Projects of using Spark - Second Part

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").set("spark.executor.memory","12g")
conf.set("spark.executor.cores","4").set("spark.memory.offHeap.enabled","true")
conf.set("spark.executor.memoryOverhead","4096").set("spark.memory.offHeap.size","4096")
sc = SparkContext(conf=conf)
```

3 Normal Section

3.1 The most frequent name

The steps are as follows:

- concatenate first name and last name, forms a pair with gender
- devided data by gender and count their frequence
- sorted the list and find the top 5 names

The codes are as follows:

```
data = sc.textFile('hdfs://localhost:9001/user/data_new.txt').map(lambda x :
    re.split("\t", x))
namepair = data.map(lambda x :(x[6]," ".join([x[2],x[3]])))
maleNameCount = namepair.filter(lambda x : x[0] == "E").map(lambda x:
    (x[1],1)).reduceByKey(lambda x, y: x+y)
femaleNameCount = namepair.filter(lambda x: x[0] == "K").map(lambda x:
    (x[1],1)).reduceByKey(lambda x, y: x+y)
print("top 5 name of male", sorted(maleNameCount.collect(), key = lambda t: t[1],
    reverse = True)[:5])
print("top 5 name of female", sorted(femaleNameCount.collect(), key = lambda t:
    t[1], reverse = True)[:5])
```

results are as follows:

```
top 5 name of male [('MEHMET YILMAZ', 15665), ('MUSTAFA YILMAZ', 13049), ('MEHMET
    KAYA', 12177), ('MEHMET DEMIR', 11947), ('AHMET YILMAZ', 10137)]
top 5 name of female [('FATMA YILMAZ', 17376), ('AYSE YILMAZ', 13475), ('EMINE
    YILMAZ', 11462), ('FATMA KAYA', 11424), ('FATMA DEMIR', 11207)]
```

3.2 Top 3 name in top 10 cities

The steps are as follows:

- concatenate first name and last name, forms a pair with gender
- group the pairs by the cities name
- rank the group based on the number of pairs, get the pairs of top 10 cities
- use Counter function to get the frequence of name within the group
- use nlargest of heapq to get the top 3 name

The codes are as follows:

```
from heapq import nlargest
from operator import add
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], " ".join([x[2],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])
top3name = 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 name in top 10 cities", top3name.collect())
```

results are as follows:

```
top 3 name in top 10 cities [('ISTANBUL', ['MURAT YILMAZ', 'MURAT KAYA', 'MUSTAFA
    YILMAZ']), ('ANKARA', ['MURAT YILMAZ', 'MURAT SAHIN', 'MUSTAFA YILMAZ']),
    ('IZMIR', ['MUSTAFA YILMAZ', 'MURAT YILMAZ', 'MEHMET YILMAZ']), ('ADANA',
    ['MEHMET YILMAZ', 'MEHMET KAYA', 'MEHMET DEMIR']), ('GAZIANTEP', ['MEHMET
    YILMAZ', 'MEHMET KAYA', 'MEHMET YILDIRIM']), ('BURSA', ['MEHMET YILMAZ',
    'AHMET YILMAZ', 'AYSE YILMAZ']), ('KAHRAMANMARAS', ['MEHMET DEMIR', 'MEHMET
    KOSE', 'FATMA DEMIR']), ('SANLIURFA', ['MEHMET DEMIR', 'FATMA DEMIR', 'MEHMET
    CIFTCI']), ('KONYA', ['MEHMET YILMAZ', 'MUSTAFA YILMAZ', 'MUSTAFA KAYA']),
    ('MALATYA', ['MEHMET DOGAN', 'MEHMET KAYA', 'MEHMET YILMAZ'])]
```

4 Hard Section

4.1 City prediction model

The steps are as follows:

- Use the address street as the feature, the random forest as model
- Use StringIndexer to label the street name to be feature
- Fit the randomforest classifier
- Evaluate using Hit@5

The codes are as follows:

```
currentdata = data.map(lambda x: [x[11], x[14].split()[0]])
print(currentdata.take(10))

