import os

import sys

import math

import numpy as np

import random

from numpy.linalg import inv

from random import shuffle

import matplotlib.pyplot as plt

alpha = 0.1

def read\_file(filename):

result = []

with open(filename, 'r') as f:

for line in f:

line = line.strip('\n')

line = line.split(',')

result.append(line)

result = np.array(result)

result = result.astype(float)

return result

def read\_file\_r(filename):

result = []

with open(filename, 'r') as f:

for line in f:

line = line.strip('\n')

result.append(line)

return result

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

# generative

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

# compute u1(class 1), u2(class 0)

def compute\_mean(data\_file, label\_file, tr\_index):

N1 = 0

N2 = 0

sum1 = 0

sum2 = 0

for i in tr\_index:

if label\_file[i] == '1':

sum1 += data\_file[i]

N1 += 1

else:

sum2 += data\_file[i]

N2 += 1

u1 = sum1 / float(N1)

u2 = sum2 / float(N2)

return u1, u2, N1, N2

# compute s1 (class 1), s2(class 0)

def compute\_s(data, data\_label, tr\_index, u1, u2):

s1 = 0

s2 = 0

for i in tr\_index:

if data\_label[i] == '1':

dif = data[i] - u1

s1 += dif[:,None]\*dif

else:

dif = data[i] - u2

s2 += dif[:,None]\*dif

return s1, s2

# compute w0 and w

def compute\_w\_ge(s, u1, u2, N1, N2):

s\_inverse = inv(s)

prod1 = np.dot(np.dot(u1, s\_inverse),u1)

prod2 = np.dot(np.dot(u2, s\_inverse), u2)

w0 = (-1) \* prod1 / 2.0 + prod2 / 2.0 + math.log(float(N1)/N2)

w = np.dot(s\_inverse, u1 - u2)

return w0, w

# compute error rate

def compute\_error\_ge(data, data\_label, index\_test, w, w0):

error = 0

for te\_i in index\_test:

a = np.dot(w, data[te\_i]) + w0

if a >= 0:

if data\_label[te\_i] != '1':

error += 1

else:

if data\_label[te\_i] != '0':

error += 1

return error / float(len(index\_test))

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

# discrimitive

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

def compute\_y (w0, phi):

a = np.dot(w0, phi)

return 1 / (1 + np.exp(-a))

# compute R

def compute\_R(y):

r = map(lambda x: x \* (1 - x), y)

return np.diagflat(r)

# compute w

def compute\_w(phi, phi\_tranpose, R, y, t, w0):

global alpha

N = len(phi[0])

p1 = inv(np.dot(alpha, np.identity(N)) + np.dot(np.dot(phi\_tranpose, R), phi))

p2 = np.dot(phi\_tranpose, (y-t)) + np.dot(alpha, w0)

w1 = w0 - np.dot(p1, p2)

return w1

def compute\_w\_sN(data, data\_label, tr\_index):

global alpha

phi = []

t = []

for i in tr\_index:

phi.append(data[i])

t.append(data\_label[i])

t = np.array(t).astype(float)

d = len(phi[0])

N = len(phi)

# add feature in the last row

phi\_p = np.ones((N, d+1))

phi\_p[:,:-1] = phi

phi = phi\_p

w0 = [0] \* (d + 1)

phi\_tranpose = np.transpose(phi)

y = compute\_y(w0, phi\_tranpose)

R = compute\_R(y)

# compute w1

w1 = compute\_w(phi, phi\_tranpose, R, y, t, w0)

#sum1 = sum(np.square(np.subtract(w1, w0))) / sum(np.square(w0))

sum1 = 1

n = 1

while n < 100 and sum1 >= 10 \*\* (-3):

w0 = w1

y = compute\_y(w0, phi\_tranpose)

R = compute\_R(y)

w1 = compute\_w(phi, phi\_tranpose, R, y, t, w0)

sum1 = sum(np.square(np.subtract(w1, w0))) / sum(np.square(w0))

n += 1

y = compute\_y(w1, phi\_tranpose)

