

## CS7CS4 Machine Learning

The assigned dataset is: # id:16-16-16

---

### Part A

(i) The provided dataset consists of two numerical features (dubbed  $X_1$  and  $X_2$ ) and a binary target label (that was -1 or +1). The features were plotted into a scatter plot, with the X-axis representing the  $X_1$  feature and the Y-axis representing the  $X_2$  feature. Data points with label as +1 are represented using cyan circles, while those with label as -1 as orange circles for clarity. Here is the graph produced:

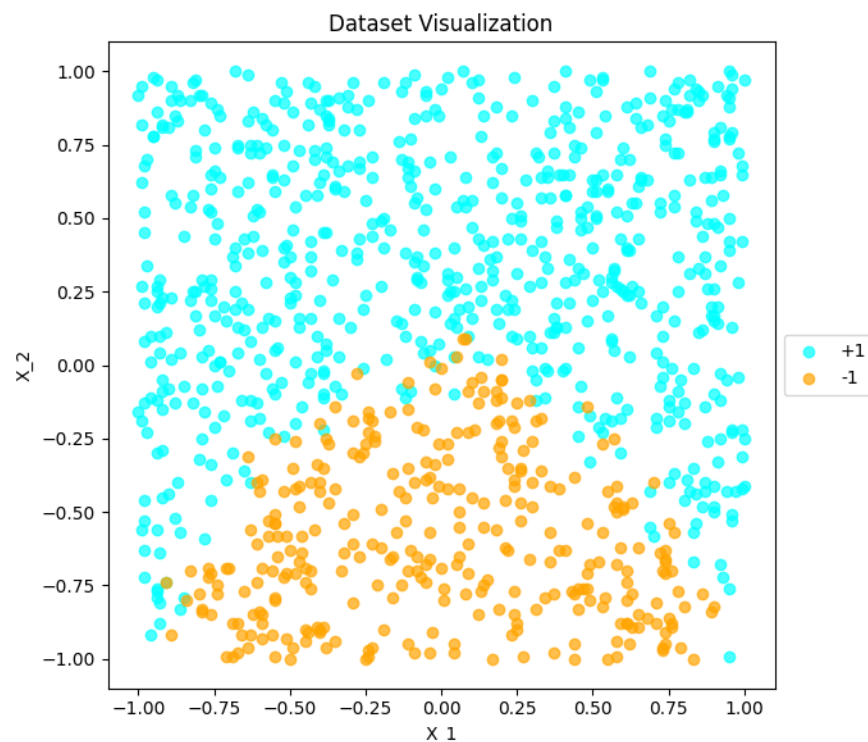


Figure 1. Scatter plot of the given data

The code segment that produced this is present under the [plot\\_given\\_data\(\)](#) function.

(ii) The Logistic Regression model was trained using both the given features, which computes the probability of a point belonging to the +1 class using the sigmoid function  $p(y = +1 | X) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2)}}$  and the -1 class's probability would be  $p(y = -1 | X) = 1 - p(y = +1 | X)$ , and  $f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$  represents the linear combination of the features  $X_1$  and  $X_2$ .

After fitting, the feature coefficients that were produced were  $\theta_1 = -0.05339$  and  $\theta_2 = 5.15001$ , while the intercept is  $\theta_0 = 1.69049$ . The accuracy of this model compared to the actual data produced a score of 0.87888 i.e. approximately 87.9%. The interpretation of the coefficients is as follows:

- $\theta_1$  is negative but small, which implies that increasing feature  $X_1$  would slightly decrease the probability of predicting +1.
- $\theta_2$  is positive and larger, which implies that increasing feature  $X_2$  would increase the probability of predicting +1.
- As such, since  $|\theta_1| < |\theta_2|$ , feature  $X_2$  would have the most influence on prediction.

This analysis aligns with how the data is structured in [Figure 1](#), where the data points appear to be vertically segregated into two distinct regions.

The code segment that produced this is present under the [train\\_log\\_regr\(\)](#) and [make\\_predictions\(\)](#) functions.

(iii) The trained model's predictions were plotted on top of the scatter plot produced by the provided data, with the predicted +1 labels appearing as red '+' markers and the predicted -1 labels appearing as green 'x' markers to display a comparison between the two:

