DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Machine Learning and Data Science

ENCS5341

Homework (2)

Prepared By: Student Name: Ibrahim Nobani

Student ID: 1190278

Instructor: Dr. Yazan Abu Farha

Section: 1

Date: 11/1/2023

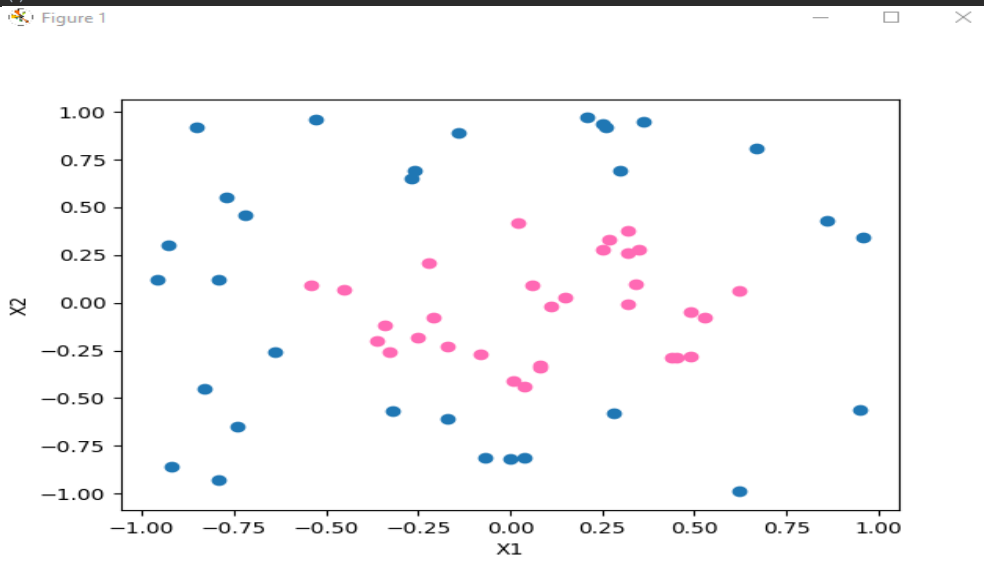
Part1:

First the CSV train file was read and appended to a trainedData list as follows:

```
trainedData=[]
trainedDataC1=[[],[]]
trainedDataC2=[[],[]]
with open("./train.csv", 'r') as file:
  csvreader = csv.reader(file)
  for row in csvreader:
    trainedData.append(row)
trainedData.pop(0)
```

Then the first scatter plot of the trained data was plotted:

```
f, axis = plt.subplots()
axis.scatter(trainedDataC1[0],trainedDataC1[1])
axis.scatter(trainedDataC2[0],trainedDataC2[1],color = 'hotpink')
axis.set_xlabel('X1')
axis.set_ylabel('X2')
plt.show()
```



Gradient descent was then applied for logistic regression to generate data for Theta0, Theta1 and Theta2:

We know gradient descent has the following model:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

With the sigmoid function:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Gradient descent for the logistic regression generates this model:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Which is applied in the code down below:

```
def gradient_descent(x, y, iterations, n,LearnRate=0.001):
  W1 = 0.1
  W0 = 0.01
  W2=0.1
  iterations = iterations
  LearnRate = LearnRate
  for i in range(iterations):
    finalPr = 1 / (1 + np.exp(-(W1 * x[0]) +-(W2 * x[1])+ -W0))
    gradient1 = -(2 / n) * sum(x[0] * (y - finalPr))
    gradient2 = -(2 / n) * sum(y - finalPr)
    gradient3= -(2 / n) * sum(x[1] * (y - finalPr))
    W1 = W1 - (LearnRate * gradient1)
    W2=W2 - (LearnRate * gradient3)
    W0 = W0 - (LearnRate * gradient2)
    print("Iteration",i," W0: ",W0,"W1: ",W1,"W2: ",W2)
  return W0, W1, W2
X=np.array(X)
Theta0,Theta1,Theta2=gradient_descent(X,Y,100000,len(Y))
```

