Bosi Cheng
PID: A53271697

```
import numpy
import urllib
import scipy.optimize
import random
from math import exp
from math import log

def parseData(fname):
    for l in urllib.urlopen(fname):
        yield eval(l)

print("Reading data...")
data = list(parseData("http://jmcauley.ucsd.edu/cse190/data/beer/beer_50000.json"))
print("done")

def feature(datum):
    feat = [1, datum['review/taste'], datum['review/appearance'], datum['review/aroma'],
datum['review/palate'], datum['review/overall']]
    return feat

X = [feature(d) for d in data]
y = [d['beer/ABV'] >= 6.5 for d in data]

def inner(x,y):
    return sum([x[i]*y[i] for i in range(len(x))])

def sigmoid(x):
    return 1.0 / (1 + exp(-x))

##################################################
# Logistic regression by gradient ascent         #
##################################################

# NEGATIVE Log-likelihood
def f(theta, X, y, lam):
    loglikelihood = 0
    for i in range(len(X)):
        logit = inner(X[i], theta)
```

```
        loglikelihood -= log(1 + exp(-logit))
        if not y[i]:
            loglikelihood -= logit
    for k in range(len(theta)):
        loglikelihood -= lam * theta[k]*theta[k]
    # for debugging
    # print("ll =" + str(loglikelihood))
    return -loglikelihood

# NEGATIVE Derivative of log-likelihood
def fprime(theta, X, y, lam):
    dl = [0]*len(theta)
    for i in range(len(X)):
        logit = inner(X[i], theta)
        for k in range(len(theta)):
            dl[k] += X[i][k] * (1 - sigmoid(logit))
            if not y[i]:
                dl[k] -= X[i][k]
    for k in range(len(theta)):
        dl[k] -= lam*2*theta[k]
    return numpy.array([-x for x in dl])

cutlen=len(data)/3
print(cutlen)
ranstart=random.sample(range(cutlen),1)[0]
print(ranstart)
x_train=X[ranstart:(ranstart+cutlen)]
y_train=y[ranstart:(ranstart+cutlen)]
x_valid=X[(ranstart+cutlen):(ranstart+2*cutlen)]
y_valid=y[(ranstart+cutlen):(ranstart+2*cutlen)]
x_test=X[(ranstart+2*cutlen):]+X[:ranstart]
y_test=y[(ranstart+2*cutlen):]+y[:ranstart]

##################################################
# Train                                          #
##################################################

def train(lam):
    theta,_,_ = scipy.optimize.fmin_l_bfgs_b(f, [0]*len(X[0]), fprime, pgtol = 10, args =
(x_train, y_train, lam))
    return theta
```

```python
###################################################
# Predict                                         #
###################################################

def performance(theta):
    res_train = [inner(theta,x) for x in x_train]
    res_valid = [inner(theta,x) for x in x_valid]
    res_test = [inner(theta,x) for x in x_test]

    pred_train = [s > 0 for s in res_train]
    pred_valid = [s > 0 for s in res_valid]
    pred_test = [s > 0 for s in res_test]

    correct_train = [(a==b) for (a,b) in zip(pred_train,y_train)]
    correct_valid = [(a==b) for (a,b) in zip(pred_valid,y_valid)]
    correct_test = [(a==b) for (a,b) in zip(pred_test,y_test)]

    acc_train = sum(correct_train) * 1.0 / len(correct_train)
    acc_valid = sum(correct_valid) * 1.0 / len(correct_valid)
    acc_test = sum(correct_test) * 1.0 / len(correct_test)
    return acc_train, acc_valid, acc_test

###################################################
# Validation pipeline                             #
###################################################

lam = 1.0
theta = train(lam)
acc_train, acc_valid, acc_test = performance(theta)
print("train=" + str(acc_train))
print("valid=" + str(acc_valid))
print("test=" + str(acc_test))
```

1.

```
train=0.8301932077283092
valid=0.6176647065882636
test=0.5680345572354212
```