# print(data.map(lambda x: x[14]).distinct().count()) # 207482
# print(data.map(lambda x: x[14].split()[0]).distinct().count()) # 119346
schema = ['city','street_address']
df = spark.createDataFrame(currentdata, schema)

indexer = [StringIndexer(inputCol = column, outputCol = column + "_index") for
    column in schema]

pipeline = Pipeline(stages = indexer)
df = pipeline.fit(df).transform(df)
df = df.drop(*schema)
df.show()

from pyspark.ml.classification import RandomForestClassifier

vectorForm = df.rdd.map(lambda row: (row[0], Vectors.dense(row[1:2])))
Vectordf = spark.createDataFrame(vectorForm, ['label', 'features'])
trainData, validData, testData = Vectordf.rdd.randomSplit([0.7,0.1,0.2])
# randomforest
rf = RandomForestClassifier(labelCol="label", featuresCol="features", numTrees=3)
rfmodel = rf.fit(trainData.toDF())
predictions = rfmodel.transform(testData.toDF())

p1 = predictions.select('probability', 'label')

hits = p1.rdd.map(lambda x: (sorted(enumerate(x[0]), key = lambda y: -y[1]),
    x[1])).map(lambda row: ([x[0] for x in row[0]], row[1])) # index
hits = hits.map(lambda x: [ 1 if x[0][i]== x[1] else 0 for i in range(5)])
total = hits.count()
hits = [hits.filter(lambda x: x[i] == 1).count()/total for i in range(5)]
print(hits)
```

results are as follows (represent Hit@1, Hit@2, Hit@3, Hit@4, Hit@5):

```
[0.22996436894098976, 0.07439285909703836, 0.04869332379240792,
    0.04343798619180908, 0.038241862958964444]
```

Detailed dataFrame are as follows:

```
[['MALATYA', 'BOGAZICI'], ['MALATYA', 'CITILBAGI'], ['MALATYA', 'BOGAZICI'],
    ['MALATYA', 'KALE'], ['MALATYA', 'CITILBAGI'], ['MALATYA', 'CITILBAGI'],
    ['MALATYA', 'KALE'], ['MALATYA', 'KALE'], ['MALATYA', 'KALE'], ['MALATYA',
    'BOGAZICI']]
+----------+--------------------+
|city_index|street_address_index|
+----------+--------------------+
|      24.0|               809.0|
|      24.0|             92338.0|
|      24.0|               809.0|
|      24.0|                79.0|
|      24.0|             92338.0|
|      24.0|             92338.0|
|      24.0|                79.0|
|      24.0|                79.0|
|      24.0|                79.0|
|      24.0|               809.0|
|      24.0|                79.0|
|      24.0|                79.0|
|      24.0|                79.0|
|      24.0|                79.0|
|      24.0|               809.0|
|      24.0|               809.0|
|      24.0|             92338.0|
|      24.0|                79.0|
|      24.0|             92338.0|
|      24.0|               809.0|
+----------+--------------------+
only showing top 20 rows
+--------------------+-----+
|         probability|label|
+--------------------+-----+
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.06093868190131...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
|[0.28781466500985...| 24.0|
+--------------------+-----+
only showing top 30 rows
```

4.2 Gender prediction model

The steps are as follows:

- Use the first_name as features, the random forest as model
- Use StringIndexer to label features
- Fit the randomforest classifier
- Evaluate using Hit@5

The codes are as follows:

```
currentdata = data.map(lambda x: [x[6], x[2]])
print(currentdata.take(10))

schema = ['gender','first_name']
df = spark.createDataFrame(currentdata, schema)

indexer = [StringIndexer(inputCol = column, outputCol = column + "_index") for
    column in schema]

pipeline = Pipeline(stages = indexer)
df = pipeline.fit(df).transform(df)
df = df.drop(*schema)
df.show()

from pyspark.ml.classification import RandomForestClassifier

vectorForm = df.rdd.map(lambda row: (row[0], Vectors.dense(row[1:2])))
Vectordf = spark.createDataFrame(vectorForm, ['label', 'features'])
trainData, validData, testData = Vectordf.rdd.randomSplit([0.7,0.1,0.2])
# randomforest
rf = RandomForestClassifier(labelCol="label", featuresCol="features", numTrees=3)
rfmodel = rf.fit(trainData.toDF())
predictions = rfmodel.transform(testData.toDF())

p1 = predictions.select('probability', 'label')

hits = p1.rdd.map(lambda x: (sorted(enumerate(x[0]), key = lambda y: -y[1]),
    x[1])).map(lambda row: ([x[0] for x in row[0]], row[1])) # index

hits = hits.map(lambda x: [ 1 if x[0][i]== x[1] else 0 for i in range(5)])# 1-5hit
total = hits.count()
hits = [hits.filter(lambda x: x[i] == 1).count()/total for i in range(5)]
print(hits)
```

results are as follows (represent Hit@1, Hit@2, Hit@3, Hit@4, Hit@5):

