Lab 5

Ido Israeli (ID - 212432439)
Jonathan Derhy (ID - 315856377)

In [ ]:
```
import numpy as np
import pandas as pd
import math
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
```

Question 1

Only loading the Xtrain and Ytrain files as we'll load the rest when they're needed.

In [ ]:
```
Xtrain = np.loadtxt('.\External\Data\Xtrain.txt', delimiter=',', skiprows=0)
Ytrain = np.loadtxt('.\External\Classifications\Ytrain.txt', delimiter=',', skip
```

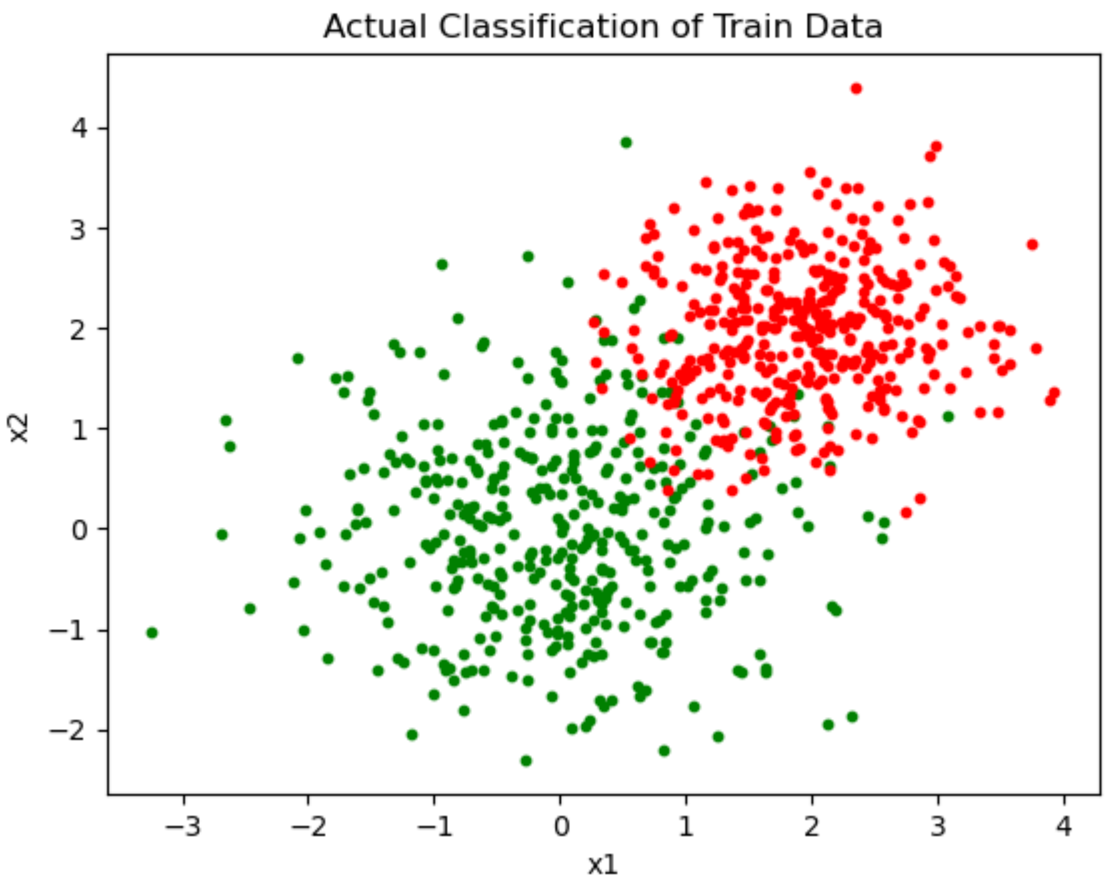
In [ ]:
```
Xtest = np.loadtxt('.\External\Data\Xtest.txt', delimiter=',', skiprows=0)
Ytest = np.loadtxt('.\External\Classifications\Ytest.txt', delimiter=',', skipro
```