2. Modify the "Predict" and "Validation pipeline" parts to the following code.

```
##################################################
# Predict                                        #
##################################################

def performance(theta):
    res_train = [inner(theta,x) for x in x_train]
    res_valid = [inner(theta,x) for x in x_valid]
    res_test = [inner(theta,x) for x in x_test]

    pred_train = [s > 0 for s in res_train]
    pred_valid = [s > 0 for s in res_valid]
    pred_test = [s > 0 for s in res_test]

    correct_train = [(a==b) for (a,b) in zip(pred_train,y_train)]
    correct_valid = [(a==b) for (a,b) in zip(pred_valid,y_valid)]
    correct_test = [(a==b) for (a,b) in zip(pred_test,y_test)]

    acc_train = sum(correct_train) * 1.0 / len(correct_train)
    acc_valid = sum(correct_valid) * 1.0 / len(correct_valid)
    acc_test = sum(correct_test) * 1.0 / len(correct_test)

    scores_test = [inner(theta, x) for x in x_test]
    predictions_test = [s>0 for s in scores_test]
    positives=sum([a for (a, b) in zip(predictions_test, y_test)])
    negatives=sum([b for (a, b) in zip(predictions_test, y_test)])
    true_positives = sum([a and b for (a, b) in zip(predictions_test, y_test)])
    false_positives = sum([a and not b for (a, b) in zip(predictions_test, y_test)])
    true_negatives = sum([not a and not b for (a, b) in zip(predictions_test, y_test)])
    false_negatives = sum([not a and b for (a, b) in zip(predictions_test, y_test)])


    return acc_train, acc_valid,
acc_test,positives,negatives,true_positives,false_positives,true_negatives,false_negatives

##################################################
# Validation pipeline                            #
##################################################

lam = 1.0
theta = train(lam)
```

```python
acc_train, acc_valid,
acc_test,positives,negatives,true_positives,false_positives,true_negatives,false_negatives
= performance(theta)
print("train=" + str(acc_train))
print("valid=" + str(acc_valid))
print("test=" + str(acc_test))
print("Positives="+str(positives))
print("Negetives="+str(negatives))
print ("True Positives="+str(true_positives))
print ("True Negatives="+str(true_negatives))
print ("False Positives="+str(false_positives))
print ("False Negatives="+str(false_negatives))
```

```
train=0.8174726989079563
valid=0.6350654026161047
test=0.5941924646028318
Positives=13413
Negetives=8245
True Positives=7447
True Negatives=2457
False Positives=5966
False Negatives=798
```

3.
Multiply likelihood with 10 while y==False and logit >0

4.

```python
    return acc_valid

#####################################################
# Validation pipeline                               #
#####################################################

lams = [0,0.01,0.1,1,100]
for lam in lams:
    theta = train(lam)
    correction_rate = performance(theta)
    print(str(lam)+':'+str(correction_rate))
```

adopt the "acc_valid" to judge the best model.

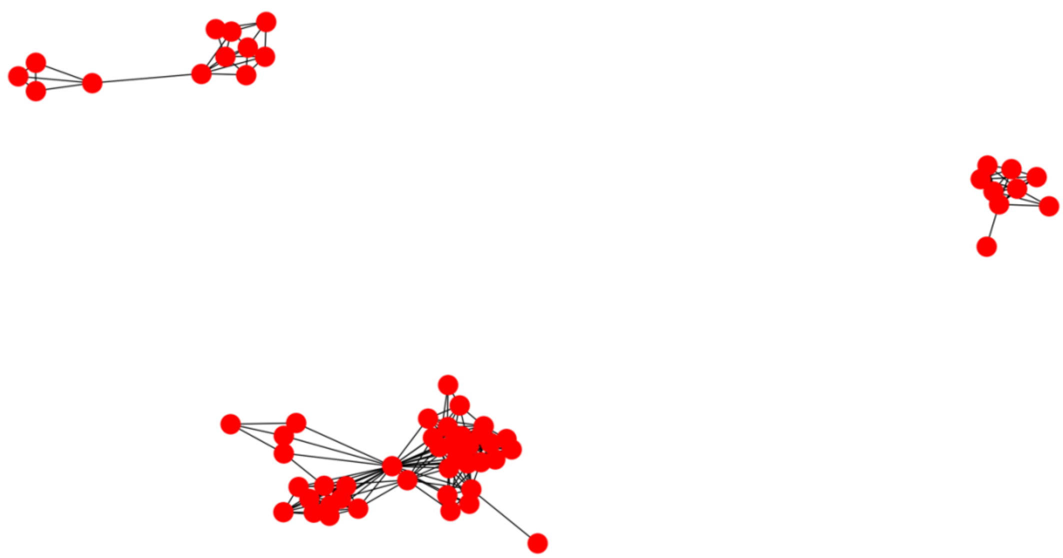
```
16666
11140
0:0.6213248529941198
0.01:0.6196447857914317
0.1:0.6194647785911437
1:0.6159246369854794
100:0.5493819752790111
```

