Question 3: Logistic Regression

In [1]:
```
#Import all the required libraries
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (10.0, 6.0)
from mpl_toolkits.mplot3d import Axes3D
import time
import math
from sklearn.linear_model import LogisticRegression
import random
```

Load the data

In [2]:
```
# Load the data
# Perform important operations on the data
x = pd.read_csv("class0-input.csv")
y = pd.read_csv("class1-input.csv")
label = pd.read_csv("labels.csv")
```

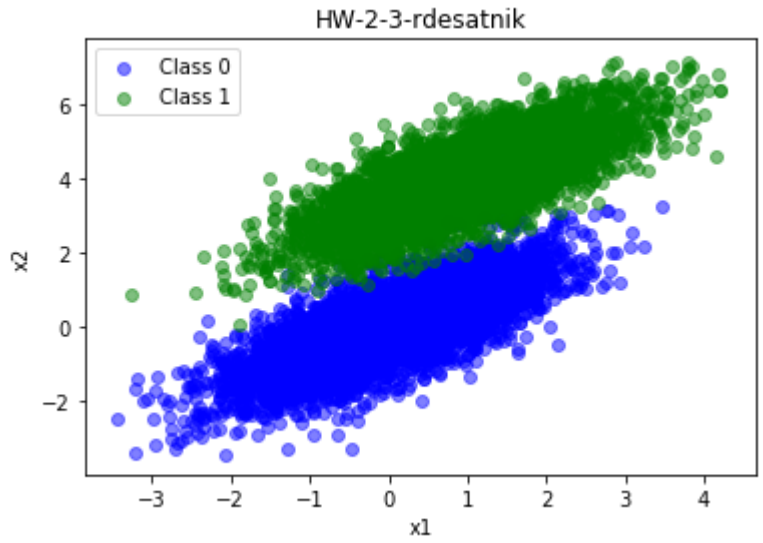
Check the shape

In [3]:
```
# Shape of X
# Shape of Y
print(x.shape)
print(y.shape)
print(label.shape)
```

```
(5000, 2)
(5000, 2)
(10000, 1)
```

Visualize the data

In [4]:
```
# Use different colors for each class
# Use plt.scatter
# Dont forget to add axes titles, graph title, legend
plt.scatter(x.x1,x.x2, c="blue", alpha=0.5, label="Class 0")
plt.scatter(y.x1,y.x2, c="green", alpha=0.5, label="Class 1")
plt.title("HW-2-3-rdesatnik")
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.show()
```



Define the required functions

In [5]:
```
import csv
x0list= []
x1list= []
with open("class0-input.csv") as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        if row[0] == 'x1':
            skip = 1
        else:
            x0 = [float(row[0]),float(row[1])]
            x0list.append(x0)

with open("class1-input.csv") as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        if row[0] == 'x1':
            skip = 1
        else:
            x1 = [float(row[0]),float(row[1])]
            x1list.append(x1)

ylistall= []
with open("labels.csv") as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        if row[0] == 'label':
            skip = 1
        else:
            labeling = float(row[0])
            ylistall.append(labeling)

xlistall = x0list+x1list
X = np.array(xlistall)
Y = np.array(ylistall)
```

In [6]:
```
# Pass in the required arguments
# Implement the sigmoid function
def sigmoid(z):
    return 1/(1+np.exp(-z))
```

In [7]:
```
def Pred(weight,X):
    z = np.array(weight[0]+weight[1]*np.array(X[:,0])+weight[2]*np.array(X[:,1]))
    return sigmoid(z)
```

In [8]:
```
# Pass in the required arguments
# The function should return the gradients
#def grad(w, X, Y):
def calculate_gradients(weight, X, Y):
        pred = Pred(weight,X)
        gradient = [0]*3
        gradient[0] = -1 * sum(Y*(1-pred) - (1-Y)*pred)
        gradient[1] = -1 * sum(Y*(1-pred)*X[:,0] - (1-Y)*pred*X[:,0])
        gradient[2] = -1 * sum(Y*(1-pred)*X[:,1] - (1-Y)*pred*X[:,1])
        return gradient
```

In [9]:
```
# Update the weights using gradients calculated using above function and learning
# The function should return the updated weights to be used in the next step
def update_weights(current_grads, prev_weights, learning_rate, iteratnum=100):
    UWlist = []
    iterat = 0
    while True:
        prev_weights = current_grads
        weight0 = prev_weights[0] - learning_rate*calculate_gradients(prev_weigh
        weight1 = prev_weights[1] - learning_rate*calculate_gradients(prev_weigh
        weight2 = prev_weights[2] - learning_rate*calculate_gradients(prev_weigh
        current_grads = [weight0, weight1, weight2]
        UWlist.append(current_grads)
        if (current_grads[0]-prev_weights[0])**2 + (current_grads[1]-prev_weights
            return current_grads, UWlist

        if iterat>iteratnum:
            return current_grads, UWlist
        iterat = iterat + 1
```

