Report for Project 1 of Using Spark

Zhan Wang
Department of Data Science
Fudan University
Shanghai, China 200433
17307110421@fudan.edu.cn

Abstract

This project completes some analysis tasks (E1-E6, N1-N4) based on the leaked mernis database. For each task, we introduce the detailed solutions and implement the corresponding codes. The analysis results are presented by directly printing and visually graphing. The whole project is done on Spark platform in RDD format.

1 Environment

This project is done on Ubuntu 16.04, JDK 18.0, python 3.6.2 and Spark 3.1.0.

2 Load & Clean Data

The original dataset contains the information for about 48 million Turkish citizens with other introductory text. Therefore, after loading the dataset as text format, we cleaned the data through filtering necessary information and storing the complete instances (core codes shown below). The final cleaned dataset contains 49,611,709 records.

```
# load and clean the data
data = sc.textFile(data_path)
data = data.map(lambda x: x.split('\t'))
data = data.filter(lambda x: len(x) == 17)
print("The Number of Data: ", data.count())
```

3 Tasks

3.1 E1: The oldest Male

First, we filter out all the information about male (Gender: 'E') in the cleaned dataset. Then, sort the filtered data ascendingly by their year of birth. The first record is the oldest man in all Turkish citizens. Thus we print the whole name (the first name and the last name) of the first people.

```
answer_E1 = data.filter(lambda x: x[6] == "E")\
            .sortBy(lambda x: int((x[8].split('/'))[2]))
print("\nAnswer for E1: ", \
        answer_E1.first()[2]+' '+answer_E1.first()[3])
```

Output of above codes is

```
Answer for E1:  CELIL UNAL
```

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

3.2 E2: The Most Common Letter in Names

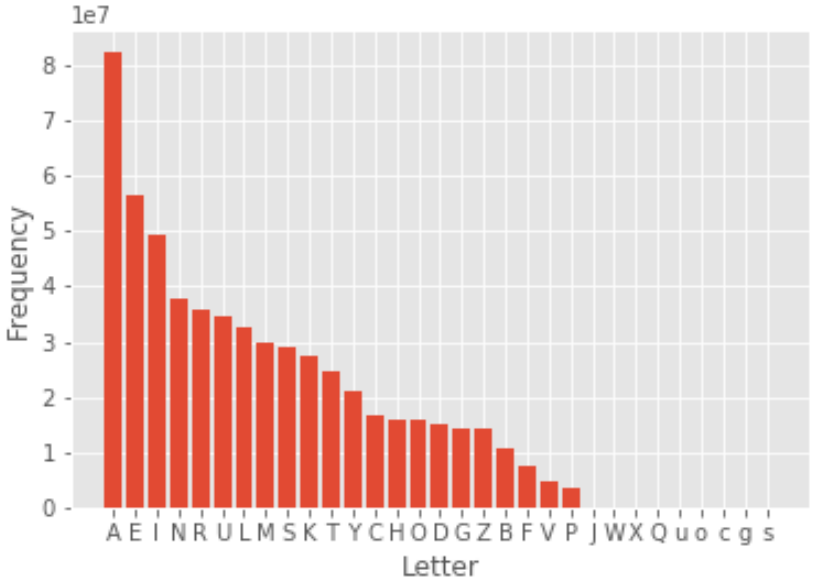
For this task, we firstly select all the names (containing first name and last name) in the dataset. Then split each name into letters and count the number of appearances for every existed letter. Since there are some strange characters such as '(', we clean the result and only maintain the information for alphabet. Sort the cleaned result in descending order by the count numbers and the first record is the expected answer for this question.

```
data_E2 = data.map(lambda x: x[2] + x[3])
answer_E2 = data_E2.flatMap(lambda x: list(x))\
                .map(lambda x: (x, 1)).reduceByKey(add).collect()
answer_E2 = [v for v in answer_E2 if v[0].isalpha()]
answer_E2.sort(key=lambda x:x[1], reverse=True)
print("\nAnswer for E2: ", \
        answer_E2[0][0], " with frequency ", answer_E2[0][1])
```

The output of above codes is

```
Answer for E2:  A  with frequency  82319942
```

The most common letter in the Turkey citizens' names is 'A' with frequency 82319942. To compare visibly, we also make a plot about all the appeared letters and their frequency in names as below.



