CS168 Assignment [6]

SUNet ID(s): 16337266
Name(s): Xu Yuan

Part 1

(a) Line graph:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \tag{1}$$

Line graph with added point:

$$L = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & -1 \\ -1 & 3 & -1 & 0 & 0 & -1 \\ 0 & -1 & 3 & -1 & 0 & -1 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ -1 & -1 & -1 & -1 & -1 & 5 \end{bmatrix} \tag{2}$$

Circle graph:

$$L = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ -1 & 0 & 0 & 0 & -1 & 2 \end{bmatrix} \tag{3}$$

Circle graph with added point:

$$L = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 3 & -1 & 0 & 0 & -1 & -1 \\ -1 & 3 & -1 & 0 & 0 & -1 \\ 0 & -1 & 3 & -1 & 0 & -1 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & 0 & 0 & -1 & 3 & -1 \\ -1 & -1 & -1 & -1 & -1 & 5 \end{bmatrix} \tag{4}$$

(b) Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import collections as cs

n=100

def part1_b_a():
    line_D=np.zeros((n,n))
    for i in range(n):
        if i==0 or i==n-1:
            line_D[i][i]=1
        else:
            line_D[i][i]=2
    eig_value,eig_vector=np.linalg.eig(line_D)
    part1_plot(eig_value,eig_vector,"part1_b_a_D","part1_b_a_D")
    line_A=np.zeros((n,n))
    for i in range(n-1):
        line_A[i][i+1]=1
        line_A[i+1][i]=1
    print(line_A)
    line_L=line_D-line_A
    eig_value,eig_vector=np.linalg.eig(line_A)
    part1_plot(eig_value,eig_vector,"part1_b_a_A","part1_b_a_A")
    return line_L

def part1_b_b():
    line_D=np.zeros((n,n))
    for i in range(n):
        if i==0 or i==n-2:
            line_D[i][i]=2
        elif i==n-1:
            line_D[i][i]=n-1
        else:
            line_D[i][i]=3
    eig_value,eig_vector=np.linalg.eig(line_D)
    part1_plot(eig_value,eig_vector,"part1_b_b_D","part1_b_b_D")
    line_A=np.zeros((n,n))
    for i in range(n-1):
        line_A[i][i+1]=1
        line_A[i+1][i]=1
        line_A[n-1][i]=1
        line_A[i][n-1]=1
    line_L=line_D-line_A
    eig_value,eig_vector=np.linalg.eig(line_A)
    part1_plot(eig_value,eig_vector,"part1_b_b_A","part1_b_b_A")
    return line_L
```

```
def part1_b_c():
    line_D=np.zeros((n,n))
```

```
    for i in range(n):
            line_D[i][i]=2
    eig_value,eig_vector=np.linalg.eig(line_D)
    part1_plot(eig_value,eig_vector,"part1_b_c_D","part1_b_c_D")
    line_A=np.zeros((n,n))
    for i in range(n-1):
        line_A[i][i+1]=1
        line_A[i+1][i]=1
    line_A[n-1][0]=1
    line_A[0][n-1]=1
    line_L=line_D-line_A
    eig_value,eig_vector=np.linalg.eig(line_A)
    part1_plot(eig_value,eig_vector,"part1_b_c_A","part1_b_c_A")
    return line_L

def part1_b_d():
    line_D=np.zeros((n,n))
    for i in range(n):
        if i==n-1:
            line_D[i][i]=n-1
        else:
            line_D[i][i]=3
    eig_value,eig_vector=np.linalg.eig(line_D)
    part1_plot(eig_value,eig_vector,"part1_b_d_D","part1_b_d_D")
    line_A=np.zeros((n,n))
    for i in range(n-1):
        line_A[i][i+1]=1
        line_A[i+1][i]=1
        line_A[n-1][i]=1
        line_A[i][n-1]=1
    line_A[0][n-2]=1
    line_A[n-2][0]=1
    line_L=line_D-line_A
    eig_value,eig_vector=np.linalg.eig(line_A)
    part1_plot(eig_value,eig_vector,"part1_b_d_A","part1_b_d_A")
    return line_L

def part1_plot(eig_value,eig_vector,title,filename):
    index_and_value=dict(enumerate(eig_value))
    counter=cs.Counter(index_and_value)
    sorted_value=counter.most_common()
    largest_value=sorted_value[0][0]
    second_largest_value=sorted_value[1][0]
    smallest_value=sorted_value[-1][0]
    second_smallest_value=sorted_value[-2][0]
    x=[i for i in range(n)]
    plt.scatter(x,eig_vector[:,largest_value],color='red',marker='o',
                                            label="largest")
    plt.scatter(x,eig_vector[:,second_largest_value],color='yellowgreen',
                                            marker='o',label="
                                            second_largest")
```