Question 2

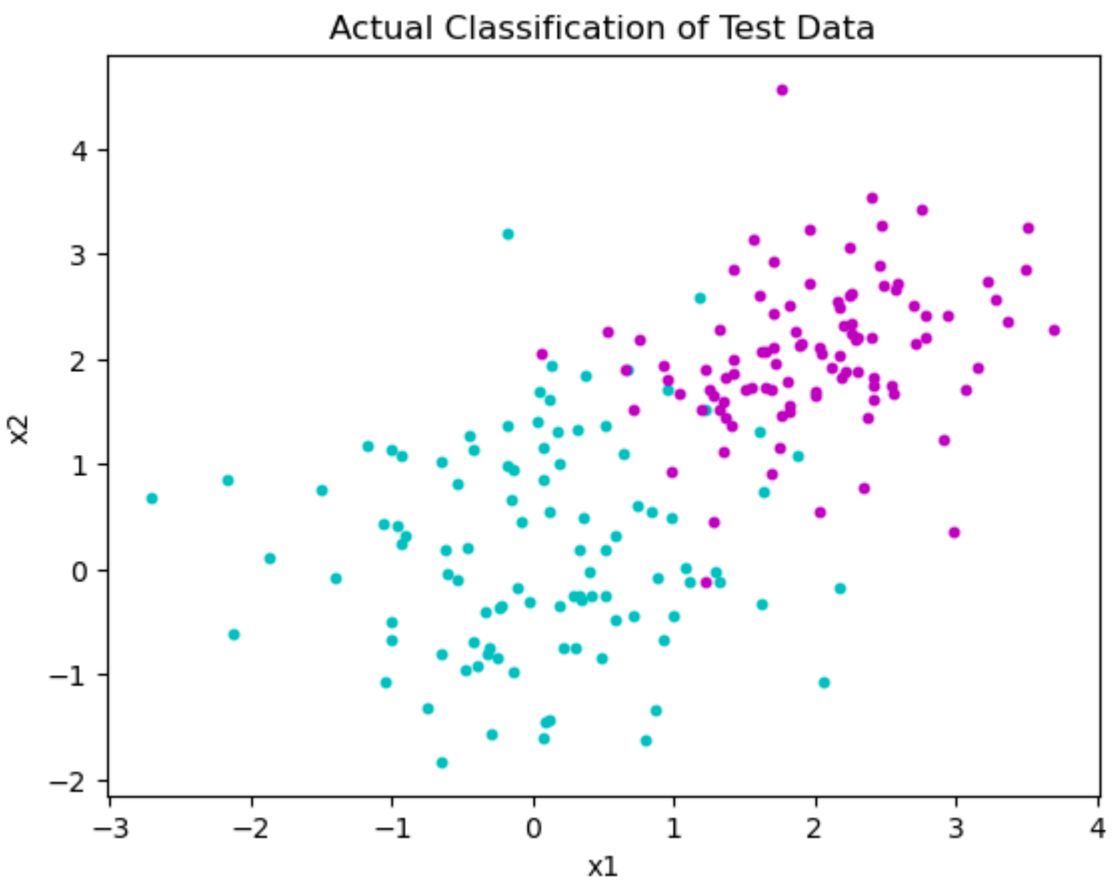
In [ ]:
```
def drawGroups(X, Y, color_0, color_1, Title):
    c = []
    for y in Y:
        c.append(color_0 if y == 0 else color_1)

    df = pd.DataFrame({'x': X[:,0],
                       'y': X[:,1],
                       'Color': c})
    testValues = df.groupby('Color')
    for name, group in testValues:
        plt.scatter(group.x, group.y, 10, color=name, label=name)
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title(Title)
    return
```

In [ ]:
```
drawGroups(Xtrain, Ytrain, 'g', 'r', "Actual Classification of Train Data")
```



In [ ]:
```
drawGroups(Xtest, Ytest, 'c', 'm', "Actual Classification of Test Data")
```



Question 3

```
In [ ]: W = np.loadtxt('.\External\Coefficients\Coefficients.txt', delimiter=',', skipro
        print('w0 = '+ str(W[0]))
        print('W = '+ str(W[1:]))
```

```
w0 = -0.85
W = [1.32 1.24]
```

