CSE 258 Assignment 2 Linbin Yang A53277054

Task(Classifier Evaluation)

1.

1.1. Split the dataset into train/test/valid, then construct the input matrix vector.

1.2. Prepare the objective function for logistic regression

1.3. Prepare the derivative for the corresponding objective function

(For task 1, the logits that the inner returns is result of one linear function, theta.T*X, while new_inner returns sigmoid(logits) in task 3)

The final result I get:

Accuracy on test dataset: `0.718948757950318`

Accuracy on valid dataset: `0.7150827933765299`

2.

The results I get are as follows

P= 11976

N= 4690

TP= 9015

TN= 2961

FP= 3362

FN= 1328

3.

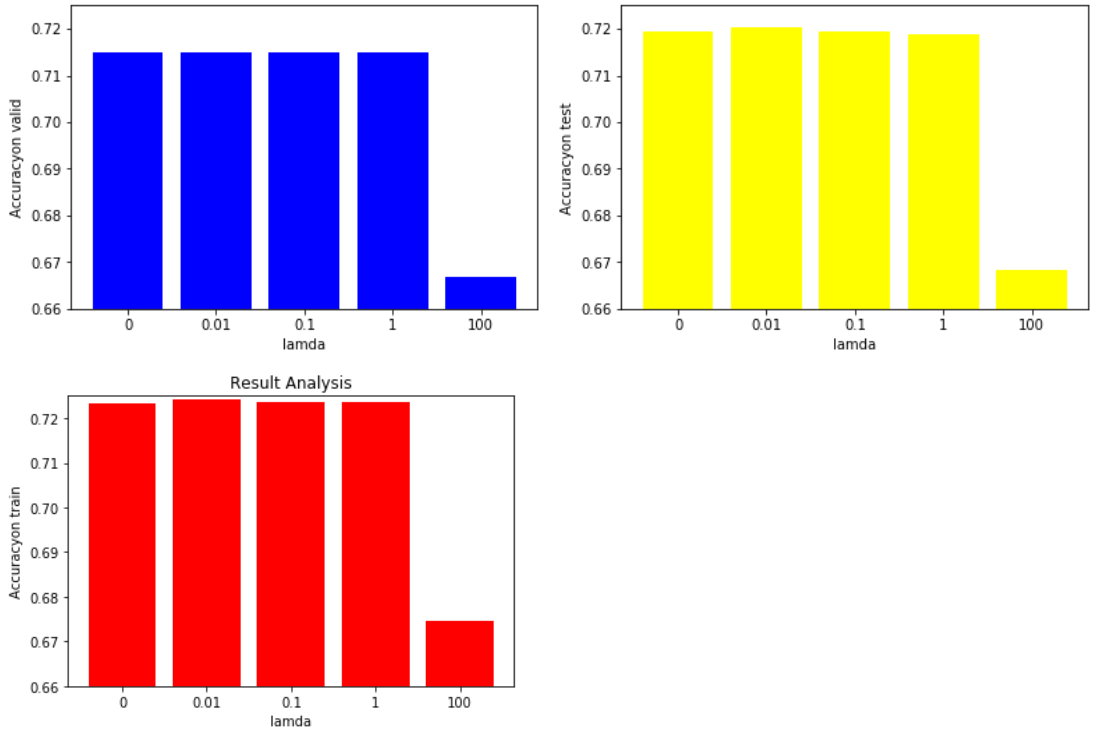
The difference between SVM and LR is that each sample in LR has same importance. This question need to attach 10 times more importance to FP compared with FN, so we reconstruct the f and fprime here. When y = 0 and logits > 0, that is FP, we multiply theta with 10, so when we do Gradient Descent, theta change more because FP datasets, it changes the total cost more.

```
# Construct New Objective
def new_f (theta, x, y, lam):
    negative_likelihood = 0
    for i in range(len(x)):
        logits = new_inner(x[i],theta)
        if y[i] == 1:
            nagative_likelihood += y[i]*math.log(logits)
        elif y[i] == 0:
            if logits > 0:
                negative_likelihood += (1-y[i])*math.log(1-My_sigmoid(sum(x[i][j]*10*theta[j] for j in range(len(theta)))))
            else:
                negative_likelihood += (1-y[i])*math.log(1-logits)
            negative_likelihood += (1-y[i])
    for k in range(len(theta)):
        negative_likelihood = negative_likelihood - lam * theta[k]*theta[k]
    return -negative_likelihood
```

```
def new_fprime(theta, x, y, lam):
    dl = [0] * len(theta)
    # all together there are len(theta) cofficients
    for i in range(len(x)):
        logits = new_inner(x[i],theta)
        for k in range(len(theta)):
            if y[i] == 1:
                dl[k] -= logits * (1-logits) * x[i][k]
            elif y[i] == 0:
                if logits > 0:
                    dl[k] -= 10*(1-y[i])*(-1)*x[i][k]
                else:
                    dl[k] -= (1-y[i])*(-1)*x[i][k]
    for k in range(len(theta)):
        dl[k] -= lam*2*theta[k]
    return np.array([-x for x in dl])
```

