Justin Jones

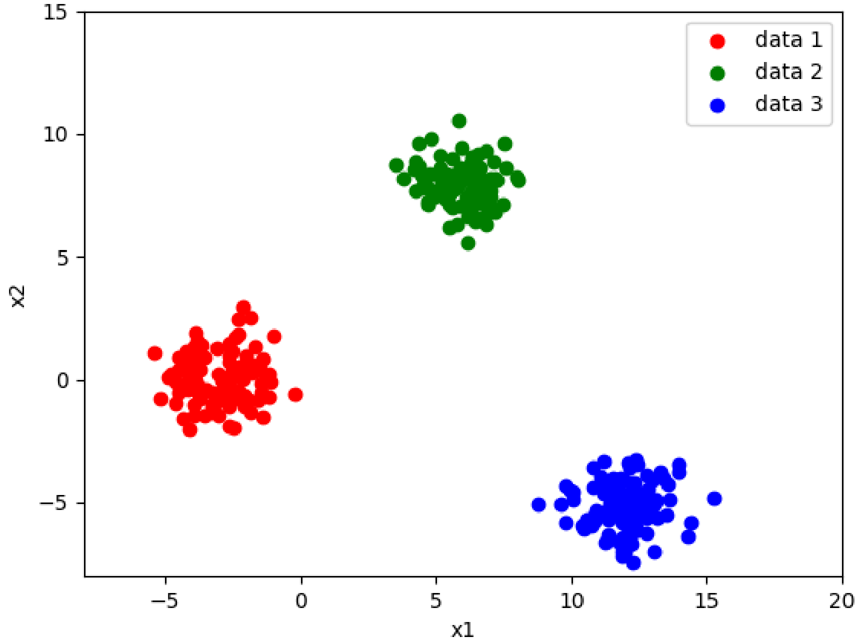
I was successful in implementing all parts of the assignment. I had to include 2 lines of code at the beginning to ignore warnings that I was getting from Numpy, however, these warnings did not affect the success of the program. When the program was tested with harder data, it decreased the accuracy, but not to the point where it became unusable. Here, I qualified easy vs hard data by how spread out and overlapping the data is, with easy data groups being close together and having no overlap with the other groups.

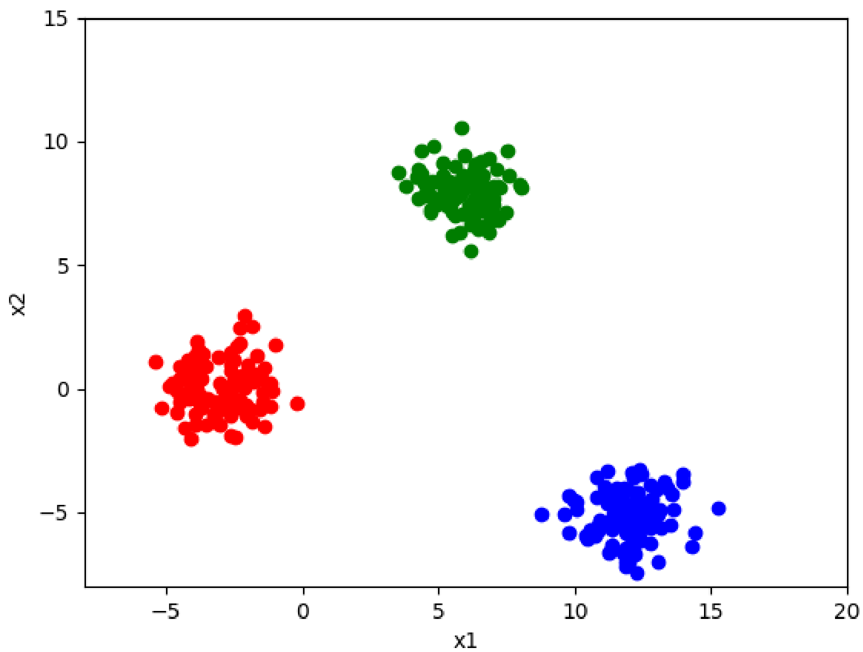
Easy Data:

```
C:\Users\justi\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\justi\PycharmProjects\pythonProject\data_groups.py
Cost Function on iteration 0: 0.6931471805599434
Cost Function on iteration 100: 0.13485449755443504
Cost Function on iteration 200: 0.0787710098762648
Cost Function on iteration 300: 0.057303511616273325
Cost Function on iteration 400: 0.04574267785529813
Cost Function on iteration 500: 0.03843257278739104
Cost Function on iteration 600: 0.03335412295685409
Cost Function on iteration 700: 0.029600338790682815
Cost Function on iteration 800: 0.026700928495447602
Cost Function on iteration 900: 0.024386677244518803
Cost Function on iteration 0: 0.6931471805599434
Cost Function on iteration 100: 0.010540837936679344
Cost Function on iteration 200: 0.0052973774016771774
Cost Function on iteration 300: 0.0035433908070937924
Cost Function on iteration 400: 0.00266433409043075
Cost Function on iteration 500: 0.002135903826023862
Cost threshold was met at iteration 535, with thetas: [0.06426533 0.37482655 0.50514956]
Cost Function on iteration 0: 0.6931471805599434
Cost Function on iteration 100: 0.006200150759436535
Cost Function on iteration 200: 0.0031336615459400655
Cost Function on iteration 300: 0.00210163576929924
Cost threshold was met at iteration 316, with thetas: [ 0.03800369  0.44582171 -0.19052934]
----- FINAL THETAS -----
Theta 1: [ 0.61317915 -1.3899572   0.0444854 ]
Theta 2: [0.06426533 0.37482655 0.50514956]
Theta 3: [ 0.03800369  0.44582171 -0.19052934]
Percent of correct evaluations: 100.0

Process finished with exit code 0
```