Question 4

```
In [ ]: def innerProduct(W, x):
            """
            This Function
            """
            return np.matmul(W[1:], x) + W[0]
```

```
In [ ]: def classify(W, X):
            """
            This Function
            """
            Ypredicted = []
            for x in X:
                Ypredicted.append(0.0 if np.sign(innerProduct(W, x)) == -1 else 1.0)
            return Ypredicted
```

```
In [ ]: def drawClassification(W, X, Y, color_0, color_1, Title):
            c = []
            for y in Y:
                c.append(color_0 if y == 0 else color_1)

            df = pd.DataFrame({'x': X[:,0],
                               'y': X[:,1],
                               'Color': c})
            testValues = df.groupby('Color')
            for name, group in testValues:
                plt.scatter(group.x, group.y, 8, color=name, label=name)
            expression = -(W[0]+W[1]*X)/W[2]
            plt.plot(X, expression, 'k')
            plt.xlabel('x1')
            plt.ylabel('x2')
            plt.title(Title)
            return
```

Question 5

```
In [ ]: def accuracyNaively(actualY, predictedY):
            """
            This Function
            """
            count = 0
            for pred_y, act_y in zip(predictedY, actualY):
                if pred_y == act_y:
                    count+=1
            return (count*100)/len(actualY)
```

```
In [ ]: def accuracy(actualY, predictedY):
            """
            This Function
            """
            return 100*(predictedY == actualY).mean()
```

```
In [ ]: predictedTrainY = classify(W, Xtrain)
```

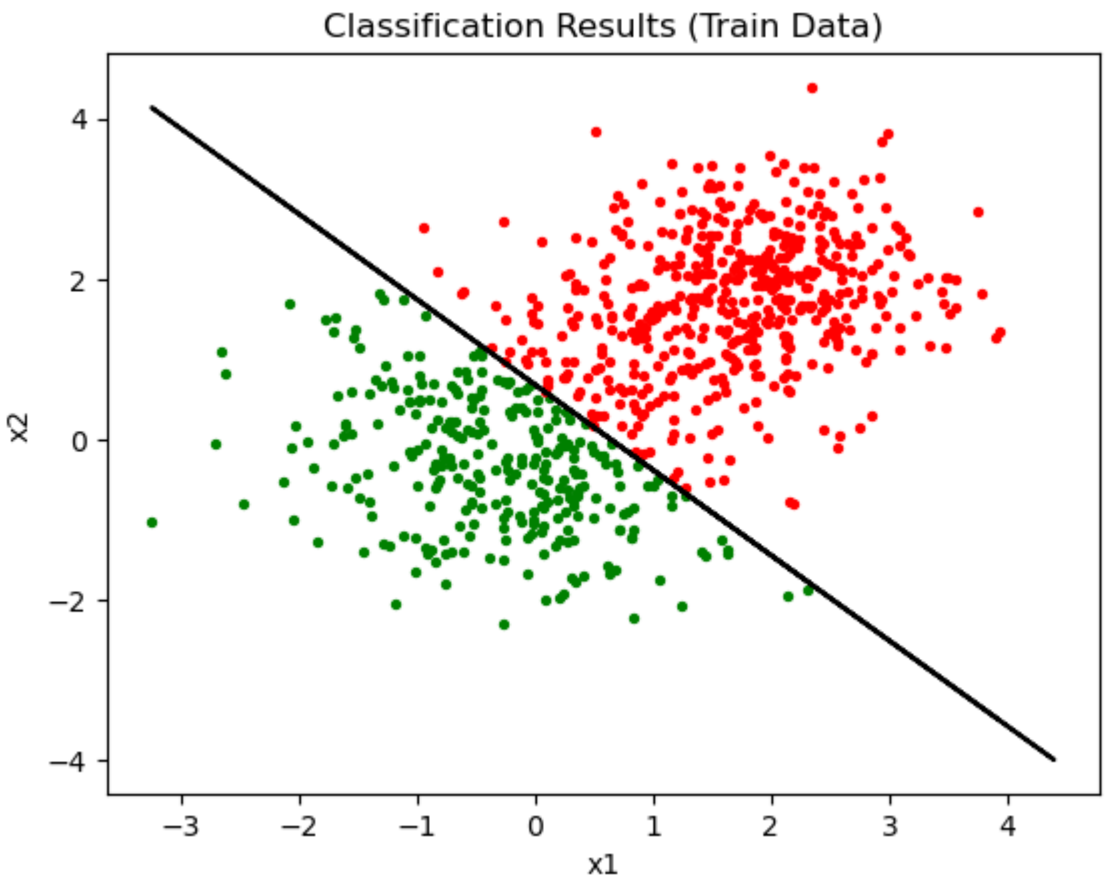
```
In [ ]: trainAccuracy1 = accuracy(Ytrain, predictedTrainY)
        print(str(trainAccuracy1)+'%')

        85.375%
```

