CSU44061 Machine Learning - Week 2

Faith Olopade - 21364066
Dataset # id:6--6-6

Appendix contains all the code.

Question (a)

(i): Data Visualisation

The dataset is visualised in Figure 2 which contains a plot of two characteristics (X1 on the x-axis and X2 on the y-axis). With a + (green) for a target value of +1 and a o (blue) for a target value of -1, each point represents a single instance. The data shows a distinct division between the classes, appearing clustered, with +1 instances often above and -1 instances below.

	X1	X2	y
0	0.89	-0.34	-1
1	0.09	0.35	1
2	-0.23	-0.88	-1
3	-0.39	0.98	1
4	0.83	0.42	-1

Figure 1: Pairs of feature values

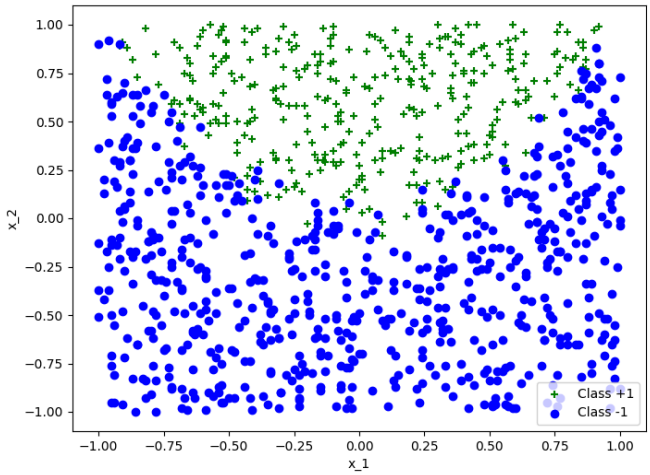

Figure 2: Plot of dataset for each pair of feature values

I used matplotlib to create the scatter plot, separating the two classes based on their target values:

```
plt.scatter(X1[positive_class], X2[positive_class], c='green',
marker='+', label='Class +1')
plt.scatter(X1[negative_class], X2[negative_class], c='blue',
marker='o', label='Class -1')
```

(ii): Logistic Regression Classifier

A logistic regression model using X1 and X2 as features and y as the target variable, which accepts values +1 and -1 to indicate two classes, was trained on the dataset using sklearn. This model uses the sigmoid function to predict whether a data point belongs to class **+1** or **-1** based on the learned coefficients.

The equation for the logistic regression model is:

$$h(x) = \sigma(\theta_0 + \theta_1 X_1 + \theta_2 X_2)$$

where the features X1 and X2 are represented by the coefficients θ_1 and θ_2, respectively, and the intercept is represented by θ_0

Model Parameters:

- **Intercept (θ_0):** -1.924
- **Coefficient for X1 (θ_1):** -0.092
- **Coefficient for X2 (θ_2):** 5.532

While the negative value for X1 indicates that it decreases the likelihood of predicting +1 as X1 increases, the larger coefficient for X2 indicates that it plays a stronger role in predicting class +1

The LogisticRegression model was trained using the fit() function from sklearn, which computes the optimal intercept and coefficients:

```
log_reg = LogisticRegression(penalty=None, solver='lbfgs')
log_reg.fit(X, y)
```

(iii): Predict Target Values

The logistic regression model was used to predict the target values for the training data. The predicted values are visualised on a plot (Figure 3) where different markers (orange triangles for class +1 predictions and red inverted triangles for class -1 predictions) were used to distinguish between the original data points and the model's predictions.

The decision boundary, which divides the two classes according to the logistic regression model, is depicted in the plot (Figure 3). The black dashed line in the plot indicates this decision boundary, which was calculated using the equation that follows, which was obtained from the model parameters by manipulating the coefficients and intercept, ensuring it reflects where the model predicts equal likelihood between the two classes

$$X_2 = -\frac{\theta_1 \cdot X_1 + \theta_0}{\theta_2} = -\frac{-0.092 \cdot x_1 - 1.924}{5.532}$$

```
decision_boundary = -(coef[0] * x_values + intercept) / coef[1]
```

This is consistent with the data points' observed separation. While some of the spots close to the boundary are incorrectly classified, the majority of the model's predictions are accurate, which is typical for logistic regression.