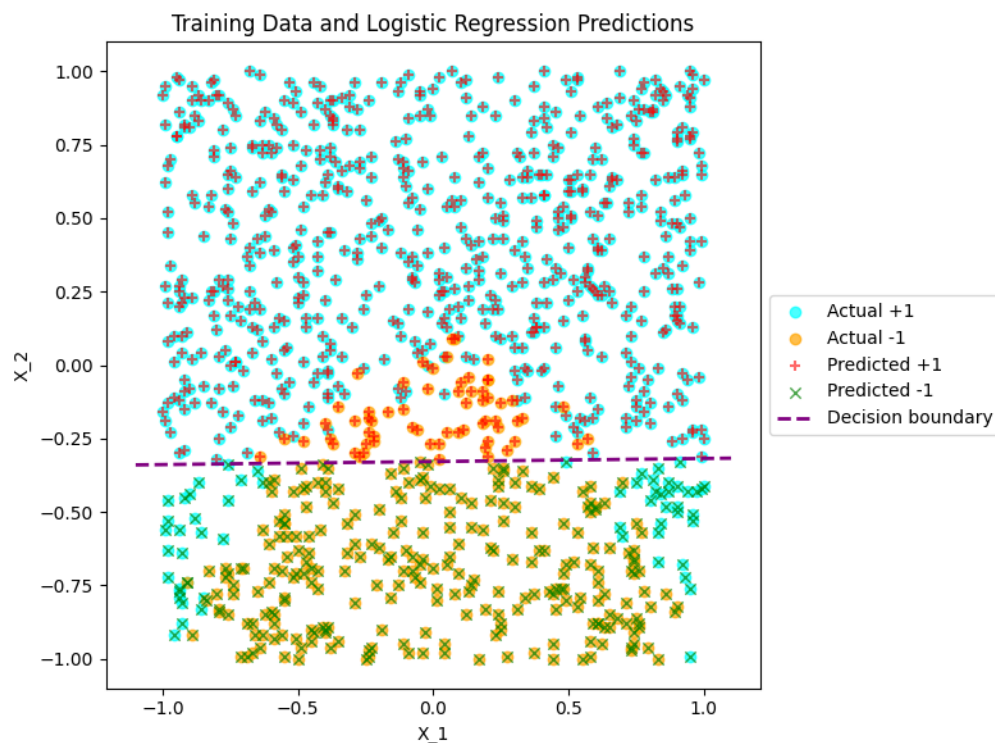


Figure 2. Scatter plot representing given data and predictions of Logistic Regression model

As mentioned in [Part A \(ii\)](#), the tendency of the predicted data showed an 87.9% overlap with the data points from the given dataset, with the predicted data being more distinctively separated into two sections thanks to the decision boundary (represented by the purple line).

The decision boundary was derived from the logistic regression parameters based on the formula mentioned in Part A (ii):  $f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$  by setting  $f(x) = 0$ , which then allows us to solve for  $x_2 = -\frac{\theta_0 + \theta_1 x_1}{\theta_2}$ , which helps us define the decision boundary i.e. line that separates the +1 and -1 classes on the graph shown in [Figure 2](#).

The code segment that produced this is present under the [plot\\_log\\_regr\\_predictions\(\)](#) function.

(iv) From [Figure 2](#) and accuracy score of 87.9%, it becomes apparent that the predicted labels align closely with the true labels. The majority of the +1 and -1 data points lie on the correct sides of the decision boundary, with the few miscalculations on either side very close to the boundary line. My observation is that this method relied primarily on the  $X_2$  feature on the vertical axis to separate the two classes, and since only two features were involved in modelling the decision process, the separation line acts as a proper split as compared to the gaussian separation of the given data set. Overall, I would say that the Logistic Regression model provided a good fit for the data with a clear linear decision boundary.