```
    plt.scatter(x,eig_vector[:,smallest_value],color='purple',marker='o',
                                            label="smallest")
    plt.scatter(x,eig_vector[:,second_smallest_value],color='dodgerblue',
                                            marker='o',label="
                                            second_smallest")
    plt.title(title)
    plt.xlabel("i")
    plt.ylabel("vector")
    plt.legend(shadow=True, loc = 0)
    plt.savefig(filename + ".png", format = 'png')
    plt.close()
```

explain:
V^tLV is the sum of squares of differences between values of neighboring nodes. Eigenvectors corresponding to the lowest eigenvalues minimize the distance between neighbors, and largest eigenvalues maximize discrepancy between neighbor values.
For the Laplacian matrix, the number of occurrences of the eigenvalue 0 is the number of connected regions of the graph, the minimum eigenvalue is always 0, and the smallest non-zero eigenvalue represents the connectivity of the graph. Four case with eight figures:

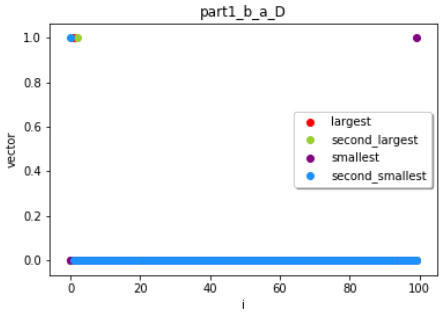

Figure 1: Line graph D

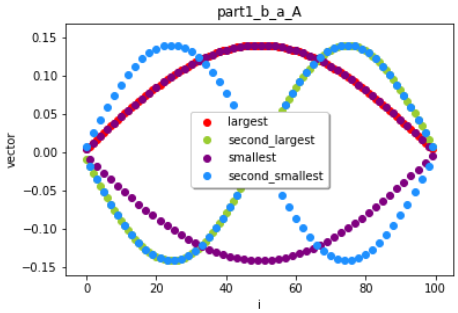

Figure 2: Line graph A

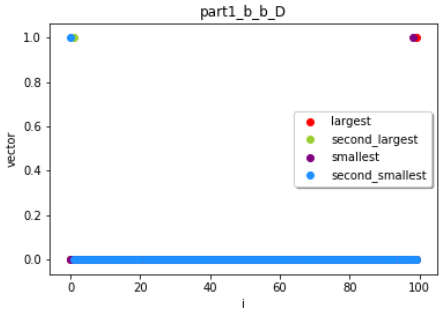

Figure 3: Line graph with added point D

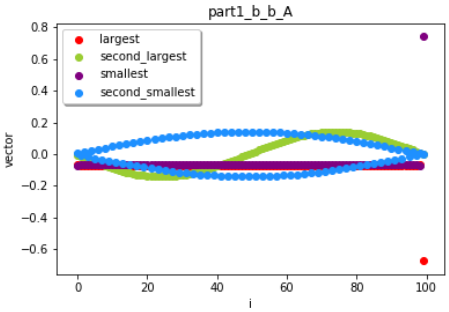