Justin Jones



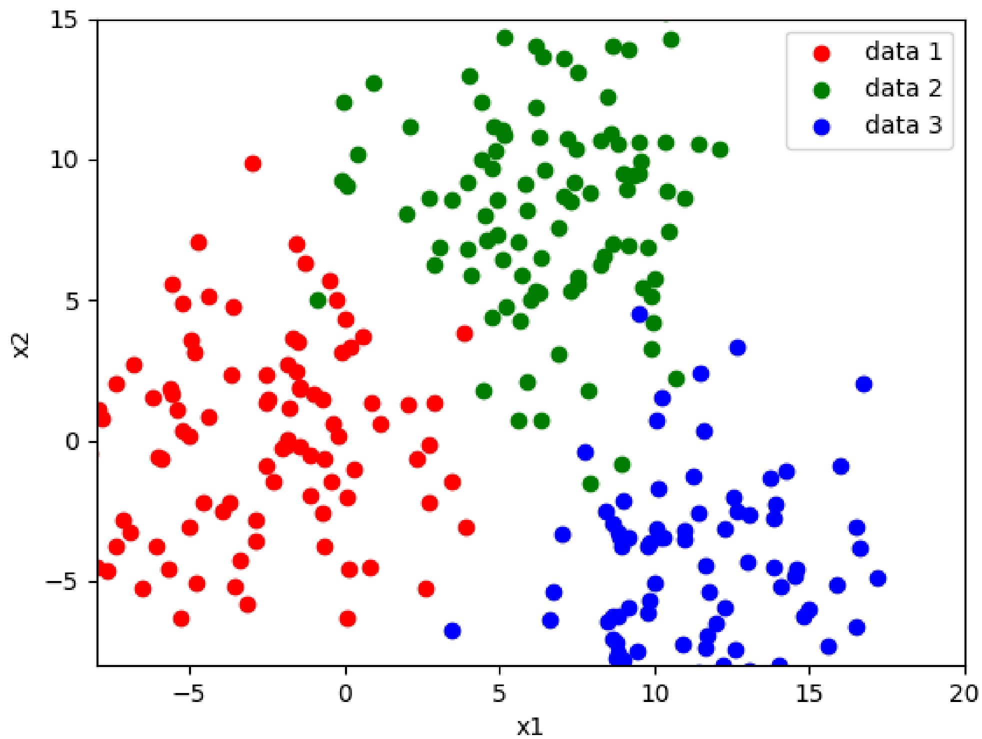


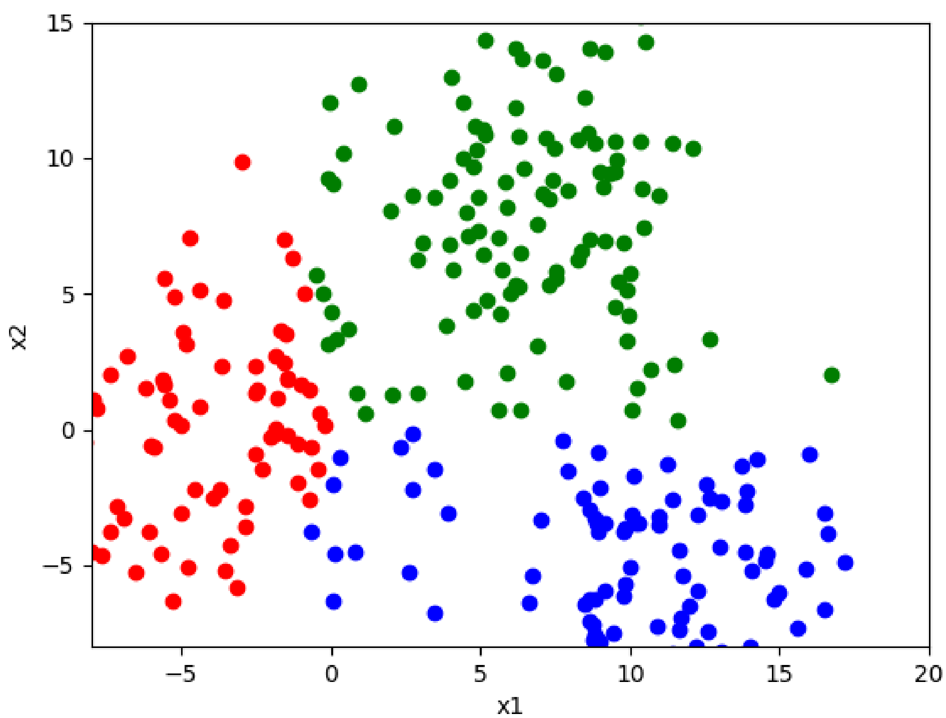
Justin Jones

Medium Data

```
C:\Users\justi\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\justi\PycharmP
Cost Function on iteration 0: 0.6931471805599434
Cost Function on iteration 100: 0.7174186070100376
Cost Function on iteration 200: 0.8540799772821641
Cost Function on iteration 300: 0.9574471800277945
Cost Function on iteration 400: 1.0416710960908222
Cost Function on iteration 500: 1.1134556734633447
Cost Function on iteration 600: 1.176455516196779
Cost Function on iteration 700: 1.2328908705941104
Cost Function on iteration 800: 1.2842170293368635
Cost Function on iteration 900: 1.3314424615345166
Cost Function on iteration 0: 0.6931471805599434
Cost Function on iteration 100: 0.018461200664075392
Cost Function on iteration 200: 0.010684089432041577
Cost Function on iteration 300: 0.007760786125234749
Cost Function on iteration 400: 0.006187827207921721
Cost Function on iteration 500: 0.005192066057484243
Cost Function on iteration 600: 0.004499382905006853
Cost Function on iteration 700: 0.003986780646580958
Cost Function on iteration 800: 0.0035904560134351862
Cost Function on iteration 900: 0.003273853651930706
Cost Function on iteration 0: 0.6931471805599434
Cost Function on iteration 100: 0.009578792414036252
Cost Function on iteration 200: 0.005155101601613322
Cost Function on iteration 300: 0.0035851052883706814
Cost Function on iteration 400: 0.002769917147100554
Cost Function on iteration 500: 0.0022672564548991744
Cost threshold was met at iteration 575, with thetas: [ 0.05554285  0.5330332  -0.23986914]
----- FINAL THETAS -----
Theta 1: [ 0.45689564 -1.44317928  0.32308605]
Theta 2: [0.10808861 0.53917555 0.55271272]
Theta 3: [ 0.05554285  0.5330332  -0.23986914]
Percent of correct evaluations: 88.66666666666667
```

