# -\*- coding: utf-8 -\*-

"""

Created on Sat Sep 15 21:19:10 2018

@author: tchat

"""

from mpl\_toolkits.mplot3d import Axes3D

import numpy as np

from sklearn import linear\_model

import matplotlib.pyplot as plt

# Import the data

x\_feature\_train = r'D:\EE 660\HW 4\Xtrain.csv'

x\_feature\_test = r'D:\EE 660\HW 4\Xtest.csv'

y\_label\_train = r'D:\EE 660\HW 4\ytrain.csv'

y\_label\_test = r'D:\EE 660\HW 4\ytest.csv'

X\_train = np.genfromtxt(x\_feature\_train, delimiter=",")

X\_test = np.genfromtxt(x\_feature\_test, delimiter=",")

Y\_train = np.genfromtxt(y\_label\_train, delimiter=",")

Y\_test = np.genfromtxt(y\_label\_test, delimiter=",")

# Standardize the data

X\_average = np.average(X\_train, axis=0)

X\_std = np.std(X\_train, axis=0)

X\_train1 = (X\_train-X\_average)/X\_std

X\_average = np.average(X\_test, axis=0)

X\_std = np.std(X\_test, axis=0)

X\_test1 = (X\_test-X\_average)/X\_std

# Transform data into log scale

X\_train2 = np.log(X\_train+0.1)

X\_test2 = np.log(X\_test+0.1)

# Binarize the data

X\_train3 = (X\_train > 0).astype(int)

X\_test3 = (X\_test > 0).astype(int)

er = np.zeros((5))

er\_test = np.zeros((5))

mer = np.zeros((5))

mer\_test = np.zeros((5))

c = [0.01, 0.1, 1, 10, 100]

# 5 fold cross validation error rate for Standardized data

for i in range (5):

for j in range (5):

idx = np.random.permutation(len(X\_train1))

x\_train\_cv,y\_train\_cv = X\_train1[idx], Y\_train[idx]

xcv\_train, xcv\_test = x\_train\_cv[:2452,:], x\_train\_cv[2452:,:]

ycv\_train, ycv\_test = y\_train\_cv[:2452], y\_train\_cv[2452:]

lr = linear\_model.LogisticRegression(C=c[i])

lr.fit(xcv\_train, ycv\_train)

pred1 = lr.predict(xcv\_train)

er[j] = (pred1.shape[0] - np.sum(pred1==ycv\_train))/(pred1.shape[0])

pred2 = lr.predict(xcv\_test)

er\_test[j] = (pred2.shape[0] - np.sum(pred2==ycv\_test))/(pred2.shape[0])

mer[i] = np.mean(er)

mer\_test[i] = np.mean(er\_test)

print ((mer))

print ((mer\_test))

# Error rates for the Full Standardized training and test data

lr2 = linear\_model.LogisticRegression(C=100)

lr2.fit(X\_train1, Y\_train)

pred1 = lr2.predict(X\_train1)

pred2 = lr2.predict(X\_test1)

er\_train = (pred1.shape[0] - np.sum(pred1==Y\_train))/(pred1.shape[0])

er\_test = (pred2.shape[0] - np.sum(pred2==Y\_test))/(pred2.shape[0])

print(er\_train)

print(er\_test)

# 5 fold cross validation error rate for log scale data

er = np.zeros((5))

er\_test = np.zeros((5))

mer = np.zeros((5))

mer\_test = np.zeros((5))

for i in range (5):

for j in range (5):

idx = np.random.permutation(len(X\_train2))

x\_train\_cv,y\_train\_cv = X\_train2[idx], Y\_train[idx]

xcv\_train, xcv\_test = x\_train\_cv[:2452,:], x\_train\_cv[2452:,:]

ycv\_train, ycv\_test = y\_train\_cv[:2452], y\_train\_cv[2452:]

lr = linear\_model.LogisticRegression(C=c[i])

lr.fit(xcv\_train, ycv\_train)

pred1 = lr.predict(xcv\_train)

er[j] = (pred1.shape[0] - np.sum(pred1==ycv\_train))/(pred1.shape[0])

pred2 = lr.predict(xcv\_test)

er\_test[j] = (pred2.shape[0] - np.sum(pred2==ycv\_test))/(pred2.shape[0])

mer[i] = np.mean(er)

mer\_test[i] = np.mean(er\_test)

print ((mer))

print ((mer\_test))