4.

I got the following result:

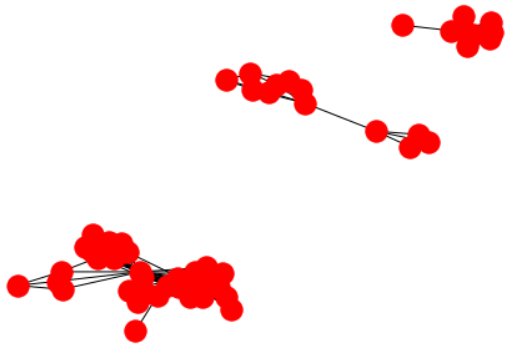


We got that when lam = 0, the model has the best performance on valid set, so we choose lam = 0.

Task (Community Detection)

5.

All together there are 3 clusters and the biggest cluster has 40 nodes. When we visualize the cluster using nx package, it shows as follows:



The result is directly got using the functions in open source nx package which you can find at: https://networkx.github.io/documentation/stable/

6.

I just sort the 40 nodes according to ID and split them into two parts,

first_half = [697, 703, 708, 713, 719, 729, 745, 747, 753, 769, 772, 774, 798, 800, 803, 804, 805, 810, 811, 819]

second_half = [823, 825, 828, 830, 840, 856, 861, 863, 864, 869, 876, 878, 880, 882, 884, 886, 888, 889, 890, 893]

and the final normalized-cut-cost (NCC) I got is

`0.42240587695133147`

7.

After the greedy algorithm, I got the final NCC:

`0.09817045961624274`

The final split is:

first_half = [697, 703, 708, 713, 719, 745, 747, 753, 769, 772, 774, 800, 803, 805, 810, 811, 819, 828, 823, 830, 840, 880, 890, 869, 856, 798]

second_half = [825, 861, 863, 864, 876, 878, 882, 884, 886, 888, 889, 893, 729, 804]

8. After reimplementing the algorithm, I got the final NCC:

`0.3326342975206613`

I find that if you directly use the split set from question 7, you got .338 or .337, but if you use the 50/50 data from question 6, you get the result above.