Justin Jones



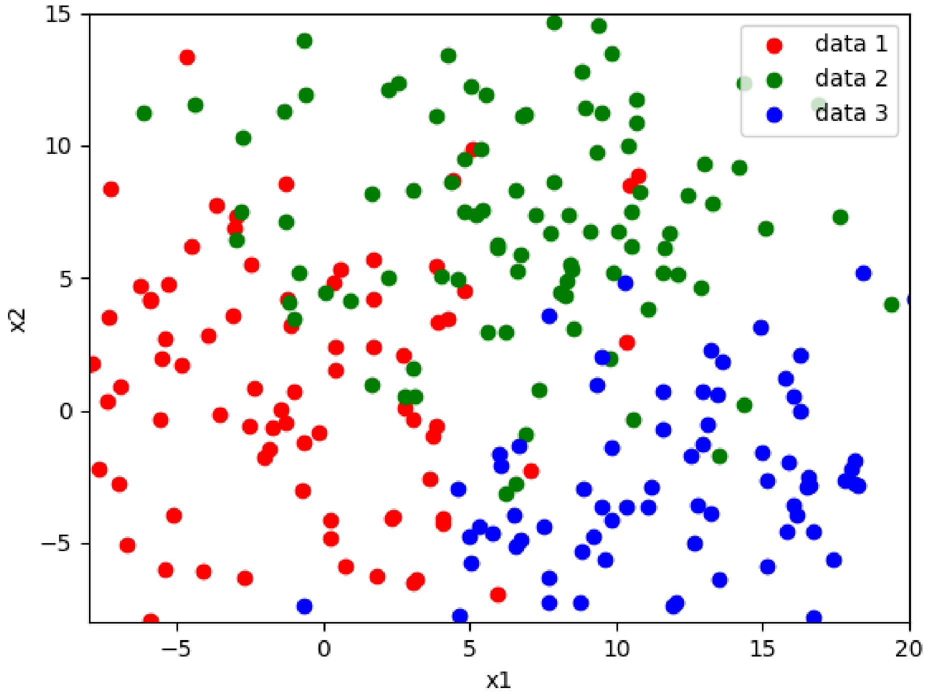


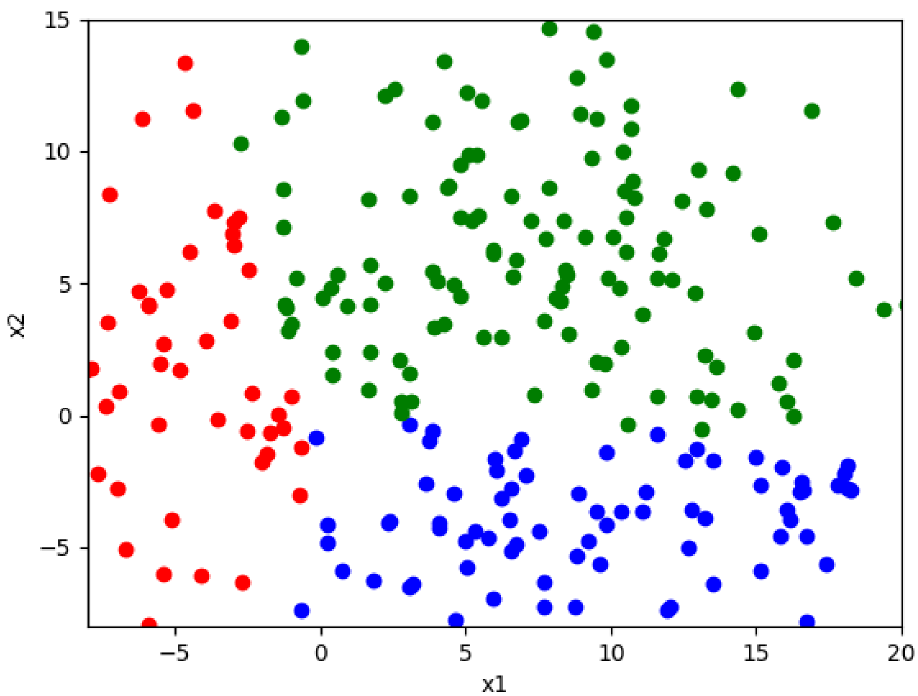
Justin Jones

Hard Data

```
Cost Function on iteration 200: 1.7853083955759168
Cost Function on iteration 300: 2.045905728042059
Cost Function on iteration 400: 2.251801016207697
Cost Function on iteration 500: 2.424525944268394
Cost Function on iteration 600: 2.574723248628655
Cost Function on iteration 700: 2.7084967426932627
Cost Function on iteration 800: 2.8296972026763605
Cost Function on iteration 900: 2.940921961333186
Cost Function on iteration 0: 0.6931471805599434
Cost Function on iteration 100: 0.10825282765451129
Cost Function on iteration 200: 0.10963066177567314
Cost Function on iteration 300: 0.11384457025335687
Cost Function on iteration 400: 0.11825878729364733
Cost Function on iteration 500: 0.12246529192321844
Cost Function on iteration 600: 0.12639276003517408
Cost Function on iteration 700: 0.1300464921808415
Cost Function on iteration 800: 0.13345045680369716
Cost Function on iteration 900: 0.13663141930827263
Cost Function on iteration 0: 0.6931471805599434
Cost Function on iteration 100: 0.016094269946741718
Cost Function on iteration 200: 0.009431683104490633
Cost Function on iteration 300: 0.006863766089456254
Cost Function on iteration 400: 0.005464860359437265
Cost Function on iteration 500: 0.004572884092686679
Cost Function on iteration 600: 0.003949666727273591
Cost Function on iteration 700: 0.0034872511550578713
Cost Function on iteration 800: 0.0031292038604340666
Cost Function on iteration 900: 0.002842995347108326
----- FINAL THETAS -----
Theta 1: [-0.01799004 -1.39539267  0.01455518]
Theta 2: [0.21783005 0.65529173 0.69445926]
Theta 3: [ 0.09235126  0.61101383 -0.37494789]
Percent of correct evaluations: 77.66666666666666
```

Justin Jones





Justin Jones

All Code:

```
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)


# Define the logistic function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))


# Define the hypothesis function
def h(x, t):
    return sigmoid(np.dot(x, t.T))


# Define the cost function for logistic regression
def cost_function(X, y, theta):
    epsilon = 1e-15  # Small constant to avoid taking log of zero
    h_theta = h(X, theta)
    return -np.mean(y * np.log(h_theta + epsilon) + (1 - y) * np.log(1 -
h_theta + epsilon))


# Implement generate three groups of training data
# group_1: ((x_mean, x_standard_deviation), (y_mean,
y_standard_deviation))
# group_2: ((x_mean, x_standard_deviation), (y_mean,
y_standard_deviation))
# group_3: ((x_mean, x_standard_deviation), (y_mean,
y_standard_deviation))
# m: the number of samples to be generated for each group
def generateData(group_1, group_2, group_3, m):