3.3 E3: Age Distribution

First, we define a `group` function to classify ages into multiple groups: 0-18, 19-28, 29-38, 39-48, 49-59, >60 and others. Then, generate an age dataset using `map()` and `group` function. Count the population in each age group and print the age distribution according to age order.

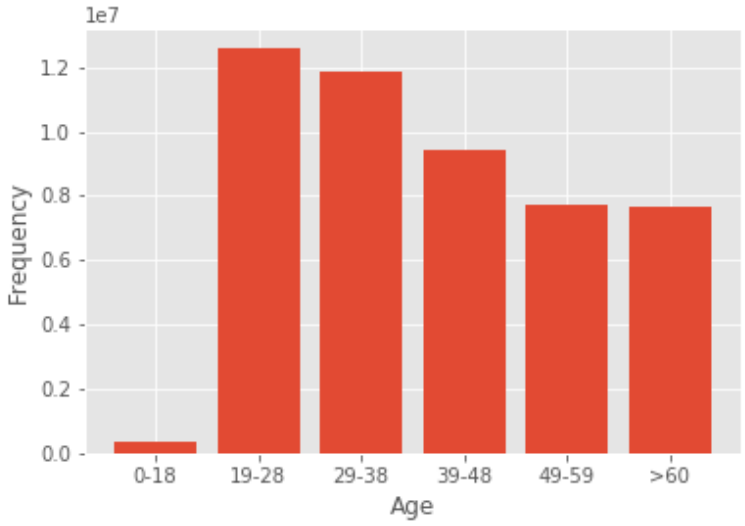
```
def group(x):
    age = int(2009-int((x[8].split("/"))[2]))
    if (age < 0):
        return ("others", 1)
    elif (age <= 18):
        return ("0-18", 1)
    elif (age <= 28):
```

```
        return ("19-28", 1)
    elif (age <= 38):
        return ("29-38", 1)
    elif (age <= 48):
        return ("39-48", 1)
    elif (age <= 59):
        return ("49-59", 1)
    else:
        return (">60", 1)

answer_E3 = data.map(group).reduceByKey(add).collect()
answer_E3.sort(key=lambda x:x[0][0])
print("\nAnswer for E3: ")
for (age, fre) in answer_E3:
    print(age, "\t",fre)
```

The output of above codes is

```
Answer for E3:
0-18     349145
19-28    12593171
29-38    11869188
39-48    9401053
49-59    7730567
>60      7668585
```

We also make a bar plot to visualize the age distribution results.



3.4 E4: Male & Female

Filter the whole dataset by gender (‘E‘ or ‘K‘) and count the number. As a result, we print the male population, female population and male to female ratio.

```
male = data.filter(lambda x: x[6] == "E")
female = data.filter(lambda x: x[6] == "K")
male_counter = male.count()
female_counter = female.count()
print("\nAnswer for E4: ")
print(male_counter, " males and ", female_counter, " females")
```

```
print("Male:Female ", male_counter/female_counter)
```

The output of above codes is

```
Answer for E4:
24534483  males and  25077226  females
Male:Female  0.9783571356736188
```

From above results, there are 24534483 males and 25077226 females and male to female ratio is about 0.978.

