```
# -*- 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()
```