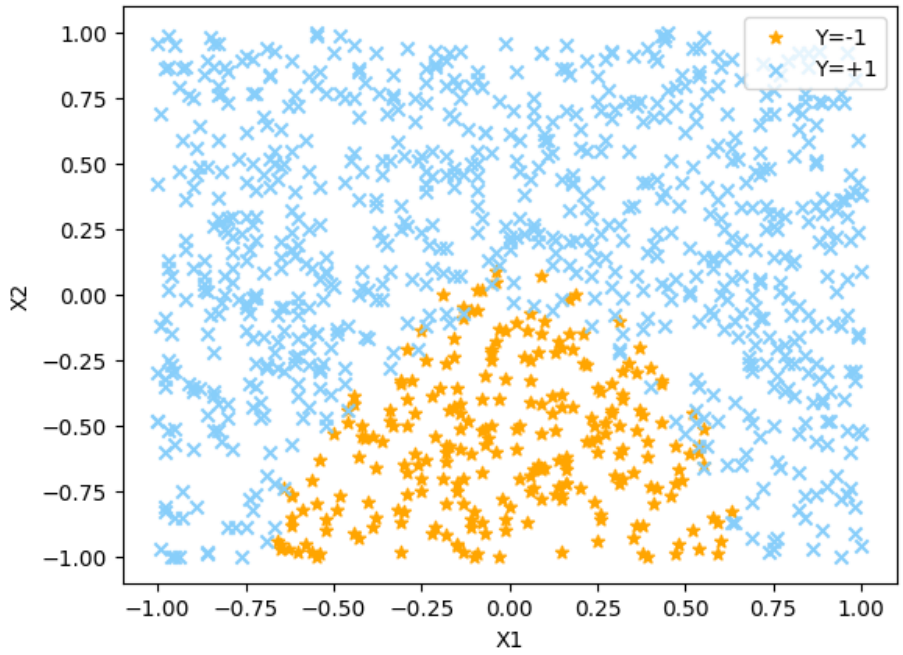
(a) (i)

Plotted a scattered graph using matplotlib for the input features from the dataset. The first feature (X1) is plotted on the X-axis, and the second feature (X2) is plotted on the Y-axis. All the points are between -1 to +1. On the graph, a blue X (x) is used to plot points having a y value of +1, while a yellow star (*) is used to indicate points having a y value of -1. A legend is displayed in the top right corner of the plot indicating this information. The graph shows dominance of Y=+1 class and it seems that implementing a curved decision boundary would give the best fit as the data does not seem to be linearly separable.



(a) (ii)

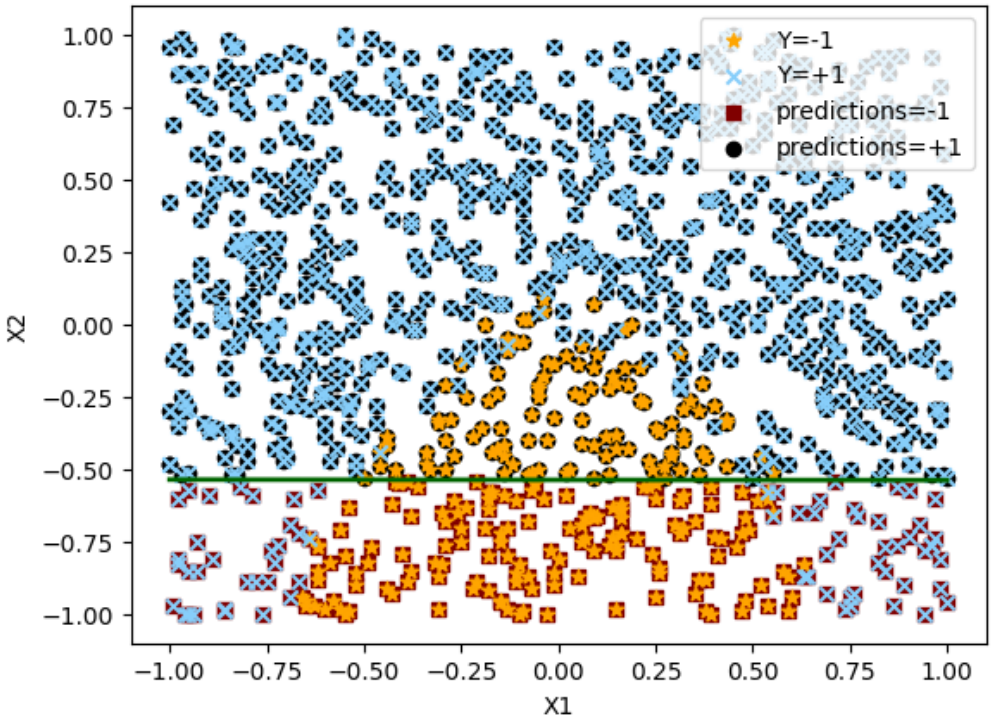
From scikit-learn, a logistic regression model was trained for the given dataset.
LogisticRegression(penalty=None) class was imported from sklearn.linear_model. An object of this class was created and trained for our data using *.fit(X, y)* method.
An intercept of 2.02871098 was obtained, favorable towards y being +1 when all features are set to 0.
Coefficients obtained for the model are 0.00337053745 and 3.84563071 for features X1 and X2, respectively. Both being positive would cause an increase in the prediction. Also, as the weight of feature X2 is large and positive, a slight increase in the value of X2 will lead to a rapid increase in the value of y. Thus, feature X2 has the most influence on the prediction. An accuracy of 82.18 % was obtained

Intercept -> [2.02871098]
Coefficients -> [[3.37053745e-03 3.84563071e+00]]
Accuracy: 0.8218218218218218

(a) (iii)

After training model with data, predictions were done using same training data and the prediction points were plotted along with input points as show in the graph below.
z = b + w1*x1 + w2*x2 is the equation of logistic regression having two features.

