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

##################################################

# 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)

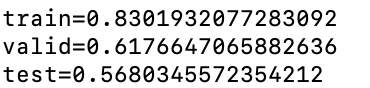
acc\_train, acc\_valid, acc\_test = performance(theta)

print("train=" + str(acc\_train))

print("valid=" + str(acc\_valid))

print("test=" + str(acc\_test))

1.



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)

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))

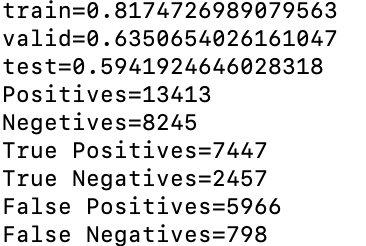
print("Negetives="+str(negatives))

print ("True Positives="+str(true\_positives))

print ("True Negatives="+str(true\_negatives))

print ("False Positives="+str(false\_positives))

print ("False Negatives="+str(false\_negatives))



3.

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

4.

return acc\_valid

##################################################

# Validation pipeline #

##################################################

lams = [0,0.01,0.1,1,100]

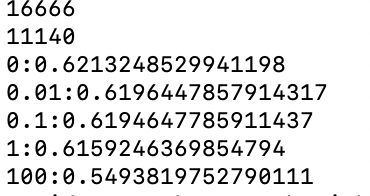
for lam in lams:

theta = train(lam)

correction\_rate = performance(theta)

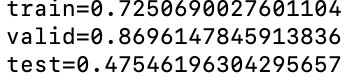
print(str(lam)+':'+str(correction\_rate))

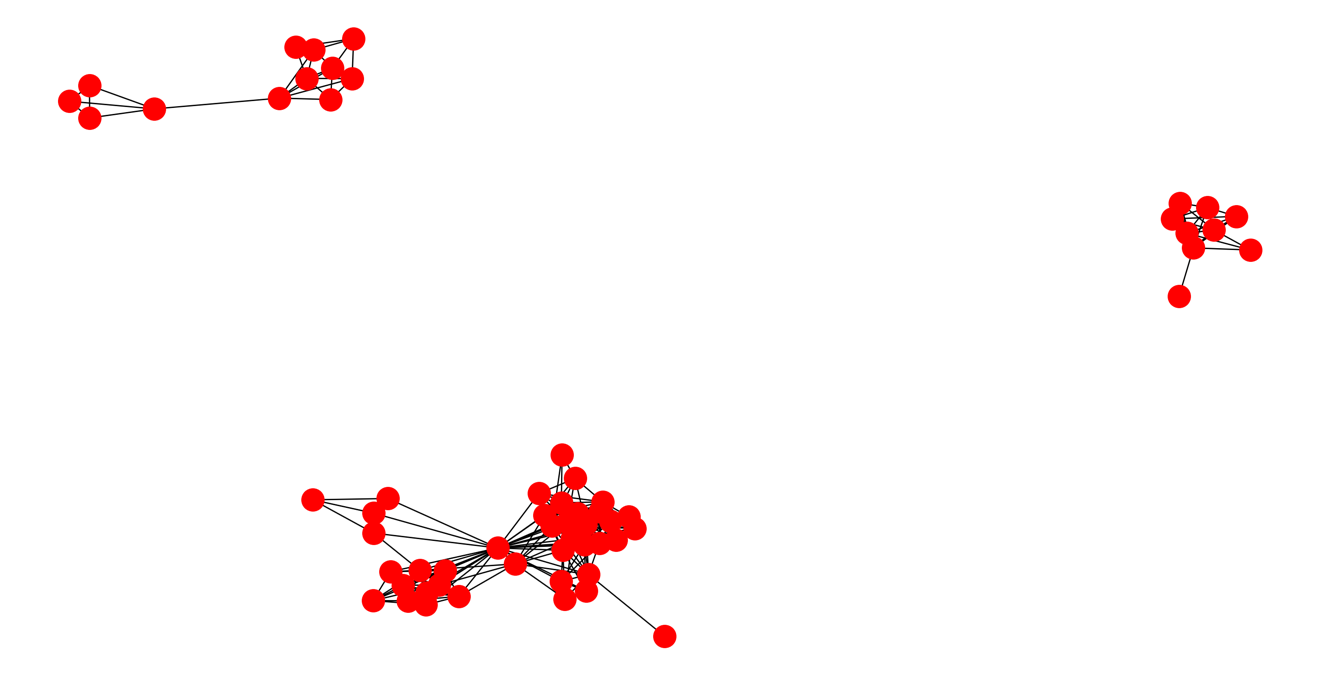
adopt the "acc\_valid" to judge the best model.