    templist = [group_1, group_2, group_3]
    datalist = []
    for i in range(len(templist)):
        tempx = np.random.normal(templist[i][0][0], templist[i][0][1], m)
        tempy = np.random.normal(templist[i][1][0], templist[i][1][1], m)
        datalist.append(np.stack((tempx, tempy), axis=1))
```

```
    data_1 = datalist[0]
    data_2 = datalist[1]
    data_3 = datalist[2]
    return data_1, data_2, data_3


data1, data2, data3 = generateData(((-3, 3), (0, 4)), ((6, 3), (8, 4)),
((12, 3), (-5, 4)), 100)


# display the three groups of training data in the coordinate
# with each group of data having different color
def dispData(d1, d2, d3):
    plt.scatter(d1[:, 0], d1[:, 1], label='data 1', color='red')
    plt.scatter(d2[:, 0], d2[:, 1], label='data 2', color='green')
    plt.scatter(d3[:, 0], d3[:, 1], label='data 3', color='blue')
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.xlim(-8, 20)
    plt.ylim(-8, 15)
    plt.legend()
    plt.show()
    return


dispData(data1, data2, data3)


# modified gradient descent from linear regression to use the sigmoid
function
def gradientDescent(X, y, theta, learning_rate, threshold=0.002,
conv_threshold=0.00015):
    for i in range(1000):
        for j in range(len(X)):
            temp = h(X[j], theta)
            if temp < 0.5:
                y[j] = 0
            else:
                y[j] = 1
        hyp = h(X, theta)
```

```
        theta1_mult = 0
        for j in range(len(X)):
            theta1_mult += hyp[j] - y[j]
        new_theta1 = theta[0] - learning_rate * (1/len(X)) * theta1_mult

        theta2_mult = 0
        for j in range(len(X)):
            theta2_mult += (hyp[j] - y[j]) * X[j, 1]
        new_theta2 = theta[1] - learning_rate * (1 / len(X)) * theta2_mult

        theta3_mult = 0
        for j in range(len(X)):
            theta3_mult += (hyp[j] - y[j]) * X[j, 2]
        new_theta3 = theta[2] - learning_rate * (1 / len(X)) * theta3_mult

        if i % 100 == 0:
            print(f'Cost Function on iteration {i}: {cost_function(X, y, theta)}')
        if cost_function(X, y, theta) < threshold:
            print(f'Cost threshold was met at iteration {i}, with thetas: {theta}')
            break

        if ((new_theta1 - theta[0]) ** 2 < conv_threshold and (new_theta2 - theta[1]) ** 2 < conv_threshold and
                (new_theta3 - theta[1]) ** 2 < conv_threshold):
            theta[0] = new_theta1
            theta[1] = new_theta2
            theta[2] = new_theta3
            print(f'Theta converged at iteration {i} with Thetas: {theta} and Cost = {cost_function(X, y, theta)}')
            break

        theta[0] = new_theta1
        theta[1] = new_theta2
        theta[2] = new_theta3

    return theta
```

