Baysian Learning
732A54

Lab 1 BDA

Guilherme Barros
guiba484

Eric Herwin
erihe068

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.

Code:

```
from pyspark import SparkContext

sc = SparkContext()

## Sorting
data = sc.textFile("/user/x_erher/data/temperature-readings.csv")
rows_temp = data_temp.map(lambda line: line.split(";"))
key_val = rows_temp.map(lambda x: (x[1][:4], float(x[3])))
key_val = key_val.reduceByKey(lambda x,y: x if x>=y else y).sortBy(lambda x: x[1])

print(key_val.take(5))
```

a)

Extend the program to include the station number (not the station name) where the maximum/minimum temperature was measured.

Code:

```
from pyspark import SparkContext

sc = SparkContext()

data = sc.textFile("/user/x_erher/data/temperature-readings.csv")
rows = data.map(lambda line: line.split(";"))

def max_temperature(x,y):
    if x[1]>=y[1]:
        return x
    else:
        return y

year_and_temp = rows.map(lambda x: (x[1][0:4], (x[0], float(x[3]))) )
max_temperatures = year_and_temp.reduceByKey(lambda x,y: max_temperature(x,y))
print(max_temperatures.take(5))
```

b)

Write the non-parallelized program in Python to find the maximum temperatures for each year without using Spark.

Code:

Non parallelized:

```
import csv
import time
```

```
start = time.time()
data = []
with open('sample-temp.csv') as csvDataFile:
    csvReader = csv.reader(csvDataFile, delimiter=';')
    for row in csvReader:
        data.append([row[1][0:4], float(row[3])])

def getKey(item):
    return item[1]

sort_data = sorted(data, key = getKey) #making a dict will take the last value, therefore sort

dicty = dict(sort_data)
end = time.time()

print(dicty)
print('--')
print(end - start) #output: 289.079863071
```

From the measure time we got that it took around 290 seconds to perform the code (on the small data).

Parallelized:

```
from pyspark import SparkContext

sc = SparkContext()

data = sc.textFile("/user/x_erher/data/temperatures-big.csv")
rows = data.map(lambda line: line.split(";"))
vals = rows.map(lambda x: (x[1][0:4], float(x[3])))
red_uce = vals.reduceByKey(lambda x,y: x if x >= y else y)


print(red_uce.take(5))
```

From the history log, we got that it took around 2.7 minutes for the code ro run on the temperatures-big.csv.

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:

Readings monthly

```
from pyspark import SparkContext

sc = SparkContext()

data = sc.textFile("/user/x_erher/data/temperature-readings.csv")
rows = data.map(lambda line: line.split(";"))
test = rows.filter(lambda x: float(x[1][0:4]) > 1950 and float(x[1][0:4]) < 2014)
```

```
test = test.filter(lambda temp: float(temp[3]) > 10 )

year_and_temp = test.map(lambda x: ((x[1][0:4],x[1][5:7]), 1) )
count = year_and_temp.reduceByKey(lambda x,y: x+y)

print(count.take(10))
```

Readings by station

```
from pyspark import SparkContext

sc = SparkContext()

data = sc.textFile("/user/x_erher/data/temperature-readings.csv")
rows = data.map(lambda line: line.split(";"))
test = rows.filter(lambda x: float(x[1][0:4]) > 1950 and float(x[1][0:4]) < 2014)
test = test.filter(lambda temp: float(temp[3]) > 10 )

year_and_temp = test.map(lambda x: ((x[0], x[1][0:4], x[1][5:7]),1) ).distinct()
output = year_and_temp.map(lambda x: (x[0][1], x[0][2], x[1]))

print(output.take(10))
```

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. In this exercise you will use the temperature-readings.csv file.

Code:

```
from pyspark import SparkContext

sc = SparkContext()

data = sc.textFile("/user/x_erher/data/temperature-readings.csv")
rows = data.map(lambda line: line.split(";"))
year_and_month = rows.map(lambda x: ((x[0],x[1][0:7]), float(x[3])) )

aTuple = (0,0)
rdd1 = year_and_month.aggregateByKey(aTuple, lambda a,b: (a[0] + b, a[1] + 1),
                            lambda a,b: (a[0] + b[0], a[1] + b[1]))
finalResult = rdd1.mapValues(lambda v: v[0]/v[1])
print(finalResult.take(5))
```

