CS7CS4 - Machine Learning - Week 2 Assignment

Student Information

Name: Tim Farrelly
Student Number: 19335376
Dataset ID: `# id:22--44--22`

Part A

Dataset Visualisation (i)

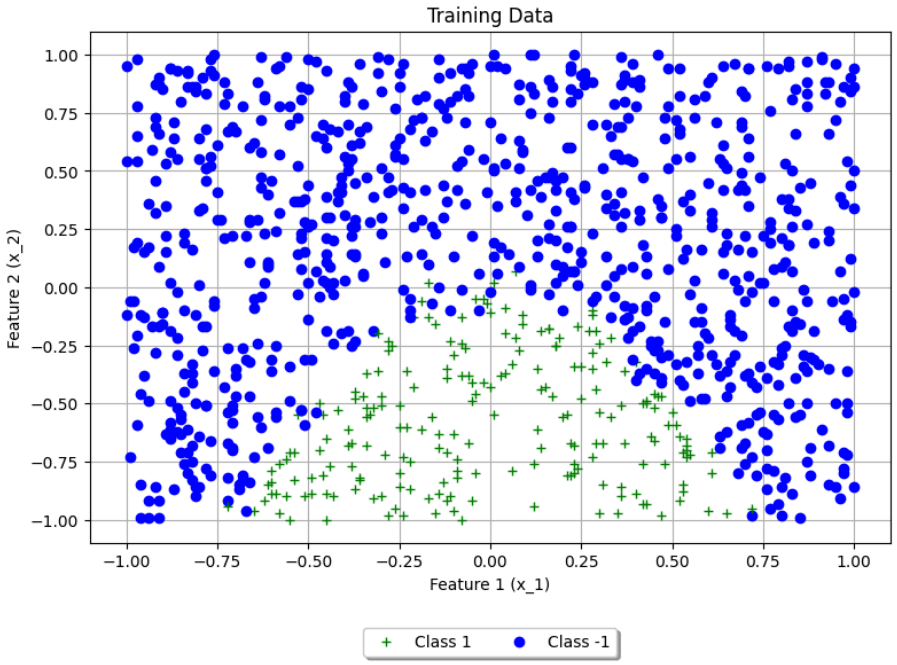

Figure 1. Initial Dataset Visualisation

Logistic Regression Classifier (ii)

We used sklearn to train a logistic regression classifier on the provided dataset. The dataset contains two features, denoted as Feature 1 (x_1) and Feature 2 (x_2), with binary labels, which are visualised above in Figure 1. After training the logistic regression model, the parameter values of the trained model were reported as follows:

- Intercept (bias term): -2.1478
- Coefficients for Feature 1 and Feature 2: 0.0349 and -3.3356, respectively.

Which feature has the most influence on the prediction? The magnitude of the coefficient reflects the influence of a feature on the model's decision, and since the

coefficient for Feature 2 is -3.3356 while Feature 1 is only 0.0349, it can be clearly seen that Feature 2 has the most influence on the model's predictions.

Which features cause the prediction to increase and which cause it to decrease? The sign of the coefficients indicates whether the feature increases or decreases the prediction. A positive coefficient means the feature increases the likelihood of predicting the positive class, while a negative coefficient reduces it. For example, Feature 1 (0.0349) has a small positive influence, meaning an increase in Feature 1 slightly increases the probability of predicting class 1. Whereas Figure 2 (-3.3356) has a strong negative influence, meaning an increase in Feature 2 decreases the probability of predicting class 1 and thus increases the likelihood of predicting class -1.

Logistic Classifier Model Predictions (iii and iv)

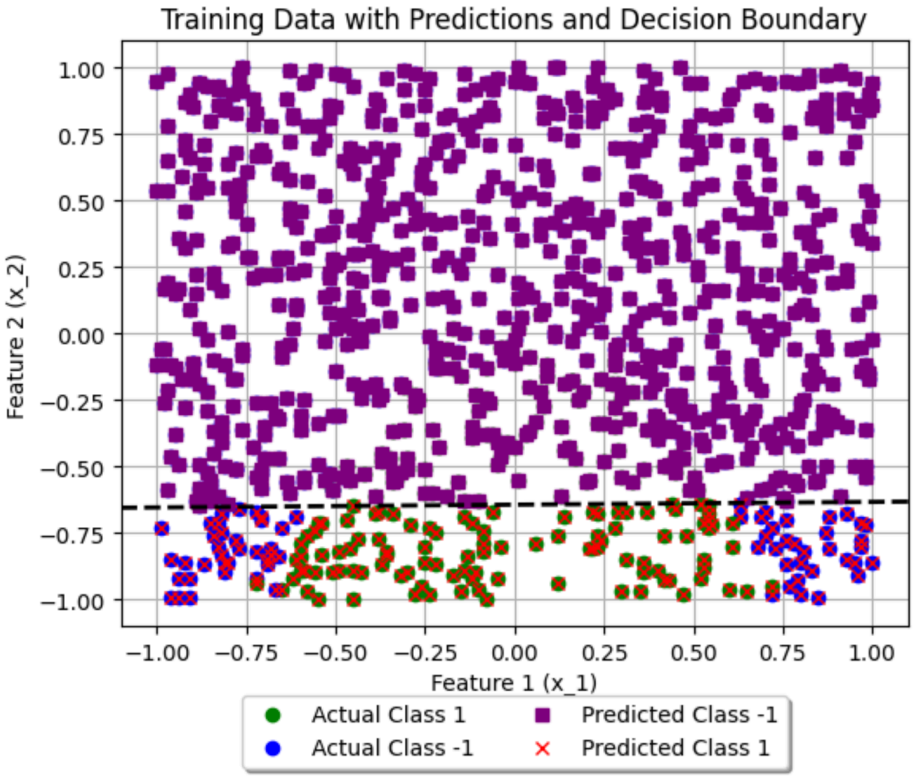

Figure 2. Class predictions using trained Logistic Classifier Model

Figure 2 above shows the predicted target values using our trained logistic classifier model. The decision boundary line can be clearly distinguished as the black dotted line. In logistic regression, the decision boundary is the line where the probability of predicting either class is 0.5. To plot this on our graph, we used matplotlib's contour function to draw a line on our graph exactly where the probability was exactly 0.5. However, a more general method to find the decision boundary would be to use the coefficients of the trained logistic regression