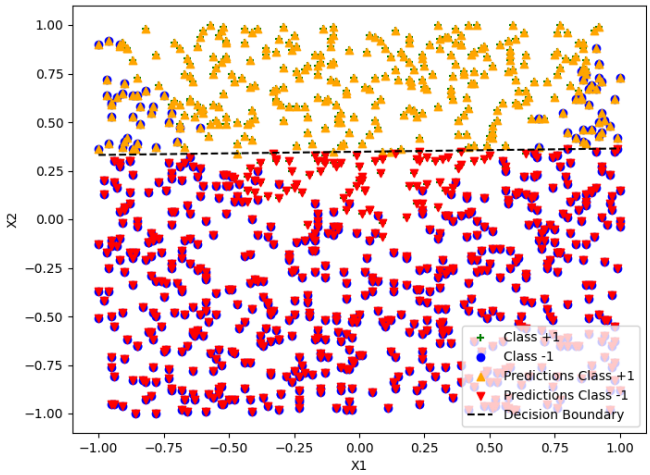

Figure 3: Plot of predicted target values

(iv): Predictions and Training

The model's predictions closely match the actual class labels, especially far from the decision boundary. Most misclassifications occur near the boundary, reflecting the model's uncertainty in

these regions. The large coefficient for X2 confirms its dominant influence in distinguishing between the two classes.

Question (b)

(i): SVM Classifier

We trained linear SVM classifiers using LinearSVC from sklearn with different values of the penalty parameter C (0.001, 1, and 100). The results were visualised through decision boundaries.

By varying the **C** parameter, the SVM classifier adjusts how strictly it penalises misclassification

```
svm_model = LinearSVC(C=C, max_iter=10000)
svm_model.fit(X, y)
```

Larger **C** values enforce stricter boundaries, while smaller values allow for a larger margin with more misclassification.

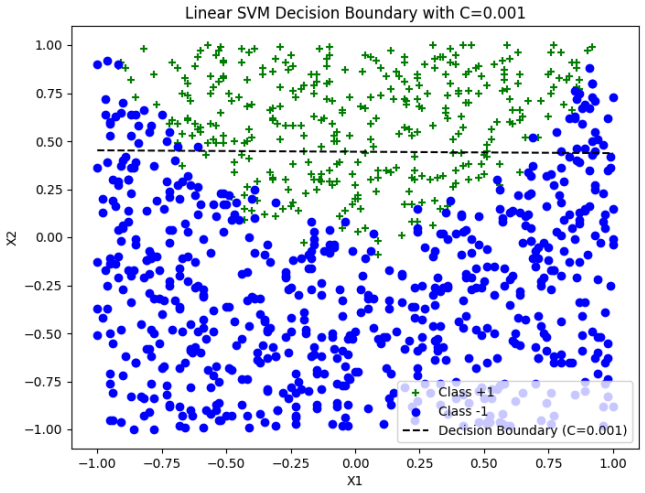

Figure 4: Linear SVM Predictions and Decision Boundary with C=0.001

C = 0.001:

- **Intercept**: -0.2106, **Coefficients**: [0.0035, 0.4733]
- The model focuses on maximising the margin, allowing for more misclassifications. The small coefficients show it relies less on feature X1.

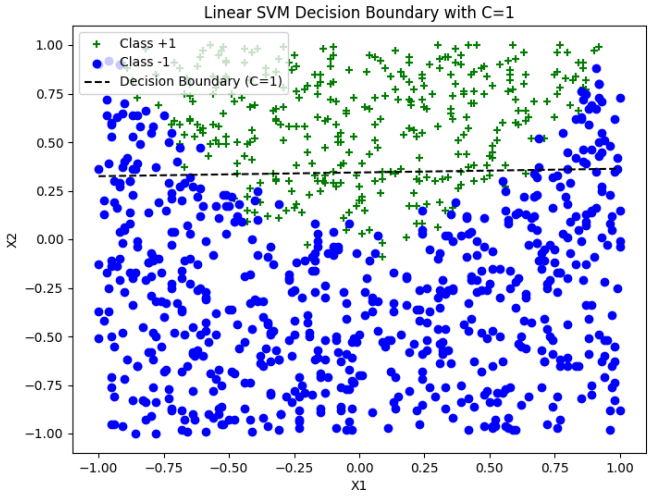

Figure 5: Linear SVM Predictions and Decision Boundary with C=1

C = 1:

- **Intercept**: -0.6294, **Coefficients**: [-0.0353, 1.8297]
- As C increases, the model focuses more on reducing misclassifications, with X2 becoming a dominant feature.

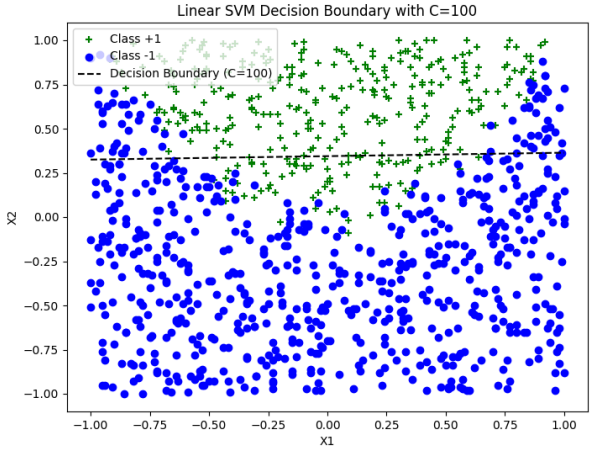

Figure 6: Linear SVM Predictions and Decision Boundary with C=100

C = 100:

- **Intercept**: -0.6365, **Coefficients**: [-0.0360, 1.8480]
- For a large C, the model aggressively reduces misclassification errors, becoming highly sensitive to X2, which may lead to overfitting.

(ii): Predict Target Values

The predictions were made using the predict() function, and the decision boundaries were recalculated similarly to logistic regression

```
predictions = svm_model.predict(X)
decision_boundary = -(coef[0] * x_values + intercept) / coef[1]
```

Each boundary reflects the effect of different C values on the model's behaviour.

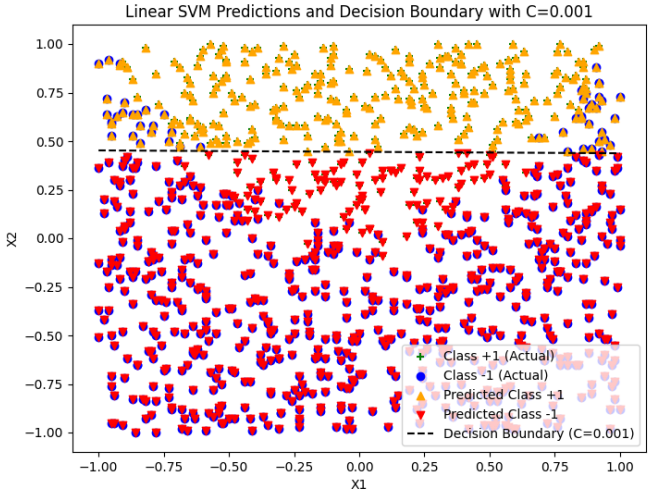

Figure 7: SVM Classifier with C=0.001

For **C = 0.001**, the model allows more misclassifications to achieve a larger margin.

