Projects of using Spark

Yanjian Zhang
School of Data Science
Fudan University
16300200020@fudan.edu.cn

Abstract

Mernis database would have over 120 million records whereas the leaked one only have about 48 million. This leak contains the following information for 49,611,709 Turkish citizens. This project is to use spark to make a statistical analysis and models for this database.

1 Data Preporcessing

Use the linux command to delete the first 76 lines and redirect to the data_new.txt file. Then delete the last 49 lines and redirect to the data_new.txt.

```
tail -n +76 data_dump.sql > data_tmp.txt
head -49611709 data_tmp.txt > data_new.txt
```

2 Environment & Context Setting

```
from __future__ import print_function,division
from pyspark import SparkConf,SparkContext
from pyspark.sql import SparkSession
from itertools import islice
import re
conf = SparkConf().setAppName("turkey").setMaster("local")
sc = SparkContext(conf=conf)
```

3 Easy Section

3.1 Find the eldest man

The steps are as follows:

- find valid year data, remove year exception like 95
- filter with male condition
- sorted the list and find the eldest one

The codes are as follows:

```
data = sc.textFile('hdfs://localhost:9001/user/data_new.txt').map(lambda x :
    re.split("\t", x))
```

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

```
validYearData = data.filter(lambda x : len(x[8].split("/")[-1])==4)
male = validYearData.filter(lambda x : x[6] == 'E')
year = male.map(lambda x : (" ".join([x[2], x[3]]),int(x[8][-4:])))
name, birthyear = sorted(year.collect(), key = lambda x: x[1])[0]
print("name",name)
print("age",2019 - int(birthyear))
```

results are as follows:

```
name CELIL UNAL
age 690
```

3.2 Find the character of max frequency in names

The steps are as follows:

- concatenate the first name and the second name
- transform string into list, flatten the word list
- sum up the frequency, and find the max one

The codes are as follows:

```
data = sc.textFile('hdfs://localhost:9001/user/data_new.txt').map(lambda x :
    re.split("\t", x))
name = data.map(lambda x : "".join([x[2],x[3]]))
count = name.flatMap(lambda x: list(x)).map(lambda x: (x, 1)).reduceByKey(lambda x,
    y: x + y)
letter,freq = sorted(count.collect(), key = lambda x: x[1])[-1]
print("letter",letter)
print("frequency",freq)
```

results are as follows:

```
letter A
frequency 82319942
```

3.3 The distribution of the age

The steps are as follows:

- find valid year data, remove year exception(such as 95)
- define the age level function and use age to get corresponding age level
- count their frequency

The codes are as follows:

```
def ageLevel(age):
   if age <=18:
      return '0-18'
   if age <= 28:
      return '19-28'
   if age <= 38:
      return '29-38'
   if age <= 48:
      return '39-48'
   if age <= 59:
      return '49-59'
   return '60+'
```

```
data = sc.textFile('hdfs://localhost:9001/user/data_new.txt').map(lambda x :
    re.split("\t", x))
validYearData = data.filter(lambda x : len(x[8].split("/")[-1])==4) # prevent the
    case like \95 instead of \1995
year = validYearData.map(lambda x : (x[8][-4:]))
age = year.map(lambda x : 2019 - int(x))
level = age.map(lambda x: (ageLevel(x),1))
levelcount = level.reduceByKey(lambda x, y: x+y)
print(levelcount.collect())
```

results are as follows:

```
[('29-38', 12593171), ('19-28', 349145), ('60+', 14434140), ('39-48', 11869188),
    ('49-59', 10366063)]
```

3.4 Birthmonth proportion

The steps are as follows:

- find valid month data, remove year exception(such as 13)
- count the frequency of each month
- count the total number of valid month, and calculate the proportion

The codes are as follows:

```
data = sc.textFile('hdfs://localhost:9001/user/data_new.txt').map(lambda x :
    re.split("\t", x))
month = data.map(lambda x : x[8].split("/")[1])
validMonth = month.filter(lambda x: int(x) in range(1,13) if x else False)
# print(month.collect()[0])

monthCount = validMonth.map(lambda x: (x,1)).reduceByKey(lambda x, y: x+y).persist()
totalCount = monthCount.map(lambda x: x[1]).reduce(lambda x,y: x+y)
monthPortion = monthCount.mapValues(lambda x: str('%.4f%%' %
    (x/float(totalCount)*100)))
print(monthPortion.collect())
```

results are as follows:

```
[('12', '5.7093%'), ('11', '5.8771%'), ('8', '6.3989%'), ('3', '10.3431%'), ('1',
    '15.7922%'), ('7', '8.5755%'), ('9', '6.9641%'), ('10', '6.9313%'), ('6',
    '6.9897%'), ('2', '9.4632%'), ('4', '8.4442%'), ('5', '8.5113%')]
```

3.5 Sex population and proportion

The steps are as follows:

- count the frequency of each sex
- count the total number of two sex, and calculate the proportion

