CSU44061 Machine Learning – 22336688 – Lucia Brown – Week 2 Assignment

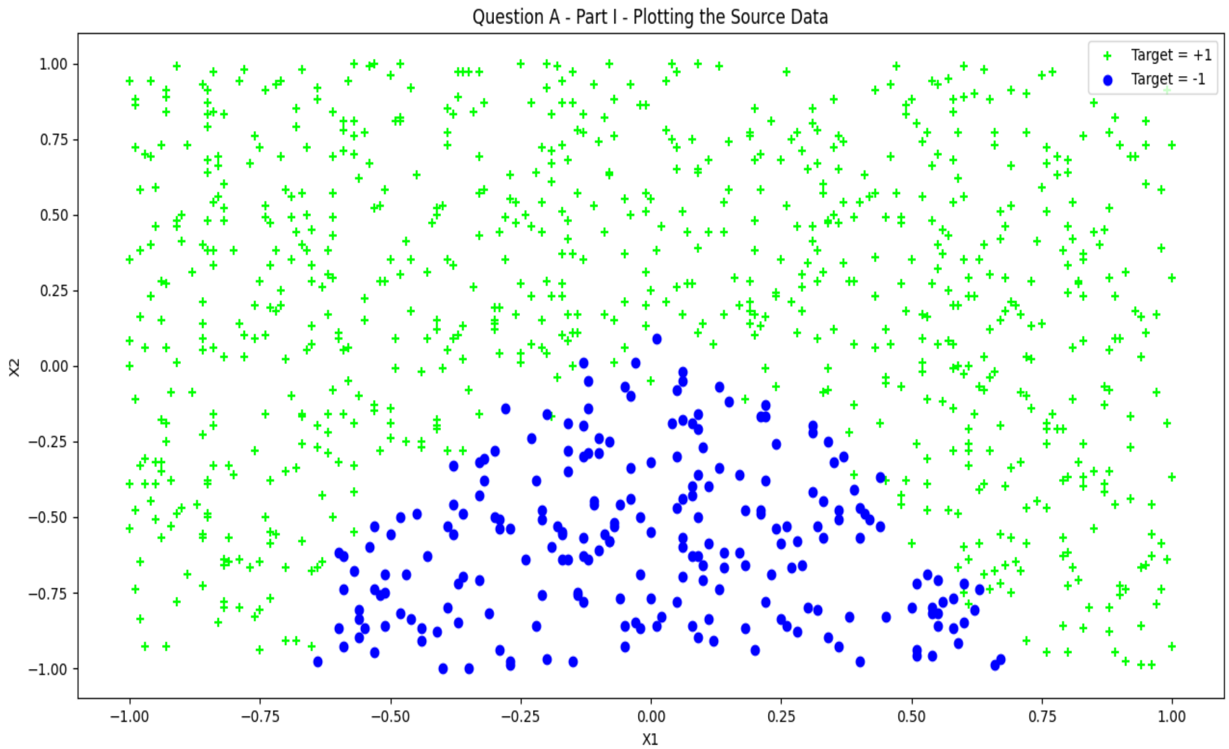
First Line (ID) of downloaded data: # id:7-14-7

****

Question A:

Part I – Visualising the Source Data:

- The source data was visualised using Matplotlib/Pyplot, as per requirements.
- The natural decision boundary of this plot appears to have a curved decision boundary i.e. a negative quadratic shape.



Part II – Training Logistical Regression Classifiers:

- The data was split into training and testing sets (80/20 split) and fit a logistic regression model using sklearn to estimate the probability of the outcome y∈{−1,+1} based on two features X1 and X2.
- The logistic regression model was implemented using `classifier=LogisticRegression(random_state=0)` from sklearn. Model parameters were recorded using `classifier.coef_` and `classifier.intercept_`
- The logistical regression model estimates the probability of the outcome (y) using model parameters:
- b_0 – The intercept – the baseline probability
- The coefficients b_i show the effect of X_i on the probability of achieving the outcome.
- The model produces a coefficient for each feature(X1,X2) indicating how X1,X2 affects the probability of y.
- The 80/20 train/test split was executed and hence is what displays in diagrams throughout this report.

The below results were found on the coefficient model parameter:

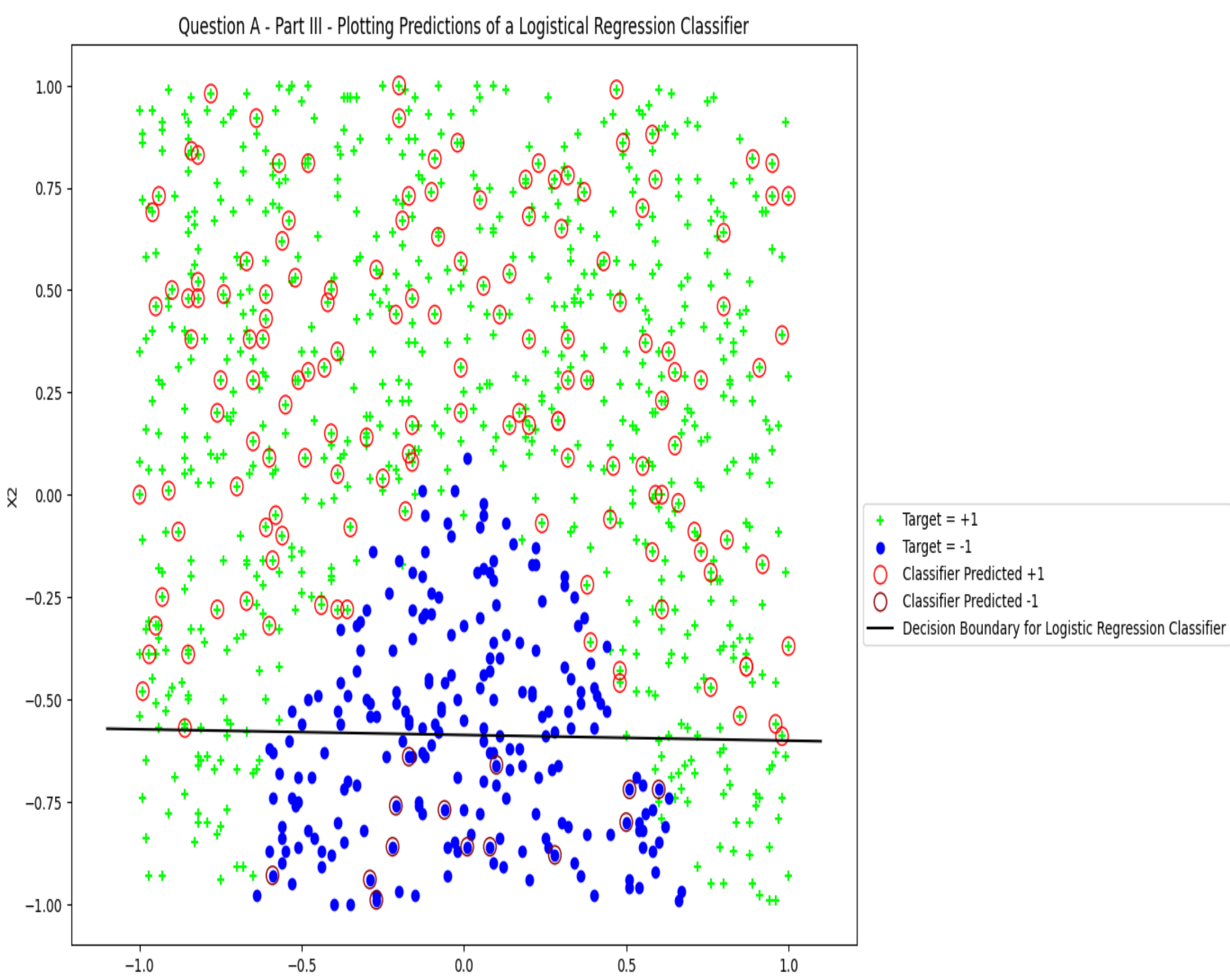
X	Coefficient	Meaning/Effect
X1	0.05	Very small influence on the outcome of y
X2	3.553	Very strong positive influence on y's outcome

- The model baseline b_0 was 2.0855176029767017, meaning that even if X1 and X2 were zero, there is still a decent chance of y being +1.
- It is clear that X2 had the greatest influence on y being +1.
- It is also clear that X1 has the least influence on y being +1.

Part III – Predicting Values with Logistical Regression:

- Predictions were generated for the data set using the trained logistical regression classifier and the 80/20 train/test split.
- To visualize how the classifier separates the two classes, a decision boundary was added.
- The decision boundary corresponds to the points where the model predicts a probability of 0.5 for y=+1. This occurs when the linear combination of inputs equals zero.
- The decision boundary was obtained with the model parameters by creating a straight line on the graph.
- The X1 coordinates for this line span the length of the X1 values from the existing plot by using numPy's linspace: `np.linspace(X1.min()-0.1, X1.max()+0.1, 200)`
- The X2 coordinates for this line were generated for each X1 value according to the following formula: $X2 = -(b_0 + b_1 X1) / b_2$
- The decision boundary represents the distinguishing point in the model. Points below the decision boundary line are identified as –1 and above the line as +1.
- The decision boundary line itself is indifferent between distinguishing –1 and +1 as the y value.

-



Part IV – Predictions vs. Training Data:

- Predictions align extremely well with the training data, predicted points mostly overlap the values on the plot.
- This accuracy is confirmed by sklearn's *accuracy_score* import which gives a score of 0.81 or 81%), indicating that the linear decision boundary captures the majority of the patterns in the data.
- Misclassifications do not seem to appear in the plotted data, the only predictions which remained unclassified from the training data are +1 values that appear below the decision boundary and –1 values that appear above the decision boundary.
- Overall, the linear decision boundary is appropriate for this dataset, as the classes are largely separable along a straight line.

****

Question B:

Part I – Training SVM Classifiers:

SVM Model for Predictions:

```
modelZero =LinearSVC(C=0.001)
modelOne=LinearSVC(C=1)
modelHundred =LinearSVC(C=100)
```

- Three linear SVM classifiers were trained at 3 different penalty parameters (C). The models use the sklearn functionality LinearSVC().
- Small C (0.001) tolerates more misclassifications.
- Larger C (100) fits the decision boundary more closely to the training data.
- The 80/20 test/train split was used.
- The margin is a buffer-zone around the decision boundary which SVM uses to separate the features. The C parameter dictates how strictly the SVM decision boundary is adhered to.

The model parameters for the three different models are given below:

C	**b_0**	**b_1**	**b_2**	**Meaning**
0.001	0.33161957	-0.00367932	0.29394636	Very small influence on X1, moderate influence on X2
1	0.72318188	0.01327877	1.2818011	X2 has strong influence, X1 minor
100	0.728805	0.01349708	1.29467031	Similar to C=1

- Similar trends are observed in SVM as in Logistical Regression.
- Higher values of C seem to produce model parameter outputs more close to that of Logistical Regression.

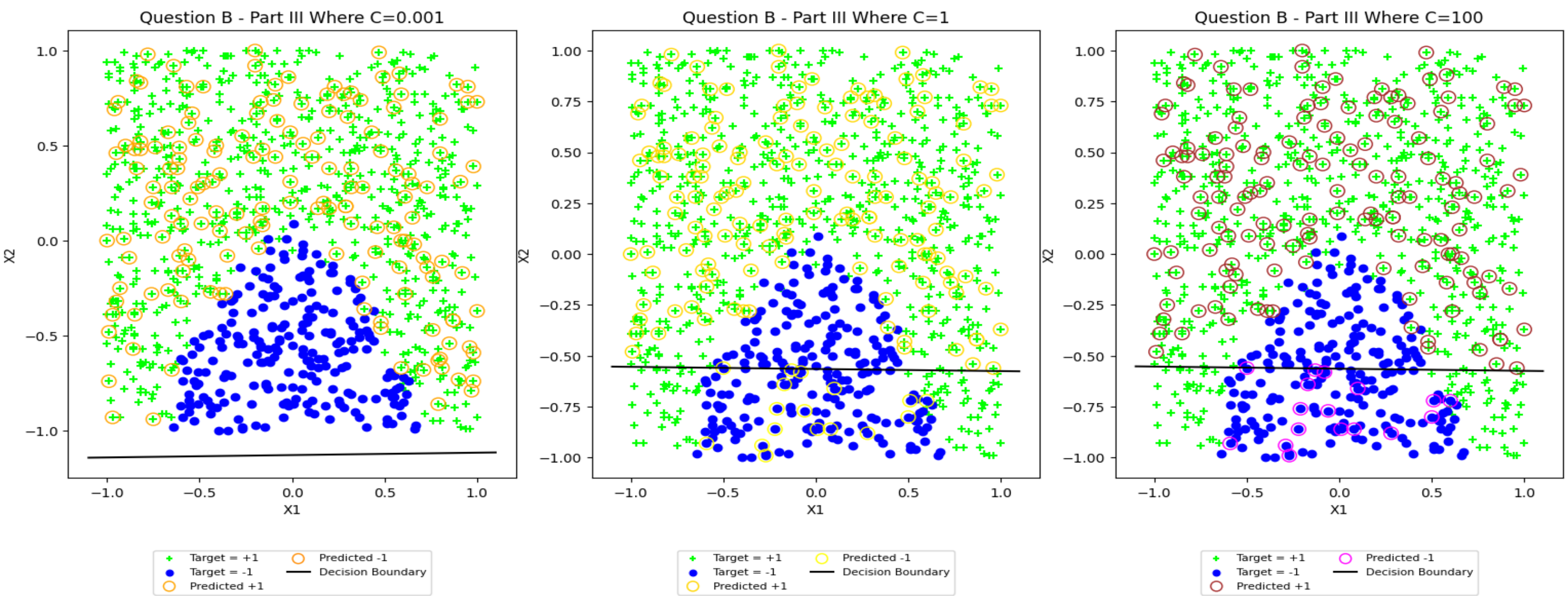
Part II – Plotting Predictions with SVM Data:

- Each model creates a straight line that divides the two classes.
- Points on one side of the line are predicted as belonging to the +1 class. Points on the other side are predicted as belonging to the –1 class.
- With C = 0.001, the line is based on the model parameters and may misclassify points.
- The decision boundary line is established using the following formula: (b0 + b1*X1 + b2*X2 = 0). i.e. where the SVM function equals zero.
- This formula was achieved in sklearn and numpy by taking a linspace of the X1 values (creating the X values of the decision boundary line) then applying the formula (on the model parameters for each of the three models) to get the y values of the decision boundary line.

```
x_vals = np.linspace(X1.min()-0.1, X1.max()+0.1, 200)
y_vals_Zero = -(b0_zero + b1_zero * x_vals) / b2_zero
```

```
y_vals_One = -(b0_one + b1_one * x_vals) / b2_one
y_vals_Hundred = -(b0_hundred + b1_hundred * x_vals) / b2_hundred
```

- With C = 1 and C=100 an extremely similar decision boundary line was generated. This means that the data is already fairly well-separated, so increasing C further doesn't change the model's boundary. This also means that the dataset is fairly clean and free from significant noise.



<u>Part III – Impact on Model Parameters by Changing C</u>:

- The C parameter controls the trade-off between the size of the model parameters and how accurately the training data is classified.
- The impact on the model parameters by changing C is small sizes of C lead to smaller coefficients and thus more mistakes in the decision boundary line.
- 1/C acts as the L_2 penalty weight, penalizing large coefficients.
- Larger values of C lead to larger coefficients and less mistakes, with a more accurate decision boundary line. When the most accurate line is reached, increasing C has minimal effects.
- In this dataset, the boundary is very imprecise for C=0.001 and the larger C=1,C=100 produce a more accurate boundary.

<u>Part IV – SVM vs. Logistical Regression:</u>

Model Parameters:

Parameter	**SVM – C=1 Example**	**Logistical Regression**
Intercept	0.72318188	2.0855176029767017
X1 Coefficient	0.01327877	0.05
X2 Coefficient	1.2818011	3.553

- Both the SVM and Logistical Regression models agree that the X2 feature has the most influence overall and is the stronger predictor.
- Logistical Regression coefficients and intercept are noticeably larger than that of SVM – these Logistical Regression numbers are based off probabilities while SVM numbers are weights for creating the best decision boundary between the classes.

Predictions:

Model	Accuracy Score
Logistical Regression	0.805
SVM C=0.001	0.8
SVM C=1	0.81
SVM C=100	0.81

- Both models appear to perform approximately equal.
- Tuning C in the SVM model does not increase accuracy by much.
- The data is quite well handled by the simple linear logistical regression
-

****

Question C:

Part I – Adding the Square of Each Feature:

- The square of X1 and the square of X2 gives a total of four features (X1, X2, $X1^2$, $X2^2$)
- A model was trained using a Logistic Regression in sklearn with the above features, similar to what was carried out in Question A with an 80/20 train/test split.

The model is given as follows:

```
classifierSquare=LogisticRegression(random_state=0)
classifierSquare.fit(X_trainSquare,y_trainSquare)
y_predSquare = classifierSquare.predict(X_testSquare)
```

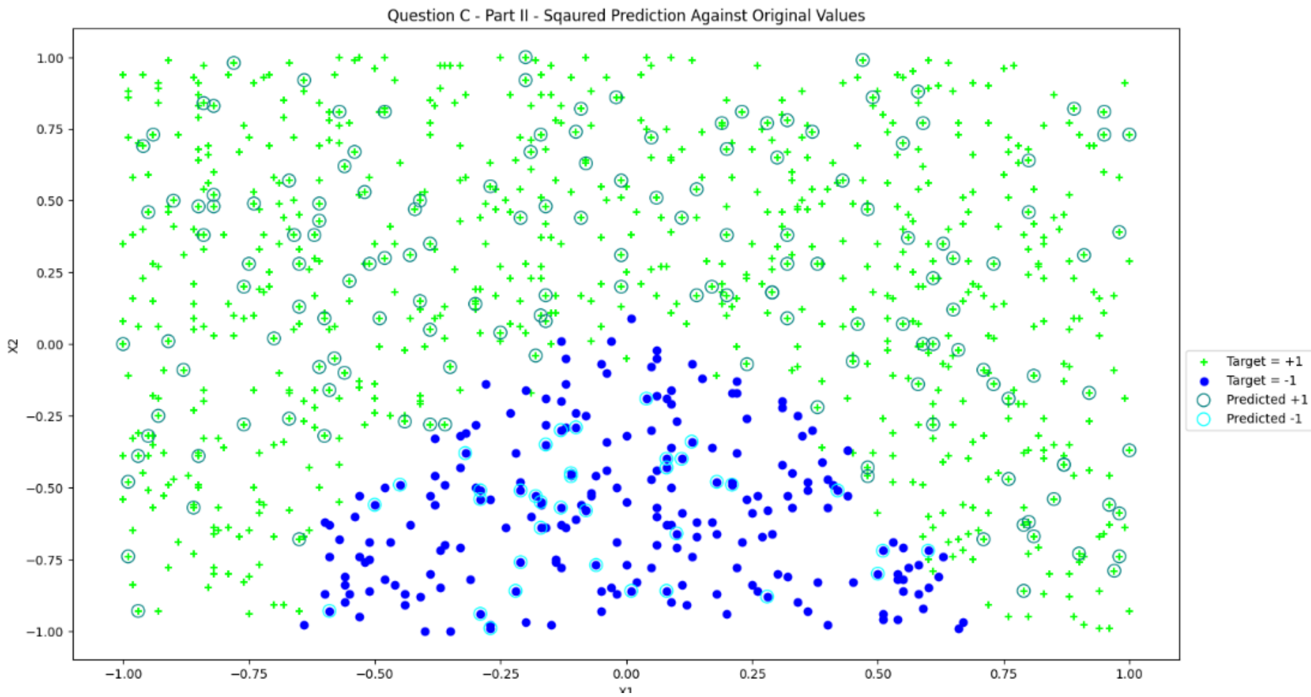
The model parameters are given as follows:

Model Parameter	Value	Meaning
Intercept (b_0)	0.6047	Baseline when all features are 0
X1 (b_1)	0.1405	Small positive influence on y = +1
X2 (b_2)	5.1718	String positive influence on y = + 1
$X1^2$ (b_1^2)	7.4917	Very strong positive influence on y = +1
$X2^2$ (b_2^2)	0.1499	Slight positive influence on y = +1

- The model achieves a test accuracy of 0.97 or 97% indicating that adding quadratic features improves flexibility while capturing nonlinear data.
- The large coefficient for $X1^2$ suggests that the outcome is highly sensitive to larger values of X1.
- X2 remains important but squared effect is small.

Part II – Plotting Predictions with Squared Data:

The model is visibly now more capable of making accurate predictions, as seen in the below plot – there is a clear curved decision boundary compared to the linear decision boundaries of previous questions.



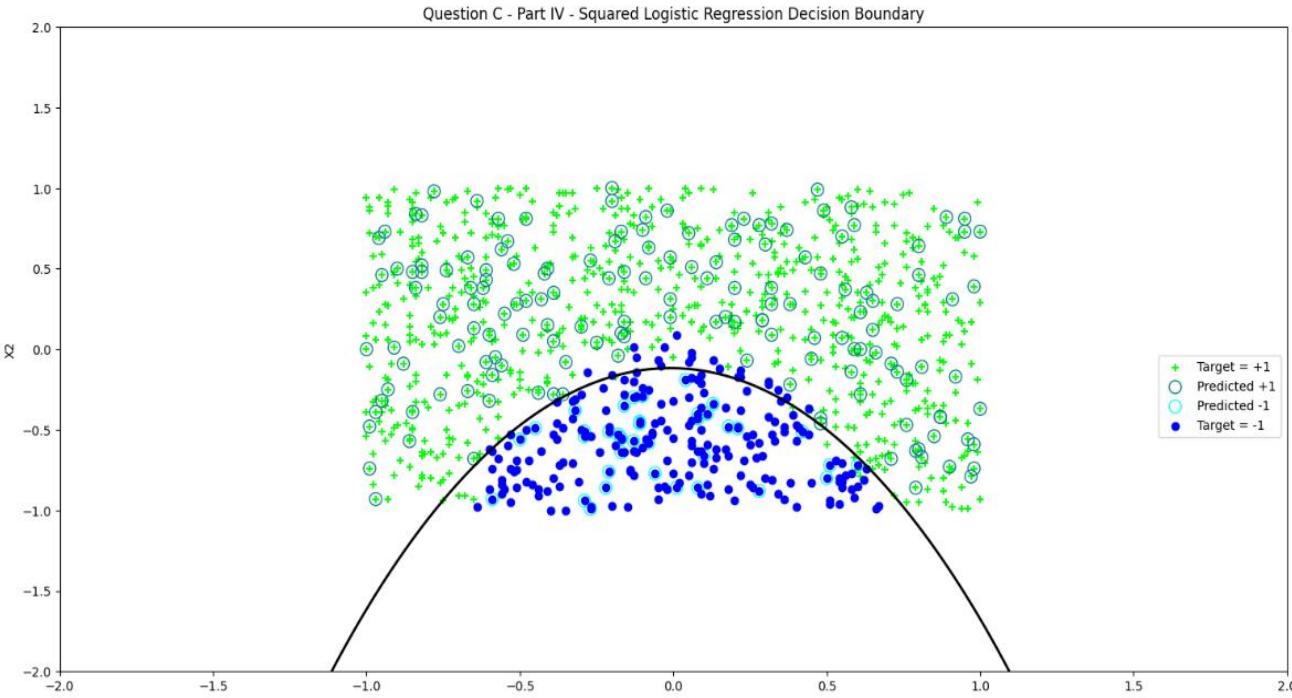
Part III – Comparing Classifier Performance Against Reasonable Baseline:

- The baseline parameter chosen always chooses the most frequent class from the training data, achieved by using Python Collections' Counter import with the .most_common() method.
- By applying the accuracy_score feature to this baseline, it achieves an accuracy of 0.8 or 80%.
- Any meaningful classifier should achieve at least this score.
- By comparison, the squared logistical regression classifier achieves an accuracy score of 0.97 or 97%.
- This proves the benefit of the logistical regression classifier over brute-force calculating the most common class.

Part IV – Plotting the Decision Boundary for a Squared Version of the Source Data:

- The decision boundary for the quadratic/squared version of this data was taken by numPy's meshgrid functionality to combine the space produced by X1 and the space produced by X2.
- The sigmoid is also computed from this linear combination. I.e. it is the probability (prob) that a data point belongs to +1.
- Thus, a linear combination could be created as follows, as a contour in pyPlot, not a line.

```
linearCombination = b0c + b1c*X1_grid + b2c*X2_grid + b1csquared*X1_grid**2 + b2csquared*X2_grid**2
prob = 1 / (1 + np.exp(-linearCombination))
plt.contour(X1_grid, X2_grid, prob, levels=[0.5], colors='black', linewidths=2, label="Decision Boundary for Squared
Logistic Regression")
```



Question C - Part IV - Squared Logistic Regression Decision Boundary
Target = +1
Predicted +1
Predicted -1
Target = -1
X2
X1

Appendix/Code:

```
# First line of the data file: # id:7-14-7

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.svm import LinearSVC
from collections import Counter

df =
pd.read_csv("Week2Assignment/week2.php.csv",header=None,comment="#",sep=",",skipinitialspace=True)
#print(df.head())
X1=df.iloc[:,0] # Col1
X2=df.iloc[:,1] # Col2
X=np.column_stack((X1,X2)) # Stack into 2D array
y=df.iloc[:,2]  # Target vals

# QUESTION A
# PART I
plt.figure(figsize=(12,12))
plt.scatter(X1[y==1],X2[y==1],marker="+",color="lime",label="Target = +1")
plt.scatter(X1[y==-1],X2[y==-1],marker="o",color="blue",label="Target = -1")
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Question A - Part I")
plt.legend()
plt.grid(False)
plt.show()

# PART II - https://youtube.com/shorts/8it0yrJzQfU?si=1UoVpwNGf7flDZeh

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=0)
X1_test = X_test[:, 0]
X2_test = X_test[:, 1]

classifier=LogisticRegression(random_state=0)
classifier.fit(X_train,y_train)
y_pred=classifier.predict(X_test)

# Get model parameters
print("Intercept (b0):", classifier.intercept_[0])
print("Coefficients (b1, b2):", classifier.coef_[0])
```

```
for i, coef in enumerate(classifier.coef_[0]):
    effect = "increases" if coef > 0 else "decreases"
    print(f"Feature X{i+1} {effect} the probability of predicting +1
(coefficient={coef:.3f})")

print("Accuracy of Logistical Regression:", round(accuracy_score(y_test, y_pred),7))

# PART III - https://youtu.be/ZsM2z0pTbnk?si=iMnwQTl_QFWL7ani

#Add predictions to plot
plt.figure(figsize=(12,12))
plt.scatter(X1[y==1],X2[y==1],marker="+",color="lime",label="Target = +1")
plt.scatter(X1[y==-1],X2[y==-1],marker="o",color="blue",label="Target = -1")
plt.scatter(X1_test[(y_test == y_pred) & (y_test == 1)], X2_test[(y_test == y_pred) & (y_test
== 1)], facecolors='none', edgecolors='red', s=100, label="Classifier Predicted +1")
plt.scatter(X1_test[(y_test == y_pred) & (y_test == -1)], X2_test[(y_test == y_pred) & (y_test
== -1)], facecolors='none', edgecolors='darkred', s=100, label="Classifier Predicted -1")

# Add decision boundary as line on plot
b0 = classifier.intercept_[0]
b1, b2 = classifier.coef_[0]
x_vals = np.linspace(X1.min()-0.1, X1.max()+0.1, 200)
y_vals = -(b0 + b1 * x_vals) / b2
plt.plot(x_vals, y_vals, color='black', linewidth=1.5, label="Decision Boundary for Logistic
Regression Classifier")

plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Question A - Part III")
plt.legend(bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.grid(False)
plt.show()

# PART IV - SEE REPORT PDF

# QUESTION B

# PART I - https://youtu.be/kPkwf1x7zpU?si=z3680NVTyFd0b_h4
svcScoreDictionary={}
cVals=[0.001,1,100]

# https://youtu.be/joTa_FeMZ2s?si=3DI8TfqasbA9BtgO
# https://youtu.be/5oVQBF_p6kY?si=8PbeAmYH38awD8_9
X_train_SVM,X_test_SVM,y_train_SVM,y_test_SVM =
train_test_split(X,y,test_size=0.2,random_state=0)
modelZero =LinearSVC(C=0.001)
modelOne=LinearSVC(C=1)
modelHundred =LinearSVC(C=100)
modelZero.fit(X_train_SVM, y_train_SVM)
modelOne.fit(X_train_SVM, y_train_SVM)
```

```
modelHundred.fit(X_train_SVM, y_train_SVM)
models = {
    0.001: modelZero,
    1: modelOne,
    100: modelHundred
}

for cVal, model in models.items():
    X1_test_SVM = X_test_SVM[:, 0]
    X2_test_SVM = X_test_SVM[:, 1]
    # Print model parameters
    print(f"\nLinear SVC Model for C = {cVal}")
    print("Model Coefficients:\n", model.coef_)
    print("Model Intercept:", model.intercept_)

# PART II - https://youtu.be/_YPScrckx28?si=fBxs_9gB27Ey7EYp

y_pred_Zero=modelZero.predict(X_test_SVM)
y_pred_One=modelOne.predict(X_test_SVM)
y_pred_Hundred=modelHundred.predict(X_test_SVM)
print("\nAccuracy Score for C=0.001: ",round(accuracy_score(y_test_SVM,y_pred_Zero),7))
print("Accuracy Score for C=1: ",round(accuracy_score(y_test_SVM,y_pred_One),7))
print("Accuracy Score for C=100: ",round(accuracy_score(y_test_SVM,y_pred_Hundred),7),"\n")

# PART III

b0_zero = modelZero.intercept_[0]
b1_zero, b2_zero = modelZero.coef_[0]

b0_one = modelOne.intercept_[0]
b1_one, b2_one = modelOne.coef_[0]

b0_hundred = modelHundred.intercept_[0]
b1_hundred, b2_hundred = modelHundred.coef_[0]

# Add decision boundary as line on plot
x_vals = np.linspace(X1.min()-0.1, X1.max()+0.1, 200)
y_vals_Zero = -(b0_zero + b1_zero * x_vals) / b2_zero

x_vals = np.linspace(X1.min()-0.1, X1.max()+0.1, 200)
y_vals_One = -(b0_one + b1_one * x_vals) / b2_one

x_vals = np.linspace(X1.min()-0.1, X1.max()+0.1, 200)
y_vals_Hundred = -(b0_hundred + b1_hundred * x_vals) / b2_hundred


# PLOT C=0.001
plt.figure(figsize=(12,12))
plt.scatter(X1[y==1], X2[y==1], marker="+", color="lime", label="Target = +1")
plt.scatter(X1[y==-1], X2[y==-1], marker="o", color="blue", label="Target = -1")
```

```
plt.scatter(X1_test_SVM[(y_test_SVM == y_pred_Zero) & (y_test_SVM == 1)],
X2_test_SVM[(y_test_SVM == y_pred_Zero) & (y_test_SVM == 1)],facecolors='none',
edgecolors='orange', s=100, label="Predicted +1")
plt.scatter(X1_test_SVM[(y_test_SVM == y_pred_Zero) & (y_test_SVM == -1)],
X2_test_SVM[(y_test_SVM == y_pred_Zero) & (y_test_SVM == -1)],facecolors='none',
edgecolors='darkorange', s=100, label="Predicted -1")
plt.plot(x_vals, y_vals_Zero, color='black', linewidth=1.5, label="Decision Boundary for
Linear SVC Model where C=0.001")
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Question B - Part III Where C=0.001")
plt.legend(bbox_to_anchor=(1, 0.5))
plt.grid(False)
plt.tight_layout()
plt.show()

# PLOT C=1
plt.figure(figsize=(12,12))
plt.scatter(X1[y==1], X2[y==1], marker="+", color="lime", label="Target = +1")
plt.scatter(X1[y==-1], X2[y==-1], marker="o", color="blue", label="Target = -1")
plt.scatter(X1_test_SVM[(y_test_SVM == y_pred_One) & (y_test_SVM == 1)],
X2_test_SVM[(y_test_SVM == y_pred_One) & (y_test_SVM == 1)],facecolors='none',
edgecolors='gold', s=100, label="Predicted +1")
plt.scatter(X1_test_SVM[(y_test_SVM == y_pred_One) & (y_test_SVM == -1)],
X2_test_SVM[(y_test_SVM == y_pred_One) & (y_test_SVM == -1)],facecolors='none',
edgecolors='yellow', s=100, label="Predicted -1")
plt.plot(x_vals, y_vals_One, color='black', linewidth=1.5, label="Decision Boundary for Linear
SVC Model where C=1")
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Question B - Part III Where C=1")
plt.legend(bbox_to_anchor=(1, 0.5))
plt.grid(False)
plt.tight_layout()
plt.show()

# PLOT C=100
plt.figure(figsize=(12,12))
plt.scatter(X1[y==1], X2[y==1], marker="+", color="lime", label="Target = +1")
plt.scatter(X1[y==-1], X2[y==-1], marker="o", color="blue", label="Target = -1")
plt.scatter(X1_test_SVM[(y_test_SVM == y_pred_Hundred) & (y_test_SVM == 1)],
X2_test_SVM[(y_test_SVM == y_pred_Hundred) & (y_test_SVM == 1)],facecolors='none',
edgecolors='brown', s=100, label="Predicted +1")
plt.scatter(X1_test_SVM[(y_test_SVM == y_pred_Hundred) & (y_test_SVM == -1)],
X2_test_SVM[(y_test_SVM == y_pred_Hundred) & (y_test_SVM == -1)],facecolors='none',
edgecolors='fuchsia', s=100, label="Predicted -1")
plt.plot(x_vals, y_vals_Hundred, color='black', linewidth=1.5, label="Decision Boundary for
Linear SVC Model where C=100")
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Question B - Part III Where C=100")
```

```python
plt.legend(bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.grid(False)
plt.show()

# PART IV - SEE REPORT PDF

# QUESTION C

# PART I
X1Square=X1**2
X2Square=X2**2
XSquare=np.column_stack((X1,X2,X1Square,X2Square)) # Stack into 4D array
X_trainSquare,X_testSquare,y_trainSquare,y_testSquare =
train_test_split(XSquare,y,test_size=0.2,random_state=0)
X1_test = X_testSquare[:, 0]
X2_test = X_testSquare[:, 1]

classifierSquare=LogisticRegression(random_state=0)
classifierSquare.fit(X_trainSquare,y_trainSquare)
y_predSquare = classifierSquare.predict(X_testSquare)

print("Squared Logistic Regression Model Parameters:")
print("Intercept (b0):", classifierSquare.intercept_[0])
print("Coefficients (b1, b2, b1^2, b2^2):", classifierSquare.coef_[0])

accuracySquare = accuracy_score(y_testSquare, y_predSquare)
print("Test Accuracy:", round(accuracySquare, 7))

# PART II
# Original Data - NOT squared
plt.figure(figsize=(12,12))
plt.scatter(X1[y==1], X2[y==1], marker="+", color="lime", label="Target = +1")
plt.scatter(X1[y==-1], X2[y==-1], marker="o", color="blue", label="Target = -1")

plt.scatter(X1_test[(y_testSquare == y_predSquare) & (y_testSquare == 1)],
X2_test[(y_testSquare == y_predSquare) & (y_testSquare == 1)],facecolors='none',
edgecolors='teal', s=100, label="Predicted +1")
plt.scatter(X1_test[(y_testSquare == y_predSquare) & (y_testSquare == -1)],
X2_test[(y_testSquare == y_predSquare) & (y_testSquare == -1)],facecolors='none',
edgecolors='aqua', s=100, label="Predicted -1")
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Question C - Part II - Sqaured Prediction Against Original Values")
plt.legend(bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.grid(False)
plt.show()

# PART III - Add Baseline Comparison
```

```
mostCommonClass = Counter(y_trainSquare).most_common(1)[0][0] # Most common class in training
set

# Predict most common class
y_predBaseline = np.full_like(y_testSquare,fill_value=mostCommonClass)
baselineAccuracy = accuracy_score(y_testSquare, y_predBaseline)
print("Baseline Accuracy (majority class predictor):", round(baselineAccuracy, 2))
print("Squared Logistic Regression Accuracy:", round(accuracySquare, 2))

# PART IV - Create the decision boundary
b0c = classifierSquare.intercept_[0]
b1c, b2c, b1csquared, b2csquared = classifierSquare.coef_[0]
x1_range = np.linspace(X1.min() - 1, X1.max() + 1, 200)
x2_range = np.linspace(X2.min() - 1, X2.max() + 1, 200)
X1_grid, X2_grid = np.meshgrid(x1_range, x2_range)


linearCombination = b0c + b1c*X1_grid + b2c*X2_grid + b1csquared*X1_grid**2 +
b2csquared*X2_grid**2
prob = 1 / (1 + np.exp(-linearCombination))
plt.figure(figsize=(12,12))
plt.scatter(X1[y==1], X2[y==1], marker="+", color="lime", label="Target = +1")
plt.scatter(X1_test[(y_testSquare == y_predSquare) & (y_testSquare == 1)],
X2_test[(y_testSquare == y_predSquare) & (y_testSquare == 1)],facecolors='none',
edgecolors='teal', s=100, label="Predicted +1")
plt.scatter(X1_test[(y_testSquare == y_predSquare) & (y_testSquare == -1)],
X2_test[(y_testSquare == y_predSquare) & (y_testSquare == -1)],facecolors='none',
edgecolors='aqua', s=100, label="Predicted -1")
plt.scatter(X1[y==-1], X2[y==-1], marker="o", color="blue", label="Target = -1")
plt.contour(X1_grid, X2_grid, prob, levels=[0.5], colors='black', linewidths=2)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Question C - Part IV - Squared Logistic Regression Decision Boundary')
plt.legend(bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.grid(False)
plt.show()
```