3.5 E5: Month with the Highest Birth Rate of Male and Female

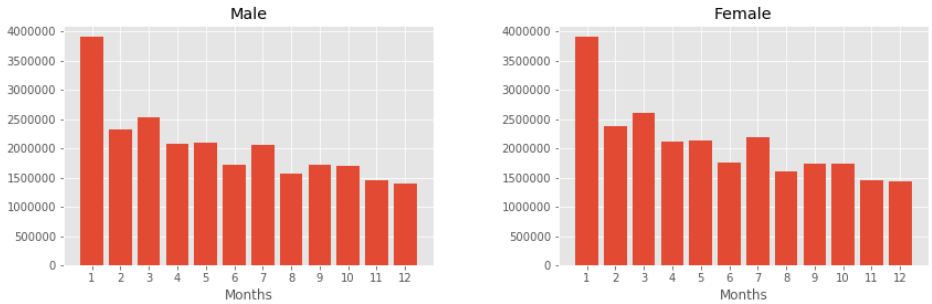
Use `map()` and `reduceByKey()` to analyse statistically male and female data. Filter the `collect` result by standard months (from January to December) and sort in descending order. The first record contains the month with the highest birth rate.

```
months = [str(i) for i in range(1, 13)]

answer_E5_E = male.map(lambda x: ((x[8].split("/"))[1], 1))\
                .reduceByKey(add).collect()
answer_E5_E = [v for v in answer_E5_E if v[0] in months]
answer_E5_E.sort(key=lambda x:x[1], reverse=True)

answer_E5_K = female.map(lambda x: ((x[8].split("/"))[1], 1))\
                .reduceByKey(add).collect()
answer_E5_K = [v for v in answer_E5_K if v[0] in months]
answer_E5_K.sort(key=lambda x:x[1], reverse=True)

print("\nAnswer for E5: ")
print("Male\t", answer_E5_E[0][0], \
    " with the highest birth rate ", int(answer_E5_E[0][1]) / male_counter)
print("Female\t", answer_E5_K[0][0], \
    " with the highest birth rate ", int(answer_E5_K[0][1]) / female_counter)
```

The output of above codes is

```
Answer for E5:
Male     1  with the highest birth rate  0.15943645521285288
Female   1  with the highest birth rate  0.15604935729334657
```

From the results, the month with the highest birth rate is January for both male and female. We also make a visualization of birth rate distribution of male and female.



(a) (b)

3.6 E6: The Most Populated Street

We use MapREduce to compute the population of every street appeared in the datase and sort the result in descending order. The first record is the street where the most people live.

```
answer_E6 = data.map(lambda x:(x[12], 1)).reduceByKey(add).collect()
answer_E6.sort(key=lambda x:x[1], reverse=True)
print("\nAnswer for E6: ", answer_E6[0][0])
```

The output of above codes is

```
Answer for E6:  CANKAYA
```

From the printing result, CANKAYA Street has the most population.

3.7 N1: The Most Common 10 Last Names of Male and Female

In this task, we use the male and female data generated before and count the appeared number of each name correspondingly for male and female through MapReduce method. Sort the result in descending order by frequency. The first 10 records are the most common 10 last names for male and female.

```
answer_N1_E = male.map(lambda x: (x[3], 1)).reduceByKey(add).collect()
answer_N1_E.sort(key=lambda x:x[1], reverse=True)


answer_N1_K = female.map(lambda x: (x[3], 1))\
                    .reduceByKey(add).collect()
answer_N1_K.sort(key=lambda x:x[1], reverse=True)

print("\nAnswer for N1: ")
print("The most common last name of male: \n", \
        [v[0] for v in answer_N1_E[:10]])
print("The most common last name of female: \n", \
        [v[0] for v in answer_N1_K[:10]])
```

The output of above codes is

```
Answer for N1:
The most common last name of male:
['YILMAZ', 'KAYA', 'DEMIR', 'SAHIN', 'CELIK', 'YILDIZ', 'YILDIRIM',
'OZTURK', 'AYDIN', 'OZDEMIR']
The most common last name of female:
['YILMAZ', 'KAYA', 'DEMIR', 'SAHIN', 'CELIK', 'YILDIZ', 'YILDIRIM',
'OZTURK', 'AYDIN', 'OZDEMIR']
```

From above results, we find male and female have the same most common 10 last names: YILMAZ, KAYA, DEMIR, SAHIN, CELIK, YILDIZ, YILDIRIM, OZTURK, AYDIN and OZDEMIR.