Now the split is (using 50/50 dataset)

first_half = [697, 703, 708, 713, 719, 745, 747, 772, 774, 800, 803, 805, 810, 819, 828, 823, 830, 840, 880, 798]

second_half = [825, 856, 861, 863, 864, 869, 876, 878, 882, 884, 886, 888, 889, 890, 893, 729, 804, 753, 811, 769]

In [12]:
```
import numpy as np
import urllib.request
import scipy.optimize
import random
import matplotlib.pyplot as plt
```

In [13]:
```
lam = 1.0
```

In [14]:
```
def parseData(fname):
    for l in urllib.request.urlopen(fname):
        yield eval(l)
```

In [15]:
```
# load dataset
print ("Loading dataset......")
data = list(parseData("http://jmcauley.ucsd.edu/cse255/data/beer/beer_50000.json"))
print ("done")
```

```
Loading dataset......
done
```

In [16]:
```
# Shuffle the data and split it into three
random.shuffle(data)
train_set = data[:int(len(data)/3)]
test_set = data[int(len(data)/3):int(len(data)/3)*2]
validate_set = data[int(len(data)/3)*2:]
```

In [17]:
```
# Construct train Input for Q1.1
X_train = [[1, x['review/taste'], x['review/appearance'], x['review/aroma'], x['review/palate'], x['review/overall']] for x in train_set]
Y_train = [True if x['beer/ABV'] >= 6.5 else False for x in train_set]
```

In [18]:
```
# Construct validation Input for Q1.1
X_valid = [[1, x['review/taste'], x['review/appearance'], x['review/aroma'], x['review/palate'], x['review/overall']] for x in validate_set]
Y_valid = [True if x['beer/ABV'] >= 6.5 else False for x in validate_set]
```

In [19]:
```
# Construct test Input for Q1.1
X_test = [[1, x['review/taste'], x['review/appearance'], x['review/aroma'], x['review/palate'], x['review/overall']] for x in test_set]
Y_test = [True if x['beer/ABV'] >= 6.5 else False for x in test_set]
```

In [20]:
```
# Construct Sigmoid Function
def My_sigmoid(x):
    return 1/(1+np.exp(-x))
```

In [21]:
```
# Inner Multiply
def inner(x, y):
    return sum(x[i] * y[i] for i in range(len(x)))
```

In [22]:
```
# Construct Objective
def f (theta, x, y, lam):
    negative_likelihood = 0
    for i in range(len(x)):
        logits = inner(x[i],theta)
        negative_likelihood = negative_likelihood - np.log(1+np.exp
(-logits))
        if not y[i]:
            negative_likelihood = negative_likelihood - logits
    for k  in range(len(theta)):
        negative_likelihood = negative_likelihood - lam * theta[k]*
theta[k]
    return -negative_likelihood
```

In [23]:
```
# Calculate the Derivative
def fprime(theta, x, y, lam):
    dl = [0] * len(theta)
    # all together there are len(theta) cofficients
    for i in range(len(x)):
        logits = inner(x[i],theta)
        for k in range(len(theta)):
            dl[k] += x[i][k] * (1-My_sigmoid(logits))
            if not y[i]:
                dl[k] -= x[i][k]
    for k in range(len(theta)):
        dl[k] -= lam*2*theta[k]
    return np.array([-x for x in dl])
```

In [24]:
```
# Traning process
def train(lam,x,y):
    theta,_,_ = scipy.optimize.fmin_l_bfgs_b(f, [0]*len(x[0]), fpri
me, pgtol = 10, args = (x, y, lam))
    return theta
```

In [25]:
```
# Predict Process
def predict(input_x, output_y, theta):
    score = [inner(theta,x ) for x in input_x]
    predictions = [s>0 for s in score]
    correct = [(a==b) for a,b in zip(predictions, output_y)]
    acc = sum(correct) * 1.0/len(correct)
    return acc
```

In [26]:
```
# Start Train on tran_set
theta = train(lam, X_train, Y_train)
acc = predict(X_test, Y_test, theta)
print ("acc on test set= "+str(acc))
```

```
acc on test set= 0.718948757950318
```

In [27]:
```
# Calculate Acc on valid_set
acc = predict(X_valid, Y_valid, theta)
print ("acc on valid set= "+str(acc))
```

```
acc on valid set= 0.7150827933765299
```

In [28]:
```
# True False Positive Negative
theta = train(lam, X_test,Y_test)
score =[inner(theta,x) for x in X_test]
predictions = [s > 0 for s in score]
correct = [(a==b) for a,b in zip(predictions, Y_test)]
```