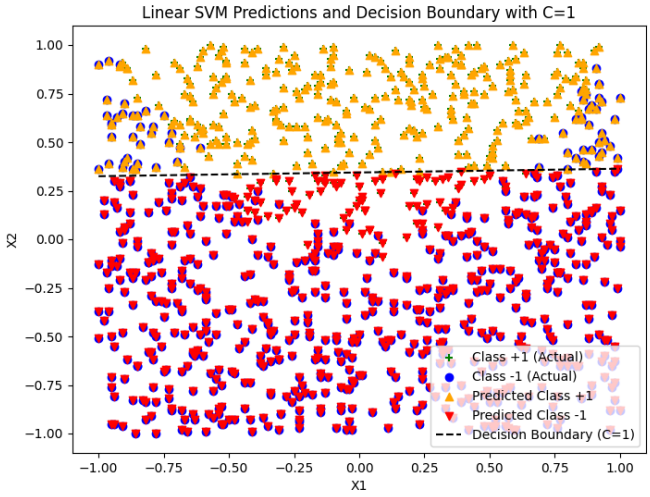

Figure 8: SVM Classifier with C=1

For **C = 1**, the decision boundary is stricter, reducing misclassifications while maintaining a balance between margin and accuracy.

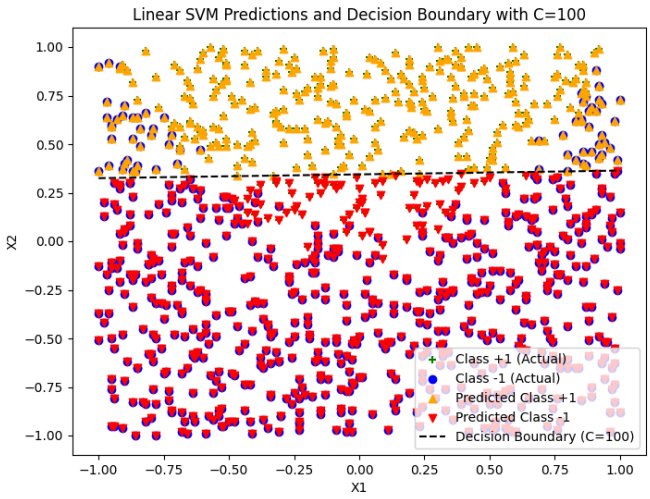

Figure 9: SVM Classifier with C=100

For C = 100, the model fits the training data very tightly, potentially overfitting and reducing its generalisation ability.

Overall, higher C values result in stricter boundaries and reduced misclassifications but can lead to overfitting.

(iii): Changing C on Model Parameters and Predictions

As C increases, the model shifts from allowing more misclassifications to minimising them. With higher C values, the model becomes more sensitive to X2, and the decision boundary becomes stricter, but at the risk of overfitting.

(iv): SVM and Logistic Regression Model Parameters and Predictions

Both SVM and logistic regression highlight the importance of X2 in classification. However, while logistic regression maintains a more stable boundary, the SVM's boundary becomes more sensitive with high C values. SVM offers more control over the error-margin trade-off, while logistic regression is more stable across all settings.

Question (c)

(i): Logistic Regression with Additional Squared Features

To capture non-linear relationships, I extended the logistic regression model by adding squared terms for X1 and X2 ($X1^2$, $X2^2$). The model parameters are:

- Intercept: -0.1441
- Coefficients:
  - X1: 0.1011
  - X2: 23.4760
  - $X1^2$: -23.0090
  - $X2^2$: -1.0298

I created the additional squared features and combined them with the original features, which were then used to train a this logistic regression model

```
X1_squared = X1 ** 2
X2_squared = X2 ** 2
X_new = np.column_stack((X1, X2, X1_squared, X2_squared))
log_reg.fit(X_new, y)
```

Interpretation

X2 and $X1^2$ have the strongest influence on the predictions. X2's large positive coefficient pushes predictions toward class +1, while $X1^2$'s negative coefficient indicates an inverse relationship with class +1. These features create a non-linear decision boundary, improving the model's flexibility.

