HW2_BigData_Q2

October 18, 2019

0.1 Question 2. Graph Analysis

[7]:
```
import numpy as np
from pyspark import *
```

[8]:
```
# Configure Spark
conf = SparkConf()
sc = SparkContext.getOrCreate(conf=conf)
# The directory for the file
filename = "q1.txt"
```

[36]:
```
# Finished. Return RDD
def getData(sc, filename):
    """
    Load data from raw text file into RDD and transform.
    Hint: transfromation you will use: map(<lambda function>).
    Args:
        sc (SparkContext): spark context.
        filename (string): hw2.txt cloud storage URI.
    Returns:
        RDD: RDD list of tuple of (<User>, [friend1, friend2, ... ]),
        each user and a list of user's friends
    """
    # read text file into RDD
    data = sc.textFile(filename)

    # TODO: implement your logic here
    data = data.map(lambda line: np.array([str(x) for x in line.replace('\n','').
↪split('\t')]))
    data = data.map(lambda p:(int(p[0]), p[1].split(',')))

    return data
```

[37]:
```
def getEdges(line):
    # similar to mapFriends() in Q1, edges are direct friendship
    friends = line[1]
    user = line[0]
    if friends != ['']:
        for i in range(len(friends)):
            # Direct friend
            yield((user, int(friends[i])))
```

0.1.1 1. Format data into edges and vertices

[38]:
```
# Get data in proper format
data = getData(sc, filename)
```

[39]:
```
# Get vertics
vertices = data.map(lambda x: (x[0],))
vertices.take(5)
```

[39]:
```
[(0,), (1,), (2,), (3,), (4,)]
```

[40]:
```
# Get edges
edges = data.flatMap(getEdges)
edges.take(5)
```

[40]:
```
[(0, 1), (0, 2), (0, 3), (0, 4), (0, 5)]
```

0.1.2 2. Convert the RDD to DataFrame

[41]:
```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .master("local[*]") \
    .appName("Learning_Spark") \
    .getOrCreate()
```

[42]:
```
# Convert vertices to DF
v = spark.createDataFrame(vertices,["id"])
v.show(5)
```

```
+---+
| id|
+---+
|  0|
|  1|
|  2|
|  3|
|  4|
+---+
only showing top 5 rows
```

```
[43]: # Convert edges to DF
      e = spark.createDataFrame(edges, ["src","dst"])
      e.show(5)
```

```
+---+---+
|src|dst|
+---+---+
|  0|  1|
|  0|  2|
|  0|  3|
|  0|  4|
|  0|  5|
+---+---+
only showing top 5 rows
```

0.1.3 3. Create graph

```
[18]: from graphframes import *
```

```
[19]: sc.setCheckpointDir('/Users/mac/Desktop/BigData/HW2')
```

```
[22]: g = GraphFrame(v, e)
```

```
[45]: g.edges.show()
```

```
+---+---+
|src|dst|
+---+---+
|  0|  1|
|  0|  2|
|  0|  3|
|  0|  4|
|  0|  5|
|  0|  6|
|  0|  7|
|  0|  8|
|  0|  9|
|  0| 10|
|  0| 11|
|  0| 12|
|  0| 13|
|  0| 14|
|  0| 15|
|   0|  16|
|   0|  17|
|   0|  18|
|   0|  19|
|   0|  20|
+----+----+
only showing top 20 rows
```

0.1.4 4. Connected Components

```
[24]: result = g.connectedComponents()
```

```
[82]: result.count()
```

```
[82]: 49995
```

(1). Number of clusters in this dataset

```
[81]: result.select("component").distinct().count()
```

```
[81]: 917
```

```
[99]: count = result.groupBy("component").count().orderBy("count",ascending=False)
```

(2) Top 10 clusters

```
[100]: count.show(10)
```

```
+---------+-----+
|component|count|
+---------+-----+
|        0|48860|
|    38403|   66|
|    18466|   31|
|    18233|   25|
|    18891|   19|
|      864|   16|
|    49297|   13|
|    19199|    6|
|     7658|    5|
|    22897|    4|
+---------+-----+
only showing top 10 rows
```

[104]:
```
# number of users in the top 10 clusters
from pyspark.sql.functions import sum as _sum
count.limit(10).agg(_sum("count")).show()
```

```
+----------+
|sum(count)|
+----------+
|     49045|
+----------+
```

[87]:
```
# confirm with another method
from pyspark.sql.functions import col
result.where(col("component").
 ↪isin({0,38403,18466,18233,18891,864,49297,19199,7658,22897})).count()
```

[87]: 49045

(3) List all 25 user IDS in cluster 18233

[113]:
```
count.filter("count=25").select("component").show()
```

```
+---------+
|component|
+---------+
|    18233|
+---------+
```