At boundary line (since P(y=+1) = 0.5) this equation becomes 0; z = 0.
thus, rearranging initial equation we get x2 = m * x1 + c. (where m = -w1 / w2 & c = -b / w2)
We have weights and intercept values from the model. We then use minimum x1 value and corresponding x2 value to mark a point, also, use maximum x1 value and corresponding x2 value to mark another point. Draw a line between these two points which becomes our decision boundary (line in green).



(a) (iv)

As seen in the plot above, the input data is not linearly separable. But logistic regression model still tries to separate the points using a single line (green line around X2=0.5), our accuracy is 82.18 %, which has an error of 17.82% due to data not being linearly separable. Thus, it can be inferred that a linear decision boundary is not the best fit for our dataset.

(b) (i)

Trained SVM classifier (LinearSVC from sklearn) for the dataset, with the penalty parameter ranging from C=0.001 to C=100 as implemented in code below.

CODE - >

```
from sklearn.svm import LinearSVC as svc

svc1 = svc(C=0.001).fit(X,y)
svc3 = svc(C=0.1).fit(X,y)
svc4 = svc(C=1).fit(X,y)
svc5 = svc(C=100). fit(X,y)
```

Below are the parameters obtained for respective values of C:

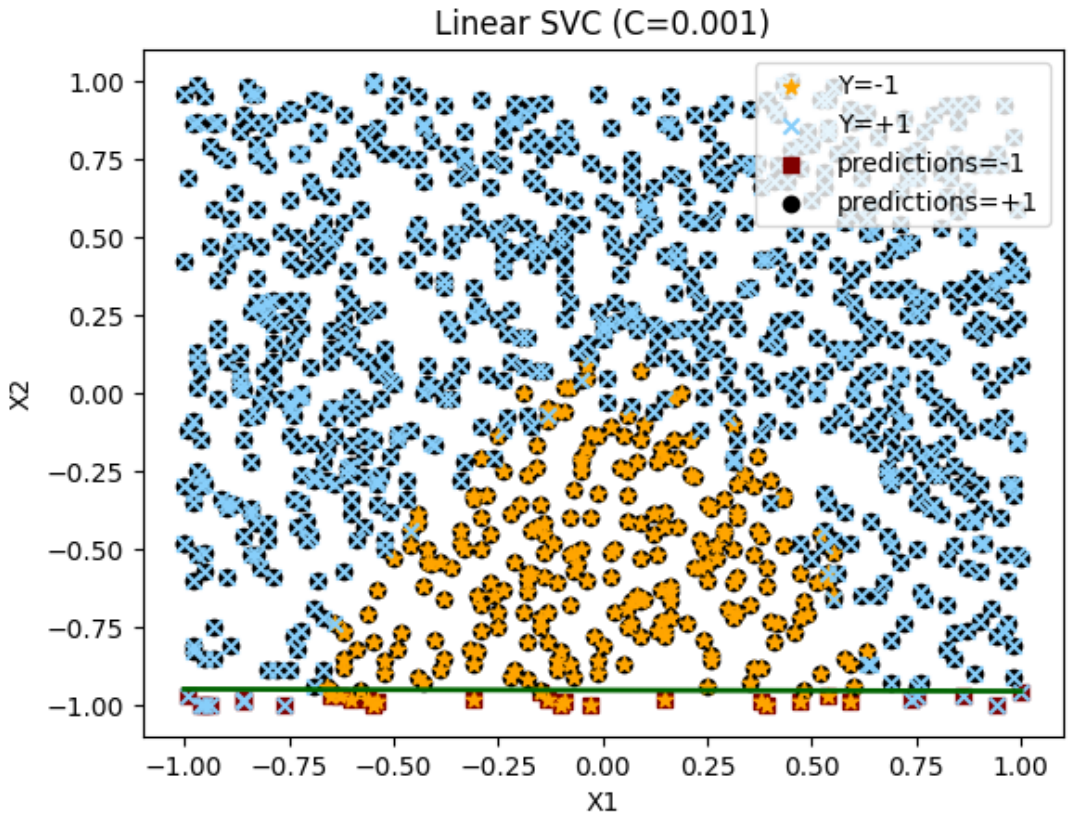
When C=0.001 -> Intercept: [0.32992226] Coefficients: [[0.00112062, 0.34625132]]
When C=0.1 -> Intercept: [0.65695954] Coefficients: [[-0.00279709, 1.2533749]]
When C=1 -> Intercept: [0.6885337] Coefficients: [[-0.00291764, 1.33082951]]
When C=100 -> Intercept: [0.69256843] Coefficients: [[-0.00292697, 1.34050896]]

A positive intercept, small feature1 coefficient and a large feature2 coefficient, are obtained here. This is line sync with the values obtained for logistic regression in a(ii). Increasing value of C, increases intercept and feature2 coefficient, forcing model to better fit the dataset, is notable.

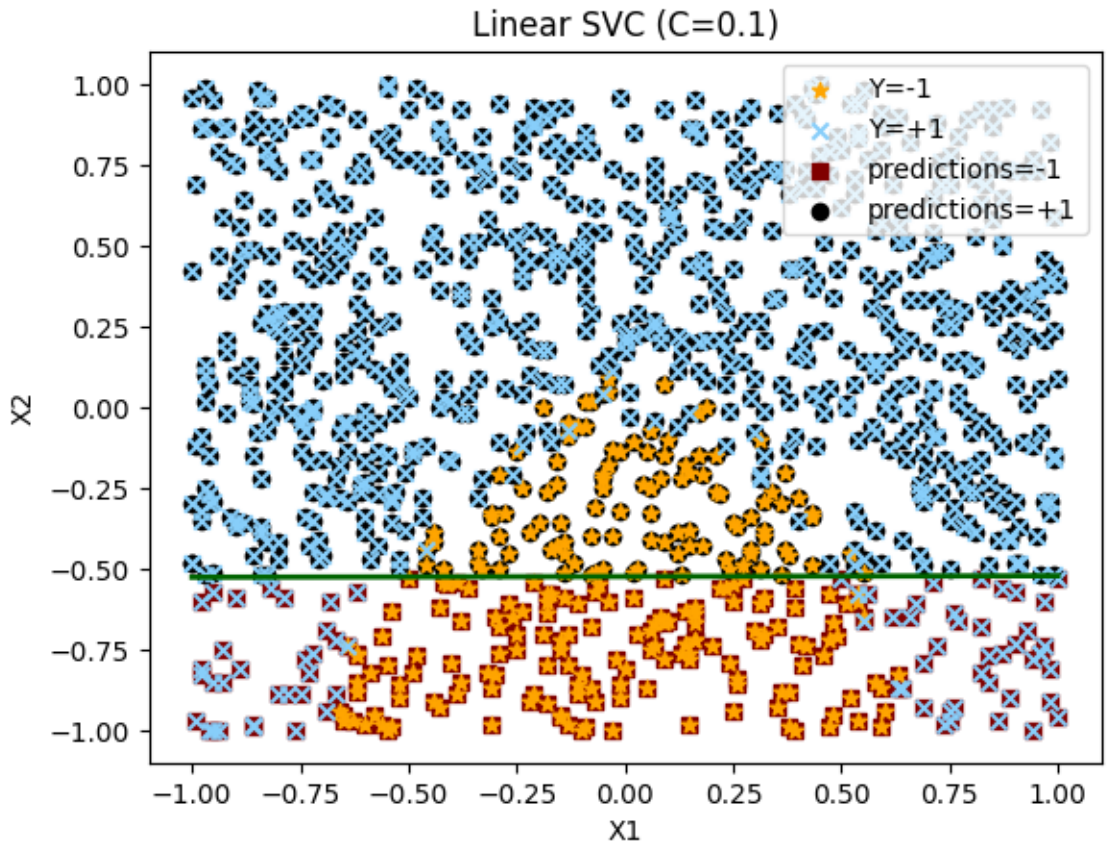
(b) (ii)

Decision boundary for all the graphs below is found in similar way, i.e., by substituting the intercept and coefficient values in the model when its outcome is undecided (w1x1+w2x2+b =0). Then, two points were found out, point1 (x1min, corresponding x2), point2 (x1max, corresponding x2). A line was draw through these two points (green line in graph) which is the decision boundary for respective value of C.

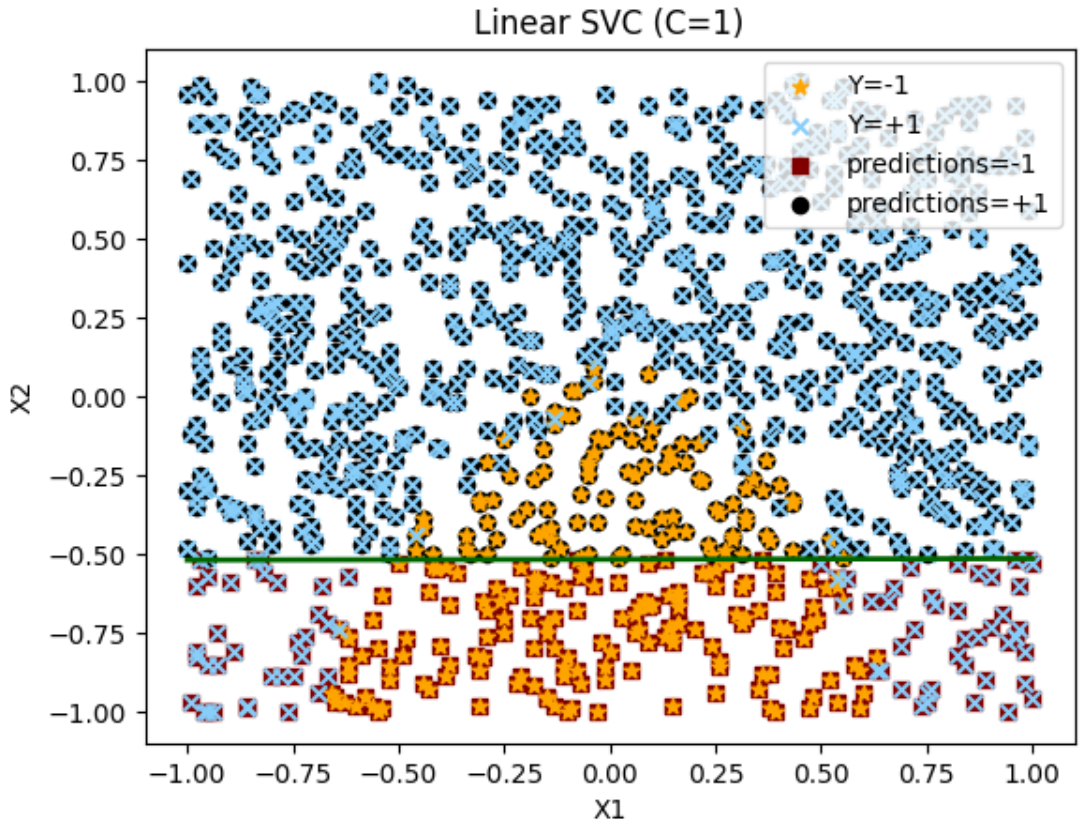
Predictions when C has a small value of 0.001. The graph shows underfitting as it is strongly regularized.



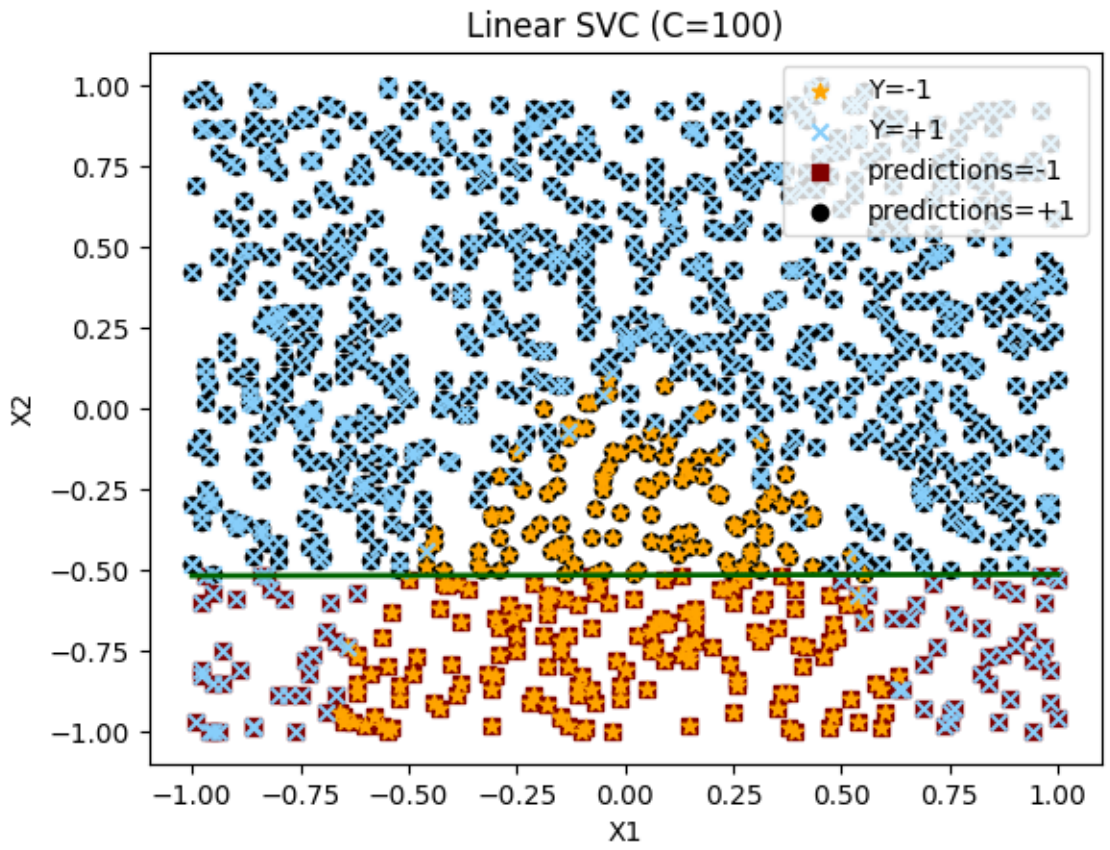
Predictions when C has a value of 0.1. The graph shows comparatively good fit to our dataset which is not linearly separable



Predictions when C has a value of 1.



Predictions when C has a value of 100.



(b) (iii)

We get different predictions for different values of C. The value of C decides the extent of regularization done in the model. An increase in C, as 1/C decreases, corresponds to decrease in regularization leading to better fit to training data. The model parameters keep on increasing with an increase in the value of C until they find a good fit, after which there is little to no change in parameters even with significant increase in value of C. The predictions made by the model also show a similar trend, accuracy of around 75% for lower C (underfitting), which increases to around 82% for optimal C value and then there is no significant change in the accuracy even when C is increased. This is due to our data not being linearly separable.

(b) (iv)

Logistic Regression: Intercept -> [2.02871098]
Coefficients -> [[3.37053745e-03 3.84563071e+00]]
Accuracy: 0.8218218218218218

SVC for best fit (C=0.1): Intercept: [0.65695954]
Coefficients: [[-0.00279709, 1.2533749]]
Accuracy: 0.8218218218218218

As the equations for logistic regression and SVM are different, we get different parameters for both models.
It is seen that for C=0.1, accuracy obtained using SVC is identical with the accuracy obtained using logistic regression. This is because both algorithms try to divide dataset linearly and our dataset not being linearly separable can best achieve around 82% accuracy for given features.

(c) (i)

Two new features were created → X3 = X1*X1 & X4 = X2*X2
A logistic regression model was trained with all the features (old features – X1 X2; new features – X3 X4).
Code:

```
X3 = X1*X1
X4 = X2*X2

X_extraF = np.column_stack((X,X3,X4))

LRmodel = LogisticRegression()
LRmodel.fit(X_extraF,y)
```

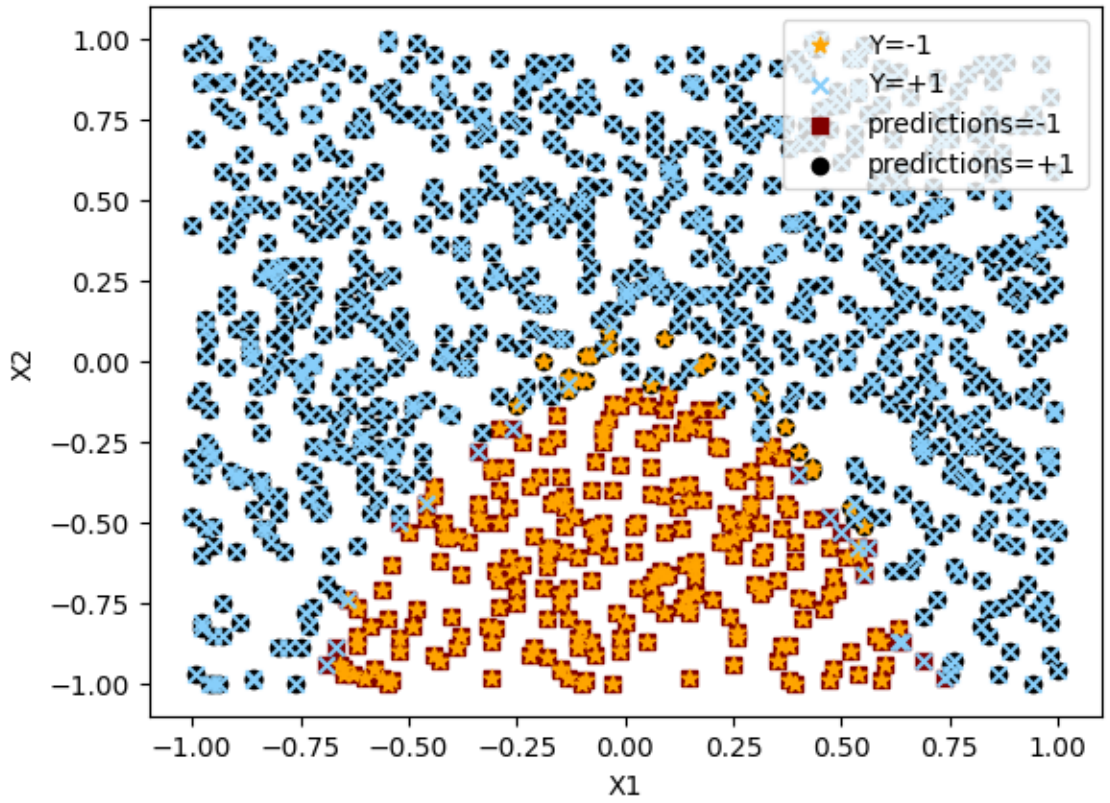
Below are the model parameters when trained with all four features:

Intercept -> [0.42026481]
Coefficients -> [[0.01169035, 5.5201553, 8.12152714, 0.49224507]]

Without feature engineering, X2 dominated the weight in prediction as seen in a(ii), but now it can be seen that, features X2 and X3 (X1 squared) both have major impact on the predictions. The model intercept also sees a significant decrease, i.e., from around 1.9 in 2-feature model to 0.4 in 4-feature model, indicating decrease in auto-bias of +1 class.

(c) (ii)

New model trained with 4 features was used for predicting target values in the training data. Below graph was obtained by plotting these predictions along with actual target values using the same plotting style as before.



Adding new features (Feature engineering) have improved the model to a very good extent. Predictions now have an improved accuracy of 96% signifying the importance of feature engineering. The decision boundary is no more linear (unlike models trained in (a) & (b)) which makes a much better fit to our dataset.

(c) (iii)

A baseline predictor was created to predict the most common class (y = +1 in our dataset case)
Baseline Predictor Accuracy → 74.47 %
4-Feature Model Accuracy → 96.19

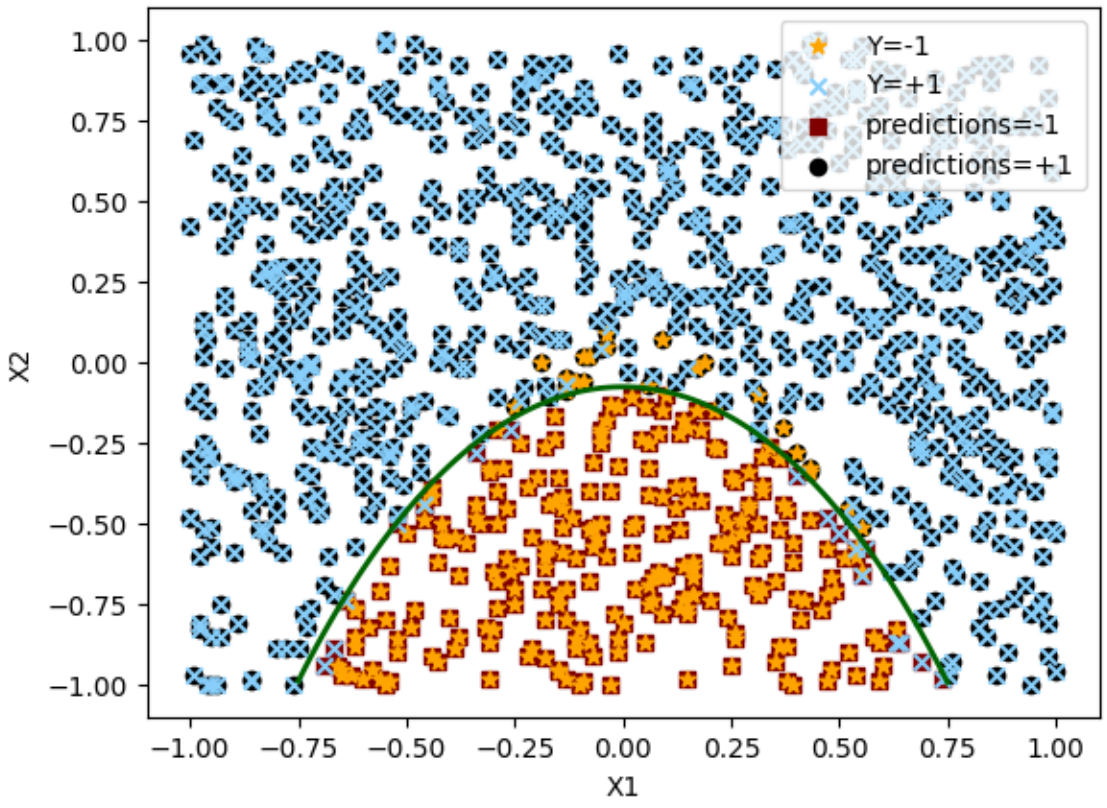
Thus, our engineered model performs better than baseline as well as performs better than basic linear regression and SVC models without feature engineering.

(c) (iv)

To plot the decision boundary for 4-feature logistic regression we need to do below math.
At decision boundary z= w1*x1 + w2*x2 + w3*x1^2 + w4*x2^2 + b, will become 0.
Thus, w1*x1 + w2*x2 + w3*x1^2 + w4*x2^2 + b = 0.
Now, we fix x1 and find corresponding values of x2 to plot the curve.
As x1 is fixed, we have value of (w1*x1 + w3*x1^2 + b), assume this value to be C (w1*x1 + w3*x1^2 + b = C).
For simplicity assume w2=A, w4=B. Thus, we get a quadratic equation in x2 → A*x2^2 + B*x2 + C = 0.
Two roots of this equation are given by; [(-B+sqrt (B^2 – 4*A*C)) / 2*A] & [(-B-sqrt (B^2 – 4*A*C)) / 2*A]

We find roots of this quadratic equation, which will be the value of x2 on decision boundary.
If root lies in the x2 limit for our graph, i.e., -1 <= x2 <= 1, then we plot it with corresponding value of x1.
If root lies outside the x2 limit, we ignore it.

Thus, plotting all values of x1 and x2 in this way we get the decision boundary as below



Appendix:

Imports →

```
import pandas as pd
import numpy as np
df = pd.read_csv("week2.csv", header=None)
print(df.head())
X1 = df.iloc[:,0]
X2 = df.iloc[:,1]
X = np.column_stack((X1,X2))
y = df.iloc[:,2]



# a(i)
import matplotlib.pyplot as plt
# negatives = plt.scatter(X1[y<0],X2[y<0], marker='o', color='red',label='Y=-1')
# positives = plt.scatter(X1[y>0],X2[y>0], marker='x', color='green', label='Y=+1')
negatives = plt.scatter(X1[y<0],X2[y<0], marker='*', color='orange',label='Y=-1')
positives = plt.scatter(X1[y>0],X2[y>0], marker='x', color='lightskyblue', label='Y=+1')
plt.xlabel("X1")
plt.ylabel("X2")
plt.legend(handles=[negatives,positives], loc="upper right")

# a(ii)

from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(penalty=None)
logreg.fit(X,y)
print("Intercept -> ", logreg.intercept_)
print("Coefficients -> ", logreg.coef_)
predictions = logreg.predict(X)
# predictions
accuracy = (predictions == y).mean()
print("Accuracy : " ,accuracy)

# a(iii)

y_predict = logreg.predict(X)
predict_N = plt.scatter(X1[y_predict<0],X2[y_predict<0], marker='s', color='maroon',label='predictions=-1')
predict_P = plt.scatter(X1[y_predict>0],X2[y_predict>0], marker='o', color='black', label='predictions=+1')
input_N = plt.scatter(X1[y<0],X2[y<0], marker='*', color='orange',label='Y=-1')
input_P = plt.scatter(X1[y>0],X2[y>0], marker='x', color='lightskyblue', label='Y=+1')
plt.xlabel("X1")
plt.ylabel("X2")
plt.legend(handles=[input_N,input_P, predict_N, predict_P], loc="upper right")
b = logreg.intercept_
w1, w2 = logreg.coef_[0]
m = - (w1/w2)
c = - (b/w2)
# print("slope - " , m)
```

```
x1_range = np.array([X1.min(),X1.max()])
corresponding_x2 = m*x1_range + c
# print("corresponding_x2 - " , corresponding_x2)
plt.plot(x1_range,corresponding_x2 , c="darkgreen",lw=2)
plt.show()
```

```
# b(i)
```

```
from sklearn.svm import LinearSVC as svc
svc1 = svc(C=0.001).fit(X,y)
svc3 = svc(C=0.1).fit(X,y)
svc4 = svc(C=1).fit(X,y)
svc5 = svc(C=100).fit(X,y)
print("When C=0.001 -> ","Intercept : ",svc1.intercept_ ,"Coefficients : ", svc1.coef_ )
print("When C=0.1   -> ","Intercept : ",svc3.intercept_ ,"Coefficients : ", svc3.coef_ )
print("When C=1     -> ","Intercept : ",svc4.intercept_ ," Coefficients : ", svc4.coef_ )
print("When C=100   -> ","Intercept : ",svc5.intercept_ ,"Coefficients : ", svc5.coef_ )
print("------------------------------------------------------------")
# Calculating Accuracy
predict = svc1.predict(X)
print("When C=0.001 -> ","Accuracy : ",(predict == y).mean())
predict = svc3.predict(X)
print("When C=0.1   -> ","Accuracy : ",(predict == y).mean())
predict = svc4.predict(X)
print("When C=1     -> ","Accuracy : ",(predict == y).mean())
predict = svc5.predict(X)
print("When C=100   -> ","Accuracy : ",(predict == y).mean())
```

```
# B(ii)
```

```
# C = 0.001 graph
y_predict = svc1.predict(X)
predict_N = plt.scatter(X1[y_predict<0],X2[y_predict<0], marker='s', color='maroon',label='predictions=-1')
predict_P = plt.scatter(X1[y_predict>0],X2[y_predict>0], marker='o', color='black', label='predictions=+1')
input_N = plt.scatter(X1[y<0],X2[y<0], marker='*', color='orange',label='Y=-1')
input_P = plt.scatter(X1[y>0],X2[y>0], marker='x', color='lightskyblue', label='Y=+1')
plt.xlabel("X1")
plt.ylabel("X2")
plt.legend(handles=[input_N,input_P, predict_N, predict_P], loc="upper right")
b = svc1.intercept_
w1, w2 = svc1.coef_[0]
m = - (w1/w2)
c = - (b/w2)
```

```
x1_range = np.array([X1.min(),X1.max()])
corresponding_x2 = m*x1_range + c
plt.plot(x1_range,corresponding_x2 , c="darkgreen",lw=2)
```

```
plt.title("Linear SVC (C=0.001)")
plt.show()


# B(ii)

# C = 0.1
y_predict = svc3.predict(X)
predict_N = plt.scatter(X1[y_predict<0],X2[y_predict<0], marker='s', color='maroon',label='predictions=-1')
predict_P = plt.scatter(X1[y_predict>0],X2[y_predict>0], marker='o', color='black', label='predictions=+1')
input_N = plt.scatter(X1[y<0],X2[y<0], marker='*', color='orange',label='Y=-1')
input_P = plt.scatter(X1[y>0],X2[y>0], marker='x', color='lightskyblue', label='Y=+1')
plt.xlabel("X1")
plt.ylabel("X2")
plt.legend(handles=[input_N,input_P, predict_N, predict_P], loc="upper right")
