```python
# HW2
# Samuel Freitas
# 2/10/21

import numpy
from numpy.random import multivariate_normal as N
import numpy as np
import matplotlib.pyplot as plt
import math
import matplotlib.patches as mpatches
from PIL import Image

# 2

# Find the result of minimize the loss
# of sum of the squared errors; however, add in a penalty for an L2
# penalty on the weights

# argmin{  sum (wTxi - yi)^2 + λ||w||^2.2}

# How does this change the solution to the original linear regression solution?
# What is the impact of adding in this penalty?
# Write your own implementation of logistic regression and
# implement your model on either real-world
# (see Github data sets:https://github.com/gditzler/UA-ECE-523-
Sp2018/tree/master/data),
# or synthetic data. If you simply use Scikit-
learn’s implementation of the logistic
# regression classifier, then you’ll receive zero points.
# A full 10/10 will be awarded to those that implement logistic
# regression using the optimization of cross-
entropy using stochastic gradient descent

# creating my own synthetic data
u1 = np.array([0, 2])
u2 = np.array([2, 0])
E = np.array([[1, 0], [0, 1]])
num_samples = 100

data1 = N(u1,E,size= num_samples)
data2 = N(u2,E,size= num_samples)

data_full = np.append(data1,data2,axis=0)
labels_full = np.append(np.zeros(len(data1)),np.ones(len(data2)),axis=0)
```

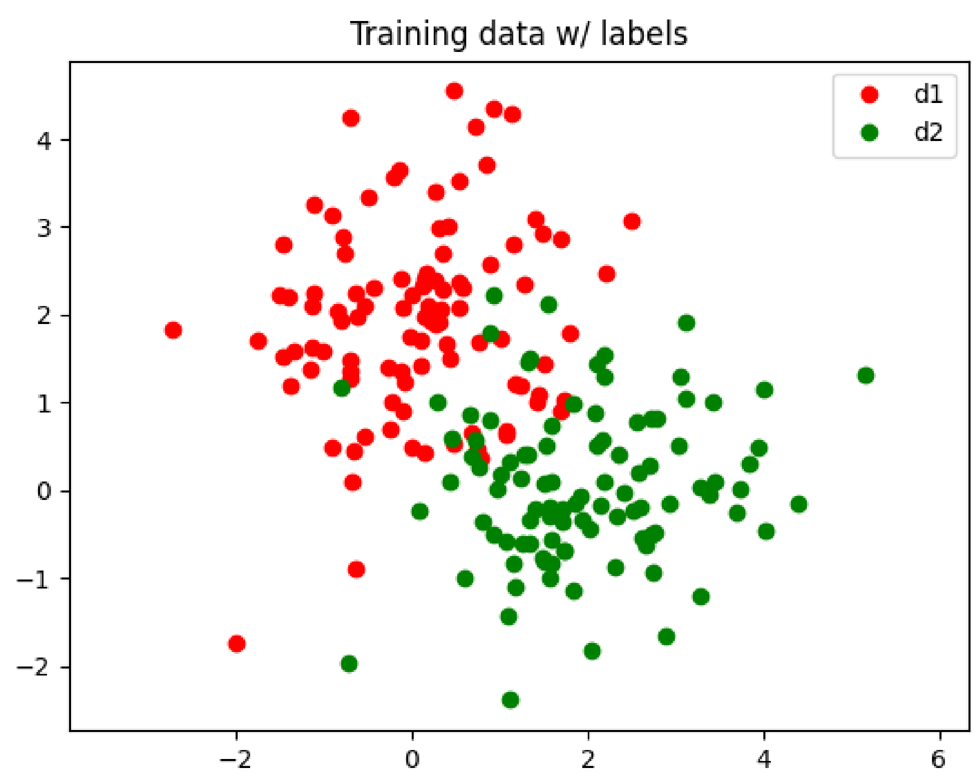
```python
def my_sigmoid(x):
    return 1/(1 + np.exp(-x))

epochs = 200
learning_rate = 0.1
X = np.asarray(data_full).T
y = np.asarray(labels_full).T
X = np.concatenate([X,np.ones([1,X.shape[-1]])],axis=0)
dims, n_data_points = X.shape

W = np.random.randn(1,dims)
loss = []

# train
for i in range(epochs):
    X_hat = np.matmul(W,X)
    y_hat = my_sigmoid(X_hat)

    cost = -np.sum(y*np.log(y_hat + 1e-5) + (1-y)*np.log(1-y_hat + 1e-5))

    if math.isnan(cost):
        cost = 0

    loss.append(cost)

    dc_dw = -np.sum((y-y_hat)*X,axis=-1)[np.newaxis,:]
    W = W - (dc_dw * learning_rate)/cost

def plot_loss(loss):
    plt.scatter(list(range(len(loss))),loss)

# predict

# for count, value in enumerate(data_full):
Z = np.asarray(data_full).T
X = np.concatenate([Z,np.ones([1,Z.shape[-1]])],axis=0)
X_hat = np.matmul(W,X)
y_hat = my_sigmoid(X_hat)


plt.ion()
# creates a figure
f1 = plt.figure(1)
# plots the data
p1 = plt.plot(data1[:,0], data1[:,1], 'o', c='r')
```