In [29]:
```
P = sum(correct)
N = len(correct) - P
TP = 0
TN = 0
FP = 0
FN = 0
for i in range(len(correct)):
    if correct[i] == True:
        if predictions[i] == True:
            TP = TP + 1
        else:
            TN = TN + 1
    else:
        if predictions[i] == True:
            FP = FP + 1
        else:
            FN = FN + 1
print ("P= "+str(P))
print ("N= "+str(N))
print ("TP= "+str(TP))
print ("TN= "+str(TN))
print ("FP= "+str(FP))
print ("FN= "+str(FN))
```

```
P= 11976
N= 4690
TP= 9015
TN= 2961
FP= 3362
FN= 1328
```

In [6]:
```
def new_inner(x, y):
    return My_sigmoid(sum(x[i] * y[i] for i in range(len(x))))
```

In [1]:
```
# Construct New Objective
def new_f (theta, x, y, lam):
    negative_likelihood = 0
    for i in range(len(x)):
        logits = new_inner(x[i],theta)
        if y[i] == 1:
            nagative_likelihood += y[i]*math.log(logits)
        elif y[i] == 0:
            if logits > 0:
                negative_likelihood += (1-y[i])*math.log(1-My_sigmo
id(sum(x[i][j]*10*theta[j] for j in range(len(theta)))))
            else:
                negative_likelihood += (1-y[i])*math.log(1-logits)
            negative_likelihood += (1-y[i])
    for k  in range(len(theta)):
        negative_likelihood = negative_likelihood - lam * theta[k]*
theta[k]
    return -negative_likelihood
```

In [ ]:
```
# Calculate the new Derivative
def new_fprime(theta, x, y, lam):
    dl = [0] * len(theta)
    # all together there are len(theta) cofficients
    for i in range(len(x)):
        logits = new_inner(x[i],theta)
        for k in range(len(theta)):
            if y[i] == 1:
                dl[k] -= logits * (1-logits) * x[i][k]
            elif y[i] == 0:
                if logits > 0:
                    dl[k] -= 10*(1-y[i])*(-1)*x[i][k]
                else:
                    dl[k] -= (1-y[i])*(-1)*x[i][k]
    for k in range(len(theta)):
        dl[k] -= lam*2*theta[k]
    return np.array([-x for x in dl])
```

In [30]:
```
acc = [] #Used to store the acc on train/valid/test for each lam
def pipeline():
    for elem in [0, 0.01, 0.1, 1, 100]:
        unit_acc = []
        theta = train(elem, X_train, Y_train)
        unit_acc.append(predict(X_train, Y_train, theta))
        unit_acc.append(predict(X_valid, Y_valid, theta))
        unit_acc.append(predict(X_test, Y_test, theta))
        acc.append(unit_acc)
```