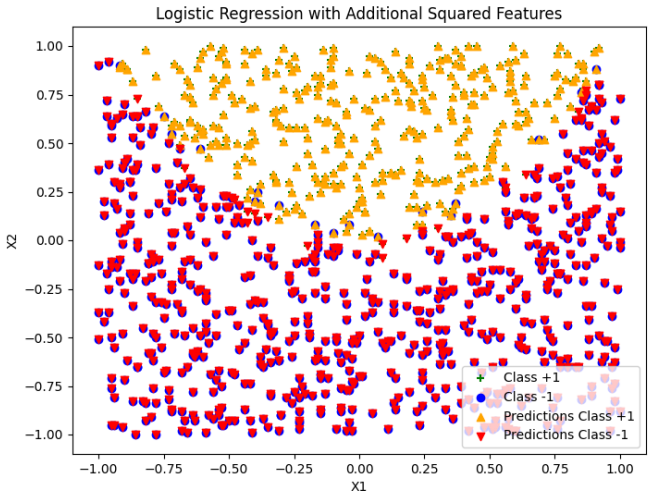

Figure 10: Additional Squared Features

The decision boundary is now non-linear, as seen in Figure 10 better capturing the complexity of the data. Although some misclassifications occur near the boundary, overall accuracy improves with the added squared features, enhancing non-linear pattern detection.

(ii): Predictions

After training, I visualised the predictions with the new features. The model parameters are the same as in (i).

Comparison to Previous Models

- **Logistic Regression without Squared Features**: In part (a), the model had a linear decision boundary, leading to misclassifications near non-linear regions. X2 had the largest influence (coefficient of 5.532).
- **SVM**: The SVM, from part (b), handled linear separations similarly but adjusted boundaries based on C values, which could lead to overfitting with high C.

The additional squared features allowed the model to create a curved boundary, which was reflected in the updated predictions and plot

```
predictions = log_reg.predict(X_new)
```

We can see the flexibility of the model increased with these new features.

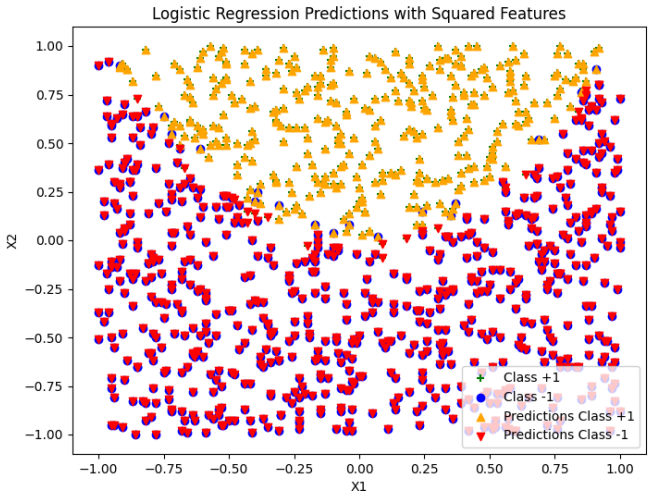

Figure 11: Squared Features

The plot in Figure 11 (visualising the predictions from the augmented logistic regression model) shows that the model can capture more complex relationships between X1 and X2 The decision boundary, no longer a straight line, better separates the data points belonging to different classes. Most of the predictions align with the actual target values, with the misclassifications primarily occurring near the boundary, as expected.

(iii): Classifier Performance

The logistic regression model with squared features achieved an accuracy of 85.67%, significantly outperforming the baseline predictor (65.03%), which only predicted the majority class. The increase in accuracy shows the value of using feature interactions to capture underlying data patterns.

I compared the accuracy of the logistic regression model against a simple baseline model that always predicts the most frequent class

```
log_reg_accuracy = accuracy_score(y, predictions)
baseline_accuracy = accuracy_score(y, np.full_like(y, fill_value=-1))
```

This confirmed that the model with squared features provided a notable performance improvement.

Baseline Predictor
The baseline always predicts the most frequent class, which doesn't account for any feature relationships. Its accuracy reflects the class distribution (65% for the majority class).