```
[0.6220115951730908, 0.3779884048269092] (Hit@1,Hit@2)
```

Detailed dataFrame are as follows:

```
[['K', 'NESLIHAN'], ['K', 'SADET'], ['K', 'GONUL'], ['E', 'MURAT'], ['K', 'SENEM'],
    ['K', 'ZOHRE'], ['K', 'ARIFE'], ['K', 'AYSE'], ['K', 'MELEK'], ['K', 'YILDIZ']]
+------------+----------------+
|gender_index|first_name_index|
+------------+----------------+
|         0.0|           196.0|
|         0.0|           700.0|
|         0.0|           130.0|
|         1.0|            10.0|
|         0.0|           461.0|
|         0.0|           453.0|
|         0.0|           107.0|
|         0.0|             3.0|
|         0.0|            43.0|
|         0.0|           241.0|
|         1.0|           399.0|
|         1.0|            56.0|
|         1.0|            92.0|
|         1.0|           122.0|
|         1.0|             2.0|
|         1.0|           160.0|
|         0.0|            44.0|
|         1.0|           590.0|
|         0.0|          3423.0|
|         1.0|           237.0|
+------------+----------------+
only showing top 20 rows
+--------------------+-----+
|         probability|label|
+--------------------+-----+
|[0.53775505166535...|  0.0|
|[0.66461526699637...|  0.0|
|[0.66461526699637...|  1.0|
|[0.53775505166535...|  1.0|
|[0.53775505166535...|  1.0|
|[0.53775505166535...|  1.0|
|[0.53775505166535...|  0.0|
|[0.65933147469178...|  0.0|
|[0.53775505166535...|  1.0|
|[0.53775505166535...|  1.0|
|[0.64171891715466...|  0.0|
|[0.52413583516642...|  1.0|
|[0.53775505166535...|  1.0|
|[0.66461526699637...|  1.0|
|[0.14706768174629...|  1.0|
|[0.52413583516642...|  1.0|
|[0.14706768174629...|  0.0|
|[0.14706768174629...|  1.0|
|[0.39298099239038...|  0.0|
|[0.52413583516642...|  1.0|
|[0.14706768174629...|  1.0|
|[0.14706768174629...|  1.0|
|[0.14706768174629...|  1.0|
|[0.14706768174629...|  0.0|
|[0.65933147469178...|  0.0|
|[0.64171891715466...|  1.0|
|[0.52413583516642...|  1.0|
|[0.14706768174629...|  1.0|
|[0.14706768174629...|  1.0|
|[0.14706768174629...|  1.0|
+--------------------+-----+
only showing top 30 rows
```

4.3 Surname prediction model

The steps are as follows:

- Use the father first name& mother first name as features, the random forest as model
- Use StringIndexer to label features
- Fit the randomforest classifier
- Evaluate using Hit@5

The codes are as follows:

```
currentdata = data.map(lambda x: [x[3], x[4][0:2], x[5][0:2]]) # only first two
    letters
print(currentdata.take(10))

schema = ['last_name','mother_first','father_first']
df = spark.createDataFrame(currentdata, schema)

indexer = [StringIndexer(inputCol = column, outputCol = column + "_index") for
    column in schema]

pipeline = Pipeline(stages = indexer)
df = pipeline.fit(df).transform(df)
df = df.drop(*schema)
df.show()

from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.classification import LogisticRegression
vectorForm = df.rdd.map(lambda row: (row[0], Vectors.dense(row[1:3])))
Vectordf = spark.createDataFrame(vectorForm, ['label', 'features'])
trainData, validData, testData = Vectordf.rdd.randomSplit([0.7,0.1,0.2])
# linear regression
# lr = LogisticRegression(regParam=0.1, elasticNetParam=1.0, family="multinomial")
# lrmodel = lr.fit(trainData.toDF())
# predictions = lrmodel.transform(testData.toDF())
# randomforest
# rf = RandomForestClassifier(labelCol="label", featuresCol="features", numTrees=2)
# rfmodel = rf.fit(trainData.toDF())
# predictions = rfmodel.transform(testData.toDF())
# bayes
nb = NaiveBayes(labelCol="label", featuresCol="features")
nbmodel = nb.fit(trainData.toDF())
predictions = nbmodel.transform(testData.toDF())
p1 = predictions.select('probability', 'label')