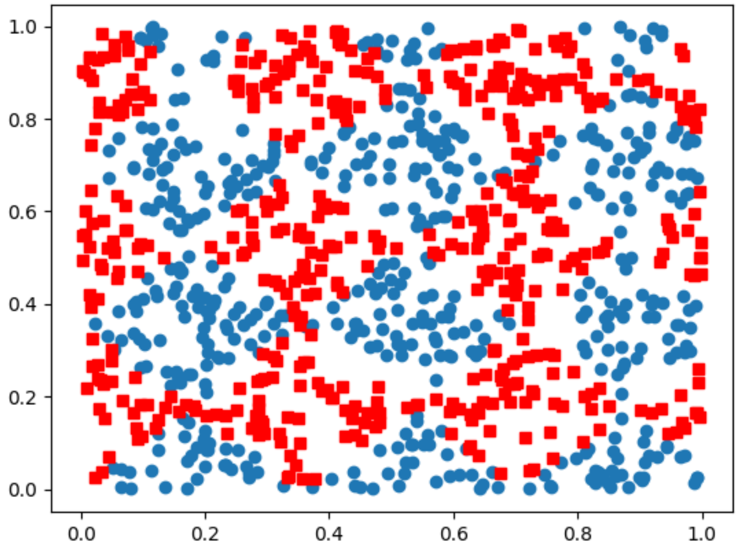
```
p2 = plt.plot(data2[:,0], data2[:,1], 'o', c='g')
# plotting options
plt.axis('equal')
plt.title('Training data w/ labels')
plt.legend(['d1','d2'])
f1.show()


plt.ion()
f2 = plt.figure(2)
# plots the data
for i in range(len(y_hat[0])):
    if np.round(y_hat[0][i]) > 0:
        plt.plot(data_full[i,0], data_full[i,1], 'o', c='g')
    else:
        plt.plot(data_full[i,0], data_full[i,1], 'o', c='r')
plt.axis('equal')
plt.title('tested w/ labels')
pop_a = mpatches.Patch(color='r', label='Population A')
pop_b = mpatches.Patch(color='g', label='Population B')
plt.title('Training data sorted with custom linear regression')
plt.legend(handles=[pop_a,pop_b])
f2.show()



# 3

# The ECE523 Lecture notes has a function for generating a checkerboard data set.

# Generate checker-board data from two classes and use any density estimate
# technique we discussed to classify newdata using
# pY|X(y|x) =ˆpX|Y(x|y)pˆY(y)/ˆpX(x)
# where pY|X(y|x) is your estimate of the posterior given you estimates of
# pX|Y(x|y) using a density estimator andˆ pY(y) using a maximum likelihood estima
tor.
# You should plotˆ pX|Y(x|y) using apseudo color plot (seehttps://goo.gl/2SDJPL).
# Note that you must modelˆ pX(x),pˆY(y), andˆ pX|Y(x|y).
# Note thatˆ pX(x)can be calculated using the Law of Total Probability.arizona.edu
4February 17, 2021

# from lecture

def gen_cb(N, a, alpha):
    """
```