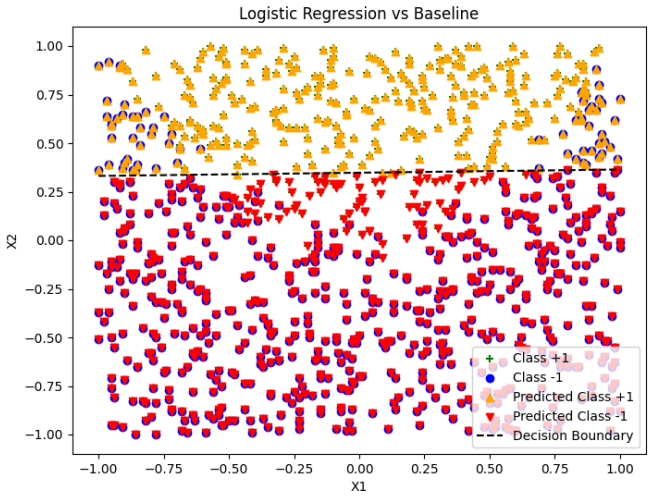

Figure 12: Logistic regression vs Baseline

The plot shows a refined decision boundary from logistic regression compared to the baseline, demonstrating the model's ability to capture relationships between features, leading to fewer misclassifications and improved overall performance.

Appendix

Key parts for question in **bold**

Question (a)

(i)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load data from CSV
df = pd.read_csv("week2.csv")
df.columns = ['X1', 'X2', 'y']
print(df.head())

# Extract the X1, X2 features and target variable y
X1 = df.iloc[:, 0]
X2 = df.iloc[:, 1]
y = df.iloc[:, 2]

# Separate the data into two classes: +1 and -1
positive_class = (y == 1)
negative_class = (y == -1)

# Plot the data points
plt.figure(figsize=(8, 6))
plt.scatter(X1[positive_class], X2[positive_class], c='green',
marker='+', label='Class +1')
plt.scatter(X1[negative_class], X2[negative_class], c='blue',
marker='o', label='Class -1')

# Add labels and legend
```

```
plt.xlabel('x_1')
plt.ylabel('x_2')
plt.legend()
plt.show()
```

(ii)

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

df = pd.read_csv("week2.csv")
df.columns = ['X1', 'X2', 'y']

X = df[['X1', 'X2']].values
y = df['y'].values

# Train  logistic regression model
log_reg = LogisticRegression(penalty=None, solver='lbfgs')  # No
regularisation for this task
log_reg.fit(X, y)

# Output  model parameters
print("Intercept:", log_reg.intercept_)
print("Coefficients:", log_reg.coef_)

# Predict with  logistic regression model
predictions = log_reg.predict(X)

#Feature importance
feature_importance = log_reg.coef_[0]
print(f"Feature X1 coefficient: {feature_importance[0]}")
print(f"Feature X2 coefficient: {feature_importance[1]}")

#Feature influence
if feature_importance[0] > feature_importance[1]:
```

```
    print("X1 has a greater influence on the predictions than X2.")
else:
    print("X2 has a greater influence on the predictions than X1.")

# Predictions  made based on the sign of the linear combination of
features.
# A positive coefficient means that an increase in that feature leads
to an increase in the likelihood of class 1.
```

(iii)

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

df = pd.read_csv("week2.csv")
df.columns = ['X1', 'X2', 'y']

X = df[['X1', 'X2']].values
y = df['y'].values

log_reg = LogisticRegression(penalty=None, solver='lbfgs')  # No
regularisation
log_reg.fit(X, y)

print("Intercept:", log_reg.intercept_)
print("Coefficients:", log_reg.coef_)



# Predict target values using the trained logistic regression model
predictions = log_reg.predict(X)

positive_class = (y == 1)
negative_class = (y == -1)

# Plot original data points with different markers for predictions
```

```
plt.figure(figsize=(8, 6))
plt.scatter(X[positive_class, 0], X[positive_class, 1], c='green',
marker='+', label='Class +1')
plt.scatter(X[negative_class, 0], X[negative_class, 1], c='blue',
marker='o', label='Class -1')

# Plot predictions with orange triangles and red inverted triangles
plt.scatter(X[predictions == 1, 0], X[predictions == 1, 1],
c='orange', marker='^', label='Predictions Class +1')
plt.scatter(X[predictions == -1, 0], X[predictions == -1, 1], c='red',
marker='v', label='Predictions Class -1')

plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()

# Decision boundary occurs where the model's output is 0: theta_0 +
theta_1 * X1 + theta_2 * X2 = 0
intercept = log_reg.intercept_[0]
coef = log_reg.coef_[0]
x_values = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
decision_boundary = -(coef[0] * x_values + intercept) / coef[1]