Justin Jones

```
# [part 3]: Implement "one-vs-all" algorithm to separate the data
def oneVsAll(data1, data2, data3, learning_rate=0.01):
    # learn theta_1, theta_2, theta_3 using gradient descent
    theta_1 = np.array([0, 0, 0], dtype=np.float64)
    theta_2 = np.array([0, 0, 0], dtype=np.float64)
    theta_3 = np.array([0, 0, 0], dtype=np.float64)

    # Does gradientdescent for each data group to train thetas
    X1 = np.c_[np.ones(len(data1)), data1]
    y1 = np.zeros((100, 1))
    theta_1 = gradientDescent(X1, y1, theta_1, learning_rate)

    X2 = np.c_[np.ones(len(data2)), data2]
    y2 = np.zeros((100, 1))
    theta_2 = gradientDescent(X2, y2, theta_2, learning_rate)

    X3 = np.c_[np.ones(len(data3)), data3]
    y3 = np.zeros((100, 1))
    theta_3 = gradientDescent(X3, y3, theta_3, learning_rate)
    # Make sure your code print out the cost function every 10 or 50 or
100 iterations
    # Your code should stop learning once the cost is very small
    print("----- FINAL THETAS -----")
    print("Theta 1:", theta_1)
    print("Theta 2:", theta_2)
    print("Theta 3:", theta_3)

    return theta_1, theta_2, theta_3


t1, t2, t3 = oneVsAll(data1, data2, data3)

# implement some code to merge the data and theta together
#   [data_1, data_2, data_3]
#   [theta_1, theta_2, theta_3]
data = np.vstack((data1, data2, data3))
theta = np.vstack((t1, t2, t3))
```

```
# data is just the original generated different groups of data merging together
# data = [[5,2], [1,3], [4,3], [12,0], ...] 300 data, with each row a sample
# data_new should be = [[5, 2, 1], [1, 3, 2], [4, 3, 2], [12, 0, 0], ...] 300 data
# each row has one more column than "data" indicating the class assignment
def classifyData(D, thetas):
    pred = np.zeros((300, 1))
    testD = np.c_[np.ones(len(D)), D]
    for i in range(300):
        ident = -1
        test0 = h(testD[i], thetas[0])
        test1 = h(testD[i], thetas[1])
        test2 = h(testD[i], thetas[2])
        if test0 > test1 and test0 > test2:
            ident = 0
        elif test1 > test2 and test1 > test2:
            ident = 1
        elif test2 > test0 and test2 > test1:
            ident = 2
        pred[i] = ident
    return np.hstack((D, pred))


new_data = classifyData(data, theta)


# Evaluate what the percentage of accurate prediction
def evaluate(data, data_new):
    num_correct = 0
    for i in range(100):
        if data_new[i][2] == 0:
            num_correct += 1
    for i in range(100):
        if data_new[i+100][2] == 1:
            num_correct += 1
    for i in range(100):
        if data_new[i+200][2] == 2:
            num_correct += 1
```

```
    return num_correct/300


print(f'Percent of correct evaluations: {100 * evaluate(data, new_data)}')


# Display the "data_new" in the figure with different color indicating
class assigned coordinates
# compare the two figures: true labels vs prediction
# Turn in document including these two figure comparison  on different
generated data (3)
def plotNewData(new_data):
    for i in range(300):
        if new_data[i][2] == 0:
            plt.scatter(new_data[i, 0], new_data[i, 1], color='red')
        elif new_data[i][2] == 1:
            plt.scatter(new_data[i, 0], new_data[i, 1], color='green')
        elif new_data[i][2] == 2:
            plt.scatter(new_data[i, 0], new_data[i, 1], color='blue')
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.xlim(-8, 20)
    plt.ylim(-8, 15)
    plt.show()
    return


plotNewData(new_data)
```