R = compute\_R(y)

s = np.dot(np.dot(phi\_tranpose, R), phi)

s0 = inv(np.identity(d + 1) / alpha)

sN = inv(s0 + s)

return w1, sN

# compute error

def compute\_error(data, data\_label, index\_test, w\_map, sN):

error = 0

for te\_i in index\_test:

temp = data[te\_i].tolist()

temp.append(1)

d = np.array(temp)

ua = np.dot(w\_map, d)

sig\_s = np.dot(np.dot(d, sN), d)

a = ua / math.sqrt(1 + np.pi \* sig\_s / 8)

if a >= 0:

if data\_label[te\_i] != '1':

error += 1

else:

if data\_label[te\_i] != '0':

error += 1

return error / float(len(index\_test))

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

def main():

data\_file = ['A.csv', 'B.csv', 'usps.csv']

label\_file = ['labels-A.csv', 'labels-B.csv', 'labels-usps.csv']

file\_number = len(data\_file)

# x, store length of train file for plot use

x = []

# three file error list (matrix)

ge\_error\_file = []

dis\_error\_file = []

for i in range(file\_number):

# store single file error (list)

ge\_error\_list = []

dis\_error\_list = []

data = read\_file(data\_file[i])

data\_label = read\_file\_r(label\_file[i])

N = len(data)

x.append(N-int(N/3))

index\_N = range(N)

# run 30 times

for t in range(30):

# set up 1/3 data set index

te\_N = int(N/3)

index\_test = np.random.choice(N, te\_N, replace = False)

# remain 2/3 train set

index\_train = [v for j, v in enumerate(index\_N) if j not in index\_test]

tr\_N = len(index\_train)

# Record performance

splits = np.arange(0.05, 1.05, 0.05)

ge\_error\_rate = []

dis\_error\_rate = []

for s in splits:

# train set index

tr\_index = np.random.choice(index\_train, int(s\*tr\_N), replace = False)

tr\_label = []

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

# generative

# u1: mean of class 1, u2: mean of class 0

u1, u2, N1, N2 = compute\_mean(data, data\_label, tr\_index)

s1, s2 = compute\_s(data, data\_label, tr\_index, u1, u2)

s1 = s1 / float(N1)

s2 = s2 / float(N2)

# compute S

Sig = (N1 / float(N)) \* s1 + (N2 / float(N)) \* s2

#d = len(Sig)

#Sig = Sig + 10\*\*(-9)\*np.identity(d)

w0, w = compute\_w\_ge(Sig, u1, u2, N1, N2)

# test file

ge\_error = compute\_error\_ge(data, data\_label, index\_test, w, w0)

ge\_error\_rate.append(ge\_error)

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

# discrimitive

w\_map, sN = compute\_w\_sN(data, data\_label, tr\_index)

# test file

error = compute\_error(data, data\_label, index\_test, w\_map, sN)

dis\_error\_rate.append(error)

ge\_error\_list.append(ge\_error\_rate)

dis\_error\_list.append(dis\_error\_rate)

ge\_error\_file.append(ge\_error\_list)

dis\_error\_file.append(dis\_error\_list)

ge\_mean =[]

ge\_std = []

dis\_mean = []

dis\_std = []

for k in range(3):

ge\_mean.append(np.mean(ge\_error\_file[k], axis = 0))

ge\_std.append(np.std(ge\_error\_file[k], axis = 0))

dis\_mean.append(np.mean(dis\_error\_file[k], axis = 0))

dis\_std.append(np.std(dis\_error\_file[k], axis = 0))

print "start to plot:"

splits = np.arange(0.05, 1.05, 0.05)

plt.figure(1)

x1 = x[0] \* splits

plt.errorbar(x1, ge\_mean[0], ge\_std[0], marker = '^')