Figure 4: Line graph with added point A

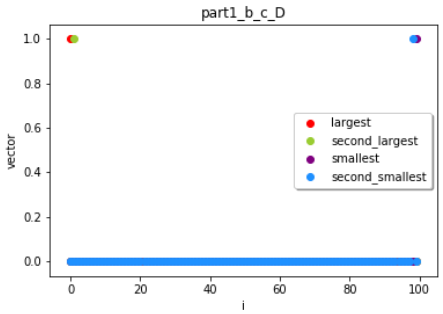

Figure 5: Circle graph D

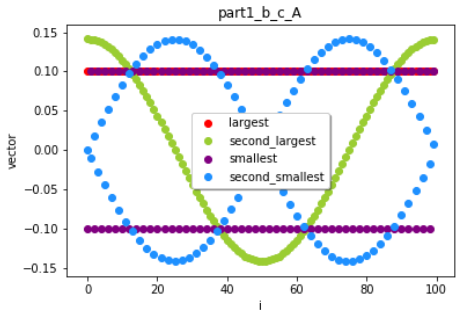

Figure 6: Circle graph A

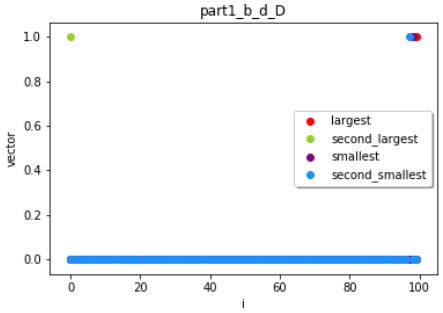

Figure 7: Circle graph with added point D

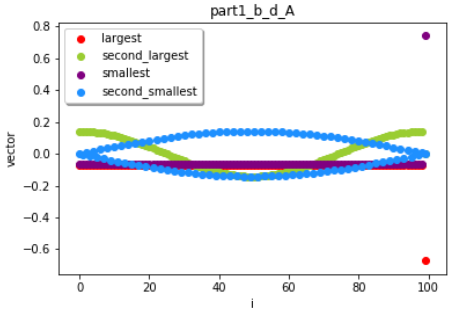

Figure 8: Circle graph with added point A

(c) The 2nd and 3st vector for part1(a):

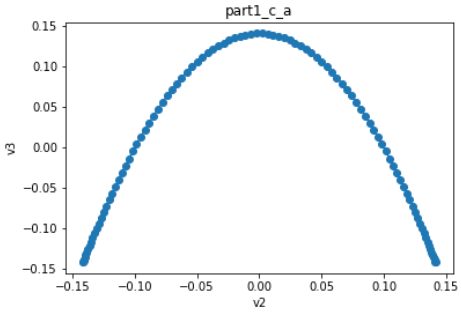

Figure 9: Line graph

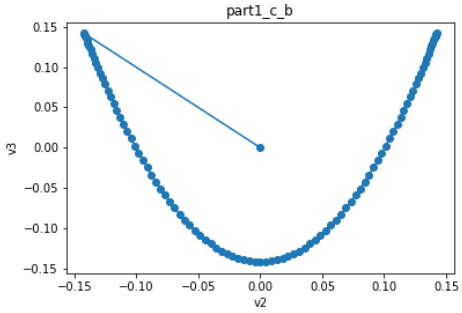

Figure 10: Line graph with added point

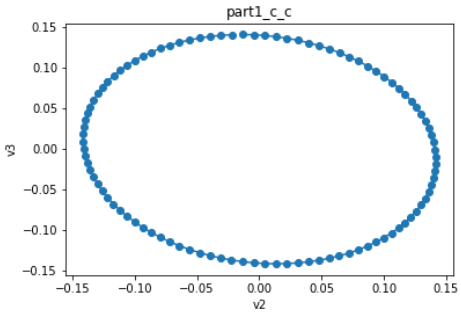

Figure 11: Circle graph

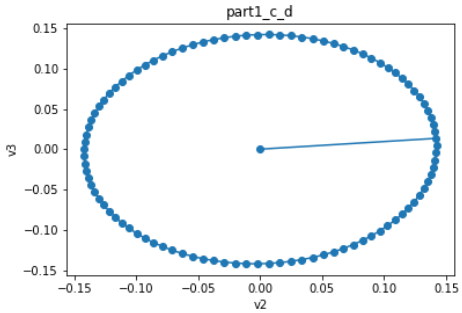

Figure 12: Circle graph with added point

(d) explain:
The blue part likes $part_c$, which corresponding to the 2nd and 3rd smallest eigenvalues.The red part is initial random point with x<0.5 and y<0.5. As we can see, those point cluster togerther in graph. The feature vector corresponding to the small eigenvalue attempts to project the adjacent points to the area where the distance is as small as possible, that is, the neighbor points are gathered together, so the eigenvectors of the Laplacian matrix can be used to achieve clustering of similar points.

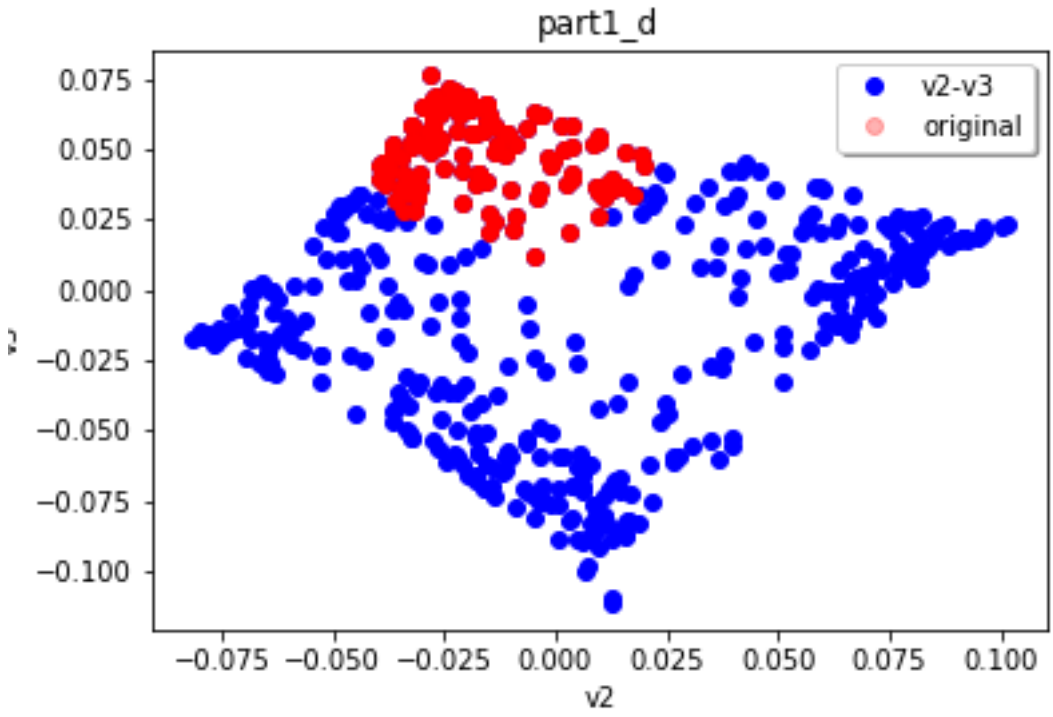

Figure 13: result

Part 2

(a)