# Plot the decision boundary
plt.plot(x_values, decision_boundary, color='black', linestyle='--',
label='Decision Boundary')

plt.legend()
plt.show()
```

Question (b)

(i)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.svm import LinearSVC

df = pd.read_csv("week2.csv")
df.columns = ['X1', 'X2', 'y']

X = df[['X1', 'X2']].values
y = df['y'].values

# Train the Linear SVM model with different penalty values (C)
C_values = [0.001, 1, 100]

for C in C_values:
    print(f"Training Linear SVM with C = {C}")

    # Train the SVM model
    svm_model = LinearSVC(C=C, max_iter=10000)  # Increase max_iter if convergence issues arise
    svm_model.fit(X, y)

    # Output the model parameters
    print(f"Intercept: {svm_model.intercept_}")
    print(f"Coefficients: {svm_model.coef_}")

    # Plot the data points and decision boundary
    plt.figure(figsize=(8, 6))
    positive_class = (y == 1)
    negative_class = (y == -1)
    plt.scatter(X[positive_class, 0], X[positive_class, 1], c='green', marker='+', label='Class +1')
    plt.scatter(X[negative_class, 0], X[negative_class, 1], c='blue', marker='o', label='Class -1')

    # Plot decision boundary
    intercept = svm_model.intercept_[0]
    coef = svm_model.coef_[0]
    x_values = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
```

```
    decision_boundary = -(coef[0] * x_values + intercept) / coef[1]
    plt.plot(x_values, decision_boundary, color='black',
linestyle='--', label=f'Decision Boundary (C={C})')

    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.legend()
    plt.title(f'Linear SVM Decision Boundary with C={C}')
    plt.show()
```

(ii)

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

df = pd.read_csv("week2.csv")
df.columns = ['X1', 'X2', 'y']

X = df[['X1', 'X2']].values
y = df['y'].values

C_values = [0.001, 1, 100]

for C in C_values:
    print(f"Training Linear SVM with C = {C}")

    svm_model = LinearSVC(C=C, max_iter=10000)  # Increase max_iter if
needed
    svm_model.fit(X, y)

    print(f"Intercept: {svm_model.intercept_}")
    print(f"Coefficients: {svm_model.coef_}")

    # Make predictions using the SVM model
```

```
    predictions = svm_model.predict(X)

# Plot the data points along with predictions
    plt.figure(figsize=(8, 6))
    positive_class = (y == 1)
    negative_class = (y == -1)
    plt.scatter(X[positive_class, 0], X[positive_class, 1], c='green',
marker='+', label='Class +1 (Actual)')
    plt.scatter(X[negative_class, 0], X[negative_class, 1], c='blue',
marker='o', label='Class -1 (Actual)')

    # Plot the predictions with different markers
    plt.scatter(X[predictions == 1, 0], X[predictions == 1, 1],
c='orange', marker='^', label='Predicted Class +1')
    plt.scatter(X[predictions == -1, 0], X[predictions == -1, 1],
c='red', marker='v', label='Predicted Class -1')

    # Decision boundary
    intercept = svm_model.intercept_[0]
    coef = svm_model.coef_[0]
    x_values = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
    decision_boundary = -(coef[0] * x_values + intercept) / coef[1]
    plt.plot(x_values, decision_boundary, color='black',
linestyle='--', label=f'Decision Boundary (C={C})')