The codes are as follows:

```
data = sc.textFile('hdfs://localhost:9001/user/data_new.txt').map(lambda x :
    re.split("\t", x))
sex = data.map(lambda x : (x[6])).map(lambda x: (x,1))
sexCount = sex.reduceByKey(lambda x, y: x+y)
totalCount = sexCount.map(lambda x: x[1]).reduce(lambda x,y: x+y)
sexPortion = sexCount.mapValues(lambda x: str('%.4f%%' % (x/float(totalCount)*100)))
print("sex population",sexCount.collect())
```

```
print("sex proportion",sexPortion.collect())
```

results are as follows:

```
sex population [('K', 25077226), ('E', 24534483)]
sex proportion [('K', '50.5470%'), ('E', '49.4530%')]
```

4 Normal Section

4.1 Top 10 surname for both sex

The steps are as follows:

- get pairs of sex and surname
- use filter to get pairs of each sex
- count the number of surname in each sex

The codes are as follows:

```
data = sc.textFile('hdfs://localhost:9001/user/data_new.txt').map(lambda x :
    re.split("\t", x))
surnamepair = data.map(lambda x :(x[6],x[3]))
male = surnamepair.filter(lambda x : x[0] == "E")
female = surnamepair.filter(lambda x: x[0] == "K")
maleCount = male.map(lambda x: (x[1], 1)).reduceByKey(lambda x, y: x + y)
femaleCount = female.map(lambda x: (x[1], 1)).reduceByKey(lambda x, y: x + y)
print("male top 10",maleCount.top(10, key = lambda x:x[1]))
print("female top 10",femaleCount.top(10, key = lambda x:x[1]))
```

results are as follows:

```
male top 10 [('YILMAZ', 352338), ('KAYA', 244272), ('DEMIR', 231289), ('SAHIN',
    201958), ('CELIK', 199622), ('YILDIZ', 195162), ('YILDIRIM', 191966),
    ('OZTURK', 178610), ('AYDIN', 177894), ('OZDEMIR', 164085)]
female top 10 [('YILMAZ', 355954), ('KAYA', 244100), ('DEMIR', 230428), ('SAHIN',
    202155), ('CELIK', 199330), ('YILDIZ', 194060), ('YILDIRIM', 192835),
    ('OZTURK', 180292), ('AYDIN', 178501), ('OZDEMIR', 165924)]
```

4.2 Cities' average age

The steps are as follows:

- get the data with valid year
- get the pairs of cities and age
- assemble into groups based on cities
- calculate the average age within cities

The codes are as follows:

```
import numpy as np
data = sc.textFile('hdfs://localhost:9001/user/data_new.txt').map(lambda x :
    re.split("\t", x))
validYearData = data.filter(lambda x : len(x[8].split("/")[-1])==4)
citypair = validYearData.map(lambda x : (x[11], 2019-int(x[8][-4:])))
cityAgeMean = citypair.groupByKey().map(lambda x: (x[0],
    int(np.mean(list(x[1]))))).persist()
print("cities' average age",cityAgeMean.collect())
```

results are as follows:

```
cities' average age [('HAKKARI', 45), ('IZMIR', 52), ('AKSARAY', 50), ('BOLU', 54),
    ('ADANA', 50), ('ADIYAMAN', 49), ('KAHRAMANMARAS', 50), ('KARAMAN', 52),
    ('SIRNAK', 45), ('CANAKKALE', 55), ('MERSIN', 51), ('GAZIANTEP', 48), ('MUS',
    47), ('SANLIURFA', 46), ('KILIS', 51), ('BALIKESIR', 55), ('NEVSEHIR', 53),
    ('VAN', 46), ('MUGLA', 53), ('KIRSEHIR', 53), ('ISPARTA', 53), ('ISTANBUL',
    50), ('SAKARYA', 52), ('AYDIN', 54), ('TOKAT', 53), ('GUMUSHANE', 53),
    ('KIRKLARELI', 55), ('KONYA', 51), ('ERZURUM', 50), ('SAMSUN', 52), ('ELAZIG',
    51), ('CORUM', 54), ('BARTIN', 54), ('KARABUK', 54), ('ANKARA', 51),
    ('BURDUR', 55), ('SIIRT', 47), ('ARDAHAN', 52), ('HATAY', 50), ('DIYARBAKIR',
    47), ('KUTAHYA', 53), ('MALATYA', 51), ('YOZGAT', 52), ('MANISA', 53),
    ('BATMAN', 46), ('YALOVA', 53), ('BILECIK', 54), ('MARDIN', 47), ('IGDIR',
    49), ('TEKIRDAG', 51), ('KIRIKKALE', 52), ('EDIRNE', 55), ('SIVAS', 52),
    ('BAYBURT', 52), ('CANKIRI', 55), ('KAYSERI', 51), ('ORDU', 54), ('ZONGULDAK',
    52), ('BINGOL', 48), ('KOCAELI', 50), ('RIZE', 53), ('OSMANIYE', 50),
    ('DUZCE', 52), ('AMASYA', 54), ('TRABZON', 53), ('AFYONKARAHISAR', 52),
    ('GIRESUN', 55), ('TUNCELI', 53), ('ERZINCAN', 53), ('KASTAMONU', 56),
    ('ARTVIN', 55), ('DENIZLI', 52), ('ANTALYA', 51), ('AGRI', 47), ('BURSA', 51),
    ('NIGDE', 51), ('SINOP', 57), ('ESKISEHIR', 53), ('BITLIS', 47), ('KARS', 50),
    ('USAK', 53)]
```