```
import pandas as pd
 def getData(filename):
 data=pd.read_csv(filename, header = None)
 return data.as_matrix()
```

(b) Code:

```
def part2_b(data):
  A=np.zeros((n,n))
  D=np.zeros((n,n))
  for i in range(len(data)):
      friend1=data[i][0]
      friend2=data[i][1]
      A[friend1-1][friend2-1]=1
      A[friend2-1][friend1-1]=1
  for i in range(n):
      count=0
      for j in range(n):
          if A[i][j]==1:
              count+=1
      D[i][i]=count
  L=D-A
  eig_value,eig_vector=np.linalg.eig(L)
  index_and_value=dict(enumerate(eig_value))
  counter=cs.Counter(index_and_value)
  sorted_value=counter.most_common()
  nums=len(sorted_value)
  min12=[]
  #print(sorted_value[-12:])
  for i in range(0,12):
      min12.append(eig_vector[:,sorted_value[nums-1-i][0]])
  for i in range(12):
      print(sorted_value[nums-1-i][1])
```

a list of the smallest 12 eigenvalues:
$-8.567981706996446e^{-14}$
$-1.503327412681734e^{-14}$
$1.053325953495575e^{-14}$
$2.1316282072803006e^{-14}$
$5.97895987393027e^{-14}$
$6.732932566596304e^{-14}$
0.014304016619472412
0.05379565273694503
0.07390297669241075
0.08128966971232651

0.12022393183754802
0.13283886699773656

(c) From part1 we learned how to use the second smallest eigenvector and the third eigenvector to cluster similar points. We use this method again to verify.
There are 6 ponits in graph, which means 6 connected components.
Because value of 10^{-12} should probably be regarded as 0.The sum of minimum nonzero eigenvalue is also 6. Tehre are:
$-8.567981706996446e^{-14}$
$-1.503327412681734e^{-14}$
$1.053325953495575e^{-14}$
$2.1316282072803006e^{-14}$
$5.97895987393027e^{-14}$
$6.732932566596304e^{-14}$
We can conclude that the minimum nonzero eigenvalue is the algebraic connectivity of the graph.

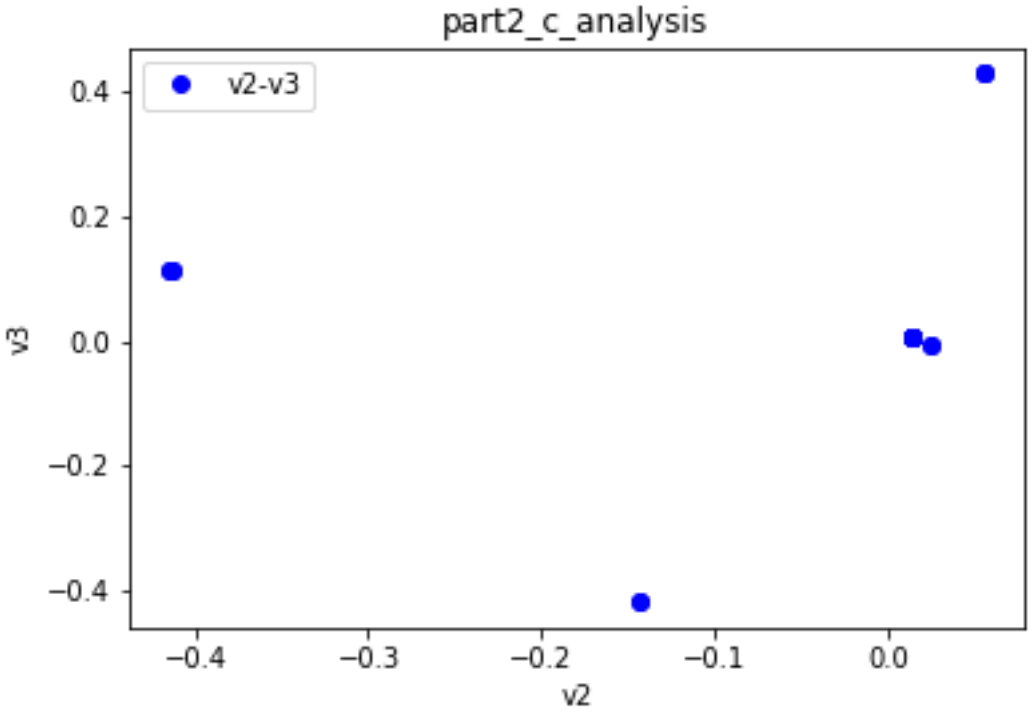

Figure 14: connected components

(d) Firstly, I try some possible eigenvector to find a suitable one which can split those points to three sets that corresponding to request. Then I calculate conductance of each set to verify if all of them are less than 0.1. We use the 8th smallest eig vector. It is obvious to split data.