In [10]:
```
randomlist = []
for i in range(0,3):
    n = random.randint(1,20)
    randomlist.append(n)
print(randomlist)
```

```
[5, 2, 16]
```

In [11]:
```
########################################
# Use the implemented functions in the main function
# 'main' fucntion should return weights after all the iterations
# Dont forget to divide by the number of datapoints wherever necessary!
# Initialize the intial weights randomly
randomlist = []
for i in range(0,3):
    n = random.randint(1,20)
    randomlist.append(n)
inputweights = randomlist

def main(X, Y, weights, learning_rate = 0.00005, num_steps = 50000):
    inputweights = weights
    UW, UWlist = update_weights(inputweights,inputweights,learning_rate,iteratnu
    return UW, UWlist

UW, UWlist = main(X=X, Y=Y, weights=inputweights, learning_rate = 0.00005, num_s
#Final Weights
print(UW)
#Weights after all iterations
print(UWlist)
```

```
[-8.693602086533597, -3.3949969533915074, 5.286215827483597]
[[3.8493885555237695, 0.923304721271198, 15.903015810840147], [3.699463962636
1954, 0.8466960880271833, 15.80587302799126], [3.550238862911615, 0.770182242
9284878, 15.708574317229306], [3.401725989298316, 0.69377141649796, 15.611122
588725358], [3.253938075043208, 0.6174718481250673, 15.513521029402625], [3.1
0688777709621794, 0.5412917132370165, 15.415773134283269], [2.960587583213247,
0.4652390715408213, 15.317882733722799], [2.815049841384938, 0.38932184848395
257, 15.21985401419319], [2.670286702638305, 0.3135478570833345, 15.121691531
384005], [2.5263101923689386, 0.23792486059024795, 15.023400215656151], [2.38
3132276312092, 0.16246066935388287, 14.924985371060004], [2.2407649542778163,
0.08716325933834225, 14.826452669964576], [2.099220362514594, 0.0120408961323
28324, 14.727808145693107], [1.9585108702340375, -0.06289775294889396, 14.629
05818546079], [1.8186491560080096, -0.13764353536037918, 14.530209525523949],
[1.6796482517523372, -0.21218677241134237, 14.43126924993414], [1.54152154637
06469, -0.28651726933478866, 14.332244793720442], [1.404282748167404, -0.3606
2437535643177, 14.233143950713288], [1.2679458140378241, -0.4344970803429981
7, 14.133974885595375], [1.132524861668897, -0.5081241228533958, 14.034746149
188546], [0.9980340848230431, -0.5814940793742396, 13.93546669556374], [0.864
4876883535013, -0.6545954110370926, 13.83614589939178], [0.7318998494215136,
```

In [12]:
```
from pandas import DataFrame
df = DataFrame(dict(x=X[:,0], y=X[:,1], label=Y))
```

In [13]:
```
# Pass in the required arguments (final weights and input)
# The function should return the predictions obtained using sigmoid function.
final_weights = UW
inputvalue = X
print("These are my final weights: " + str(final_weights))
singleinputvalue = X[0]
def predict(final_weights, singleinputvalue):
    z = np.array(final_weights[0]+final_weights[1]*np.array(singleinputvalue[0])
    return sigmoid(z)
```

```
These are my final weights: [-8.693602086533597, -3.3949969533915074, 5.2862158
27483597]
```

In [14]:
```
# Use the final weights to perform prediction using predict funtion
# Convert the predictions to '0' or '1'
# Calculate the accuracy using predictions and labels
print(final_weights)
label = Y
size_of_dataset, = label.shape
misclassifiedlist = []
correctlist = []
for value in range(size_of_dataset):
    singleinputvalue = X[value]
    sigmoid_pred = predict(final_weights, singleinputvalue)
    if sigmoid_pred < 0.5:
        prediction = 0
        if prediction == label[value]:
            correctlist.append(1)
        else:
            misclassifiedlist.append(singleinputvalue.tolist())

    elif sigmoid_pred >= 0.5:
        prediction = 1
        if prediction == label[value]:
            correctlist.append(1)
        else:
            misclassifiedlist.append(singleinputvalue.tolist())
    else:
        print("Definitly an error")


Accuracy = (len(correctlist)/size_of_dataset)*100
print("Accuracy: " + str(Accuracy) + "percent")
```

```
[-8.693602086533597, -3.3949969533915074, 5.286215827483597]
Accuracy: 99.45percent
```

