EECS E6893 Big Data Analytic HW2

Chong Hu ch3467

October 18, 2019

Problem 1. `Friends Recommendation`

Result:

```
19/10/11 01:12:02 WARN Nati
Using Spark's default log4j
Setting default log level t
To adjust logging level use
For user 13420:
(4736, 3)
(7651, 3)
(10469, 3)
(14264, 3)
(351, 2)
(2101, 2)
(2554, 2)
(7608, 2)
(8508, 2)
(8711, 2)

For user 8942:
(8939, 3)
(8940, 1)
(8943, 1)
(8944, 1)

For user 5850:
(5819, 3)
(5805, 2)
(5811, 2)
(5815, 2)
(5828, 2)
(5831, 2)
(5836, 2)
(219, 1)
(576, 1)
(639, 1)
```

```
For user 49824:
(49846, 3)
(41581, 2)
(43382, 2)
(49786, 2)
(49788, 2)
(49789, 2)
(49814, 2)
(49819, 2)
(49834, 2)
(16, 1)

For user 924:
(439, 1)
(2409, 1)
(6995, 1)
(11860, 1)
(15416, 1)
(43748, 1)
(45881, 1)

For user 44410:
(4231, 3)
(44462, 3)
(351, 2)
(4302, 2)
(6318, 2)
(8221, 2)
(9095, 2)
(10328, 2)
(10370, 2)
(10462, 2)
```

```
For user 8974:
(8960, 12)
(12241, 12)
(8774, 10)
(6973, 7)
(8969, 6)
(8980, 4)
(8982, 4)
(8984, 4)
(8978, 3)
(8979, 3)

For user 8941:
(8943, 2)
(8944, 2)
(8940, 1)

For user 9019:
(9022, 2)
(317, 1)
(9023, 1)

For user 9993:
(9991, 5)
(13134, 1)
(13478, 1)
(13877, 1)
(34299, 1)
(34485, 1)
(34642, 1)
(37941, 1)
```

Codes:

```
1  from pyspark import SparkConf, SparkContext
2  import pyspark
3  import sys
4
5
6  def getData(sc, filename):
7      """
8      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)
    data = data.map(lambda line: line.split("\t")).map(
        lambda line: (int(line[0]), [int(x) for x in line[1].split(",")] if len(
            line[1]) else []))
    return data


def map_friends(line):
    """
    map function to construct edge between friends
    construct a pair of ((friend1, friend2) -> common friends list)
    if two friends are already direct friends, then common friends list
    is empty.
    :param line: tuple of (<User>, [friend1, friend2, ... ]),
                each user and a list of user's friends
    :return:
    """
    user = line[0]
    friends = line[1]
    yield ((user, user), [])
    for i in range(len(friends)):
        yield ((user, friends[i]), [])
        for j in range(len(friends)):
            yield ((friends[i], friends[j]), [user])
            yield ((friends[j], friends[i]), [user])


def reduce_friends_pair(pair1, pair2):
    """
    reduce function to reduce same friend-friend pairs.
    If the common friend list is None, which means they are direct friends,
    still return empty list
    :param pair1: reduceByKey first pair
    :param pair2: reduceByKey second pair
    :return: a pair of ((friend1, friend2) -> common friends list)
            if two friends are already direct friends,
```

```
54              then common friends list is empty.
55      """
56      if len(pair1) == 0 or len(pair2) == 0:
57          return []
58      common_friends = set(pair1).union(set(pair2))
59      return list(common_friends)
60
61
62  def find_mutual(line):
63      """
64      map mutual edges into (user, (friend, common_friend_num))
65      :param line: a pair of ((friend1, friend2) -> common friends list)
66              if two friends are already direct friends,
67              then common friends list is empty.
68      :return: (user, (friend, common_friend_num))
69      """
70      return line[0][0], (line[0][1], len(line[1]))
71
72
73  def sort_top_friends(line):
74      """
75      sort friend has most common friends
76      :param line:  (user, [(friend, common_friend_num), ...])
77      :return: recommendations result for the user
78      """
79      user = line[0]
80      friends = sorted(line[1], key=lambda x: (x[1], -x[0]), reverse=True)
81      while len(friends) > 0:
82          if friends[-1][1] == 0:
83              friends.pop()
84          else:
85              break
86      if len(friends) > 10:
87          friends = friends[0:10]
88      return user, friends
89
90
91  def main():
92      # Configure Spark
93      sc = pyspark.SparkContext.getOrCreate()
94      # The directory for the file
95      filename = "q1.txt"
96
97      # Get data in proper format
98      data = getData(sc, filename)
```

```
99
100     # Get set of all mutual friends
101     mapData = data.flatMap(map_friends).reduceByKey(reduce_friends_pair)
102     # print(mapData.take(10))
103     # For each person, get top 10 mutual friends
104     getFriends = mapData.map(find_mutual).groupByKey().map(sort_top_friends)
105     # print(getFriends.take(5))
106     # Only save the ones we want
107     wanted = [924, 8941, 8942, 9019, 49824, 13420, 44410, 8974, 5850, 9993]
108     result = getFriends.filter(lambda x: x[0] in wanted).collect()
109     for res in result:
110         print("For user %d:" % res[0])
111         for recommendation in res[1]:
112             print(recommendation)
113         print()
114     sc.stop()
115
116
117 if __name__ == "__main__":
118     main()
```