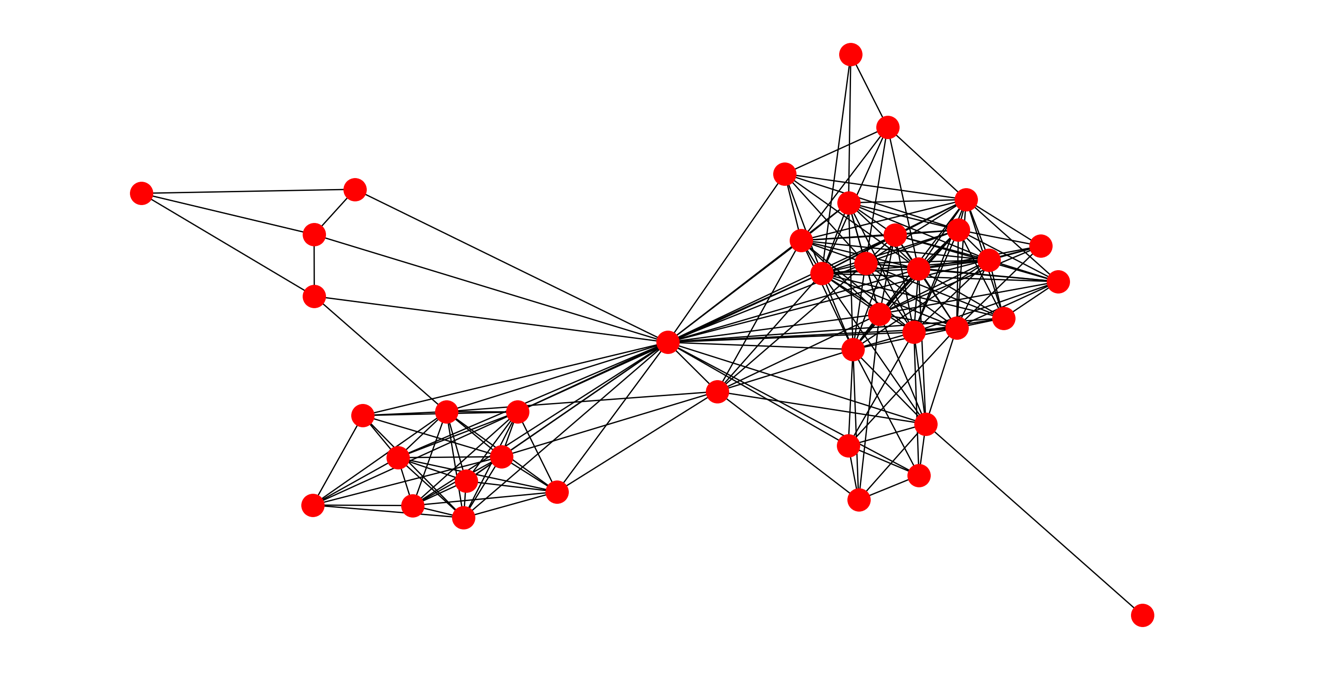
But we can see from here that the performance is very close. However, we are picking 0 for λ.

Then repeat question1:



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

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

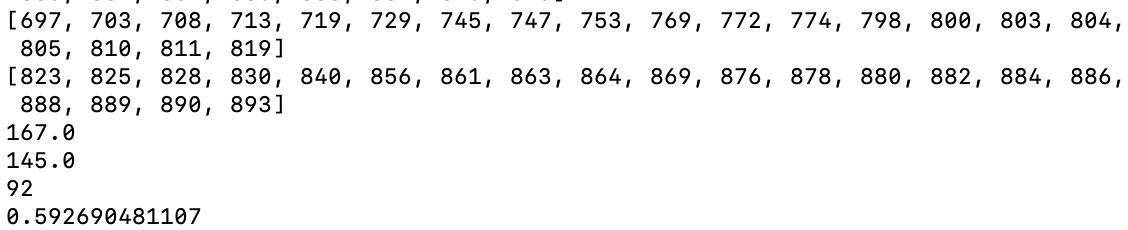
edge\_count+=1

normalized\_cut=(edge\_count/degree1+edge\_count/degree2)/2

return normalized\_cut

minimum=normalized\_cut(edges,cluster1,cluster2)

print(minimum)



0.422

7.

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

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

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:

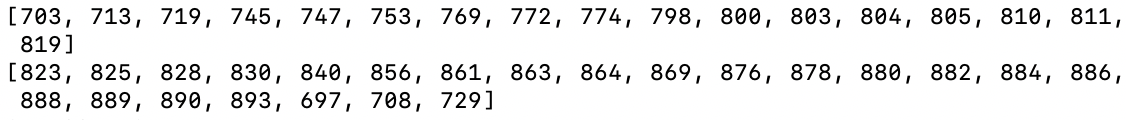
cluster1.append(node)

cluster2.remove(node)

print(cluster1)

print(cluster2)

print(minimum)



0.396