b = svc3.intercept_
w1, w2 = svc3.coef_[0]
m = - (w1/w2)
c = - (b/w2)
# print("slope - " , m)
x1_range = np.array([X1.min(),X1.max()])
corresponding_x2 = m*x1_range + c
# print("corresponding_x2 - " , corresponding_x2)
plt.plot(x1_range,corresponding_x2 , c="darkgreen",lw=2)
plt.title("Linear SVC (C=0.1)")
plt.show()


# B(ii)

# C = 1 graph
y_predict = svc4.predict(X)
predict_N = plt.scatter(X1[y_predict<0],X2[y_predict<0], marker='s', color='maroon',label='predictions=-1')
predict_P = plt.scatter(X1[y_predict>0],X2[y_predict>0], marker='o', color='black', label='predictions=+1')
input_N = plt.scatter(X1[y<0],X2[y<0], marker='*', color='orange',label='Y=-1')
input_P = plt.scatter(X1[y>0],X2[y>0], marker='x', color='lightskyblue', label='Y=+1')
plt.xlabel("X1")
plt.ylabel("X2")
plt.legend(handles=[input_N,input_P, predict_N, predict_P], loc="upper right")
b = svc4.intercept_
w1, w2 = svc4.coef_[0]
m = - (w1/w2)
c = - (b/w2)
# print("slope - " , m)
x1_range = np.array([X1.min(),X1.max()])
corresponding_x2 = m*x1_range + c
# print("corresponding_x2 - " , corresponding_x2)
plt.plot(x1_range,corresponding_x2 , c="darkgreen",lw=2)
plt.title("Linear SVC (C=1)")
plt.show()
```

```
# B(ii)

# C = 100 graph
y_predict = svc5.predict(X)
predict_N = plt.scatter(X1[y_predict<0],X2[y_predict<0], marker='s', color='maroon',label='predictions=-1')
predict_P = plt.scatter(X1[y_predict>0],X2[y_predict>0], marker='o', color='black', label='predictions=+1')
input_N = plt.scatter(X1[y<0],X2[y<0], marker='*', color='orange',label='Y=-1')
input_P = plt.scatter(X1[y>0],X2[y>0], marker='x', color='lightskyblue', label='Y=+1')
plt.xlabel("X1")
plt.ylabel("X2")
plt.legend(handles=[input_N,input_P, predict_N, predict_P], loc="upper right")
b = svc5.intercept_
w1, w2 = svc5.coef_[0]
m = - (w1/w2)
c = - (b/w2)
# print("slope - " , m)
x1_range = np.array([X1.min(),X1.max()])
corresponding_x2 = m*x1_range + c
# print("corresponding_x2 - " , corresponding_x2)
plt.plot(x1_range,corresponding_x2 , c="darkgreen",lw=2)
plt.title("Linear SVC (C=100)")
plt.show()



# c(i)
X3 = X1*X1
X4 = X2*X2
X_extraF = np.column_stack((X,X3,X4))
LRmodel = LogisticRegression()
LRmodel.fit(X_extraF,y)
print("Intercept -> ", LRmodel.intercept_)
print("Coefficients -> ", LRmodel.coef_)


# c(ii)

new_predict = LRmodel.predict(X_extraF)
predict_N = plt.scatter(X1[new_predict<0],X2[new_predict<0], marker='s', color='maroon',label='predictions=-1')
predict_P = plt.scatter(X1[new_predict>0],X2[new_predict>0], marker='o', color='black', label='predictions=+1')
input_N = plt.scatter(X1[y<0],X2[y<0], marker='*', color='orange',label='Y=-1')
input_P = plt.scatter(X1[y>0],X2[y>0], marker='x', color='lightskyblue', label='Y=+1')
plt.xlabel("X1")
plt.ylabel("X2")
plt.legend(handles=[input_N,input_P, predict_N, predict_P], loc="upper right")
plt.show()
```

```
# c(iii)
baseline = max((y == -1).mean(),(y == 1).mean())
print("Baseline Predictor Accuracy - " , baseline)
accuracy = (new_predict == y).mean()
print("4 Feature Model Accuracy - " ,accuracy)


# c(iv)

w1,w2,w3,w4 = LRmodel.coef_.T
interC = LRmodel.intercept_
def find_X2(value):
# fix x1
a = w4
B = w2
c = w1*value + w3*(value**2) + interC
# Find determinant
D = (B**2) - (4*a*c)
if D < 0:
  return None
else:
  r1 = (-B+np.sqrt(D)) / (2*a)
  r2 = (-B-np.sqrt(D)) / (2*a)

    # check root range
    if r1 >= -1 and r1 <= 1:

      return r1

    else:

      return r2
x1_point = np.linspace(-0.75,0.75, 100)
x2_root = []
for x in x1_point:
  root = find_X2(x)
  if root is not None:
    x2_root.append(root)
x2_root = np.array(x2_root)

predict_N = plt.scatter(X1[new_predict<0],X2[new_predict<0], marker='s', color='maroon',label='predictions=-1')
predict_P = plt.scatter(X1[new_predict>0],X2[new_predict>0], marker='o', color='black', label='predictions=+1')
input_N = plt.scatter(X1[y<0],X2[y<0], marker='*', color='orange',label='Y=-1')
input_P = plt.scatter(X1[y>0],X2[y>0], marker='x', color='lightskyblue', label='Y=+1')
plt.xlabel("X1")
plt.ylabel("X2")
plt.plot(x1_point,x2_root ,c="darkgreen",lw=2)
plt.legend(handles=[input_N,input_P, predict_N, predict_P], loc="upper right")
plt.show()
```