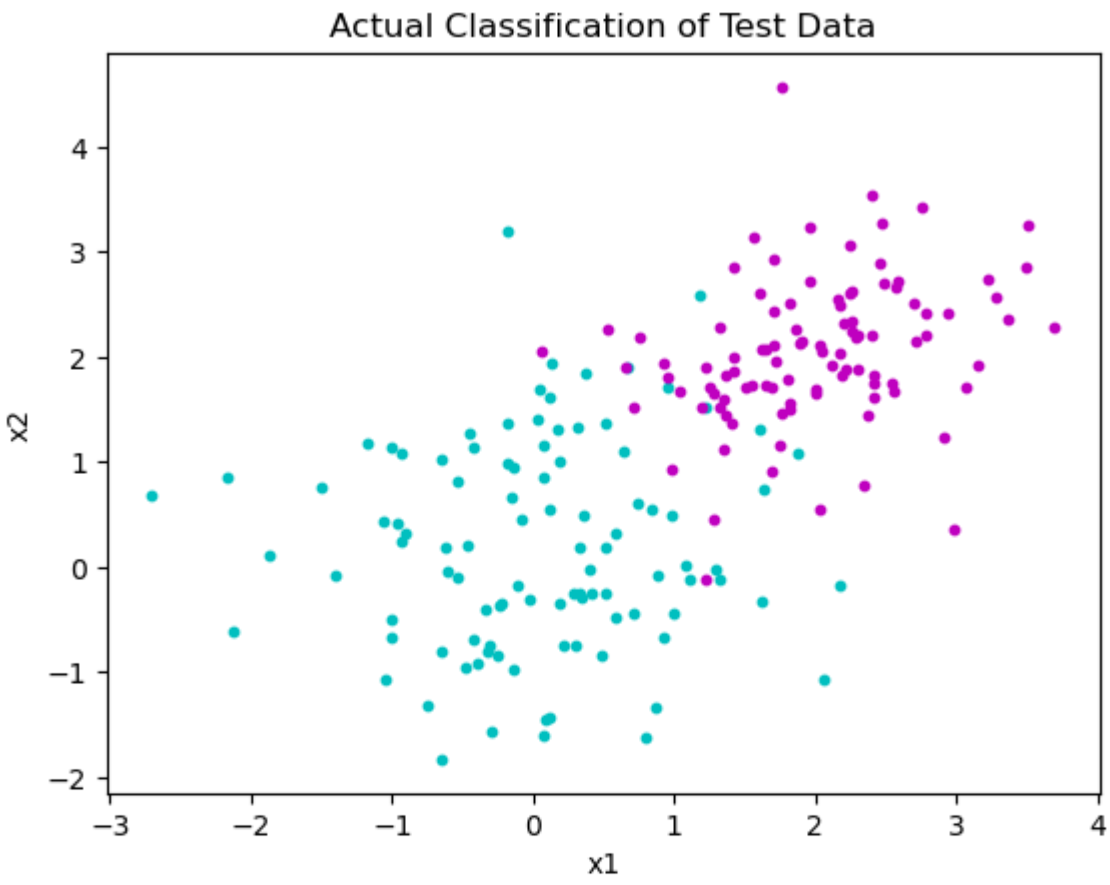
```
In [ ]: trainAccuracy2 = accuracyNaively(Ytrain, predictedTrainY)
        print(trainAccuracy1 - trainAccuracy2)

        0.0
```

```
In [ ]: drawClassification(W, Xtrain, predictedTrainY, 'g', 'r', "Classification Results
```



```
In [ ]: drawGroups(Xtest, Ytest, 'c', 'm', "Actual Classification of Test Data")
```