3.8 N2: Average Age and Aging Degree of Cities

3.8.1 Average Age

First, we group data by city and count every city's population. Then, compute the sum of citizens' age for each city through MapReduce method. The average age of each city's citizens can be obtained through dividing the sum of ages by population of the corresponding city. We print cities and their average ages.

```
city = data.map(lambda x: (x[9], 1)).reduceByKey(add).collect()
city_age = data.map(lambda x: (x[9], 2009-int((x[8].split("/"))[2])))\
            .reduceByKey(add).collect()
```

```
answer_N2_avg = []
for i in city:
    for j in city_age:
        if (i[0] == j[0]):
            answer_N2_avg.append((i[0], j[1]/i[1]))
print("\nAnswer for N2: ")
print("Average age of cities: ")
for i in answer_N2_avg:
    print('%-15s %-f'%(i[0], i[1]))
```

The part output (10 cities) of above codes is

```
Answer for N2:
Average age of cities:
BALIKESIR       44.658449
TEKIRDAG        44.464410
GUMUSHANE       41.039857
BOLU            43.816993
MALATYA         40.641307
RIZE            42.089144
CANAKKALE       45.463005
AYDIN           45.066654
ESKISEHIR       43.968123
IGDIR           38.583228
```

3.8.2 Aging Degree

First, we separately count the number of people over 60 yeas old and over 65 years old. For each city, we detect whether the ratio of the population over the age of 60 to the total population is greater than 0.1 or the ration of the population over the age of 65 to the total population is greater than 0.07. If yes, the population of this country or region is in an aging society, otherwise it's not. For results, we print the cities, the ration of the population over the age of 60 to the total population, the ration of the population over the age of 65 to the total population and whether it's in an aging society.

```
city_60 = data.filter(lambda x: 2009-int((x[8].split("/"))[2]) > 60 )\
        .map(lambda x: (x[9], 1)).reduceByKey(add).collect()
city_65 = data.filter(lambda x: 2009-int((x[8].split("/"))[2]) > 65 )\
            .map(lambda x: (x[9], 1)).reduceByKey(add).collect()
answer_N2_old = []
for i in city_60:
    for j in city_65:
        if (i[0] == j[0]):
            answer_N2_old.append([i[0], i[1], j[1]])
for i in city:
    for j in answer_N2_old:
        if (i[0] == j[0]):
            j[1] /= i[1]
            j[2] /= i[1]
            if (j[1] > 0.1) | (j[2] > 0.07):
                j.append(True)
            else:
                j.append(False)
print("Aging cities or not: ")
for i in answer_N2_old:
    print('%-15s %-f %-f %-s'%(i[0], i[1], i[2], i[3]))
```

The part output (10 cities) of above codes is

```
Aging cities or not:
BALIKESIR       0.194284 0.138414 True
```

```
TEKIRDAG        0.180609 0.126749 True
GUMUSHANE       0.134135 0.098622 True
BOLU            0.179323 0.129823 True
MALATYA         0.126919 0.086964 True
RIZE            0.142559 0.100572 True
AYDIN           0.202086 0.145008 True
CANAKKALE       0.204942 0.146162 True
ESKISEHIR       0.177397 0.122831 True
IGDIR           0.103356 0.068410 True
```

From the results, we can find Turkey is facing serious aging problem.

3.9 N3: 2 Months Birthdays Most Concentrated in for Top 10 Most Populous Cities

First, we generate the top 10 populated cities information by filtering. Then, count for every pair (‘month of birth‘, ‘city‘). Lastly, for each city, print 2 months birthdays most concentrated in and the corresponding count.

```
city_top10 = [v[0] for v in \
            sorted(city, key=lambda x:x[1], reverse=True)[:10]]
answer_N3 = data.filter(lambda x: x[9] in city_top10)\
            .map(lambda x: ((x[9], (x[8].split("/"))[1]), 1))\
            .reduceByKey(add).collect()

for i in range(len(answer_N3)):
    answer_N3[i] = [answer_N3[i][0][0], answer_N3[i][0][1], \
                    answer_N3[i][1]]
answer_N3.sort(key=itemgetter(0,2), reverse=True)

print("\nAnswer for N3: ")
for i in city_top10:
    tmp = 0
    for j in answer_N3:
        if (i == j[0]):
            print('%-10s %-s  %-d'%(i, j[1], j[2]))
            tmp += 1
        if (tmp == 2):
            break
```