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

sc = SparkContext()

data = sc.textFile("/user/x_erher/data/temperature-readings.csv")
data_perce = sc.textFile("/user/x_erher/data/precipitation-readings.csv")

#temperature
rows_4 = data.map(lambda line: line.split(";"))
maxim = rows_4.map(lambda x: (x[0], float(x[3])))
maxim = maxim.reduceByKey(lambda x,y: x if x >= y else y)
test_4 = maxim.filter(lambda temp: float(temp[1]) > 25 and float(temp[1]) < 30)

#percept
rows_4_percepr = data_perce.map(lambda line: line.split(";"))
filtered = rows_4_percepr.map(lambda x: ((x[1],x[0]), float(x[3])) )
daily_precipitation = filtered.reduceByKey(lambda x,y: x+y )
sum_4_percepr = daily_precipitation.filter(lambda temp: float(temp[1]) > 100 and
float(temp[1]) < 200)
sum_4_percepr_final = sum_4_percepr.map(lambda x: (x[0][1], float(x[1]))) #extract all the stations

data_final = sum_4_percepr_final.join(test_4) # the join

print(test_4.take(5))
print("--")
print(sum_4_percepr_final.take(5))
print("--")
print(data_final.take(5))
```

5)

Calculate the average monthly precipitation for the Östergotland region (list of stations is provided in the separate file) for the period 1993-2016. 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:

```
stations_oster = sc.textFile("data/stations-Ostergotland.csv")
precip = sc.textFile("data/precipitation-readings.csv")

stations_oster_rows = stations_oster.map(lambda line: line.split(";"))
precip_rows = precip.map(lambda line: line.split(";"))
stations_oster_list = sc.broadcast(stations_oster_rows.map(lambda x: (x[0] )).collect())
precip_rows_oster = precip_rows.filter(lambda x: x[0] in stations_oster_list.value)

precip_rows_oster_clean = precip_rows_oster.map(lambda x: (x[1][0:7], float(x[3])) )
aTuple = (0,0)

rdd2 = precip_rows_oster_clean.aggregateByKey(aTuple, lambda a,b: (a[0] + b, a[1] + 1),
                            lambda a,b: (a[0] + b[0], a[1] + b[1]))
finalResult2 = rdd2.mapValues(lambda v: v[0]/v[1])#
print(finalResult2.take(5))
```

6)

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

Code:

```
from pyspark import SparkContext

sc = SparkContext()

data_temp = sc.textFile("/user/x_erher/data/temperature-readings.csv")
data_ost = sc.textFile("/user/x_erher/data/stations-Ostergotland.csv")

#temperature
rows_temp = data_temp.map(lambda line: line.split(";"))
data_extract = rows_temp.map(lambda x: ((x[0], x[1][:4], x[1][5:7]), float(x[3])))

#stations ost
rows_stations = data_ost.map(lambda line: line.split(";"))
stations = rows_stations.map(lambda x: ((x[0], x[5][:4], x[5][5:7]),1))

#filter for year
data_extract = data_extract.filter(lambda x: float(x[0][1]) > 1950 and float(x[0][1]) < 1980)

#join to only get ostergotland stations
joined = data_extract.leftOuterJoin(stations)
joined = joined.mapValues(lambda v: v[0])

#averaging - for month
rdd_month = joined.aggregateByKey((0,0), lambda a,b: (a[0] + b, a[1] + 1),
                  lambda a,b: (a[0] + b[0], a[1] + b[1]))
finRes_month = rdd_month.mapValues(lambda v: v[0]/v[1])

# Make month into a key keeping year and temperature as a tuple: (MM, (YY, Temp))
finRes_month = finRes_month.map(lambda x: (x[0][2],(x[0][1],x[1])))

# Make another RDD with just month: (MM, Temp)
finRes_month2 = finRes_month.map(lambda x: (x[0],x[1][1]))

# Take average of the second RDD with MM as a key
finRes_month2_average = finRes_month2.aggregateByKey((0,0), lambda a,b: (a[0] + b, a[1] + 1),
                                    lambda a,b: (a[0] + b[0], a[1] + b[1]))
long_average = finRes_month2_average.mapValues(lambda v: v[0]/v[1])

# Join long term average and year average using month as a key
final = finRes_month.join(long_average)

# Reorganize rdd and take difference
final = final.map(lambda x: ((x[1][0][0],x[0]),x[1][0][1]-x[1][1]))
```

```
print(final.take(10))
```