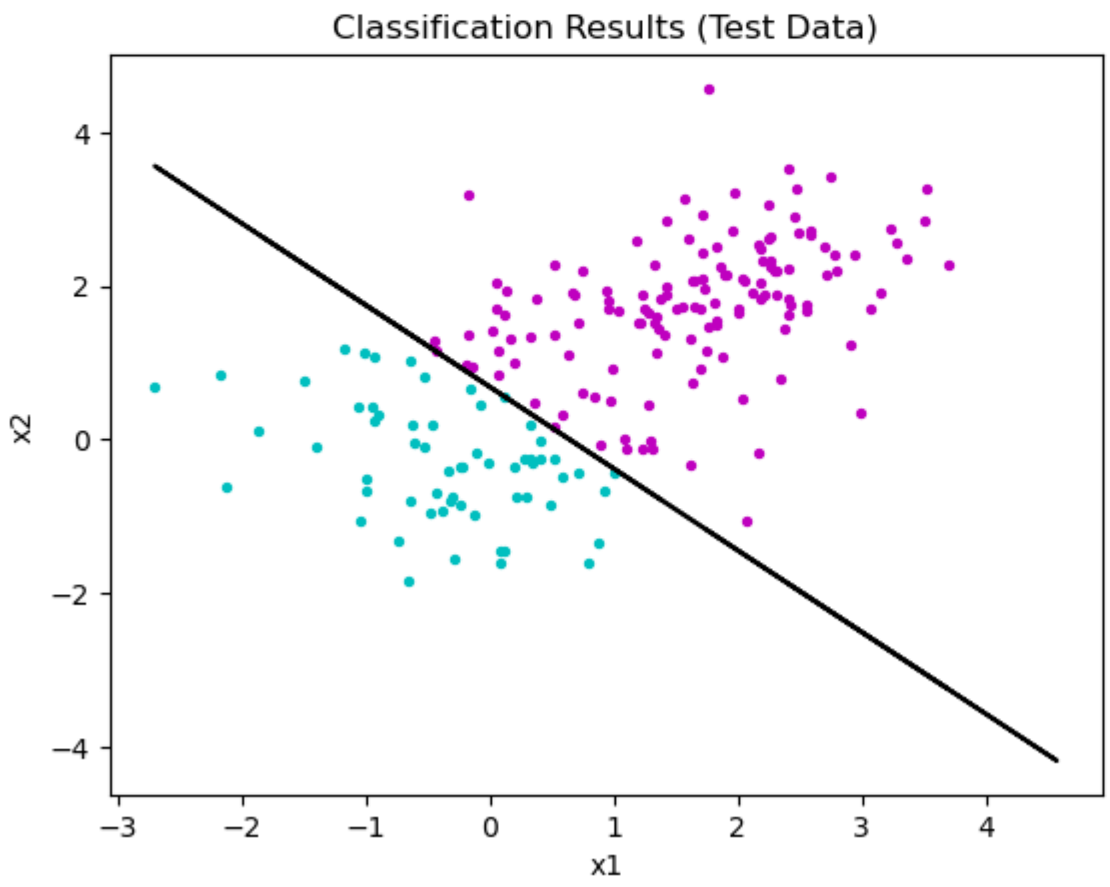
```
In [ ]: predictedTestY = classify(W, Xtest)
In [ ]: testAccuracy1 = accuracyNaively(Ytest, predictedTestY)
        print(str(testAccuracy1)+'%')
```

```
80.5%
```

```
In [ ]: testAccuracy2 = accuracy(Ytest, predictedTestY)
        print(testAccuracy1 - testAccuracy2)
```

```
0.0
```

```
In [ ]: drawClassification(W, Xtest, predictedTestY, 'c', 'm', "Classification Results (
```