plt.errorbar(x1, dis\_mean[0], dis\_std[0], marker = '\*')

plt.title('A data set')

plt.xlabel('train size')

plt.ylabel('error')

plt.legend(['generative', 'discriminative'], loc = 0)

plt.savefig("A.png")

plt.clf()

plt.figure(2)

x2 = x[1] \* splits

plt.errorbar(x2, ge\_mean[1], ge\_std[1], marker = '^')

plt.errorbar(x2, dis\_mean[1], dis\_std[1], marker = '\*')

plt.title('B data set')

plt.xlabel('train size')

plt.ylabel('error')

plt.legend(['generative', 'discriminative'], loc = 0)

plt.savefig("B.png")

plt.clf()

plt.figure(3)

x3 = x[2] \* splits

plt.errorbar(x3, ge\_mean[2], ge\_std[2], marker = '^')

plt.errorbar(x3, dis\_mean[2], dis\_std[2], marker = '\*')

plt.title('USPS data set')

plt.xlabel('train size')

plt.ylabel('error')

plt.legend(['generative', 'discriminative'], loc = 0)

plt.savefig("USPS.png")

plt.clf()

main()

Task 2:

import os

import sys

import math

import numpy as np

import random

from numpy.linalg import inv

from random import shuffle

import datetime

import matplotlib.pyplot as plt

alpha = 0.1

eta = 10\*\*(-3)

def read\_file(filename):

result = []

with open(filename, 'r') as f:

for line in f:

line = line.strip('\n')

line = line.split(',')

result.append(line)

result = np.array(result)

result = result.astype(float)

return result

def read\_file\_r(filename):

result = []

with open(filename, 'r') as f:

for line in f:

line = line.strip('\n')

result.append(line)

return result

# compute y

def compute\_y (w0, phi):

a = np.dot(w0, phi)

return 1 / (1 + np.exp(-a))

# compute R

def compute\_R(y):

r = map(lambda x: x \* (1 - x), y)

return np.diagflat(r)

# compute w

def compute\_w(tr, tr\_r, w0):

global alpha

global eta

#w0 = [0] \* len(tr[0])

t = np.array(tr\_r).astype(float)

y = compute\_y(w0, np.transpose(tr))

tr\_transpose = np.transpose(tr)

p1 = np.dot(tr\_transpose, (y - t)) + np.dot(alpha, w0)

w1 = w0 - np.dot(eta, p1)

return w1

# compute sN

def compute\_sN(w, tr):

y = compute\_y(w, np.transpose(tr))

R = compute\_R(y)

l = len(y)

s = 0

global alpha

for i in range(l):

p1 = np.dot(R[i][i], tr[i])

p2 = p1[:,None]\*tr[i]

s += p2

s0 = inv(np.identity(len(tr[0])) / alpha)

sN = inv(s0 + s)

return sN

# compute error

def compute\_error(w, tr, te\_N, data, data\_label):

sN = compute\_sN(w, tr)

error = 0

for i in range(te\_N):

ua = np.dot(w, data[i])

sig\_s = np.dot(np.dot(data[i], sN), data[i])

a = ua / (math.sqrt(1 + np.pi \* sig\_s / 8))

if a >= 0:

if data\_label[i] == '0':

error += 1

else:

if data\_label[i] == '1':

error += 1

return error /float(te\_N)

def main():

data\_file = ['A.csv', 'usps.csv']

label\_file = ['labels-A.csv', 'labels-usps.csv']

file\_number = len(data\_file)

global eta

global alpha

file\_time = []

file\_error =[]

w\_file = []

for i in range(file\_number):

data = read\_file(data\_file[i])

data\_label = read\_file\_r(label\_file[i])

N = len(data)

d = len(data[0])

temp = np.ones((N, d+1))

temp[:,:-1] = data

data = temp

te\_N = int(N/3)

tr = data[te\_N:]

tr\_r = data\_label[te\_N:]

error\_list = []