4.3 5 youngest cities

The steps are as follows:

- use the result in previous question
- sort the result and get top 5

The codes are as follows:

```
print("youngest age cities", sorted(cityAgeMean.collect(),key = lambda x :
    x[1])[:5])
cityAgeMean.unpersist()
```

results are as follows:

```
youngest age cities [('HAKKARI', 45), ('SIRNAK', 45), ('SANLIURFA', 46), ('VAN',
    46), ('BATMAN', 46)]
```

4.4 Top 3 surname in top 10 cities

The steps are as follows:

- construct pairs of cities and surnames
- assemble into groups based on cities, sort by the number of item and get the top 10 cities
- use counter to count the surnames frequency, can turn into tuplelist
- use the nlargest within heapq to find the top 3 surnames with highest frequency

The codes are as follows:

```
from heapq import nlargest
from collections import Counter
def counterTupleList(l_:list):
   dic = Counter(l_)
   keys=dic.keys()
   vals=dic.values()
   tupleList=[(key,val) for key,val in zip(keys,vals)]
   return tupleList
```

```
data = sc.textFile('hdfs://localhost:9001/user/data_new.txt').map(lambda x :
    re.split("\t", x))
cityNamePair = data.map(lambda x :(x[7], x[3]))
top10cityGroup = cityNamePair.groupByKey() \
                   .sortBy(lambda x : len(x[1]),ascending=False) \
                   .zipWithIndex() \
                   .filter(lambda x: x[1] < 10) \
                   .map(lambda x : x[0])
top3surname = top10cityGroup.mapValues(lambda x : counterTupleList(x)) \
                   .mapValues(lambda x : nlargest(3, x, key = lambda t : t[1])) \
                   .mapValues(lambda x : [t[0] for t in x])
print("top 3 surname in top 10 cities", top3surname.collect())
```

results are as follows:

```
top 3 surname in top 10 cities [('ISTANBUL', ['YILMAZ', 'KAYA', 'OZTURK']),
    ('ANKARA', ['YILMAZ', 'SAHIN', 'OZTURK']), ('IZMIR', ['YILMAZ', 'KAYA',
    'OZTURK']), ('ADANA', ['YILMAZ', 'KAYA', 'CELIK']), ('GAZIANTEP', ['YILMAZ',
    'KAYA', 'DEMIR']), ('BURSA', ['YILMAZ', 'AYDIN', 'YILDIZ']), ('KAHRAMANMARAS',
    ['DEMIR', 'YILMAZ', 'KOSE']), ('SANLIURFA', ['DEMIR', 'CIFTCI', 'KAYA']),
    ('KONYA', ['YILMAZ', 'CELIK', 'KAYA']), ('MALATYA', ['YILMAZ', 'DOGAN',
    'KAYA'])]
```

4.5 Top 2 month in top 10 cities

The steps are as follows:

- construct pairs of cities and months
- assemble into groups based on cities, sort by the number of item and get the top 10 cities
- use counter to count the months frequency, can turn into tuplelist
- use the nlargest within heapq to find the top 2 months with highest frequency

The codes are as follows:

```
from heapq import nlargest
from collections import Counter
def counterTupleList(l_:list):
   dic = Counter(l_)
   keys=dic.keys()
   vals=dic.values()
   tupleList=[(key,val) for key,val in zip(keys,vals)]
   return tupleList
data = sc.textFile('hdfs://localhost:9001/user/data_new.txt').map(lambda x :
    re.split("\t", x))
cityMonthPair = data.map(lambda x :(x[7], x[8].split("/")[1]))
top10cityGroup = cityMonthPair.groupByKey() \
                   .sortBy(lambda x : len(x[1]),ascending=False) \
                   .zipWithIndex() \
                   .filter(lambda x: x[1] < 10) \
                   .map(lambda x : x[0])
top2month= top10cityGroup.mapValues(lambda x : counterTupleList(x)) \
                   .mapValues(lambda x : nlargest(2, x, key = lambda t : t[1])) \
                   .mapValues(lambda x : [t[0] for t in x])
print("top 2 month in top 10 cities", top2month.collect())
```

results are as follows:

```
top 2 month in top 10 cities [('ISTANBUL', ['1', '7']), ('ANKARA', ['1', '3']),
    ('IZMIR', ['1', '3']), ('ADANA', ['1', '3']), ('GAZIANTEP', ['1', '3']),
```

```
('BURSA', ['1', '3']), ('KAHRAMANMARAS', ['1', '2']), ('SANLIURFA', ['1',
'2']), ('KONYA', ['1', '3']), ('MALATYA', ['1', '3'])]
```