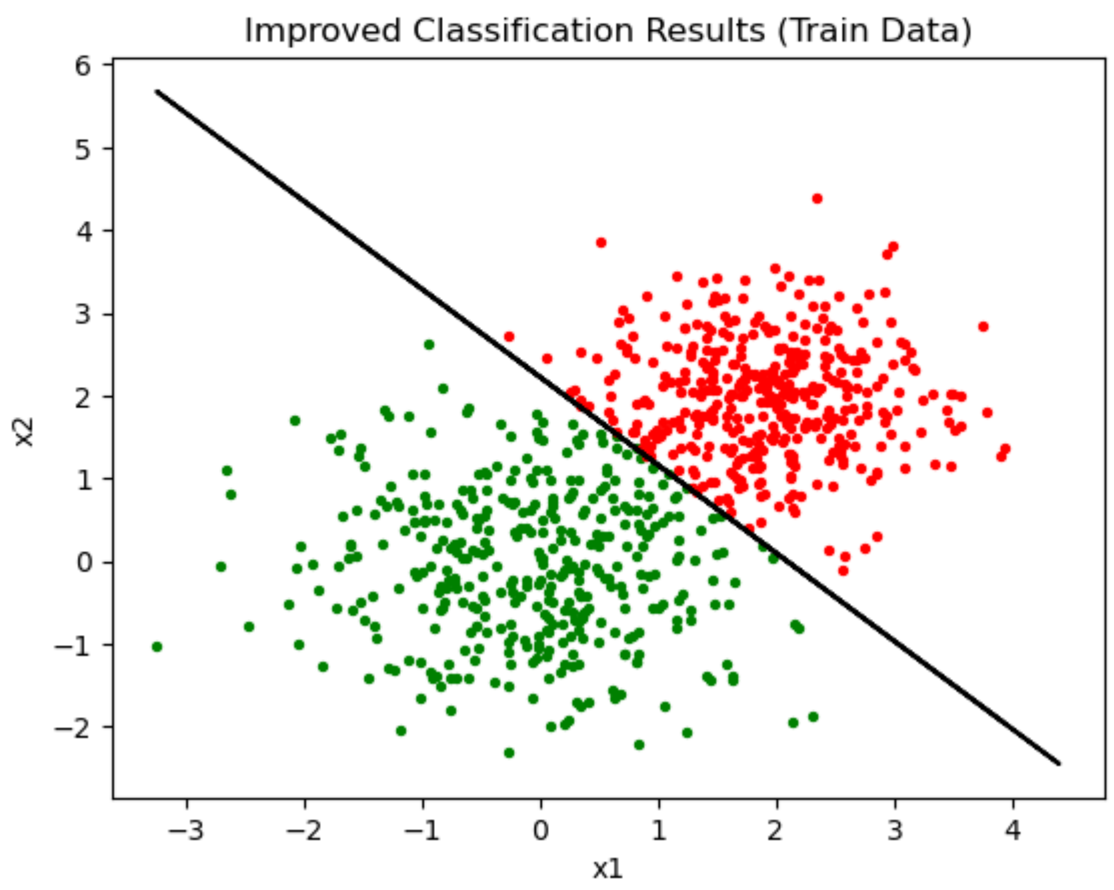


Question 6

Trying to improve the Classification of the Train

```
In [ ]: new_W = [W[0]-1.9085, W[1], W[2]]
        improvedPredictedTrainY = classify(new_W, Xtrain)
        drawClassification(new_W, Xtrain, improvedPredictedTrainY, 'g', 'r', "Improved C
        print(str(accuracy(Ytrain, improvedPredictedTrainY))+'%')
```

```
94.625%
```