After Running the gradient descent with 100000 iterations, the following values were obtained:

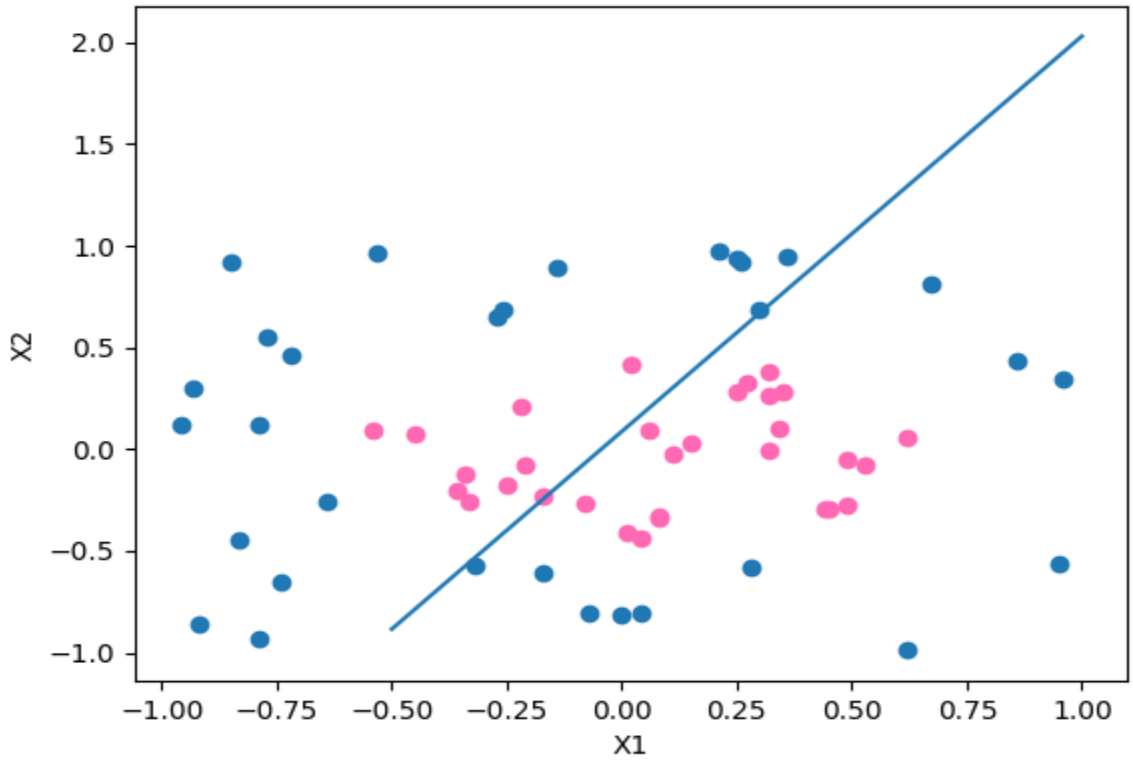
```
Iteration 99999  W0:  -0.05180900571531209 W1:  -1.1621171622480848 W2:  0.5978027589756287
Theta0:  -0.05180900571531209 Theta1:  -1.1621171622480848 Theta2:  0.5978027589756287
```

```
The logistic model obtained is = -0.05180900571531209 + -1.1621171622480848 x1 + 0.5978027589756287 x2
```