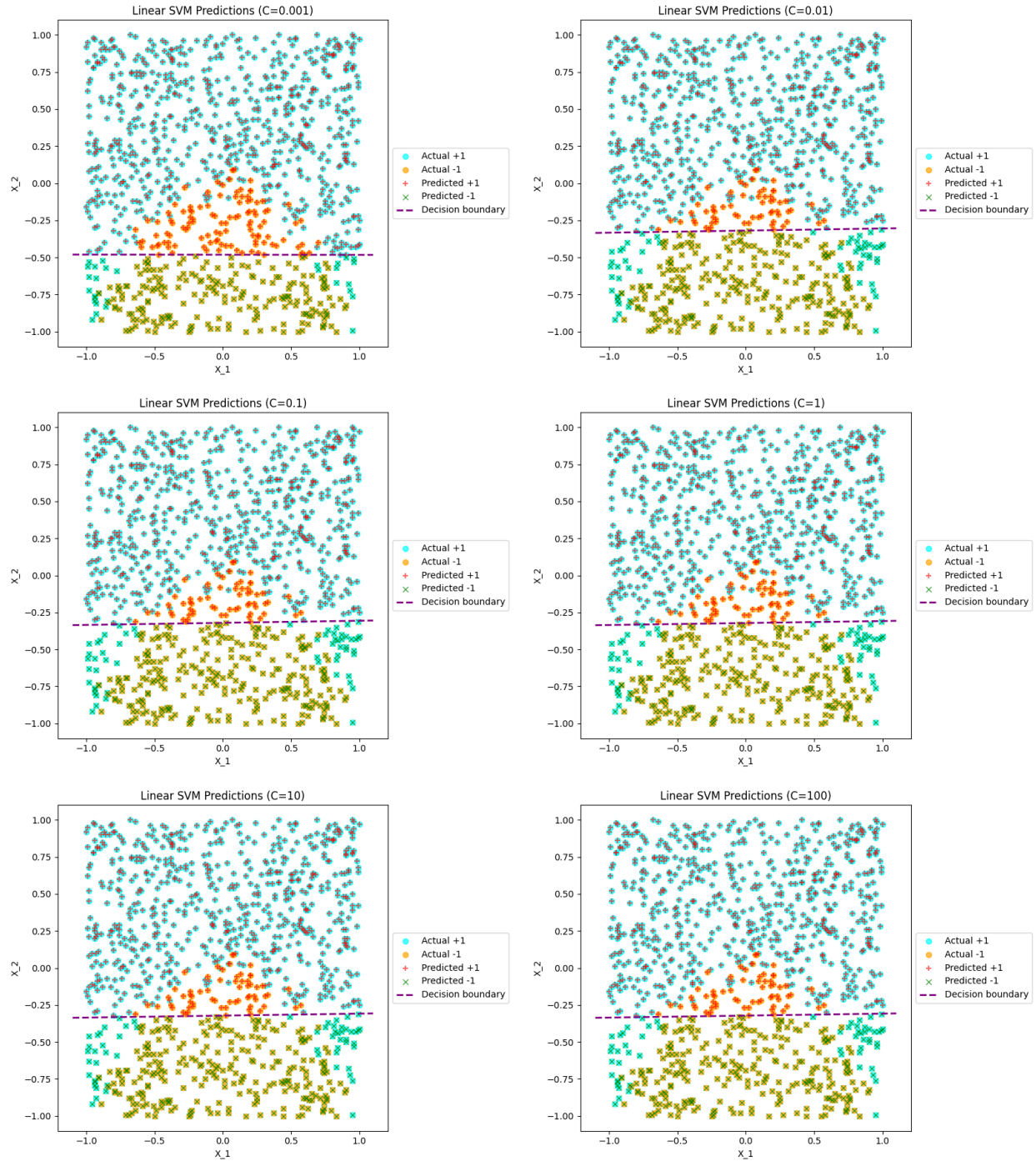
## **Part B**

(i) I trained a total of 7 Support Vector Machine classifiers on the same dataset, with the weighting parameter  $C$  having values of 0.001, 0.01, 0.1, 1, 10, 100, 1000.

The decision function for just two features ( $X_1$  and  $X_2$ ) can be computed as  $f(x) = w_1 x_1 + w_2 x_2 + b$ , where the weights learned by the SVM are  $w_1$  and  $w_2$ , while  $b$  represents the intercept or bias, and each SVM uses this to predict the class of the data point based on the sign of the calculated  $f(x)$ . The resulting parameters for all SVMs are as follows:

C	$w_1$	$w_2$	bias	accuracy
0.001	0.00040	0.48143	0.23188	0.85686
0.01	-0.01701	1.19233	0.38027	0.87888
0.1	-0.02430	1.72052	0.55092	0.87888
1	-0.02498	1.86583	0.59995	0.87788
10	-0.02524	1.88392	0.60616	0.87788
100	-0.02527	1.88578	0.60679	0.87788
1000	-0.02527	1.88597	0.60685	0.87788

(ii) The graphs generated using these are as follows:



Figures 3-8. Scatter plot representing given data and predictions of Support Vector Machine for  $C = 0.001, 0.01, 0.1, 1, 10, 100$

The code segment that produced [Part B \(i\)](#) is present under the `train_linear_svm()` function, and the one that produced these graphs is present under the `plot_svm_predictions()` function.