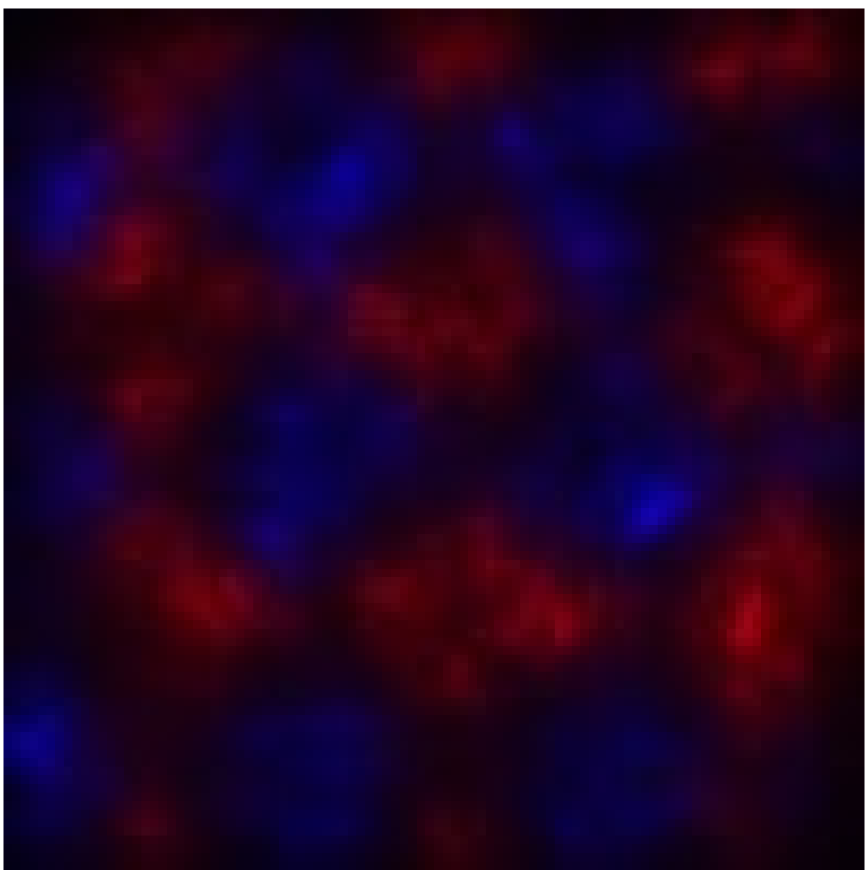
```
    N: number of points on the checkerboard
    a: width of the checker board (0<a<1)
    alpha: rotation of the checkerboard in radians
    """
    d = np.random.rand(N, 2).T  # THIS IS THE LINE OF CODE THAT IS DIFFERENT
    d_transformed = np.array([d[0]*np.cos(alpha)-d[1]*np.sin(alpha),
                              d[0]*np.sin(alpha)+d[1]*np.cos(alpha)]).T
    s = np.ceil(d_transformed[:,0]/a)+np.floor(d_transformed[:,1]/a)
    lab = 2 - (s%2)
    data = d.T
    return data, lab


points_to_gen = 1000
X,y = gen_cb(points_to_gen, 0.25, np.pi/4)

f3 = plt.figure(3)
plt.plot(X[np.where(y==1)[0],0], X[np.where(y==1)[0],1], 'o')
plt.plot(X[np.where(y==2)[0],0], X[np.where(y==2)[0],1], 's',c = 'r')

# split data

X_blue = X[np.where(y==1)]
X_red = X[np.where(y==2)]

p_b_y = len(X_blue)/points_to_gen
p_r_y = len(X_red)/points_to_gen

# pX|Y(x|y)
# P = k/n /V

k = 10
n = points_to_gen

points_between = 100

x = np.linspace(0,1,points_between)
y = np.linspace(0,1,points_between)

B_density = np.zeros((points_between,points_between))
R_density = np.zeros((points_between,points_between))

for i in range(points_between):
    for j in range(points_between):
        this_point = [x[i],y[j]]
```

```python
        B_dist = np.zeros((len(X_blue),))
        R_dist = np.zeros((len(X_red),))
        for o, val in enumerate(X_blue):
            B_dist[o] = np.linalg.norm(this_point-val)
        for o, val in enumerate(X_red):
            R_dist[o] = np.linalg.norm(this_point-val)
        B_dist = np.sort(B_dist)
        R_dist = np.sort(R_dist)

        r_blue = B_dist[k]
        r_red = R_dist[k]

        B_density[i][j] = (k/n)/(np.pi*r_blue*r_blue)
        R_density[i][j] = (k/n)/(np.pi*r_red*r_red)

scaler = max([R_density.max(),B_density.max()])
rgb_uint8 = (np.dstack((R_density/scaler,G,B_density/scaler)) * 255.999) .astype(
np.uint8)

img = Image.fromarray(rgb_uint8,'RGB')
img.save('HW2_KNN_density.png')

# now find the probability of blue or red of a random point
random_point = np.random.random(2)
print('Given random point', random_point)

idx_x =  (np.abs(x - random_point[0])).argmin()
idx_y =  (np.abs(y - random_point[1])).argmin()

P_blue = B_density[idx_x][idx_y]
P_red = R_density[idx_x][idx_y]

print('pX|Y(Blue|red)', P_red)
print('pX|Y(Red|Blue)', P_blue)

if P_blue > P_red:
    print('Most likely Blue')
else:
    print('Most likely Red')

plt.ioff()

plt.show()
```


Training data sorted with custom linear regression
Population A
Population B
4
3
2
1
0
−1
−2
−2
0
2
4
6


Training data w/ labels
d1
d2
4
3
2
1
0
−1
−2
−2
0
2
4
6





Raw density map given blue and red from above checkerboard used to predict any random value between [0,1] on the XY axis