# run 3 times

time\_list = []

w\_list =[]

for t in range(3):

run\_time = []

w0 = [0] \* len(tr[0])

# clock time start

a = datetime.datetime.now()

w = compute\_w(tr, tr\_r, w0)

b = datetime.datetime.now()

run\_time.append((b-a).total\_seconds())

if t == 0:

w\_list.append(w)

error\_rate = compute\_error(w, tr, te\_N, data, data\_label)

error\_list.append(error\_rate)

sum1 = 1

n = 1

while n < 6000 and sum1 >= eta:

w0 = w

a = datetime.datetime.now()

w = compute\_w(tr, tr\_r, w0)

b = datetime.datetime.now()

run\_time.append(run\_time[-1] + (b-a).total\_seconds())

if t == 0:

w\_list.append(w)

error\_rate = compute\_error(w, tr, te\_N, data, data\_label)

error\_list.append(error\_rate)

sum1 = sum(np.square(np.subtract(w, w0))) / sum(np.square(w0))

n += 1

time\_list.append(run\_time)

w\_file.append(w\_list)

time\_list = np.mean(time\_list, axis = 0)

file\_time.append(time\_list)

file\_error.append(error\_list)

"""

print "Gradient:"

for k in range(2):

print data\_file[k]

print file\_time[k]

print file\_error[k]

#print w\_file[k]

a\_newton\_time = [ 0.01537367, 0.02855167, 0.04108567, 0.05452367, 0.068228 ]

a\_newton\_error = [0.05855855855855856, 0.04804804804804805, 0.046546546546546545, 0.046546546546546545, 0.046546546546546545]

plt.figure(1)

x = file\_time[0]

y = file\_error[0]

plt.plot(x, y)

plt.plot(a\_newton\_time, a\_newton\_error)

plt.xlabel('time line')

plt.ylabel('error rate')

plt.title('gradient A data')

plt.gcf().autofmt\_xdate()

plt.savefig('task2\_A.png')

plt.clf()

plt.figure(2)

usps\_newton\_time = [ 0.02759267, 0.05277067 , 0.078221 , 0.10310333, 0.12794833 , 0.15363867,

0.17856 , 0.204489 , 0.22854267]

usps\_newton\_error = [0.04093567251461988, 0.042884990253411304, 0.03313840155945419, 0.037037037037037035, 0.03898635477582846, 0.037037037037037035, 0.03313840155945419, 0.03313840155945419, 0.03508771929824561]

x = file\_time[1]

y = file\_error[1]

plt.plot(x, y)

plt.plot(usps\_newton\_time, usps\_newton\_error)

plt.xlabel('time line')

plt.ylabel('error rate')

plt.title('gradient USPS data')

plt.gcf().autofmt\_xdate()

plt.savefig('task2\_usps.png')

plt.clf()

"""

main()

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

# Newton method

import os

import sys

import math

import numpy as np

import random

from numpy.linalg import inv

from collections import OrderedDict

from random import shuffle

import datetime

import matplotlib.pyplot as plt

alpha = 0.1

def read\_file(filename):

result = []

with open(filename, 'r') as f:

for line in f:

line = line.strip('\n')

line = line.split(',')

result.append(line)

result = np.array(result)

result = result.astype(float)

return result

def read\_file\_r(filename):

result = []

with open(filename, 'r') as f:

for line in f:

line = line.strip('\n')

result.append(line)

return result

# compute y

def compute\_y (w0, phi):

a = np.dot(w0, phi)

return 1.0 / (1 + np.exp(-a))

# compute R

def compute\_R(y):

r = map(lambda x: x \* (1 - x), y)

return np.diagflat(r)

# compute w

def compute\_w(tr, tr\_r, w0):

global alpha

t = np.array(tr\_r).astype(float)

y = compute\_y(w0, np.transpose(tr))