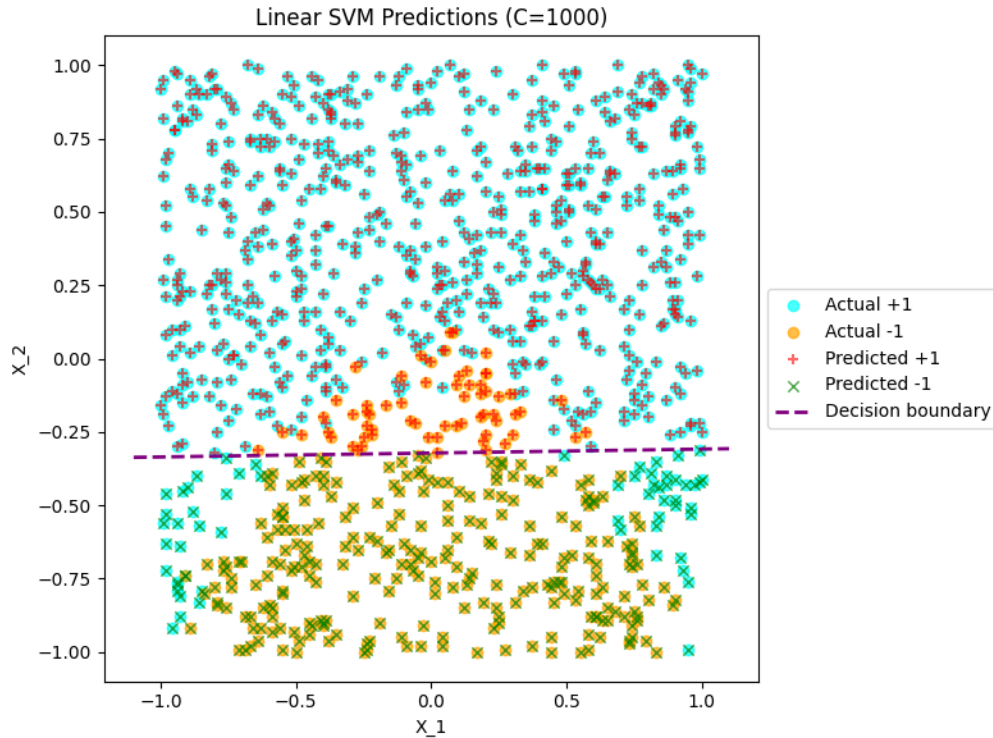


Figure 9. Scatter plot representing given data and predictions of Support Vector Machine for  $C = 1000$

(iii) The weighting parameter  $C$  is used in the cost function  $J(\theta) = \frac{1}{m} \sum_{i=1}^m \max(0, 1 - y^i \theta^T x^i) + \frac{\theta^T \theta}{C}$ , where  $\frac{\theta^T \theta}{C}$  represents the regularization, that describes an inverse relationship where bigger  $C$  makes the penalty less important, and as a result SVM prioritizes correctly classifying points, whereas a smaller  $C$  would increase the regularization and result in a wider margin for potential misclassifications.

My interpretation of the table in [Part B \(i\)](#) and [Figures 3-9](#) is as follows:

- Smaller  $C$  values such as 0.001 and 0.01 (0.1 as well, to an extent) show smaller weights associated with the features and the biases, signifying the decision boundary to be shallower and some amount of misclassification. This is further represented by the accuracy of these models, especially  $C = 0.001$  to be lower than that of the other SVM models with higher  $C$  values.
- Higher  $C$  values ranging from 1 to 1000 display much less variance in the values of weights and stronger biases than when  $C < 1$ , and the accuracy scores also appear to converge into 0.87788 or approximately 87.9%. This shows how the parameters appear to stabilize after a certain point, indicating the data is well separated and further increasing  $C$  has very little effect on how the SVM model computes its predictions.

- The decision boundary was computed using  $f(x) = w_1x_1 + w_2x_2 + b$ , which can be derived by setting  $f(x) = 0$ , which then allows us to solve for  $x_2 = -\frac{b+w_1x_1}{w_2}$ . The data points that appear close to the decision boundary reflect how the change in variance in the different  $C$  values can affect the predicted labels.
- Similar to the Logistic Regression model,  $|w_1| < |w_2|$ , and once again feature  $X_2$  seems to have a greater influence on prediction.
- To conclude,  $C$  offers a unique influence in controlling the model's complexity in an inverse relationship, where the more you increase it, the more the model minimizes misclassification and gives a more complex boundary.

(iv) Both the Logistic Regression model and the SVM models learn a linear decision boundary, and their orientation is very similar, especially for moderate to large  $C$  values, as well as near identical accuracy, with Logistic Regression being incredibly slightly more accurate on my data.

Although the SVM weights are scaled differently due to the hinge loss formulation, with the values themselves being smaller than the Logistic Regression model, their ratio still produces a similar boundary orientation, as well as the trend of the weight associated with  $X_2$  being larger and thus having a greater influence on the prediction remained consistent.

As such, despite the numerical differences, both methods displayed consistency on this dataset for up to 87.9% accuracy, with Logistic Regression providing a probabilistic interpretation and SVM emphasizing margin-based decisions.

### Part C

(i) The Logistic Regression model was trained with two additional features that were the squares of the original input features  $X_1$  and  $X_2$  to allow the model to learn using a quadratic combination of the features as  $f(x) = \theta_0 + \theta_1x_1 + \theta_2x_2 + \theta_3x_1^2 + \theta_4x_2^2$ .

After fitting, the feature coefficients that were produced were  $\theta_1 = -0.1399$ ,  $\theta_2 = 6.82814$ ,  $\theta_3 = 6.28563$  and  $\theta_4 = -0.64524$ , while the intercept is  $\theta_0 = 0.33656$ . The accuracy of this model compared to the actual data produced a score of 0.96296 i.e. approximately 96.3%.

The code segment that produced this is present under the [train\\_log\\_regr\\_with\\_sq\(\)](#) function.