    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.legend()
    plt.title(f'Linear SVM Predictions and Decision Boundary with
C={C}')
    plt.show()
```

Question (c)

(i)

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

df = pd.read_csv("week2.csv")
df.columns = ['X1', 'X2', 'y']

X1 = df['X1'].values
X2 = df['X2'].values
y = df['y'].values

# Create additional squared features to capture non-linear
relationships
X1_squared = X1 ** 2
X2_squared = X2 ** 2

# Combine the original and squared features into a new feature matrix
X_new = np.column_stack((X1, X2, X1_squared, X2_squared))

# Train the logistic regression model using the new feature set
log_reg = LogisticRegression(penalty=None, solver='lbfgs')
log_reg.fit(X_new, y)

print("Intercept:", log_reg.intercept_)
print("Coefficients:", log_reg.coef_)

predictions = log_reg.predict(X_new)

plt.figure(figsize=(8, 6))
positive_class = (y == 1)
negative_class = (y == -1)
```

```
plt.scatter(X1[positive_class], X2[positive_class], c='green',
marker='+', label='Class +1')
plt.scatter(X1[negative_class], X2[negative_class], c='blue',
marker='o', label='Class -1')

plt.scatter(X1[predictions == 1], X2[predictions == 1], c='orange',
marker='^', label='Predictions Class +1')
plt.scatter(X1[predictions == -1], X2[predictions == -1], c='red',
marker='v', label='Predictions Class -1')

plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.title('Logistic Regression with Additional Squared Features')

plt.show()
```

(ii)

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

df = pd.read_csv("week2.csv")
df.columns = ['X1', 'X2', 'y']

X1 = df['X1'].values
X2 = df['X2'].values
y = df['y'].values

X1_squared = X1 ** 2
X2_squared = X2 ** 2

X_new = np.column_stack((X1, X2, X1_squared, X2_squared))
```

```
log_reg = LogisticRegression(penalty=None, solver='lbfgs')
log_reg.fit(X_new, y)

print("Intercept:", log_reg.intercept_)
print("Coefficients:", log_reg.coef_)

# Make predictions using the updated logistic regression model
predictions = log_reg.predict(X_new)

positive_class = (y == 1)
negative_class = (y == -1)

# Plot original data points along with predictions
plt.figure(figsize=(8, 6))
plt.scatter(X1[positive_class], X2[positive_class], c='green',
marker='+', label='Class +1')
plt.scatter(X1[negative_class], X2[negative_class], c='blue',
marker='o', label='Class -1')

# Predicted points using different markers
plt.scatter(X1[predictions == 1], X2[predictions == 1], c='orange',
marker='^', label='Predictions Class +1')
plt.scatter(X1[predictions == -1], X2[predictions == -1], c='red',
marker='v', label='Predictions Class -1')

plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.title('Logistic Regression Predictions with Squared Features')

plt.show()
```

(iii)

```
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

df = pd.read_csv("week2.csv")
df.columns = ['X1', 'X2', 'y']

X = df[['X1', 'X2']].values
y = df['y'].values

log_reg = LogisticRegression(penalty=None, solver='lbfgs')  # No
regularisation
log_reg.fit(X, y)

predictions = log_reg.predict(X)

# Accuracy for the logistic regression model
log_reg_accuracy = accuracy_score(y, predictions)
print(f"Logistic Regression Accuracy: {log_reg_accuracy * 100:.2f}%")

# Baseline predictor: always the most common class (-1)
most_common_class = np.full_like(y, fill_value=-1)

# Accuracy for the baseline model
baseline_accuracy = accuracy_score(y, most_common_class)
print(f"Baseline Accuracy: {baseline_accuracy * 100:.2f}%")

# Original training data and predictions for logistic regression
plt.figure(figsize=(8, 6))
positive_class = (y == 1)
negative_class = (y == -1)
plt.scatter(X[positive_class, 0], X[positive_class, 1], c='green',
marker='+', label='Class +1')
plt.scatter(X[negative_class, 0], X[negative_class, 1], c='blue',
marker='o', label='Class -1')
```

```
# Logistic regression predictions with a different marker
plt.scatter(X[predictions == 1, 0], X[predictions == 1, 1],
c='orange', marker='^', label='Predicted Class +1')
plt.scatter(X[predictions == -1, 0], X[predictions == -1, 1], c='red',
marker='v', label='Predicted Class -1')

intercept = log_reg.intercept_[0]
coef = log_reg.coef_[0]
x_values = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
decision_boundary = -(coef[0] * x_values + intercept) / coef[1]

plt.plot(x_values, decision_boundary, color='black', linestyle='--',
label='Decision Boundary')

plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.title('Logistic Regression vs Baseline')
plt.show()
```