R = compute\_R(y)

tr\_transpose = np.transpose(tr)

p1 = inv(np.dot(alpha, np.identity(len(tr[0]))) + np.dot(np.dot(tr\_transpose, R), tr))

p2 = np.dot(tr\_transpose, (y-t)) + np.dot(alpha, w0)

w1 = w0 - np.dot(p1, p2)

return w1

# compute sN

def compute\_sN(w, tr):

y = compute\_y(w, np.transpose(tr))

R = compute\_R(y)

l = len(y)

s = 0

global alpha

s = np.dot(np.dot(np.transpose(tr), R), tr)

s0 = inv(np.identity(len(tr[0])) / alpha)

sN = inv(s0 + s)

return sN

# compute error

def compute\_error(w, tr, te\_N, data, data\_label):

# compute sN

sN = compute\_sN(w, tr)

error = 0

for i in range(te\_N):

ua = np.dot(w, data[i])

sig\_s = np.dot(np.dot(data[i], sN), data[i])

a = ua / float(math.sqrt(1 + np.pi \* sig\_s / 8))

if a >= 0:

if data\_label[i] == '0':

error += 1

else:

if data\_label[i] == '1':

error += 1

return error /float(te\_N)

def main():

data\_file = ['A.csv', 'usps.csv']

label\_file = ['labels-A.csv', 'labels-usps.csv']

file\_number = len(data\_file)

global alpha

file\_time = []

file\_error =[]

w\_file = []

for i in range(file\_number):

error\_list = []

data = read\_file(data\_file[i])

N = len(data)

d = len(data[0])

print N

print d

temp = np.ones((N, d+1))

temp[:,:-1] = data

data = temp

data\_label = read\_file\_r(label\_file[i])

N = len(data)

te\_N = int(N/3)

tr = data[te\_N:]

tr\_r = data\_label[te\_N:]

time\_list = []

w\_list = []

# run 3 times

for t in range(3):

run\_time = []

w0 = [0] \* len(tr[0])

# clock time start

a = datetime.datetime.now()

w = compute\_w(tr, tr\_r, w0)

b = datetime.datetime.now()

run\_time.append((b-a).total\_seconds())

if t == 2:

w\_list.append(w)

error\_rate = compute\_error(w, tr, te\_N, data, data\_label)

error\_list.append(error\_rate)

sum1 = 1

n = 1

while n <= 100 and sum1 >= 10 \*\* (-3):

w0 = w

a = datetime.datetime.now()

w = compute\_w(tr, tr\_r, w0)

b = datetime.datetime.now()

run\_time.append(run\_time[-1] + (b-a).total\_seconds())

if t == 2:

w\_list.append(w)

error\_rate = compute\_error(w, tr, te\_N, data, data\_label)

error\_list.append(error\_rate)

sum1 = sum(np.square(np.subtract(w, w0))) / sum(np.square(w0))

n += 1

time\_list.append(run\_time)

#print w\_list[-1]

w\_file.append(w\_list)

time\_list = np.mean(time\_list, axis = 0)

file\_time.append(time\_list)

file\_error.append(error\_list)

print "Newton"

for k in range(2):

print data\_file[k]

print file\_time[k]

print file\_error[k]

#print w\_file[k]

plt.figure(1)

#x = [datetime.datetime.now() + datetime.timedelta(seconds = i) for i in file\_time[0]]

x = file\_time[0]

y = file\_error[0]

plt.plot(x, y)

plt.xlabel('time line')

plt.ylabel('error rate')

plt.title('Newton A data')

plt.gcf().autofmt\_xdate()

plt.savefig('newton\_A.png')

plt.clf()

plt.figure(2)

#x = [datetime.datetime.now() + datetime.timedelta(seconds = i) for i in file\_time[1]]

x = file\_time[1]

y = file\_error[1]

plt.plot(x, y)

plt.xlabel('time line')

plt.ylabel('error rate')

plt.title('Newton USPS data')

plt.gcf().autofmt\_xdate()

plt.savefig('newton\_USPS.png')

plt.clf()

main()