(ii) The trained model's predictions were plotted on top of the scatter plot produced by the provided data, with the predicted +1 labels appearing as red '+' markers and the predicted -1 labels appearing as green 'x' markers to display a comparison between the two:



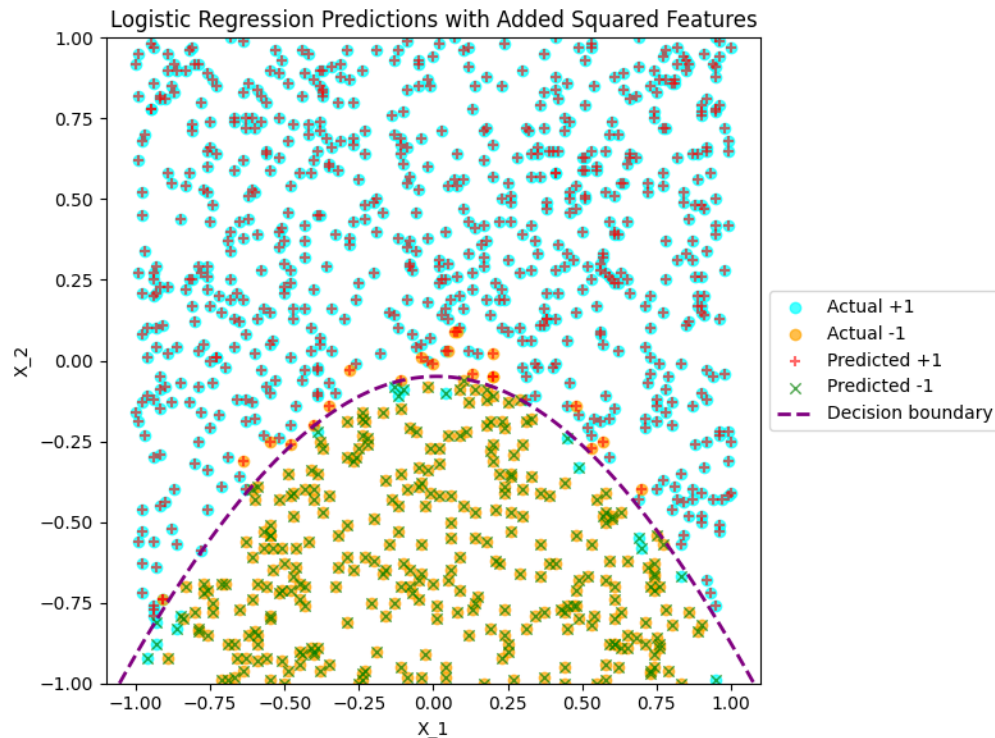


Figure 10. Scatter plot representing given data and predictions of Logistic Regression model involving squared features

The code segment that produced this is present under the [plot\\_sq\\_predictions\(\)](#) function.

The interpretation of the coefficients mentioned in [Part C \(i\)](#) is as follows:

- $\theta_1$  is negative but small, which implies that increasing feature  $X_1$  would slightly decrease the probability of predicting +1.
- $\theta_2$  is positive and larger, which implies that increasing feature  $X_2$  would increase the probability of predicting +1.
- $\theta_3$  is positive and larger, which implies that increasing feature  $X_1^2$  would increase the probability of predicting +1.
- $\theta_4$  is negative but small, which implies that increasing feature  $X_2^2$  would slightly decrease the probability of predicting +1.
- Based on the absolute value of the coefficients, feature  $X_2$  would have the most influence on prediction, followed by  $X_1^2$ .

The key difference that the quadratic function introduced for the model to be more refined than the ones in Parts A and B was that the non-linear decision surface allows the classifier to better separate data that wasn't perfectly linearly separable thanks to increased flexibility. This is also reflected in the values of the parameters associated with the weights being much larger, resulting in an improved accuracy score of 96.3% compared to the 87.9% of the previous models.

One key thing that I observed was how the squared features' coefficients reflected an inverse relationship with their linear counterparts. However, it becomes more intuitive once visualizing them on the graph:

- Data points that are classified as -1 in the given data occur in a small region where  $X_2 < 0$  and  $-0.85 < X_1 < 0.85$ , and other points form the larger region of +1 labels.
- Points that are farther from the central point of  $X_1$  are more likely to be classified as +1, which is reflected in the graph where the given data points at  $X_1 < -0.85$  and  $X_1 > 0.85$  are indeed labelled as +1.
- Points that are farther from the central point of  $X_2$  are more likely to be classified as -1, which is reflected in the graph where the given data points at  $X_2 < 0$  are mostly -1.

This also explains why we yield a curved decision boundary and gives model increased flexibility.

(iii) I coded a very trivial baseline predictor that always predicts the most common class in the training data, which was +1, and it produced an accuracy score of 0.68268 i.e. about 68.2%. This showed just how big a jump the current classifier provided in terms of accuracy, demonstrating how powerful it is to add non-linear features to a Logistic Regression model.