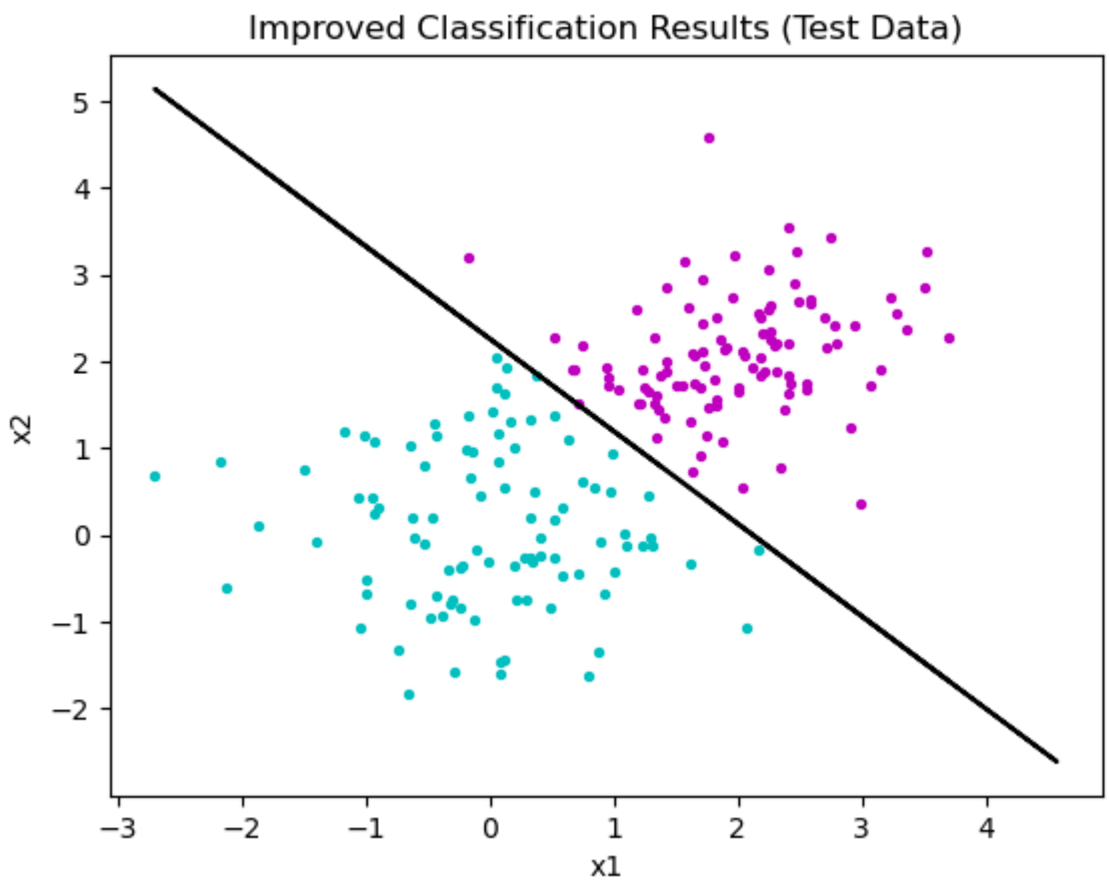
We tried eyeballing the first value, and we decided to deduct 1.5 from W[0]. Then we tried deducting 2. From then on we used a meathod that roughly resembles 'Binary Search' - we pick a value roughly in the middle of the interva [-2, -1.5], say -1.75, and then we try classifing with W[0]-{a value that is in the interval [-2, -1.75]} and see wether we improved our accuracy or not. We then do the same but this time the value is from the interval [-1.75, -1.5]. Then we recalibrate our interval based on the better improvement, and repeat the entire process.

This is 100% not a good and efficient method but it did a rather good job, as we managed to improve our accuracy from 85.375% to 94.625%.

Trying to improve the Classification of the Test

In [ ]:
```
new_W = [W[0]-1.95, W[1], W[2]]
improvedPredictedTestY = classify(new_W, Xtest)
drawClassification(new_W, Xtest, improvedPredictedTestY, 'c', 'm', "Improved Cla
print(str(accuracy(Ytest, improvedPredictedTestY))+'%')
```

```
94.0%
```



We tried eyeballing the first value, and we decided to deduct 1.5 from W[0]. Then we tried deducting 2. From then on we used a meathod that roughly resembles 'Binary Search' - we pick a value roughly in the middle of the interva [-2, -1.5], say -1.75, and then we try classifing with W[0]-{a value that is in the interval [-2, -1.75]} and see wether we improved our accuracy or not. We then do the same but this time the value is from the interval [-1.75, -1.5]. Then we recalibrate our interval based on the better improvement, and repeat the entire process.

This is 100% not a good and efficient method but it did a rather good job, as we managed to improve our accuracy from 80.5% to 94.0%.

In [ ]:
```
W = new_W
```

Question 7

In [ ]:
```
def confusionMatrix(actualY, predictedY):
    actualY = np.array(actualY)
    predictedY = np.array(predictedY)
    confusion_matrix = np.zeros([2, 2])
    values = [0, 1]
    for actual in values:
        for pred in values:
            # check both the * and the ==
            confusion_matrix[actual, pred] = ((actualY == actual)*(predictedY ==
    return confusion_matrix
```

```
In [ ]: print(f"\nConfusion Matrix of Xtrain:\n{str(confusionMatrix(Ytrain, classify(W,
```

```
Confusion Matrix of Xtrain:
[[374.  26.]
 [ 21. 379.]]
```

```
In [ ]: print("\nConfusion Matrix of Xtest:\n" + str(confusionMatrix(Ytest, classify(W,
```

```
Confusion Matrix of Xtest:
[[92.  8.]
 [ 4. 96.]]
```