```
def part2_d(L):
    eig_value,eig_vector=np.linalg.eig(L)
    index_and_value=dict(enumerate(eig_value))
    counter=cs.Counter(index_and_value)
    sorted_value=counter.most_common()
    x=[x for x in range(1,n+1)]
    for i in range(5,9):
        y=eig_vector[:,sorted_value[-1*i][0]]
        plt.scatter(x,y)
        plt.title("personID and vector")
        plt.xlabel("person")
        plt.ylabel("vector")
        plt.legend(loc = 0)
        plt.show()
        filename='part2_d_find_'+str(i)
        plt.savefig(filename + ".png", format = 'png')
        plt.close()
    #choose eig_vec_(-7)
    best_vec=eig_vector[:,sorted_value[-8][0]]
    set1=[]
    set2=[]
    set3=[]
    for i in range(n):
        if best_vec[i]<=-0.03 and best_vec[i]>-0.05:
            set1.append(i)
        elif best_vec[i]>=0 and best_vec[i]<0.01:
            set2.append(i)
        elif best_vec[i]>=0.07 and best_vec[i]<0.09:
            set3.append(i)
    set1_=[i for i in range(n) if i not in set1]
    set2_=[i for i in range(n) if i not in set2]
    set3_=[i for i in range(n) if i not in set3]
    cdt1=nx.algorithms.cuts.conductance(Graph,set1,set1_)
    cdt2=nx.algorithms.cuts.conductance(Graph,set2,set2_)
    cdt3=nx.algorithms.cuts.conductance(Graph,set3,set3_)
    print("10 items in set1:")
    print(set1[:10])
    print("the conductance of set1:",cdt1 )
    print("10 items in set2:")
    print(set2[:10])
    print("the conductance of set1:",cdt2 )
    print("10 items in set3:")
    print(set3[:10])
    print("the conductance of set1:",cdt3 )
    #print result
```