But we can see from here that the performance is very close. However, we are picking 0 for λ.

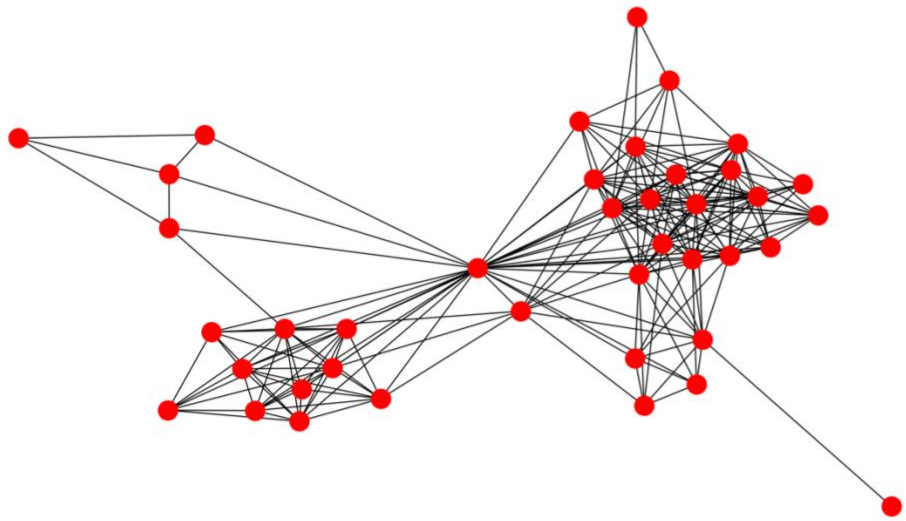
Then repeat question1:

```
train=0.7250690027601104
valid=0.8696147845913836
test=0.47546196304295657
```

5.



Obviously there are three connected components, and 40 nodes are in the largest one.



6.

```
import numpy

import urllib

import scipy.optimize

import random

from math import exp

from math import log

import networkx as nx

import matplotlib.pyplot as plt

import collections
```

```
# Karate club
G = nx.karate_club_graph()
nx.draw(G)
# plt.show()
plt.clf()

edges = set()
nodes = set()
for edge in urllib.urlopen("http://jmcauley.ucsd.edu/cse255/data/facebook/egonet.txt", 'r'):
  x,y = edge.split()
  x,y = int(x),int(y)
  edges.add((x,y))
  edges.add((y,x))
  nodes.add(x)
  nodes.add(y)

G = nx.Graph()
for e in edges:
  G.add_edge(e[0],e[1])
# nx.draw(G)
```

```
# plt.show()
plt.clf()


print("start bfs")


visited={}
graphs=[]


def BFS(node,nodes,visited):
    graph=set()
    queue=collections.deque([node])
    while queue:
        curr=queue.popleft()
        graph.add(curr)
        visited[curr]=True
        for next_node in nodes:
            if (curr,next_node) in edges and not visited[next_node]:
                queue.append(next_node)
    return graph
```

```python
for node in nodes:
    visited[node]=False
    print(node)
print("build graphs")
for node in nodes:
    if not visited[node]:
        temp=BFS(node,nodes,visited)
        print(temp)
        graphs.append(temp)
print(graphs)


largest=sorted(list(graphs[0]))
print(largest)
cluster1=largest[:(len(largest)/2)]
cluster2=largest[(len(largest)/2):]
print(cluster1)
print(cluster2)
def normalized_cut(edges,cluster1,cluster2):
    edge_count=0
    degree1,degree2=0,0
```

```
    for edge in edges:

        if edge[0] in cluster1 and edge[1] in cluster1:

            degree1+=0.5

        elif edge[0] in cluster2 and edge[1] in cluster2:

            degree2+=0.5

        elif edge[0] in cluster1 and edge[1] in cluster2:

            degree1+=1

            degree2+=1

            edge_count+=1



    normalized_cut=(edge_count/degree1+edge_count/degree2)/2

    return normalized_cut



minimum=normalized_cut(edges,cluster1,cluster2)

print(minimum)
```

```
[697, 703, 708, 713, 719, 729, 745, 747, 753, 769, 772, 774, 798, 800, 803, 804,
 805, 810, 811, 819]
[823, 825, 828, 830, 840, 856, 861, 863, 864, 869, 876, 878, 880, 882, 884, 886,
 888, 889, 890, 893]
```