Visualize the misclassification

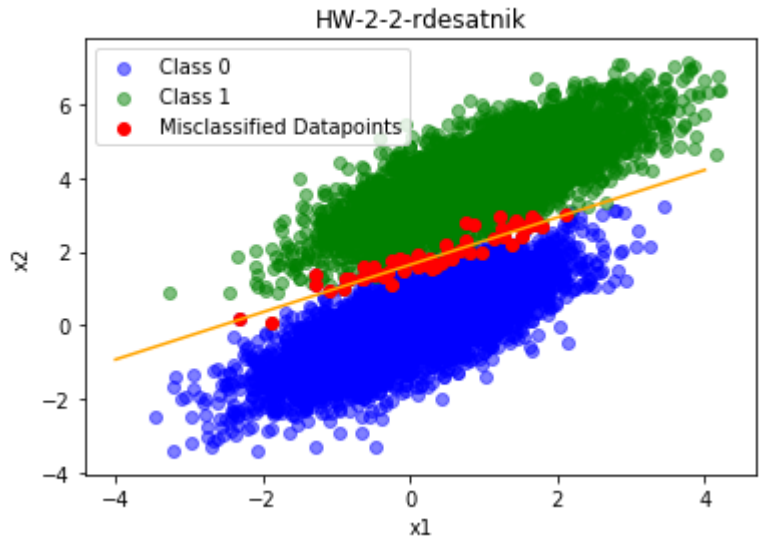
In [15]:
```
# Use different colors for class 0, class 1 and misclassified datapoints
# Use plt.scatter
# Dont forget to add axes titles, graph title, legend

#print(misclassifiedlist)
missed = np.array(misclassifiedlist)

def misclassline(formula, x_range):
    x = np.array(x_range)
    y = formula(x)
    plt.plot(x,y, c="orange")

def my_formula(x):
    return(-final_weights[0]-final_weights[1]*x)/final_weights[2]

misclassline(my_formula, range(-4,5))
#Misclassified Points
plt.scatter(x.x1,x.x2, c="blue", alpha=0.5, label="Class 0")
plt.scatter(y.x1,y.x2, c="green", alpha=0.5, label="Class 1")
plt.scatter(missed[:,0],missed[:, 1], c="red", label="Misclassified Datapoints")
plt.legend()
plt.title("HW-2-2-rdesatnik")
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()
```



Compare the results with sklearn's Logistic Regression

In [16]:

```
#Importing data for sklearn
import csv
x10list= []
x20list= []
with open("class0-input.csv") as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        if row[0] == 'x1':
            skip = 1
        else:
            x10 = float(row[0])
            x20 = float(row[1])
            x10list.append(x10)
            x20list.append(x20)
x11list= []
x21list= []
with open("class1-input.csv") as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        if row[0] == 'x1':
            skip = 1
        else:
            x11 = float(row[0])
            x21 = float(row[1])
            x11list.append(x11)
            x21list.append(x21)
y0list= []
y1list= []
with open("labels.csv") as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        if row[0] == 'label':
            skip = 1
        elif float(row[0]) == 0:
            labeling = float(row[0])
            y0list.append(labeling)
        else:
            labeling = float(row[0])
            y1list.append(labeling)


print(len(y0list))
print(len(y1list))
x10 = np.array(x10list)
x20 = np.array(x20list)
x11 = np.array(x11list)
x21 = np.array(x21list)
ylabel2 = np.array(y0list)
ylabel1 = np.array(y1list)


col_names = ['x1','x2','label']
conx1 = []
conx2 = []
conlabel = []
conx1 = x10.tolist() + x11.tolist()
```

```
conx2 = x20.tolist() + x21.tolist()
conlabel = ylabel2.tolist() + ylabel1.tolist()
datadic = {'x1': conx1,'x2': conx2,'label': conlabel}
dataset = pd.DataFrame(data = datadic)
dataset.head()
```

```
5000
5000
```

Out[16]:

	x1	x2	label
0	-0.201517	-0.683358	0.0
1	0.374519	-0.828082	0.0
2	-0.161895	-1.247107	0.0
3	0.037711	-0.047303	0.0
4	-0.260479	1.770204	0.0

In [17]:
```
# import sklearn and necessary libraries
# Print the accuracy obtained by sklearn and your model
# import sklearn and necessary libraries
# Print the accuracy obtained by sklearn and your model
#LR
features = ['x1','x2']
x = dataset[features]
y = dataset.label
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=0).fit(x,y)
SKlearn_accuracy = (clf.score(x,y))*100
My_accuracy = Accuracy
print("sklearn accuracy is: " +str(SKlearn_accuracy)+ " My logistic regression: "
```

```
sklearn accuracy is: 99.5 My logistic regression: 99.45

C:\Users\rdesa\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

In [18]:
```
#The sklearn accuracy is higher then my algorithms accuracy
#because sklearn runs more iterations at a more effienct learning rate which incr
```

In [ ]:
```
```