# Error rates for the Full Log Scale training and test data

lr2 = linear\_model.LogisticRegression(C=10)

lr2.fit(X\_train2, Y\_train)

pred1 = lr2.predict(X\_train2)

pred2 = lr2.predict(X\_test2)

er\_train = (pred1.shape[0] - np.sum(pred1==Y\_train))/(pred1.shape[0])

er\_test = (pred2.shape[0] - np.sum(pred2==Y\_test))/(pred2.shape[0])

print(er\_train)

print(er\_test)

# 5 fold cross validation error rate for Binarized data

er = np.zeros((5))

er\_test = np.zeros((5))

mer = np.zeros((5))

mer\_test = np.zeros((5))

for i in range (5):

for j in range (5):

idx = np.random.permutation(len(X\_train3))

x\_train\_cv,y\_train\_cv = X\_train3[idx], Y\_train[idx]

xcv\_train, xcv\_test = x\_train\_cv[:2452,:], x\_train\_cv[2452:,:]

ycv\_train, ycv\_test = y\_train\_cv[:2452], y\_train\_cv[2452:]

lr = linear\_model.LogisticRegression(C=c[i])

lr.fit(xcv\_train, ycv\_train)

pred1 = lr.predict(xcv\_train)

er[j] = (pred1.shape[0] - np.sum(pred1==ycv\_train))/(pred1.shape[0])

pred2 = lr.predict(xcv\_test)

er\_test[j] = (pred2.shape[0] - np.sum(pred2==ycv\_test))/(pred2.shape[0])

mer[i] = np.mean(er)

mer\_test[i] = np.mean(er\_test)

print ((mer))

print ((mer\_test))

# Error rates for the Full Log Scale training and test data

lr2 = linear\_model.LogisticRegression(C=1)

lr2.fit(X\_train3, Y\_train)

pred1 = lr2.predict(X\_train3)

pred2 = lr2.predict(X\_test3)

er\_train = (pred1.shape[0] - np.sum(pred1==Y\_train))/(pred1.shape[0])

er\_test = (pred2.shape[0] - np.sum(pred2==Y\_test))/(pred2.shape[0])

print(er\_train)

print(er\_test)

# Plotting Scatter Plot for test data in the Binarized Case

x\_var = np.zeros(X\_test3.shape[0])

y\_var = np.zeros(X\_test3.shape[0])

for k in range (48):

x\_var += X\_test3[:,k]

for m in range (6):

y\_var += X\_test3[:,48+m]

LABEL\_COLOR\_MAP = {0 : 'r', 1 : 'b',}

label\_color = [LABEL\_COLOR\_MAP[i] for i in Y\_test]

plt.scatter(x\_var, y\_var, c=label\_color)

# Plot 3D Histogram for emails labeled spam

x\_var = list(x\_var)

y\_var = list(y\_var)

X = []

Y = []

for i in range (len(X\_test3)):

if (Y\_test[i]==0):

X.append(x\_var[i])

Y.append(y\_var[i])

X = np.asarray(X)

Y = np.asarray(Y)

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

hist, xedges, yedges = np.histogram2d(X, Y, bins=4, range=[[0, 4], [0, 4]])

xpos, ypos = np.meshgrid(xedges[:-1] + 0.25, yedges[:-1] + 0.25)

xpos = xpos.flatten('F')

ypos = ypos.flatten('F')

zpos = np.zeros\_like(xpos)

dx = 0.5 \* np.ones\_like(zpos)

dy = dx.copy()

dz = hist.flatten()

ax.bar3d(xpos, ypos, zpos, dx, dy, dz, color='b')

plt.show()

# Plot 3D Histogram for emails labeled not spam

x\_var = list(x\_var)

y\_var = list(y\_var)

X = []

Y = []

for i in range (len(X\_test3)):

if (Y\_test[i]==1):

X.append(x\_var[i])

Y.append(y\_var[i])

X = np.asarray(X)

Y = np.asarray(Y)

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

hist, xedges, yedges = np.histogram2d(X, Y, bins=4, range=[[0, 4], [0, 4]])

xpos, ypos = np.meshgrid(xedges[:-1] + 0.25, yedges[:-1] + 0.25)

xpos = xpos.flatten('F')

ypos = ypos.flatten('F')

zpos = np.zeros\_like(xpos)

dx = 0.5 \* np.ones\_like(zpos)

dy = dx.copy()

dz = hist.flatten()

ax.bar3d(xpos, ypos, zpos, dx, dy, dz, color='b')

plt.show()