0.422

7.

```
import numpy
```

```python
import urllib

import scipy.optimize

import random

from math import exp

from math import log

import networkx as nx

import matplotlib.pyplot as plt

import collections




# Karate club

G = nx.karate_club_graph()

nx.draw(G)

# plt.show()

plt.clf()


edges = set()

nodes = set()

for edge in urllib.urlopen("http://jmcauley.ucsd.edu/cse255/data/facebook/egonet.txt", 'r'):

    x,y = edge.split()
```

```
    x,y = int(x),int(y)

    edges.add((x,y))

    edges.add((y,x))

    nodes.add(x)

    nodes.add(y)



G = nx.Graph()

for e in edges:

    G.add_edge(e[0],e[1])

# nx.draw(G)

# plt.show()

plt.clf()



print("start bfs")



visited={}

graphs=[]



def BFS(node,nodes,visited):

    graph=set()
```

```
    queue=collections.deque([node])
    while queue:
        curr=queue.popleft()
        graph.add(curr)
        visited[curr]=True
        for next_node in nodes:
            if (curr,next_node) in edges and not visited[next_node]:
                queue.append(next_node)
    return graph


for node in nodes:
    visited[node]=False
    print(node)
print("build graphs")
for node in nodes:
    if not visited[node]:
        temp=BFS(node,nodes,visited)
        print(temp)
        graphs.append(temp)
print(graphs)
```

```python
largest=sorted(list(graphs[0]))
print(largest)
cluster1=largest[:(len(largest)/2)]
cluster2=largest[(len(largest)/2):]
print(cluster1)
print(cluster2)
def normalized_cut(edges,cluster1,cluster2):
    edge_count=0
    degree1,degree2=0,0
    for edge in edges:
        if edge[0] in cluster1 and edge[1] in cluster1:
            degree1+=0.5
        elif edge[0] in cluster2 and edge[1] in cluster2:
            degree2+=0.5
        elif edge[0] in cluster1 and edge[1] in cluster2:
            degree1+=1
            degree2+=1
            edge_count+=1
    print(degree1)
```

```
    print(degree2)

    print(edge_count)

    normalized_cut=(edge_count/degree1+edge_count/degree2)/2

    return normalized_cut



minimum=normalized_cut(edges,cluster1,cluster2)

print(minimum)

change1,change2=[],[]

for i in range(len(cluster1)):

    temp1=cluster1[:i]+cluster1[i:]

    temp2=cluster2+[cluster1[i]]

    print(temp1,temp2)

    temp=normalized_cut(edges,temp1,temp2)

    if temp<minimum:

        minimum=temp

        change1.append(cluster1[i])

for move_node in cluster2:

    temp1=cluster1+[cluster2[i]]

    temp2=cluster2[:i]+cluster2[i:]

    temp=normalized_cut(edges,temp1,temp2)
```

```
    if temp<minimum:

        minimum=temp

        change2.append(move_node)

for node in change1:

    cluster1.remove(node)

    cluster2.append(node)

for node in change2:

    cluster1.append(node)

    cluster2.remove(node)

print(cluster1)

print(cluster2)

print(minimum)
```

```
[703, 713, 719, 745, 747, 753, 769, 772, 774, 798, 800, 803, 804, 805, 810, 811,
 819]
[823, 825, 828, 830, 840, 856, 861, 863, 864, 869, 876, 878, 880, 882, 884, 886,
 888, 889, 890, 893, 697, 708, 729]
```

0.396