,-1.34710218737']
```

plot:


Plot showing the Average monthly difference in temperature
avg_diff
7.5
5.0
2.5
0.0
−2.5
1960
1980
2000
year