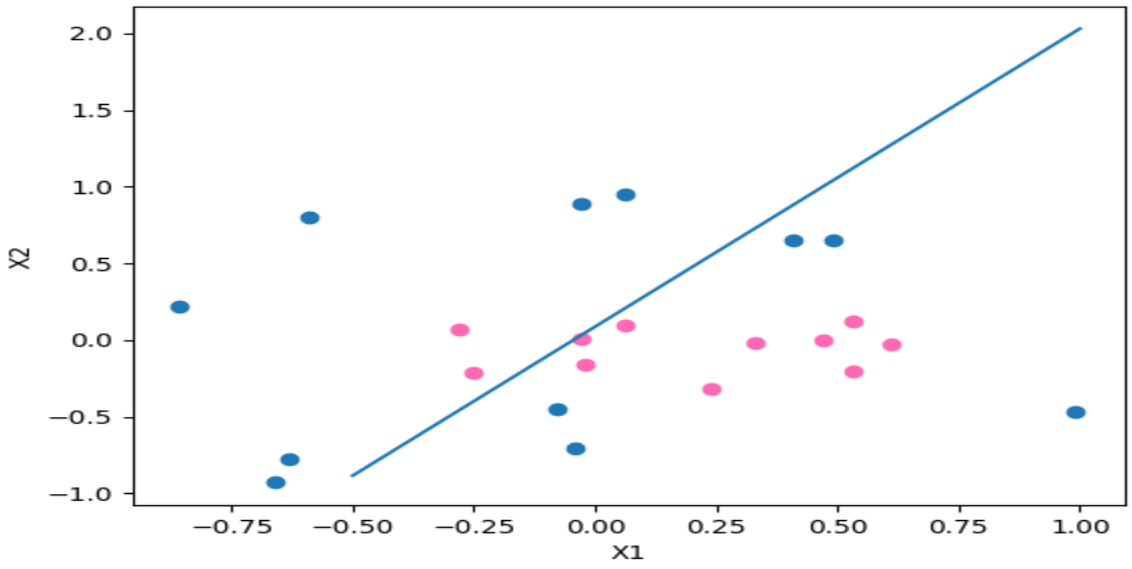
The decision boundary was then plotted as follows:

```
f, axis = plt.subplots()
axis.scatter(trainedDataC1[0],trainedDataC1[1])
axis.scatter(trainedDataC2[0],trainedDataC2[1],color = 'hotpink')
axis.set_xlabel('X1')
```

```
axis.set_ylabel('X2')
X1 = np.linspace(-0.5,1)
X2=-(Theta0+(Theta1*X1))/(Theta2)
plt.plot(X1,X2)
plt.show()
```



Plotting the test data:



To calculate the accuracy for both the train data and test data, the test data was read from the csv file, using the following function we calculated the accuracy:

```
#Accuracy functions that takes input X and Y.
def accuracy (X,Y):
  count = 0
  for i in range(len(X[0])):
    Y2 = Theta0 + Theta1 * X[0][i] + Theta2 * X[1][i]
    p = 0
    if Y2 >= 0:
        p = 1
    if p == Y[i]:
        count+=1
  accuracy = count / len(Y)
  return accuracy
print("The accuracy for the trained data is: ",accuracy(X,Y))
print("The accuracy for the test data is: ",accuracy(XT,YT))
```

```
The accuracy for the trained data is:  0.6612903225806451
The accuracy for the test data is:  0.6818181818181818
```

Part 2:

Input X was squared first, then gradient descent was applied and the results of theta were obtained as follows:

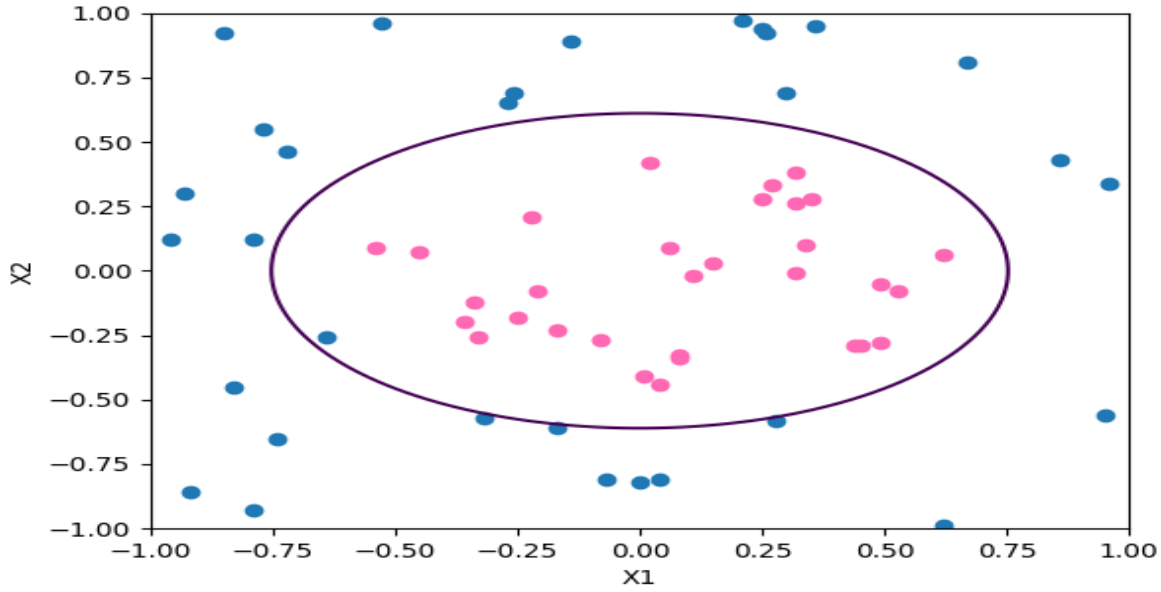
```
Xsq = np.square(X)
Xsq=np.array(Xsq)
Theta0,Theta1,Theta2=gradient_descent(Xsq,Y,100000,len(Y))
```

```
Iteration 99999  W0:  -2.6446032684652816 W1:  4.6595123504592415 W2:  7.088818963290747
Theta0:  -2.6446032684652816 Theta1:  4.6595123504592415 Theta2:  7.088818963290747
The logistic model obtained is = -2.6446032684652816 + 4.6595123504592415 x1^2 + 7.088818963290747 x2^2
```