[114]:
```
result.filter("component=18233").select("id").show(25)
```

```
+-----+
|   id|
+-----+
|18233|
|18234|
|18235|
|18236|
|18237|
|18238|
|18239|
|18240|
|18241|
|18242|
|18243|
|18244|
|18245|
|18246|
|18247|
|18248|
|18249|
|18250|
|18251|
|18252|
|18253|
|18254|
|18255|
|18256|
|18257|
+-----+
```

0.1.5 5. Page rank

(4). Top 10 important users

```
[52]: pr = g.pageRank(tol=0.01)
```

```
[63]: pr.vertices.select("id", "pagerank").orderBy("pagerank",ascending=False).
       ↪show(10)
```

```
+-----+------------------+
|   id|          pagerank|
+-----+------------------+
|10164|17.315312963089895|
|15496|14.866327204150846|
|14689|12.685692559698428|
|24966| 12.26882183906656|
| 7884|11.827780808752543|
|  934| 11.49589135687648|
|45870| 11.27397140801791|
| 5148|11.222433130678017|
|20283| 11.14062997830236|
|46039| 11.02696924843223|
+-----+------------------+
only showing top 10 rows
```

(5). Try different parameters

```
[116]: pr1 = g.pageRank(resetProbability=0.15, tol=0.01)
       pr1.vertices.select("id", "pagerank").orderBy("pagerank",ascending=False).
        ↪show(10)
```

```
+-----+------------------+
|   id|          pagerank|
+-----+------------------+
|10164|17.315312963089895|
|15496|14.866327204150846|
|14689|12.685692559698428|
|24966| 12.26882183906656|
| 7884|11.827780808752543|
|  934| 11.49589135687648|
|45870| 11.27397140801791|
| 5148|11.222433130678017|
|20283| 11.14062997830236|
|46039| 11.02696924843223|
+-----+------------------+
only showing top 10 rows
```

[117]:
```
pr2 = g.pageRank(resetProbability=0.5, tol=0.01)
pr2.vertices.select("id", "pagerank").orderBy("pagerank",ascending=False).
↪show(10)
```

```
+-----+------------------+
|   id|          pagerank|
+-----+------------------+
|10164|18.539756319902864|
|15496|15.895700017529919|
|14689|13.814565627780183|
|24966|12.594967254720714|
| 5148| 12.13232924938358|
|38123|12.107079705652753|
| 7884|11.988217312291413|
|  934|11.939041942106776|
|  910|11.207783548336854|
|44815|11.092504432507283|
+-----+------------------+
only showing top 10 rows
```

[118]:
```
pr3 = g.pageRank(resetProbability=0.15, tol=0.1)
pr3.vertices.select("id", "pagerank").orderBy("pagerank",ascending=False).
↪show(10)
```

```
+-----+------------------+
|   id|          pagerank|
+-----+------------------+
|10164|19.200290615258158|
|15496|16.546851217080825|
|14689|14.940716809515001|
|24966|13.124783956624656|
| 5148|12.759229785981626|
|38123|12.556966112921204|
|  934|12.430209408516708|
| 7884|12.380173406826115|
|  910|11.995515035966134|
|44815|11.990097101490727|
+-----+------------------+
only showing top 10 rows
```

[120]:
```
pr4 = g.pageRank(resetProbability=0.15, tol=0.01, sourceId=10164)
pr4.vertices.select("id", "pagerank").orderBy("pagerank",ascending=False).
 ↪show(10)
```

```
+-----+--------------------+
|   id|            pagerank|
+-----+--------------------+
|10164|  0.5405405405405407|
|10239|0.004594594594594596|
|10182|0.004594594594594596|
|10246|0.004594594594594596|
|10178|0.004594594594594596|
|10176|0.004594594594594596|
|10168|0.004594594594594596|
|10166|0.004594594594594596|
|10237|0.004594594594594596|
|  222|0.004594594594594596|
+-----+--------------------+
only showing top 10 rows
```

[121]:
```
result.filter("id=10164").show()
```

```
+-----+---------+
|   id|component|
+-----+---------+
|10164|        0|
+-----+---------+
```

[130]:
```
g.edges.filter("src=10164").count()
```

[130]:
```
100
```

[131]:
```
g.edges.filter("dst=10164").count()
```

[131]:
```
100
```

[123]:
```
result.filter("id=15496").show()
```

```
+-----+---------+
|   id|component|
+-----+---------+
|15496|        0|
+-----+---------+
```

[124]:
```
g.edges.filter("src=15496" or "dst=15496").count()
```

[124]:
```
100
```

[125]:
```
result.filter("id=46039").show()
```

```
+-----+---------+
|   id|component|
+-----+---------+
|46039|        0|
+-----+---------+
```

[132]:
```
g.edges.filter("src=1000").count()
```

[132]:
```
25
```

[ ]: