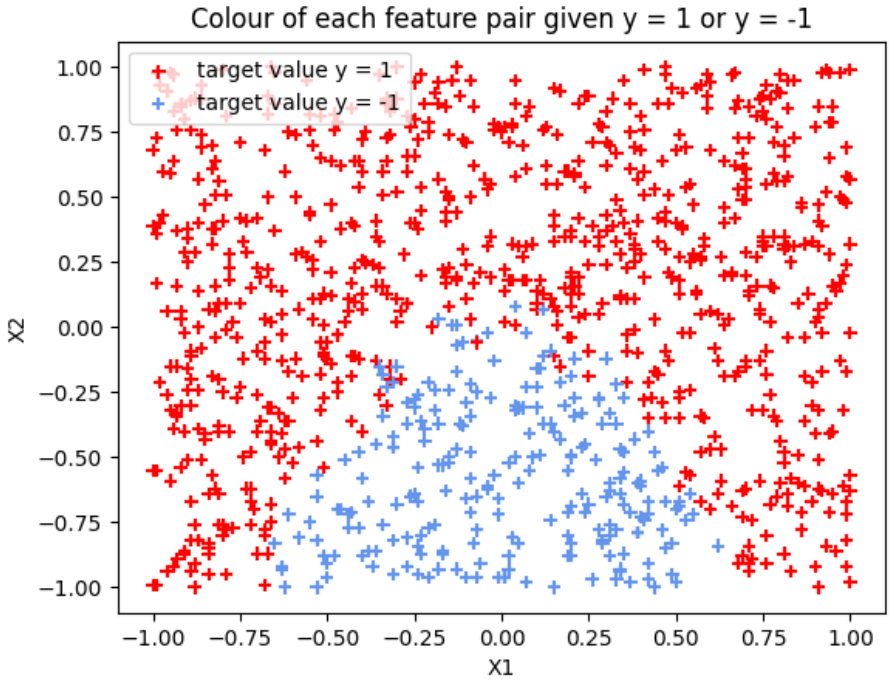
Machine Learning Assignment #2

Alex Kennedy *17328638*

 # id:2-4-2

A(i)

For each point, I needed to determine whether the point had a target value of y = 1 or y = -1. To do this I created a list of colours the size of X1. I went through each value of y and whenever y = -1, I changed the colour. So, I had a list of colours where when y = 1, the points are red and they are blue if y = -1.



A(ii)

After training a logistic regression classifier on the data given using sklearn the following parameter values were obtained:

```
Parameter Values for a(ii):
Intercept (theta0): [2.09423891]Coefficients (theta1 and theta2): [[-0.17132633  3.5261049 ]]
```

These values are used in creating a decision boundary which will create a margin between the two classifications when predicting values based on this logistic regression.

A(iii)

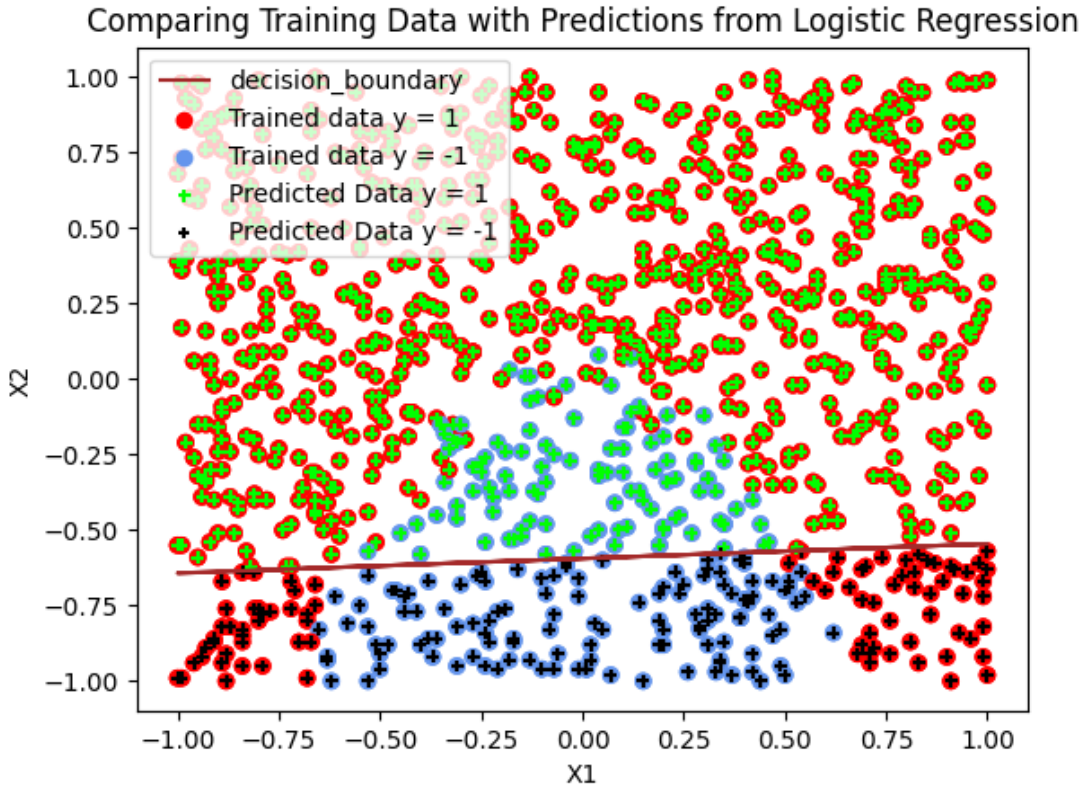
We use sklearn's predict() function to produce predicted target values. Once again, we use different colours and markers to differentiate the data as they are on the same points. I created the decision boundary through the equation $\theta^t x = 0$ where:

$$\theta^t x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n, with\ x_0 = 1.$$

Because the equation needs to be in terms of X2 we need to rearrange the equation to be

$$X_2 = -\frac{(\theta_0 * 1) + (\theta_1 x_1)}{\theta_2}$$

After plugging in our parameter values into this equation, we get our decision boundary. In the plot below, we compare the predicted target. values from a Logistic Regression classifier with actual target values from the data. The decision shows that the data is linearly seperable.



A(iv)

One can see that the actual target takes on a more quadratic form (i.e if we were to draw a line around the border of the actual target value y = -1 the line would be a parabola) , while the predicted values are more linear. This can be seen quite easily by the decision boundary. When

predicting the values, the model parameters used are used in the equation of a line and so the target values will always be in a linear form.

B(i)

After training linear classifiers for the values of the penalty parameter value C = 0.001, C = 0.01 and C = 1000, the parameter values achieved were as follows:

```
SVM with C = 0.001
Intercept (theta0): [0.35809959]Coefficients (theta1 and theta2): [[-0.00708755  0.32395396]]

SVM with C = 0.01
Intercept (theta0): [0.54717689]Coefficients (theta1 and theta2): [[-0.04395708  0.79403724]]

SVM with C = 1000
Intercept (theta0): [0.69704394]Coefficients (theta1 and theta2): [[-0.07436229  1.1871619 ]]
```

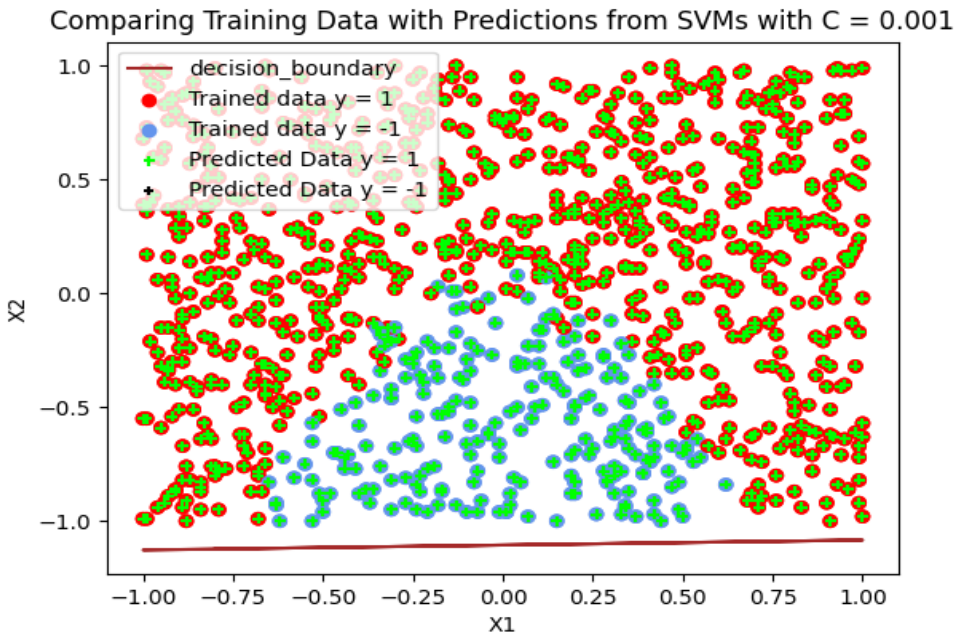
I had to increase the maximum number of iterations to 100,000 when performing the SVM for C = 1000 as it failed to converge.

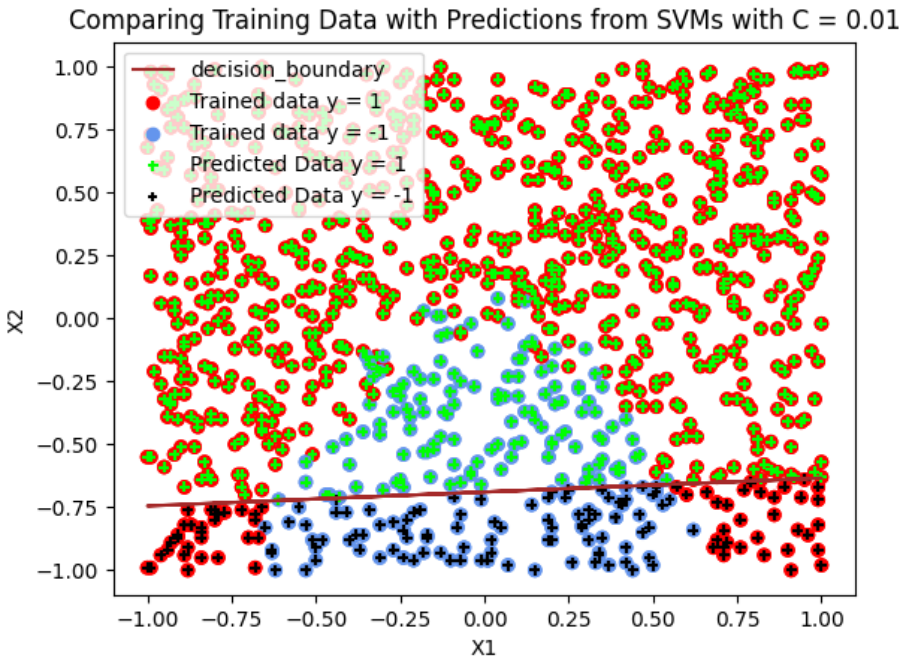
B(ii)

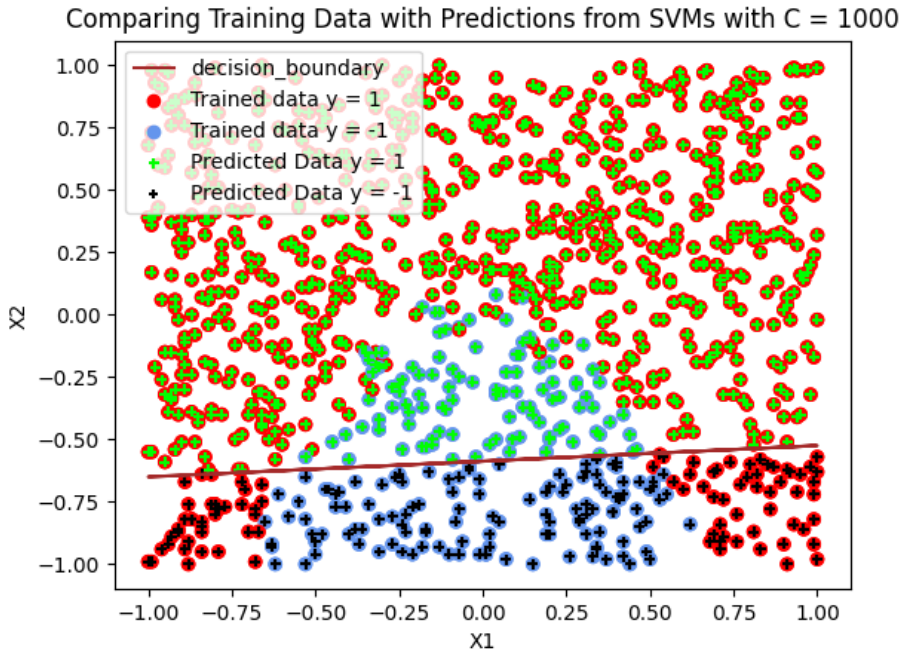
Once again we use sklearn's predict() function after training linear SVM classifiers on the data to get predicted target values and we create the decision boundary using the equation.

$$X_2 = -\frac{(\theta_0 * 1) + (\theta_1 x_1)}{\theta_2}$$

Here are the three plots for each C value:







B(iii)

The model parameter values of C will increase as the value of C increases, this increase will eventually slow down and converge towards a value. Although not plotted, The difference between the model parameter value between C = 1 and C = 1000 is extremely small

```
SVM with C = 1
Intercept (theta0): [0.69366974]Coefficients (theta1 and theta2): [[-0.07390787  1.17914049]]

SVM with C = 1000
Intercept (theta0): [0.69703948]Coefficients (theta1 and theta2): [[-0.07436571  1.18715085]]
```

The impact SVM has is that it will try to find the most optimal margin that will separate the classes correctly and help reduce errors in classification. The greater the C value, the smaller margin this is as you can see from the plot. With a low C value, the predicted data is vastly different than when there is a higher C value.

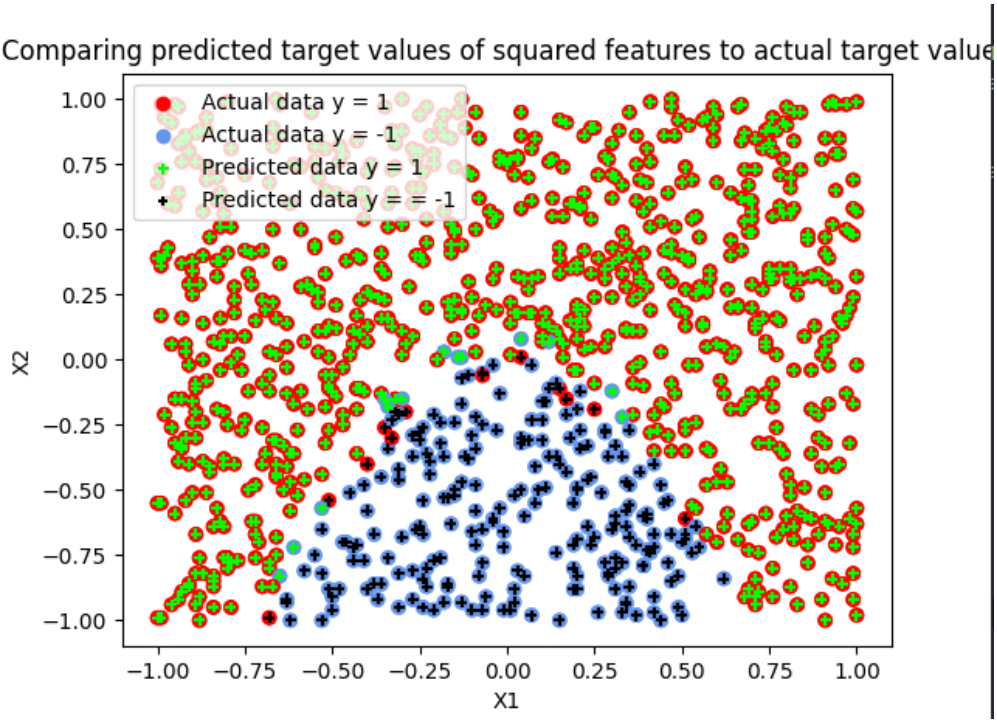
C(i)

To do this, I squared each value of X1 and X2 and used column_stack() to put 4 pieces of data into each row. This can be seen in the squareFeatures() function. After training a logistic regression classifier on these pieces of data, these were the parameter values i obtained:

```
Parameter Values for c(i):
Intercept (theta0): [-0.403679]Coefficients (theta1, theta2, theta3, theta4): [[ 0.85663063 20.62651978 46.3385383  -1.09602391]]
```

C(ii)

After using sklearn's predict() function to get the predicted target values of the logistic regression classifier using the squared features, this is the plot that was obtained:



As one can see, the predicted values are extremely similar to the actual target values from the dataset. This is due to squared features making giving becoming quadratic rather than just using linear features of X1 and X2.

C(iii)

Code Appendix

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as mpl
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC

def readFiles():
    df = pd.read_csv("week2.txt", comment = '#')
    X1 = df.iloc[:,0]
    X2 = df.iloc[:,1]
    X = np.column_stack((X1,X2))
    y = df.iloc[:,2]
    return X1, X2, X, y

#function to generate colour arrays to use to compare predicted target values to
actual target values.
def classifier(yPredict, color1, color2):
    predictColours = [color1] * len(yPredict)
    for i in range(len(yPredict)):
        if(yPredict[i] == -1):
            predictColours[i] = color2
    return predictColours

def get_boundary(X1, X2, intercept, coefficient, decision_boundary):
    decision_boundary = -
((intercept*1) + (coefficient[0][0] * X1))/coefficient[0][1]
    return decision_boundary

def squareFeatures(X1, X2):
    X3 = [0] * len(X1)
    X4 = [0] * len(X1)
    for i in range(len(X1)):
        X3[i] = (X1[i]**2)
    for j in range(len(X2)):
        X4[j] = (X2[j]**2)
    x_square = np.column_stack((X1,X2,X3,X4))
    return x_square, X3, X4

#attempting to construct a decision boundary for the square features but couldn't
 get it to work :(
def get_boundary_square(X1, X2, X3, X4, intercept, coefficient, square_boundary):
```

```
    square_boundary = -
((intercept*1) + (coefficient[0][0] * X1) + (coefficient[0][2] * X3)+ (coefficien
t[0][3]*X4))/coefficient[0][1]
    return square_boundary

def firstPlot(X1, X2, pointColours):
    for labels in ["target value y = 1", "target value y = -1"]:
        mpl.scatter(X1, X2, c = pointColours, marker = '+', label = labels)
    mpl.title("Colour of each feature pair given y = 1 or y = -1")
    mpl.xlabel("X1")
    mpl.ylabel("X2")
    mpl.legend()
    mpl.gca()
    legend = mpl.legend()
    legend.legendHandles[1].set_color('cornflowerblue')
    mpl.show()


def plot(X1, X2, decision_boundary, predictColours, pointColours):
    ax = mpl.subplot()
    for labels in ["Trained data y = 1", "Trained data y = -1"]:
        ax.scatter(X1, X2, color=pointColours, label = labels)
    for labels1 in ["Predicted Data y = 1", "Predicted Data y = -1"]:
        ax.scatter(X1, X2, color=predictColours, marker = '+', label = labels1, s
 = 5**2)
    ax.plot(X1, decision_boundary, label = "decision_boundary", c = 'brown')
    mpl.legend()
    ax = mpl.gca()
    leg = ax.get_legend()
    leg.legendHandles[2].set_color('cornflowerblue')
    leg.legendHandles[4].set_color('black')
    mpl.ylabel("X2")
    mpl.xlabel("X1")
    mpl.title("Comparing Training Data with Predictions from Logistic Regression"
)
    mpl.show()

def svm_plot0(X1, X2, decision_boundary, predictColours, pointColours):
    ax = mpl.subplot()
    for labels in ["Trained data y = 1", "Trained data y = -1"]:
        ax.scatter(X1, X2, color=pointColours, label = labels)
    for labels1 in ["Predicted Data y = 1", "Predicted Data y = -1"]:
        ax.scatter(X1, X2, color=predictColours, marker = '+', label = labels1, s
 = 5**2)
    ax.plot(X1, decision_boundary, label = "decision_boundary", c = 'brown')
```

```
    mpl.legend()
    ax = mpl.gca()
    leg = ax.get_legend()
    leg.legendHandles[2].set_color('cornflowerblue')
    leg.legendHandles[4].set_color('black')
    mpl.ylabel("X2")
    mpl.xlabel("X1")
    mpl.title("Comparing Training Data with Predictions from SVMs with C = 0.001"
)
    mpl.show()

def svm_plot1(X1, X2, decision_boundary, predictColours, pointColours):
    ax = mpl.subplot()
    for labels in ["Trained data y = 1", "Trained data y = -1"]:
        ax.scatter(X1, X2, color=pointColours, label = labels)
    for labels1 in ["Predicted Data y = 1", "Predicted Data y = -1"]:
        ax.scatter(X1, X2, color=predictColours, marker = '+', label = labels1, s
 = 5**2)
    ax.plot(X1, decision_boundary, label = "decision_boundary", c = 'brown')
    mpl.legend()
    ax = mpl.gca()
    leg = ax.get_legend()
    leg.legendHandles[2].set_color('cornflowerblue')
    leg.legendHandles[4].set_color('black')
    mpl.ylabel("X2")
    mpl.xlabel("X1")
    mpl.title("Comparing Training Data with Predictions from SVMs with C = 0.01")
    mpl.show()

def svm_plot2(X1, X2, decision_boundary, predictColours, pointColours):
    ax = mpl.subplot()
    for labels in ["Trained data y = 1", "Trained data y = -1"]:
        ax.scatter(X1, X2, color=pointColours, label = labels)
    for labels1 in ["Predicted Data y = 1", "Predicted Data y = -1"]:
        ax.scatter(X1, X2, color=predictColours, marker = '+', label = labels1, s
 = 5**2)
    ax.plot(X1, decision_boundary, label = "decision_boundary", c = 'brown')
    mpl.legend()
    ax = mpl.gca()
    leg = ax.get_legend()
    leg.legendHandles[2].set_color('cornflowerblue')
    leg.legendHandles[4].set_color('black')
    mpl.ylabel("X2")
    mpl.xlabel("X1")
    mpl.title("Comparing Training Data with Predictions from SVMs with C = 1000")
```

```
    mpl.show()

def square_plot(X1, X2, square_pred_colours, pointColours):
    for labels in ["Actual data y = 1", 'Actual data y = -1']:
        mpl.scatter(X1, X2, c = pointColours, label = labels)
    for labels in ["Predicted data y = 1", 'Predicted data y = = -1']:
        mpl.scatter(X1, X2, c = square_pred_colours, label = labels, marker = "+"
, s  =5**2)
    mpl.xlabel("X1")
    mpl.ylabel("X2")
    mpl.title("Comparing predicted target values of squared features to actual ta
rget values")
    mpl.legend()
    mpl.gca()
    legend = mpl.legend()
    legend.legendHandles[1].set_color('cornflowerblue')
    legend.legendHandles[3].set_color('black')


    mpl.show()

def main():
    X1, X2, X, y = readFiles()
    pointColours = ['red'] * len(y)
    for i in range(len(pointColours)):
        if y[i] == -1:
            pointColours[i] = 'cornflowerblue'

    firstPlot(X1, X2, pointColours)
#FOR ASSIGNMENT (A) LOGISTIC REGRESSION
    model = LogisticRegression(penalty = 'none', solver='lbfgs')
    model.fit(X, y)
    yPredict = model.predict(X)
    predictColours = classifier(yPredict, 'lime', 'black')
    print("Parameter Values for a(ii):")
    print("Intercept (theta0): " +str(model.intercept_) + "Coefficients (theta1 a
nd theta2): " +str(model.coef_) +'\n')
    decision_boundary = [0] * len(X2)
    decision_boundary = get_boundary(X1, X2, model.intercept_, model.coef_, decis
ion_boundary)
    plot(X1, X2, decision_boundary, predictColours, pointColours)

#FOR PART (B) USING SVM'S
    svm_model0 = LinearSVC(C = 0.001).fit(X, y)
    print("SVM with C = 0.001")
```

```
    print("Intercept (theta0): " +str(svm_model0.intercept_) + "Coefficients (the
ta1 and theta2): " +str(svm_model0.coef_) +'\n')
    svm_decision0 = [0] * len(X2)
    svm_decision0 = get_boundary(X1, X2, svm_model0.intercept_, svm_model0.coef_,
 svm_decision0)
    svm_predict0 = svm_model0.predict(X)
    #print(svm_predict0)
    svm_pred_color0 = classifier(svm_predict0, 'lime', 'black')
    svm_plot0(X1, X2, svm_decision0, svm_pred_color0, pointColours)

    svm_model1 = LinearSVC(C = 0.01).fit(X, y)
    print("SVM with C = 0.01")
    print("Intercept (theta0): " +str(svm_model1.intercept_) + "Coefficients (the
ta1 and theta2): " +str(svm_model1.coef_) + "\n")
    svm_decision1 = [0] * len(X2)
    svm_decision1 = get_boundary(X1, X2, svm_model1.intercept_, svm_model1.coef_,
 svm_decision1)
    svm_predict1 = svm_model1.predict(X)
    # print(svm_predict1)
    svm_pred_color1 = classifier(svm_predict1, 'lime', 'black')
    svm_plot1(X1, X2, svm_decision1, svm_pred_color1, pointColours)

    test_svm_model2 = LinearSVC(C = 1).fit(X, y,)
    print("SVM with C = 1")
    print("Intercept (theta0): " +str(test_svm_model2.intercept_) + "Coefficients
 (theta1 and theta2): " +str(test_svm_model2.coef_) + "\n")

    svm_model2 = LinearSVC(C = 1000, max_iter=100000).fit(X, y,)
    print("SVM with C = 1000")
    print("Intercept (theta0): " +str(svm_model2.intercept_) + "Coefficients (the
ta1 and theta2): " +str(svm_model2.coef_) + "\n")
    svm_decision2 = [0] * len(X2)
    svm_decision2 = get_boundary(X1, X2, svm_model2.intercept_, svm_model2.coef_,
 svm_decision2)
    svm_predict2 = svm_model2.predict(X)
    #print(svm_predict2)
    svm_pred_color2 = classifier(svm_predict2, 'lime', 'black')
    svm_plot2(X1, X2, svm_decision2, svm_pred_color2, pointColours)

#FOR PART (C)
    #squaring each X1 and X2 value and adding it to an array.
    X3 = [0] * len(X1)
    X4 = [0] * len(X2)
    x_square, X3, X4 = squareFeatures(X1, X2)
    square_model = LogisticRegression(penalty = 'none', solver='lbfgs')
```

```
    square_model.fit(x_square, y)
    print("Parameter Values for c(i):")
    print("Intercept (theta0): " +str(square_model.intercept_) + "Coefficients (t
heta1, theta2, theta3, theta4): " +str(square_model.coef_) +'\n')
    #getting the predicted values
    square_predict = square_model.predict(x_square)
    square_pred_colours = classifier(square_predict, 'lime', 'black')
    #square_boundary = [0] * len(X2)
    square_plot(X1, X2, square_pred_colours, pointColours)
    # square_boundary = get_boundary_square(X1, X2, X3, X4, square_model.intercep
t_, square_model.coef_, square_boundary)
    # print(square_boundary)


if __name__ == "__main__":
    main()
```