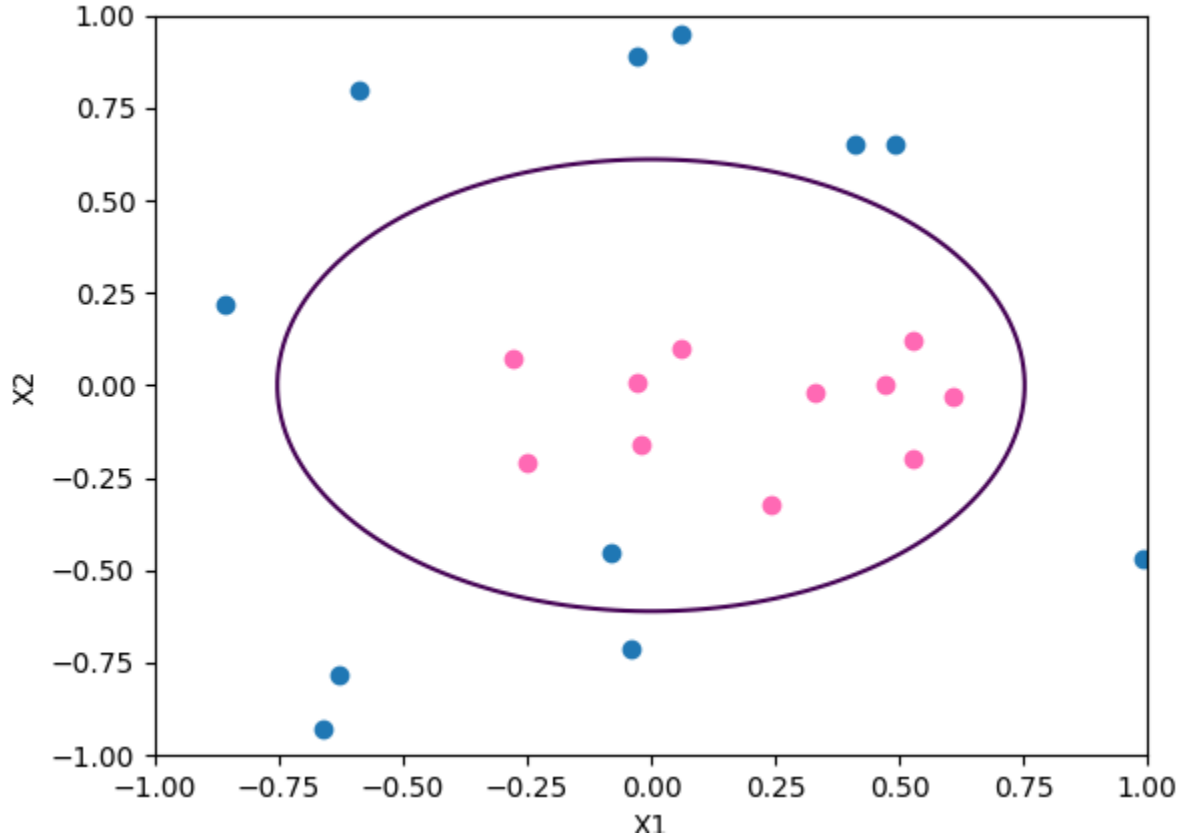
After obtaining the values of theta, we plot using the following code:

```
f, axis = plt.subplots()
axis.scatter(trainedDataC1[0],trainedDataC1[1])
axis.scatter(trainedDataC2[0],trainedDataC2[1],color = 'hotpink')
axis.set_xlabel('X1')
axis.set_ylabel('X2')
X1 = np.linspace(-1, 1, 100)
X2 = np.linspace(-1, 1, 100)
X1, X2 = np.meshgrid(X1, X2)
F = Theta0 + Theta1 * pow(X1,2) + Theta2 * pow(X2,2)
F = np.array(F)
#F = F.reshape((len(X1), len(X2)))
plt.contour(X1, X2, F, [0])
#plt.plot(X1,X2,F)
plt.show()
```

The following figure was obtained, you can see the boundary on it:



Plotting the Test data outputs this figure:



Then using the following accuracy code, the accuracies for both the train and test values were generated:

```
def accuracySQ (X,Y):
  count = 0
  for i in range(len(X[0])):
    Y2 = Theta0 + Theta1 * pow(X[0][i],2) + Theta2 * pow(X[1][i],2)
    p = 0
    if Y2 >= 0:
        p = 1
    if p == Y[i]:
        count+=1
  accuracy = count / len(Y)
  return accuracy
```

```
The accuracy for the trained data is:  0.9838709677419355
The accuracy for the test data is:  0.9545454545454546
```

Part 3:

In the case of underfitting, the train accuracy is usually low, while the test accuracy is also low but similar to the train accuracy. This is because the model is not able to capture the complexity of the data, and so it performs poorly on both the training set and new data.

In the case of overfitting, the train accuracy is usually high, while the test accuracy is significantly lower. This is because the model is able to fit the training data very well by memorizing it, but it doesn't generalize well to new, unseen data.

-In the first model we have the following accuracies:

The accuracy for the trained data is: 0.6612903225806451

The accuracy for the test data is: 0.6818181818181818

Looking at them we can conclude that both accuracies are low and close to each other, which is what the underfitting states, therefore its underfitting.

A more complex model decision boundary would be a more optimal solution for this underfitting problem.

-In the second model we have the following accuracies:

The accuracy for the trained data is: 0.9838709677419355

The accuracy for the test data is: 0.9545454545454546

We can notice that both values are relatively high accuracies (so no underfitting here), both are also close to each other, so as we can see no overfitting occurs here.

What can be concluded is that when the model got more complex it solved the underfitting and overfitting.