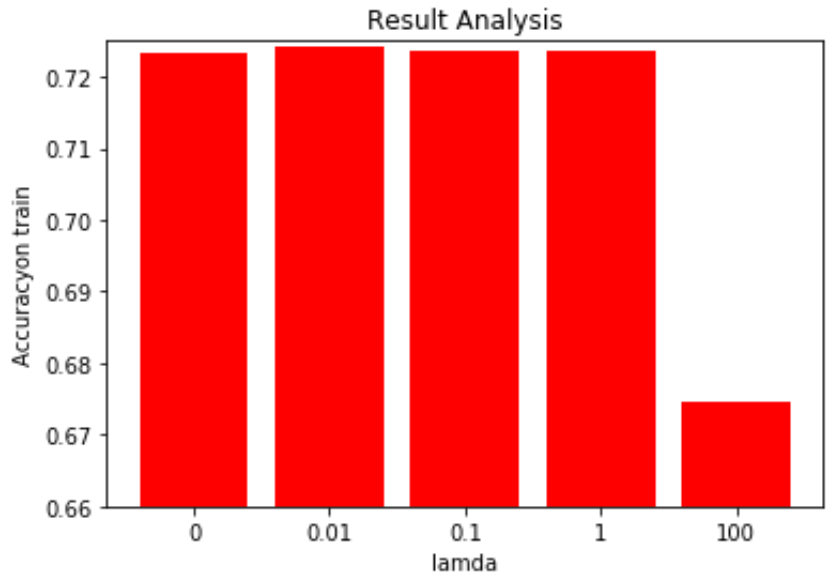
In [31]:
```
pipeline()
```

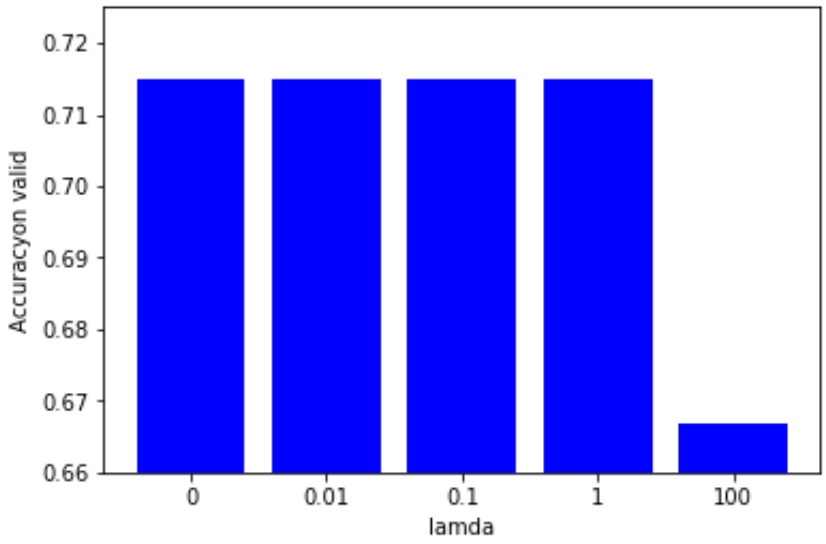
In [32]:
```
print (acc)
```

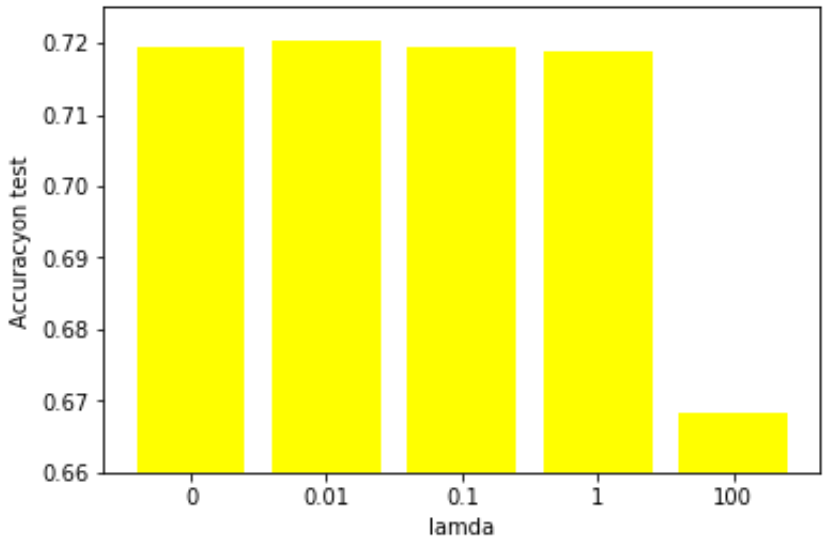
```
[[0.7234489379575183, 0.7150227981761459, 0.7193087723508941], [0.
7242289691587663, 0.714962802975762, 0.7203888155526221], [0.72350
89403576143, 0.714902807775378, 0.71936877475099], [0.723628945157
8063, 0.7150827933765299, 0.718948757950318], [0.6746069842793712,
0.6666666666666666, 0.6683667346693868]]
```

In [40]:
```
new_acc = []
for i in range(3):
    unit_acc = []
    for elem in acc:
        unit_acc.append(elem[i])
    new_acc.append(unit_acc)
print (new_acc)
lam_list = ['0', '0.01','0.1', '1', '100']
color_list = ['red','blue','yellow','green','black']
dataset = ['train','valid','test']
plt.title('Result Analysis')
for i in range(3):
    plt.bar(lam_list,new_acc[i],color=color_list[i])
    plt.xlabel('lamda')
    plt.ylabel('Accuracy'+"on "+dataset[i])
    plt.ylim(0.66,0.725)
    plt.show()
```

```
[[0.7234489379575183, 0.7242289691587663, 0.7235089403576143, 0.72
36289451578063, 0.6746069842793712], [0.7150227981761459, 0.714962
802975762, 0.714902807775378, 0.7150827933765299, 0.66666666666666
66], [0.7193087723508941, 0.7203888155526221, 0.71936877475099, 0.
718948757950318, 0.6683667346693868]]
```



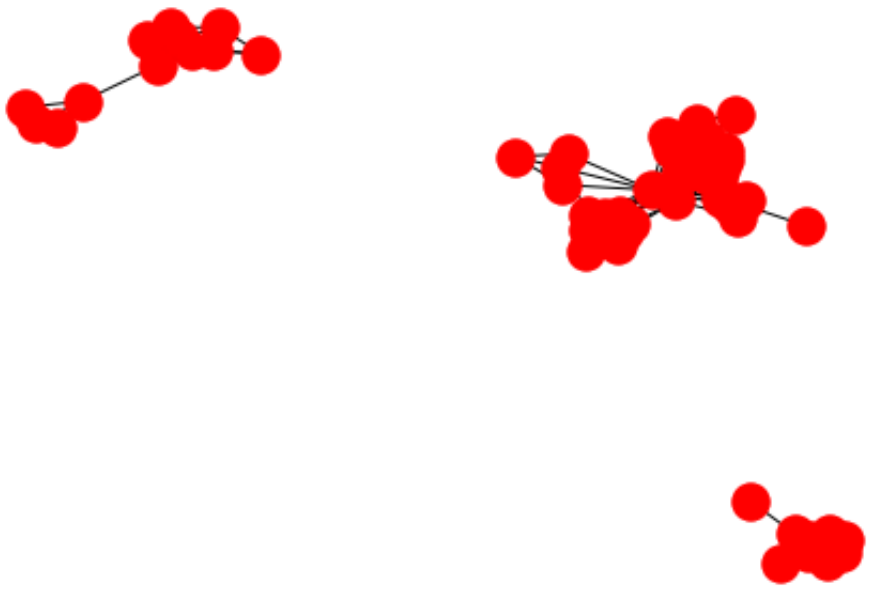

0.72
0.71
0.70
0.69
0.68
0.67
0.66
Accuracyon valid
0
0.01
0.1
1
100
lamda


0.72
0.71
0.70
0.69
0.68
0.67
0.66
Accuracyon test
0
0.01
0.1
1
100
lamda

In [1]:
```
import numpy
import urllib.request
import scipy.optimize
import random
import networkx as nx
import matplotlib.pyplot as plt
from collections import defaultdict
```

In [2]:
```
edges = set()
nodes = set()
for edge in urllib.request.urlopen("http://jmcauley.ucsd.edu/cse255
/data/facebook/egonet.txt", data = None):
  x,y = edge.split()
  x,y = int(x),int(y)
  edges.add((x,y))
  edges.add((y,x))
  nodes.add(x)
  nodes.add(y)
```