This code segment that produced this is present in the [baseline\\_accuracy\(\)](#) function

(iv) Using the formula stated in [Part C \(i\)](#), the decision boundary can be derived by setting  $f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2$  to 0. We can solve for  $x_2$  using the quadratic equation formula that would turn the equation into  $\theta_4 x_2^2 + \theta_2 x_2 + (\theta_0 + \theta_1 x_1 + \theta_3 x_1^2) = 0$ , which would give

$$x_2 = \frac{-\theta_2 \pm \sqrt{\theta_2^2 - 4\theta_4(\theta_0 + \theta_1 x_1 + \theta_3 x_1^2)}}{2\theta_4}$$
, but upon doing some research on how curves are plotted in matplotlib, I learned that it is done using contours by developing a mesh grid of the given data points and the process is more straightforward using inbuilt functions.

This code segment is already present in [plot\\_sq\\_predictions\(\)](#) function, and the decision boundary has been displayed in Figure 10 as a dashed purple curve.

## Conclusion

Across all the different models, feature  $X_2$  consistently exhibited the strongest influence on classification, driving the decision boundary primarily along the vertical axis. Linear Logistic Regression and Linear SVM produced nearly identical decision boundaries and accuracy, which demonstrated that the given dataset is linearly separable. Introducing squared features allowed Logistic Regression model to fit the non-linear structure better, bringing up the accuracy to 96.3%, showing the effectiveness of feature transformations for complex decision boundaries.



Appendix – Code

```
'''
Author: Priyansh Nayak
CS7CS4 Machine Learning
Week 2 Assignment
'''

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

def plot_given_data(X, Y):
    '''
    Plot the given dataset with +1 and -1 points in different colors.
    '''
    plt.figure(figsize=(7,6)) # larger figure
    # Plot +1 points
    plt.scatter(X[Y == 1, 0], X[Y == 1, 1], marker='o', color='cyan', alpha=0.7, label='+1')

    # Plot -1 points
    plt.scatter(X[Y == -1, 0], X[Y == -1, 1], marker='o', color='orange', alpha=0.7, label='-1')

    # Labels, title, legend
    plt.xlabel("X_1")
    plt.ylabel("X_2")
    plt.title("Dataset Visualization")
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.tight_layout()
    plt.show()
    #plt.savefig("Week2/given_data.png")

def train_log_regr(X, Y):
    '''
    Train a Logistic Regression model on the given data and print its parameters.
    '''
    # Create and train model
    model = LogisticRegression()
    model.fit(X, Y)

    # relevent parameters
    intercept = model.intercept_[0]
```

```
thetas = model.coef_[0]
# make a string of all the feature coefficients
theta_str = ", ".join(f"X{i+1} = {np.round(theta, 5)}" for i, theta in enumerate(thetas))

accuracy = model.score(X, Y) # no. of correct predictions / total predictions

# print model parameters
print("\nLogistic Regression Model Parameters:")
print(f"Feature coefficients: {theta_str}") # theta1, theta2
print(f"Intercept: {intercept:.5f}")      # theta0
print(f"Accuracy of model on full dataset: {accuracy:.5f}")

make_predictions(thetas)

return model

def make_predictions(thetas):
    """
    Make predictions based on feature coefficients and print interpretations.
    """
    # interpretation of feature coefficients for predicting +1
    print("\nInterpretation:")
    for i, theta in enumerate(thetas):
        if theta > 0:
            print(f"- Feature X{i+1} increases the probability of predicting +1")
        elif theta < 0:
            print(f"- Feature X{i+1} decreases the probability of predicting +1")
        else:
            print(f"- Feature X{i+1} has no influence")

    # most influential feature
    most_influential = np.argmax(np.abs(thetas)) # index of largest |theta|
    print(f"Feature X{most_influential+1} has the most influence on prediction\n")

def plot_log_regr_predictions(X, Y, model):
    """
    Plot the logistic regression decision boundary and the data points.
    """
    # relevent parameters
    intercept = model.intercept_[0]
    thetas = model.coef_[0]
    theta1 = thetas[0]
    theta2 = thetas[1]
```

```

plt.figure(figsize=(8,6)) # larger figure

# original data points
plt.scatter(X[Y == 1, 0], X[Y == 1, 1], marker='o', color='cyan', alpha=0.7, label='Actual +1')
plt.scatter(X[Y == -1, 0], X[Y == -1, 1], marker='o', color='orange', alpha=0.7, label='Actual -1')

# predicted points
Y_pred = model.predict(X)
plt.scatter(X[Y_pred == 1, 0], X[Y_pred == 1, 1], marker='+', color='red', alpha=0.6, linewidths=1.5,
label='Predicted +1')
plt.scatter(X[Y_pred == -1, 0], X[Y_pred == -1, 1], marker='x', color='green', alpha=0.7, linewidths=1,
label='Predicted -1')

# decision boundary:  $X_2 = -(\theta_0 + \theta_1 X_1) / \theta_2$ 
x_min, x_max = X[:,0].min(), X[:,0].max()
margin = 0.05 * (x_max - x_min) # 5% of data range as margin
x_values = np.linspace(x_min - margin, x_max + margin, 100)
y_values = -(intercept + theta1*x_values)/theta2
plt.plot(x_values, y_values, linestyle='--', color='purple', linewidth=2, label='Decision boundary')

plt.xlabel("X_1")
plt.ylabel("X_2")
plt.title("Training Data and Logistic Regression Predictions")
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()
#plt.savefig("Week2/logistic_regression_predictions.png")

def train_linear_svm(X, Y):
    """
    Train a Linear SVM model on the given data and print its parameters.
    """
    # try different C values
    C_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
    models = {}

    # print header for the table for visual comparison
    print("Linear SVM Model Parameters for different C values:")
    print(f"{'C':>8} | {'weight_X1':>10} | {'weight_x2':>10} | {'bias':>10} | {'accuracy':>10}")
    print("-"*60)

    for C in C_values:
        model = LinearSVC(C=C, max_iter=10000)