Question 8

```
In [ ]: def probabilisticLogRegClassifier(W, X):
            """
            This Function
            """
            return 1/(1+np.exp(-(X@W[1:] + W[0])))
```

Question 9

```
In [ ]: def finalClassification(prb_Ypredicted_equals_one, th):
            if th > 1 or th < 0:
                print("th should be 0<=th<=1")
                return
            Y = []
            for prb_y_is_one in prb_Ypredicted_equals_one:
                Y.append(1.0 if prb_y_is_one > th else 0.0)
            return np.array(Y)
```

Question 10

```
In [ ]: def print_confusion_matrix(W, X, Y, th):
            final_classification = finalClassification(probabilisticLogRegClassifier(W,
            confusion_matrix = confusionMatrix(Y, final_classification)
            print("\nConfusion Matrix with th="+ str(th) +":\n" + str(confusion_matrix))
            return
```

```
In [ ]: print("Confusion Matrices of Train:")
        print_confusion_matrix(W, Xtrain, Ytrain, 0.5)
        print_confusion_matrix(W, Xtrain, Ytrain, 0.1)
        #print_confusion_matrix(W, Xtrain, Ytrain, 0.8) #Not that interesting and with i
        print_confusion_matrix(W, Xtrain, Ytrain, 0.8791)
        print_confusion_matrix(W, Xtrain, Ytrain, 0.9)
```

```
Confusion Matrices of Train:

Confusion Matrix with th=0.5:
[[374.  26.]
 [ 21. 379.]]

Confusion Matrix with th=0.1:
[[263. 137.]
 [  0. 400.]]

Confusion Matrix with th=0.8791:
[[397.   3.]
 [166. 234.]]

Confusion Matrix with th=0.9:
[[398.   2.]
 [194. 206.]]
```

As a reminder, the confusion matrix for the linear classifier:

In [ ]:
```
print(f"\nConfusion Matrix of Xtrain:\n{str(confusionMatrix(Ytrain, classify(W,
```

```
Confusion Matrix of Xtrain:
[[374.  26.]
 [ 21. 379.]]
```

As we can see, the confusion matrix of the linear classifier nets us the same result as the confusion matrix that uses logistical regression with a threshold th=0.5 . This makes sense, as having a threshold of 0.5 means that every value that exceeds 0.5 will be classified as 1, and every value that is below 0.5 will be classified as 0. This is similar to what the linear classifier does, as it classifies every value whos inner product with W is negative as a 0, and otherwise as a 1.

In [ ]:
```
print("Confusion Matrices of Test:")
print_confusion_matrix(W, Xtest, Ytest, 0.5)
print_confusion_matrix(W, Xtest, Ytest, 0.1)
#print_confusion_matrix(W, Xtest, Ytest, 0.8) #Not that interesting and with it
print_confusion_matrix(W, Xtest, Ytest, 0.8791)
print_confusion_matrix(W, Xtest, Ytest, 0.9)
```

```
Confusion Matrices of Test:

Confusion Matrix with th=0.5:
[[92.  8.]
 [ 4. 96.]]

Confusion Matrix with th=0.1:
[[ 57.  43.]
 [  0. 100.]]

Confusion Matrix with th=0.8791:
[[100.   0.]
 [ 42.  58.]]

Confusion Matrix with th=0.9:
[[100.   0.]
 [ 44.  56.]]
```

As a reminder, the confusion matrix for the linear classifier:

```
In [ ]: print("\nConfusion Matrix of Xtest:\n" + str(confusionMatrix(Ytest, classify(W,
```

```
Confusion Matrix of Xtest:
[[92.  8.]
 [ 4. 96.]]
```

As we can see, the confusion matrix of the linear classifier nets us the same result as the confusion matrix that uses logistical regression with a threshold th=0.5 . This makes sense, as having a threshold of 0.5 means that every value that exceeds 0.5 will be classified as 1, and every value that is below 0.5 will be classified as 0. This is similar to what the linear classifier does, as it classifies every value whos inner product with W is negative as a 0, and otherwise as a 1.