In [3]:
```
G = nx.Graph()
for e in edges:
  G.add_edge(e[0],e[1])
nx.draw(G)
plt.show()
plt.clf()
```



<Figure size 432x288 with 0 Axes>

In [4]:
```
# To find the largest connected component
index =0
for c in sorted (nx.connected_components(G), key=len, reverse=True)
:
    if index == 0:
        goal_cluster = c
        index = index + 1
        break
```

In [5]:
```
print (goal_cluster)
print (len(c))
```

```
{769, 772, 774, 798, 800, 803, 804, 805, 810, 811, 819, 823, 697,
825, 828, 830, 703, 708, 840, 713, 719, 856, 729, 861, 863, 864, 8
69, 745, 747, 876, 878, 880, 753, 882, 884, 886, 888, 889, 890, 89
3}
40
```

In [24]:
```
# Calculate the Normalized Cut cost
goal_cluster_list = list(goal_cluster)
goal_cluster_list.sort()
first_half = goal_cluster_list[:20]
second_half = goal_cluster_list[20:]
Degree_fir = sum([G.degree(v) for v in first_half])
Degree_sec = sum([G.degree(v) for v in second_half])
cut_edge = nx.cut_size(G, set(first_half), set(second_half))
Normalized_cut = (cut_edge/Degree_fir + cut_edge/Degree_sec)/2
print (Normalized_cut)
```

```
0.42240587695133147
```

In [7]:
```
print (first_half)
```

```
[697, 703, 708, 713, 719, 729, 745, 747, 753, 769, 772, 774, 798,
800, 803, 804, 805, 810, 811, 819]
```

In [8]:
```
print (second_half)
```

```
[823, 825, 828, 830, 840, 856, 861, 863, 864, 869, 876, 878, 880,
882, 884, 886, 888, 889, 890, 893]
```

In [9]:
```
def Compute_Normalized(G, l1, l2,node):
    first = [elem for elem in l1]
    second = [elem for elem in l2]
    first.append(node)
    second.remove(node)
    Degree_fir = sum([G.degree(v) for v in first])
    Degree_sec = sum([G.degree(v) for v in second])
    Numsofedge = nx.cut_size(G, set(first), set(second))
    return (Numsofedge/Degree_fir + Numsofedge/Degree_sec)/2
```

In [10]:
```
print (Compute_Normalized(G, second_half, first_half, 747))
```

```
0.4591003946362528
```

In [11]:
```
# Deploy the greedy algorithm
def FindSmallest(G, l1, l2):
    # each time we move node from second_half to first_half
    goal_node = 0
    cut_small = 100
    for elem in l2:
        temp_cut = Compute_Normalized(G, l1, l2, elem)
        if temp_cut < cut_small:
            cut_small = temp_cut
            goal_node = elem
    return str(cut_small)+"%"+str(goal_node)
```

In [12]:
```
FindSmallest(G, second_half, first_half)
```

Out[12]:
```
'0.38733196627933347%729'
```

In [13]:
```
def stop():
    Degree_fir = sum([G.degree(v) for v in first_half])
    Degree_sec = sum([G.degree(v) for v in second_half])
    cut_edge = nx.cut_size(G, set(first_half), set(second_half))
    return (cut_edge/Degree_fir + cut_edge/Degree_sec)/2
```

In [14]:
```
def GreedyProcess(G, first_half, second_half):
    res1 = FindSmallest(G, first_half, second_half)
    res2 = FindSmallest(G, second_half, first_half)
    if float(res1.split("%")[0]) < float(res2.split("%")[0]):
        first_half.append(int(res1.split("%")[1]))
        second_half.remove(int(res1.split("%")[1]))
    else:
        second_half.append(int(res2.split("%")[1]))
        first_half.remove(int(res2.split("%")[1]))
    return stop()
```

In [1]:
```
acc = 1e-3
while 1:
    acc1 = GreedyProcess(G, first_half, second_half)
    acc2 = GreedyProcess(G, first_half, second_half)
    if abs(acc1 - acc2) > acc:
        continue
    else:
        print (acc2)
        break
```