hits = p1.rdd.map(lambda x: (sorted(list(enumerate(x[0]))[0:5], key = lambda y:
    -y[1]), x[1])).map(lambda row: ([x[0] for x in row[0]], row[1])) # index
# hits = hits.map(lambda x: [ 1 if x[0][i]== x[1] else 0 for i in range(5)])
total = hits.count()
hits1 = hits.map(lambda x: 1 if x[0][0]== x[1] else 0).filter(lambda x: x ==
    1).count()/total
print(hits1)
hits2 = hits.map(lambda x: 1 if x[0][1]== x[1] else 0).filter(lambda x: x ==
    1).count()/total
print(hits2)
hits3 = hits.map(lambda x: 1 if x[0][2]== x[1] else 0).filter(lambda x: x ==
    1).count()/total
print(hits3)
hits4 = hits.map(lambda x: 1 if x[0][3]== x[1] else 0).filter(lambda x: x ==
    1).count()/total
print(hits4)
hits5 = hits.map(lambda x: 1 if x[0][4]== x[1] else 0).filter(lambda x: x ==
    1).count()/total
print(hits5)
```

results are as follows (represent Hit@1, Hit@2, Hit@3, Hit@4, Hit@5):

```
0.011681659170414792
0.011706646676661669
0.0114880059970015
0.010476011994002998
0.010950774612693653
```

Detailed dataFrame are as follows:

```
[['ZENGIN', 'ZE', 'OS'], ['YILDIRIM', 'ZO', 'IS'], ['CETIN', 'ES', 'HA'], ['GENC',
    'EL', 'IS'], ['YILDIRIM', 'SE', 'MU'], ['YILDIRIM', 'SE', 'MU'], ['GUZEL',
    'KA', 'OS'], ['DURMAZ', 'YE', 'SU'], ['ZENGIN', 'MA', 'ME'], ['AKBULUT', 'SA',
    'IH']]
+---------------+-----------------+-----------------+
|last_name_index|mother_first_index|father_first_index|
+---------------+-----------------+-----------------+
|          128.0|              2.0|             16.0|
|            7.0|             65.0|             12.0|
|           15.0|             43.0|              1.0|
|          117.0|             12.0|             12.0|
|            7.0|              4.0|              4.0|
|            7.0|              4.0|              4.0|
|          101.0|             22.0|             16.0|
|          150.0|             46.0|             11.0|
|          128.0|             24.0|              0.0|
|           50.0|              9.0|             52.0|
|         2909.0|              0.0|              0.0|
|          101.0|             38.0|              7.0|
|         2909.0|              0.0|              0.0|
|         2909.0|              0.0|              0.0|
|          128.0|              6.0|             12.0|
|        13599.0|              2.0|              0.0|
|          145.0|              6.0|             35.0|
|          128.0|             10.0|              7.0|
|          128.0|             32.0|              0.0|
|          128.0|             25.0|              0.0|
+---------------+-----------------+-----------------+
only showing top 20 rows
+--------------------+------+
|         probability| label|
+--------------------+------+
|[4.56094295119874...|   7.0|
|[0.01285558774628...|   7.0|
|[1.74650656636327...| 150.0|
|[7.01122768393193...| 128.0|
|[0.01201613983682...|2909.0|
|[4.14413525343956...| 128.0|
|[0.01234441380727...| 101.0|
|[1.54563240750777...| 356.0|
|[0.00640652405548...| 145.0|
|[0.00803767106449...| 128.0|
|[2.92601392388087...|  50.0|
|[0.00533342826855...|  63.0|
|[0.00341756709651...| 889.0|
|[0.01207441688951...| 949.0|
|[0.01314199722417...| 128.0|
|[0.01176314471367...| 356.0|
|[5.33703247045961...|1245.0|
|[2.98716181818661...|1245.0|
|[0.01671362328769...| 843.0|
|[0.01270495960130...|  40.0|
|[0.01266454781561...|1245.0|
|[5.16572881061125...|   0.0|
|[1.03287097433738...| 731.0|
|[2.90398256540169...|  47.0|
|[0.01025616804462...| 446.0|
|[0.00228672423637...| 446.0|
|[2.20954751004919...|1540.0|
|[2.92601392388087...|5761.0|
|[5.61650265072350...|1245.0|
|[2.45754050154942...|1245.0|
+--------------------+------+
only showing top 30 rows
```

4.4 Further Detail

We use Naive Bayers, LogisticRegression and Random forest for those three tasks in Hard section. Random forest performs better in previous two tasks.

However, due to the limitation of the computation resource and too large quantities of classes for the feature and the predicting label in task 3, I make a compromise in the model and the features: we only use Naive Bayes and we only use first two letters in mother_first feature and father_first feature.