Problem 2. Graph Analysis

Connected Components:

```
clusters amount:  917

number of users in top 10 cluster
cluster id: 0   number of users:    48860
cluster id: 38403   number of users:    66
cluster id: 18466   number of users:    31
cluster id: 18233   number of users:    25
cluster id: 18891   number of users:    19
cluster id: 864 number of users:    16
cluster id: 49297   number of users:    13
cluster id: 19199   number of users:    6
cluster id: 7658    number of users:    5
cluster id: 22897   number of users:    4
Total number of users in top 10 cluster:    49045

user ids for the cluster which has 25 users
[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]
```

(1) How many clusters / connected components in total for this dataset?

```
clusters amount:  917
```

(2) How many users in the top 10 clusters? There are different number of users in each clusters, so rank them and give the top 10 clusters with the largest amount of users.

```
number of users in top 10 cluster
cluster id:         0         number of users:         48860
cluster id:         38403         number of users:         66
```

```
cluster id:        18466        number of users:        31
cluster id:        18233        number of users:        25
cluster id:        18891        number of users:        19
cluster id:        864        number of users:        16
cluster id:        49297        number of users:        13
cluster id:        19199        number of users:        6
cluster id:        7658        number of users:        5
cluster id:        22897        number of users:        4
Total number of users in top 10 cluster:        49045
```

(3) What are the user ids for the cluster which has 25 users? Basically, list out all the 25 user IDs in that cluster.

```
user ids for the cluster which has 25 users
[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]
```

Page rank

```
a list of 10 important users (User ID) in this network:
[10164, 15496, 14689, 24966, 7884, 934, 45870, 5148, 20283, 46039]
The most important one is 10164

Using different parameter:
a list of 10 important users (User ID) in this network:
[10164, 15496, 14689, 24966, 7884, 934, 45870, 20283, 46039, 14996]
The most important one is 10164
```

(4) Provide a list of 10 important users (User ID) in this network. Who is the most important one? Order by the "PageRank" value.

```
a list of 10 important users (User ID) in this network:
[10164, 15496, 14689, 24966, 7884, 934, 45870, 5148, 20283, 46039]
The most important one is 10164
```

(5) By using different parameter settings for PageRank, is there any difference? This is an open question, you can try as many as you want. Provide the screenshots of your tests.

```
Using different parameter:
a list of 10 important users (User ID) in this network:
[10164, 15496, 14689, 24966, 7884, 934, 45870, 20283, 46039, 14996]
The most important one is 10164
```

Although we used different parameter settings for PageRank, we can still get similar result. We can see many common users in both result and the most important one is the same.

6 Why this user become the most important one? What are the possible reasons? This is an open question, basically, understand how PageRank works. You can also use the result from the connected component to explain it.

From the connected component analysis, I can find this user is in the largest cluster. This makes this user relatively important. Beyond that, most of this user's friend might have high PageRank value. This user's friends who have high PageRank value contributes to this user.