In [15]:
```
for i in range(100):
    acc1 = GreedyProcess(G, first_half, second_half)
print (acc1)
```

```
0.09817045961624274
```

In [16]:
```
print (first_half)
```

```
[697, 703, 708, 713, 719, 745, 747, 753, 769, 772, 774, 800, 803,
805, 810, 811, 819, 828, 823, 830, 840, 880, 890, 869, 856, 798]
```

In [17]:
```
print (second_half)
```

```
[825, 861, 863, 864, 876, 878, 882, 884, 886, 888, 889, 893, 729,
804]
```

In [18]:
```
# edges denotes the total set of edges in this graph
new_edge_set = set()
for i in range(len(edges)):
    unit_set = []
    if list(list(edges)[i])[0] in goal_cluster_list and list(list(e
dges)[i])[1] in goal_cluster_list:
        unit_set.append(list(list(edges)[i])[0])
        unit_set.append(list(list(edges)[i])[1])
        new_edge_set.add(tuple(unit_set))
```

In [19]:
```
def calculateModularity(G, first_half, second_half,node):
    first = [elem for elem in first_half]
    first.append(node)
    second = [elem for elem in second_half]
    second.remove(node)
    e11 = 0
    e22 = 0
    a1 = 0
    a2 = 0
    N = len(new_edge_set)
    for elem in list(new_edge_set):
        if list(elem)[0] in first:
            a1 = a1 + 1
            if list(elem)[1] in first:
                e11 = e11 + 1
        if list(elem)[0] in second:
            a2 = a2 + 1
            if list(elem)[1] in second:
                e22 = e22 + 1
    return e11/N - (a1/N)**2 + e22/N - (a2/N)**2
```

In [20]:
```
def FindLargest(G, l1, l2):
    # each time we move node from second_half to first_half
    goal_node = 0
    G_Modularity = -2
    for elem in l2:
        temp_GM = calculateModularity(G, l1, l2, elem)
        if temp_GM > G_Modularity:
            G_Modularity = temp_GM
            goal_node = elem
    return str(G_Modularity)+"%"+str(goal_node)
```

In [21]:
```
def stop_new():
    e11 = 0
    e22 = 0
    a1 = 0
    a2 = 0
    N = len(new_edge_set)
    for elem in list(new_edge_set):
        if list(elem)[0] in first_half:
            a1 = a1 + 1
            if list(elem)[1] in first_half:
                e11 = e11 + 1
        if list(elem)[0] in second_half:
            a2 = a2 + 1
            if list(elem)[1] in second_half:
                e22 = e22 + 1
    return e11/N - (a1/N)**2 + e22/N - (a2/N)**2
```

In [22]:
```
def newGreedyProcess(G, first_half, second_half):
    res1 = FindLargest(G, first_half, second_half)
    res2 = FindLargest(G, second_half, first_half)
    if float(res1.split("%")[0]) > float(res2.split("%")[0]):
        first_half.append(int(res1.split("%")[1]))
        second_half.remove(int(res1.split("%")[1]))
    else:
        second_half.append(int(res2.split("%")[1]))
        first_half.remove(int(res2.split("%")[1]))
    return stop_new()
```

In [25]:
```
for i in range(99):
    newGreedyProcess(G, first_half, second_half)
print (newGreedyProcess(G, first_half, second_half))
```

```
0.3326342975206613
```

In [26]:
```
print (first_half)
```

```
[697, 703, 708, 713, 719, 745, 747, 772, 774, 800, 803, 805, 810,
819, 828, 823, 830, 840, 880, 798]
```

In [27]:
```
print (second_half)
```

```
[825, 856, 861, 863, 864, 869, 876, 878, 882, 884, 886, 888, 889,
890, 893, 729, 804, 753, 811, 769]
```