```

```
model.fit(X, Y)

# extract model parameters
weights = model.coef_[0]
weight_X1 = weights[0] # weight for X1
weight_X2 = weights[1] # weight for X2
bias = model.intercept_[0] # bias term
acc = model.score(X, Y) # accuracy on training data

# print parameters in tabulated format for visual comparison
print(f"{C:8.3f} | {weight_X1:10.5f} | {weight_X2:10.5f} | {bias:10.5f} | {acc:10.5f}")

models[C] = model

return models

def plot_svm_predictions(X, Y, svm_models):
    """
    Plot the SVM decision boundaries and the data points.
    """
    for C, model in svm_models.items():
        # relevent parameters
        weights = model.coef_[0]
        weight_X1 = weights[0]
        weight_x2 = weights[1]
        bias = model.intercept_[0]

        plt.figure(figsize=(8,6)) # larger figure

        # original data points
        plt.scatter(X[Y == 1, 0], X[Y == 1, 1], marker='o', color='cyan', alpha=0.7, label='Actual +1')
        plt.scatter(X[Y == -1, 0], X[Y == -1, 1], marker='o', color='orange', alpha=0.7, label='Actual -1')

        # predicted points
        Y_pred = model.predict(X)
        plt.scatter(X[Y_pred == 1, 0], X[Y_pred == 1, 1], marker='+', color='red', alpha=0.6, linewidths=1.5,
label='Predicted +1')
        plt.scatter(X[Y_pred == -1, 0], X[Y_pred == -1, 1], marker='x', color='green', alpha=0.7, linewidths=1,
label='Predicted -1')

        # decision boundary:  $X_2 = -(bias + weight\_X1 * X_1) / weight\_x2$ 
        x_min, x_max = X[:,0].min(), X[:,0].max()
        margin = 0.05 * (x_max - x_min) # 5% of data range as margin
        x_values = np.linspace(x_min - margin, x_max + margin, 100)
```

```
y_values = -(bias + weight_X1 * x_values) / weight_x2
plt.plot(x_values, y_values, linestyle='--', color='purple', linewidth=2, label='Decision boundary')

plt.xlabel("X_1")
plt.ylabel("X_2")
plt.title(f"Linear SVM Predictions (C={C})")
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()

#plt.savefig(f"Week2/svm_predictions_C_{C}.png")

def train_log_regr_with_sq(X_sq, Y):
    """
    Train a Logistic Regression model with squared features on the given data and print its parameters.
    """
    # create and train model with extended features
    model = LogisticRegression()
    model.fit(X_sq, Y)

    # relevent parameters
    intercept = model.intercept_[0]
    thetas = model.coef_[0]

    # make a string of all the feature coefficients
    theta_str = ", ".join(f"X{i+1} = {np.round(theta, 5)}" for i, theta in enumerate(thetas))

    accuracy = model.score(X_sq, Y) # no. of correct predictions / total predictions

    # print model parameters
    print("\nLogistic Regression with Squared Features Model Parameters:")
    print(f"Feature coefficients: {theta_str}") # theta1, theta2, theta3, theta4
    print(f"Intercept: {intercept:.5f}") # theta0
    print(f"Accuracy of model on full dataset: {accuracy:.5f}")

    make_predictions(thetas)

    return model

def plot_sq_predictions(X, Y, X_sq, model):
    """
    Plot the logistic regression decision boundary and the data points.
    """
    plt.figure(figsize=(8,6)) # larger figure
```

```
# original data points
plt.scatter(X[Y == 1, 0], X[Y == 1, 1], marker='o', color='cyan', alpha=0.7, label='Actual +1')
plt.scatter(X[Y == -1, 0], X[Y == -1, 1], marker='o', color='orange', alpha=0.7, label='Actual -1')

# predicted points
Y_pred = model.predict(X_sq)
plt.scatter(X[Y_pred == 1, 0], X[Y_pred == 1, 1], marker='+', color='red', alpha=0.6, linewidths=1.5,
label='Predicted +1')
plt.scatter(X[Y_pred == -1, 0], X[Y_pred == -1, 1], marker='x', color='green', alpha=0.7, linewidths=1,
label='Predicted -1')

# decision boundary: theta0 + theta1*X1 + theta2*X2 + theta3*X1^2 + theta4*X2^2 = 0
# nonlinear boundary, so we need to plot a contour
x_min, x_max = X[:, 0].min(), X[:, 0].max()
y_min, y_max = X[:, 1].min(), X[:, 1].max()

# 5% of data range as margin
x_margin = 0.05 * (x_max - x_min)
x_min, x_max = x_min - x_margin, x_max + x_margin

# create a grid of points
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200), np.linspace(y_min, y_max, 200))

# compute the decision on a grid
X_sq_grid = np.column_stack((xx.ravel(), yy.ravel(), xx.ravel()**2, yy.ravel()**2))
Z = model.decision_function(X_sq_grid)
Z = Z.reshape(xx.shape)

# plot decision boundary and margins
plt.contour(xx, yy, Z, levels=[0], colors='purple', linestyle='--', linewidths=2)

# adding proxy line because contour doesn't create a legend entry
plt.plot([], [], color='purple', linestyle='--', linewidth=2, label='Decision boundary')

plt.xlabel("X_1")
plt.ylabel("X_2")
plt.title("Logistic Regression Predictions with Added Squared Features")
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()
#plt.savefig("Week2/logistic_regression_sq_predictions.png")

def baseline_accuracy(Y):
    ...
```



```
    Compute and print the baseline accuracy of a model that always predicts the most common class.
    '''

    most_common = np.sign((Y == 1).sum() - (Y == -1).sum()) # +1 if more +1s, -1 if more -1s
    baseline_pred = np.full_like(Y, most_common) # figure out which label is more common
    acc = np.mean(baseline_pred == Y)

    print(f"Baseline (always predicts {most_common}): {acc:.5f}")

    return acc

def main():
    # path for csv file
    csv_path = "Week2/week2.csv"

    print("test")

    # load CSV, skip comment line, no header
    df = pd.read_csv(csv_path, comment="#", header=None)
    print(df.head(), end="\n\n") # preview data

    # extract features and labels
    X1 = df.iloc[:,0] # feature 1
    X2 = df.iloc[:,1] # feature 2
    X = np.column_stack((X1, X2)) # first 2 columns as 2D array
    Y = df.iloc[:, 2] # third column as labels

    plot_given_data(X, Y)

    # train Logistic Regression
    log_model = train_log_regr(X, Y)
    plot_log_regr_predictions(X, Y, log_model)

    # train SVM
    svm_models = train_linear_svm(X, Y)
    plot_svm_predictions(X, Y, svm_models)

    # train Logistic Regression with new features
    # extend features for prediction
    X_sq = np.column_stack((X1, X2, X1**2, X2**2))
    sq_model = train_log_regr_with_sq(X_sq, Y)
    plot_sq_predictions(X, Y, X_sq, sq_model)

    baseline_accuracy(Y) # compare against trivial accuracy

main()
```