7 PageRank Calculation

	ID1	ID2	ID3	ID4	ID5
Iteration 0	0.2	0.2	0.2	0.2	0.2
Iteration 1	0.07	0.29	0.41	0.07	0.16
Iteration 2	0.09	0.45	0.25	0.09	0.12
Iteration 3	0.13	0.29	0.29	0.13	0.16
Iteration 4	0.09	0.34	0.33	0.09	0.15

Table 2: PageRank Calculation Details

Here is whole code for part II.

```
import pyspark
from pyspark import SparkConf, SparkContext
from pyspark import SQLContext
import os
from graphframes import *


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)
    data = data.map(lambda line: line.split("\t")).map(
        lambda line: (int(line[0]), [int(x) for x in line[1].split(",")] if len(
            line[1]) else []))
    return data


def get_vertices(data, sqlcontext):
    """
    get vertices
    :param data: RDD list of tuple of (<User>, [friend1, friend2, ... ]),
        each user and a list of user's friends
    :param sqlcontext: SQLContext
    :return: dataframe
    """
    vertices = data.map(lambda line: (line[0],))

    return sqlcontext.createDataFrame(vertices, schema=["id"])


def get_edges(data, sqlcontext):
    """
    get edges
    :param data: RDD list of tuple of (<User>, [friend1, friend2, ... ]),
        each user and a list of user's friends
    :param sqlcontext: SQLContext
    :return:
    """

    def map_friends(line):
        """
        map function to construct edge between friends
        construct a pair of ((friend1, friend2) -> common friends list)
        if two friends are already direct friends, then common friends list
        is empty.
        :param line: tuple of (<User>, [friend1, friend2, ... ]),
                    each user and a list of user's friends
        :return: friend pair
        """
        user = line[0]
        friends = line[1]
        for i in range(len(friends)):
            yield (user, friends[i])

```

```
64      edges = data.flatMap(map_friends)
65      return sqlcontext.createDataFrame(edges, schema=["src", "dst"])
66
67
68  def connected_components(graph):
69      """
70      run connected components on graph
71      :param graph: Graph contains vertices and edges
72      :return:
73      """
74      print("connected components")
75      result = graph.connectedComponents()
76
77      # How many clusters / connected components in total for this dataset
78      cluster_num = result.select("component").distinct().count()
79      print("clusters amount: ", cluster_num)
80      print()
81
82      # How many users in the top 10 clusters?
83      print("number of users in top 10 cluster")
84      res1 = result.groupBy("component").count().orderBy('count',
85                                                         ascending=False)
86      res2 = res1.head(10)
87      total = 0
88      for row in res2:
89          total += row["count"]
90          print("cluster id:\t%d\tnumber of users:\t%d" % (
91              row["component"], row["count"]))
92      print("Total number of users in top 10 cluster:\t", total)
93      print()
94
95      # What are the user ids for the cluster which has 25 users?
96      print("user ids for the cluster which has 25 users")
97      cluster_id = res1.where(res1["count"] == 25).select("component").collect()
98      cluster_id = [row["component"] for row in cluster_id]
99      user_list = result.where(result["component"].isin(cluster_id)).select(
100         "id").collect()
101     user_ls = [row["id"] for row in user_list]
102     print(user_ls)
103     print()
104     return
105
106
107 def page_rank(graph):
108     """
```

```
    run PageRank on graph
    :param graph: Graph contains vertices and edges
    :return:
    """
    print("PageRank:")
    result = graph.pageRank(resetProbability=0.15, tol=0.01)

    # Provide a list of 10 important users (User ID) in this network.
    print("a list of 10 important users (User ID) in this network:")
    user_list = result.vertices.select("id", "pagerank") \
        .orderBy('pagerank',
                 ascending=False).head(10)
    user_ls = [row["id"] for row in user_list]
    print(user_ls)
    print("The most important one is %d" % user_ls[0])
    print()

    # using different parameter settings for PageRank
    print("Using different parameter:")
    result = graph.pageRank(resetProbability=0.1, maxIter=20)
    print("a list of 10 important users (User ID) in this network:")
    user_list = result.vertices.select("id", "pagerank") \
        .orderBy('pagerank',
                 ascending=False).head(10)
    user_ls = [row["id"] for row in user_list]
    print(user_ls)
    print("The most important one is %d" % user_ls[0])


def main():
    # Configure Spark
    if not os.path.isdir("checkpoints"):
        os.mkdir("checkpoints")
    conf = SparkConf().setMaster('local').setAppName('connected components')
    sc = SparkContext(conf=conf)
    sqlcontext = SQLContext(sc)
    SparkContext.setCheckpointDir(sc, "checkpoints")

    # The directory for the file
    filename = "../q1/q1.txt"

    # Get data in proper format
    data = getData(sc, filename)
    edges = get_edges(data, sqlcontext)
    vertices = get_vertices(data, sqlcontext)
```

```
154     graph = GraphFrame(vertices, edges)
155     connected_components(graph=graph)
156     page_rank(graph=graph)
157 
158 
159 if __name__ == '__main__':
160     main()
```