```
#Result:
10 items in set1:
[1, 5, 7, 12, 16, 20, 54, 59, 66, 70]
the conductance of set1: 0.008974964572508267
10 items in set2:
[0, 2, 4, 6, 8, 9, 11, 13, 14, 15]
the conductance of set1: 0.006294256490952006
10 items in set3:
[3, 10, 24, 121, 255, 275, 278, 313, 351, 398]
the conductance of set1: 0.023746701846965697
```

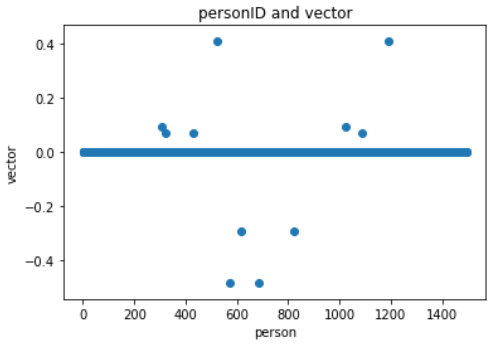

Figure 15: the 5th smallest eig_vec

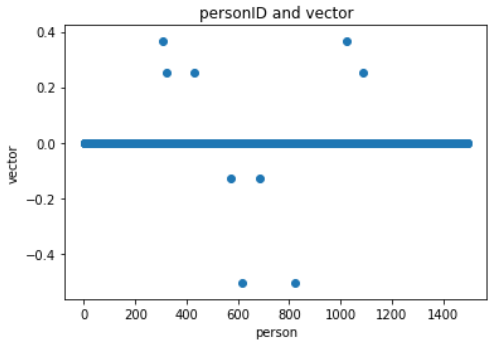

Figure 16: the 6th smallest eig_vec

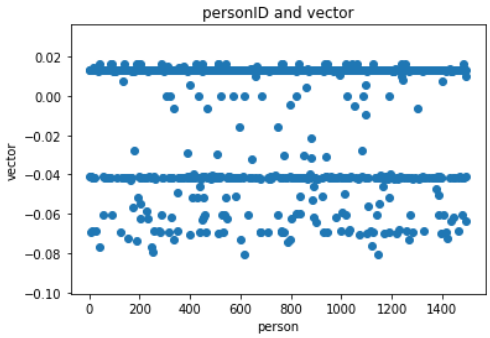

Figure 17: the 7th smallest eig_vec

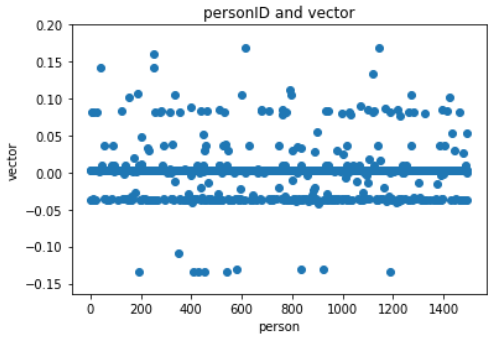

Figure 18: the 8th smallest eig_vec

(e) Code:

```
def part2_e(L):
    random_set=random.sample(range(0,n),150)
    random_set_=[x for x in range(n) if x not in random_set]
    cdt=nx.algorithms.cuts.conductance(Graph,random_set,random_set_)
    print("random result:",cdt)
```

```
#result:
random result: 0.8856132075471698
```

Compared with $part2_d$, random's conductance is very high. And its tight-knit is low, because these points come from different groups.