The output of above codes is shown in the below table:

City	Month	Count	City	Month	Count
ISTANBUL	7	232190	KONYA	1	233455
	1	210683		3	160557
IZMIR	1	182326	ANKARA	1	179389
	7	160248		3	135286
BURSA	1	159285	SIVAS	1	162517
	7	133978		3	129923
SAMSUN	1	211699	AYDIN	1	154536
	3	122190		3	120063
ADANA	1	213863	SANLIURFA	1	309939
	3	107823		2	91797

3.10 N4: Last Name & City

3.10.1 Top 3 Last Names in the Top 10 Cities

After filtering the data by top 10 cities, we count the number of each pair (‘last name‘, ‘city‘). Then, print the top 3 used last names for each 10 cites and the corresponding used count.

```
answer_N4 = data.filter(lambda x: x[9] in city_top10)\
        .map(lambda x: ((x[9], x[3]), 1))\
        .reduceByKey(add).collect()
for i in range(len(answer_N4)):
    answer_N4[i] = [answer_N4[i][0][0], answer_N4[i][0][1], \
                    answer_N4[i][1]]
answer_N4.sort(key=itemgetter(0,2), reverse=True)

print("\nAnswer for N4: ")
answer_N4_top3 = []
answer_N4.sort(key=itemgetter(0,2), reverse=True)
for i in city_top10:
    tmp = 0
    for j in answer_N4:
        if (i == j[0]):
            print('%-10s %-10s  %-d'%(i, j[1], j[2]))
            answer_N4_top3.append([i, j[1], j[2]])
            tmp += 1
        if (tmp == 3):
            break
```

The output of above codes is shown in the below table:

City	Last Name	Count	City	Last Name	Count
ISTANBUL	YILMAZ	22028	KONYA	YILMAZ	16156
	OZTURK	15302		CELIK	11330
	AYDIN	10770		YILDIRIM	10826
IZMIR	YILMAZ	15399	ANKARA	YILMAZ	22503
	OZTURK	9905		OZTURK	15195
	KAYA	8579		OZDEMIR	12473
BURSA	YILMAZ	18134	SIVAS	SAHIN	23224
	AYDIN	13926		YILMAZ	23148
	OZTURK	12654		KAYA	21177
SAMSUN	YILMAZ	22509	AYDIN	YILMAZ	11892
	SAHIN	17619		KAYA	8583
	KAYA	15104		OZTURK	8533
ADANA	YILMAZ	13825	SANLIURFA	DEMIR	23582
	KAYA	8408		KAYA	15032
	SAHIN	8099		KILIC	13561

3.10.2 Correlation Analysis

In this task, we use the information of top 10 cities and top 3 last names for each city. We first convert strings of city and last name into categorical value with dictionaries. Then create a dataframe and implement a Chi-Square Test.

```
city_top10_dict = {}
for i in city_top10:
    city_top10_dict[i] = city_top10.index(i)
last_name_dict = {}
tmp = 0
for j in answer_N4_top3:
    if j[1] not in last_name_dict.keys():
        last_name_dict[j[1]] = tmp
        tmp += 1

data_N4 = []
for v in answer_N4_top3:
    data_N4 += [(city_top10_dict[v[0]], \
```

```
                Vectors.dense([last_name_dict[v[1]]])) \
                for _ in range(v[2])]
spark =SparkSession(sc)
dataset = spark.createDataFrame(data_N4, ["label", "features"])
r = ChiSquareTest.test(dataset, 'features', 'label').head()
print("pValues: " + str(r.pValues))
print("degreesOfFreedom: " + str(r.degreesOfFreedom))
print("statistics: " + str(r.statistics))
```

The result of Chi-Square Test is shown below:

```
pValues: [0.0]
degreesOfFreedom: [81]
statistics: [962961.358706]
```

From the result, we can conclude that the last name and city is relevant with p-value nearly 0.

References

[1] https://spark.apache.org/docs/latest/rdd-programming-guide.html