Expected Text Output

Logistic Regression Model Parameters:

Feature coefficients: X1 = -0.05339, X2 = 5.15001

Intercept: 1.69049

Accuracy of model on full dataset: 0.87888

Interpretation:

- Feature X1 decreases the probability of predicting +1

- Feature X2 increases the probability of predicting +1

Feature X2 has the most influence on prediction

Linear SVM Model Parameters for different C values:

C	weight_X1	weight_x2	bias	accuracy
0.001	0.00040	0.48143	0.23188	0.85686
0.010	-0.01701	1.19233	0.38027	0.87888
0.100	-0.02430	1.72052	0.55092	0.87888
1.000	-0.02498	1.86583	0.59995	0.87788
10.000	-0.02524	1.88392	0.60616	0.87788
100.000	-0.02527	1.88578	0.60679	0.87788
1000.000	-0.02527	1.88597	0.60685	0.87788

Logistic Regression with Squared Features Model Parameters:

Feature coefficients: X1 = -0.13999, X2 = 6.82814, X3 = 6.28563, X4 = -0.64524

Intercept: 0.33656

Accuracy of model on full dataset: 0.96296

Interpretation:

- Feature X1 decreases the probability of predicting +1

- Feature X2 increases the probability of predicting +1

- Feature X3 increases the probability of predicting +1

- Feature X4 decreases the probability of predicting +1

Feature X2 has the most influence on prediction

Baseline (always predicts 1): 0.68268