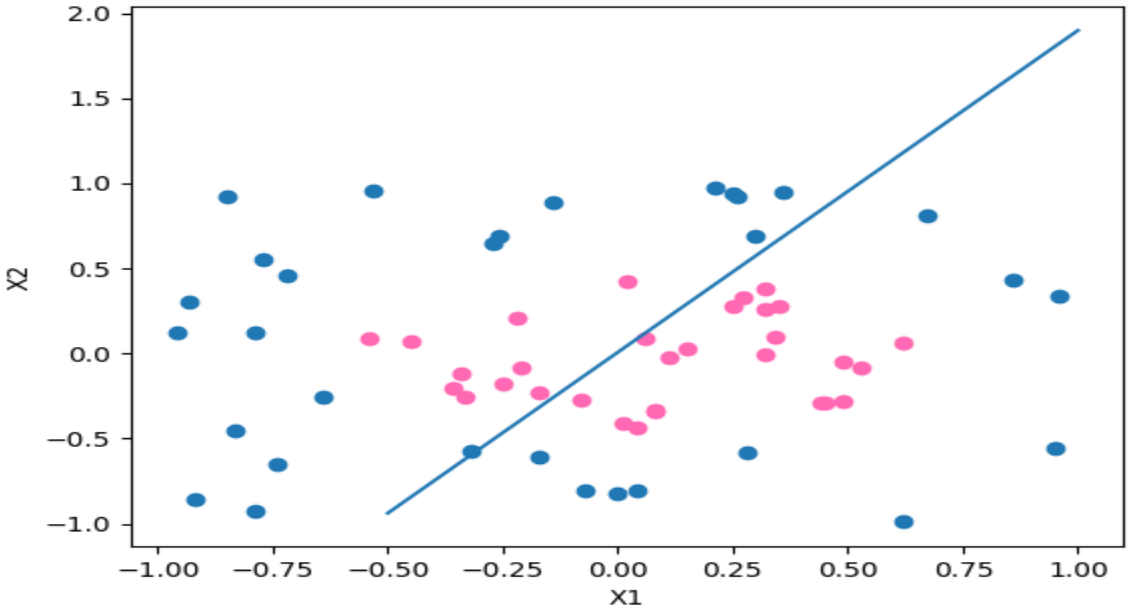
Part 4:

Using scikit linear model with logistic regression with the following code:

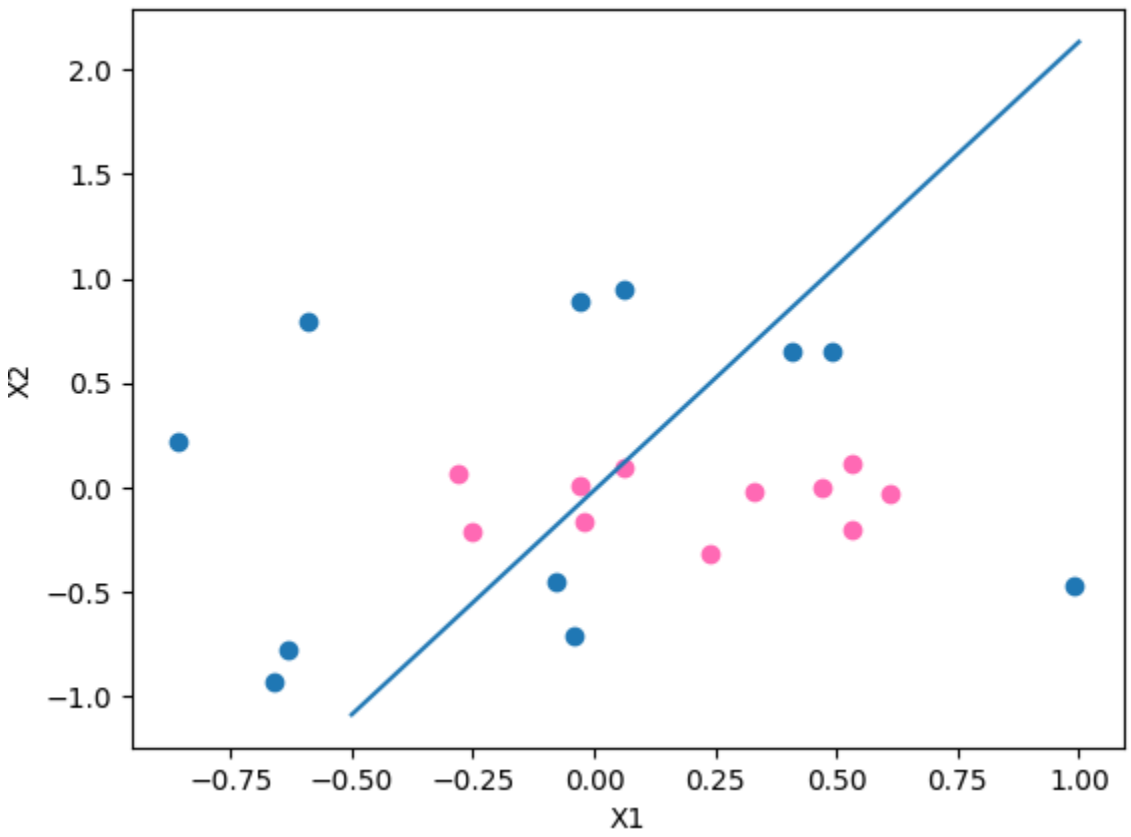
```
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
def logisticRegSK(X,Y,XT,YT,C):
  model = LogisticRegression(C=C, solver="lbfgs")
  X2T = XT.transpose()
  model.fit(X2T, YT)
  scoreT = model.score(X2T, YT)
  X2=X.transpose()
  model.fit(X2,Y)
  theta = model.coef_[0]
  print(model.intercept_[0],theta)
  print("Theta0: ",model.intercept_[0],"Theta1 and Theta2: ",theta)
  score = model.score(X2, Y)
  print("The accuracy of this train model: ",score)
  print("The accuracy of this test model: ", scoreT)
  return model.intercept_[0],model.coef_[0][0],model.coef_[0][1]
Theta=logisticRegSK(X,Y,np.array(XT),YT,0.001)
plotN(Theta[0],Theta[1],Theta[2],trainedDataC1,trainedDataC2)
plotN(Theta[0],Theta[1],Theta[2],testDataC1,testDataC2)
Theta=logisticRegSK(np.array(Xsq),Y,np.array(np.square(XT)),YT,1e2)
plotSQ(Theta[0],Theta[1],Theta[2],trainedDataC1,trainedDataC2)
plotSQ(Theta[0],Theta[1],Theta[2],testDataC1,testDataC2)
```

These results were obtained:

-Trained Data:

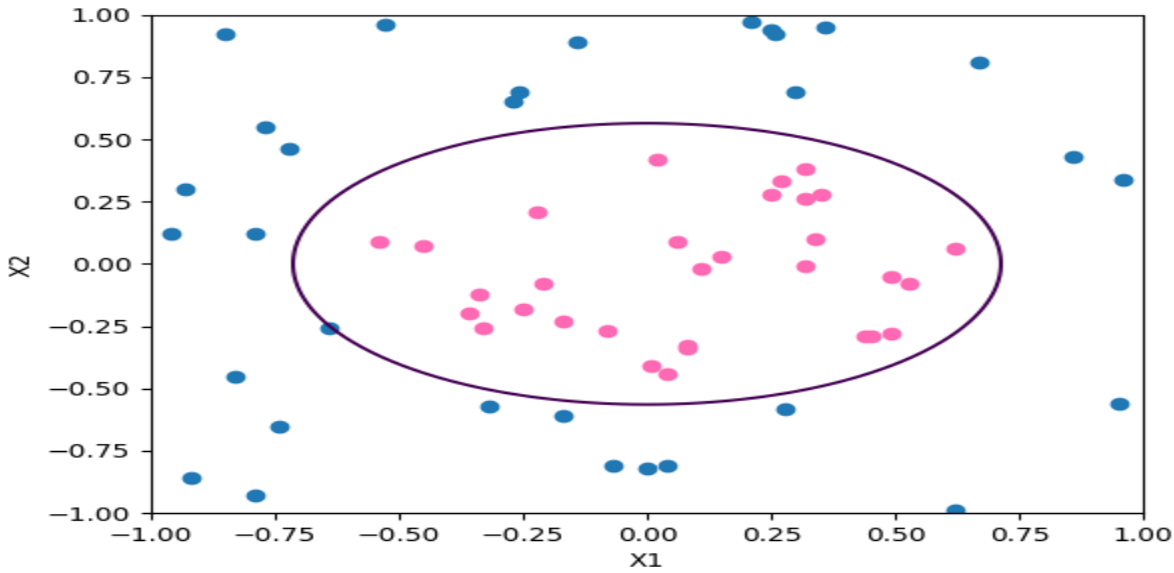


-Test Data:

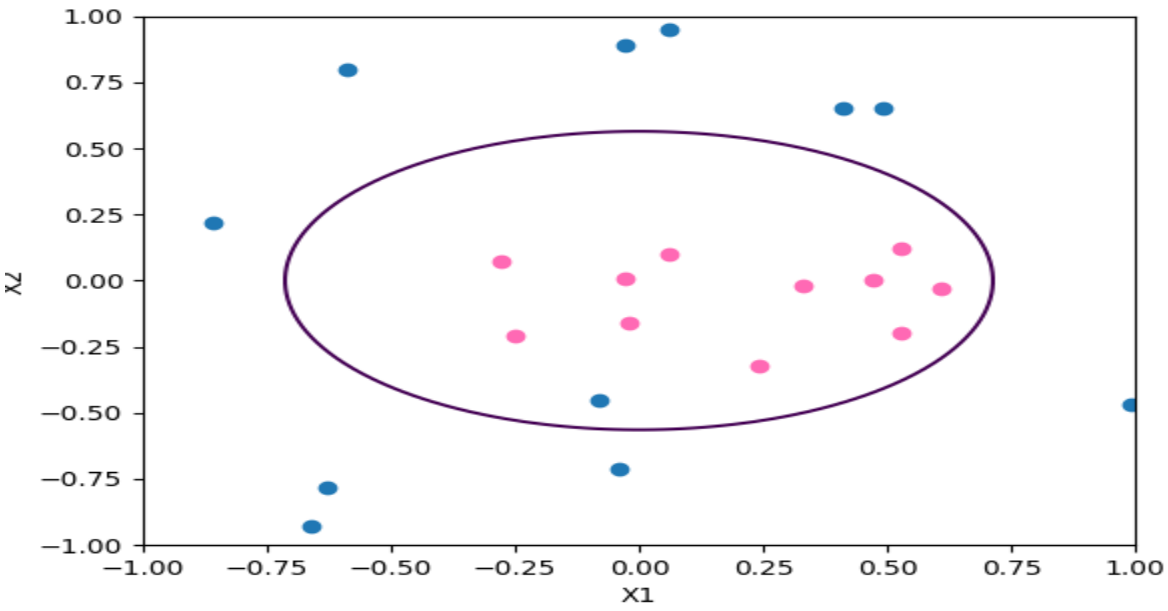



```
The following is the linear regression using the sci-kit library...
-1.4193968427571736e-05 [-0.00385303  0.00203662]
Theta0:  -1.4193968427571736e-05 Theta1 and Theta2:  [-0.00385303  0.00203662]
The accuracy of this train model:  0.6774193548387096
The accuracy of this test model:  0.6363636363636364
```

-Squared Trained Data:



Squared Test Data:




```
Theta0:  -6.108537664520157 Theta1 and Theta2:  [11.99221337 19.19137057]
The accuracy of this train model:  1.0
The accuracy of this test model:  0.9545454545454546
```