model directly. In a two-feature model, the decision boundary is a line that can be expressed as:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$$

Where β_0 is the intercept, and β_1 and β_2 are the coefficients for features x_1 and x_2 respectively. These values are obtained from the trained model. We can rearrange this equation to solve for x_2:

$$x_2 = -(\beta_0/\beta_2) - (\beta_1/\beta_2)x_1$$

This gives us the equation of the decision boundary line. We can then plot this line by choosing a range of x_1 values and calculating the corresponding x_2 values using this equation. This method directly uses the model parameters to define the decision boundary, providing a straightforward way to visualise it on our graph

What is the efficacy/ performance of the logistic regression model on the training data? The model achieved an accuracy of 82.46% on the training data, indicating a good match between the predictions and the training data. However accuracy alone is not a comprehensive metric, for added context we use sklearns metrics library to get the confusion matrix and a classification report for our model, which reveal much more information.

Data imbalance: Here is the confusion matrix for our model:

	Predicted Negative	Predicted Positive
Actual Negative	720	67
Actual Positive	108	103

Table 1. Logistic Classifier Confusion Matrix

The dataset is imbalanced, with 787 instances of class -1 and only 211 instances of class 1. This imbalance affects the interpretation of our metrics.

Precision and Recall: Results differ greatly for each class.

For class -1: Precision is 0.87 and recall is 0.91, resulting in an F1-score of 0.89.
For class 1: Precision is 0.61 and recall is 0.49, resulting in a lower F1-score of 0.54.

This indicates that the model performs much better on the majority class (-1) than on the minority class (1).

In conclusion, the model's accuracy is quite strong, however it struggles to reliably predict the minority class. This suggests that improvements could be made through techniques like class weighting or over sampling the minority class during training.

Part B

Linear SVM Classifiers and Results (i and ii)

C = 0.001

Model parameters for SVM with C=0.001:
Intercept: -0.3809
Coefficients: [-0.00503 -0.30697]

Here are the predicted values for SVM with C = 0.001

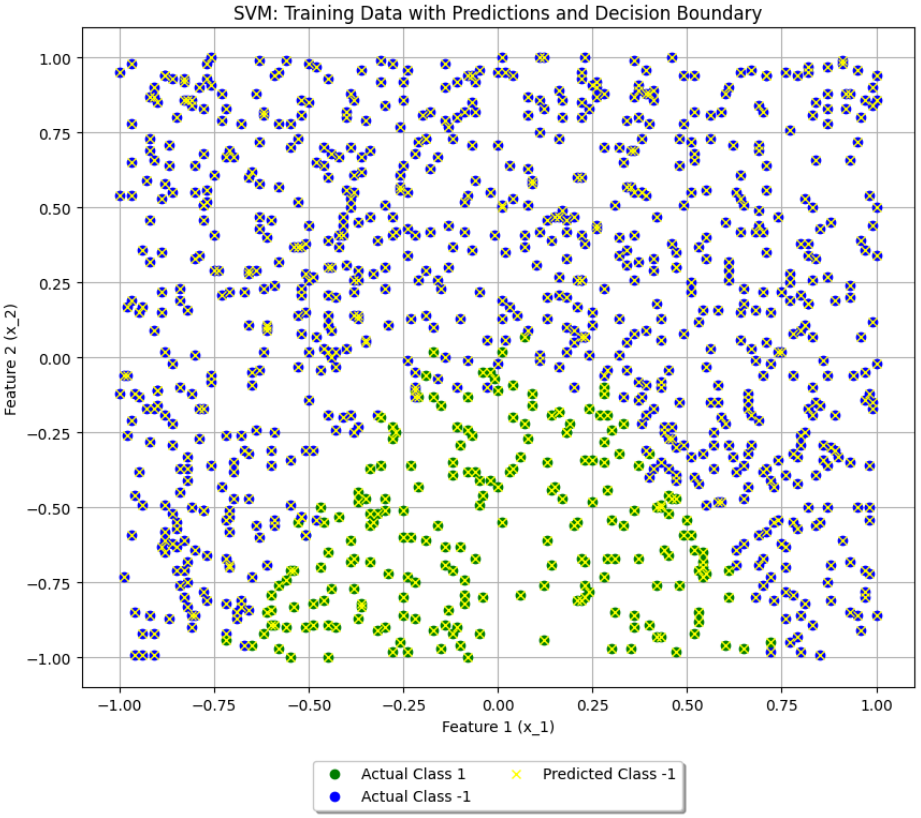

Figure 3. Predicted values for SVM with C = 0.001

Performance metrics:
Accuracy for C=0.001: 78.86%
Confusion Matrix:

	Predicted Negative	Predicted Positive
Actual Negative	787	0
Actual Positive	211	0

C = 1

Model parameters for SVM with C = 1:
Intercept: -0.74236
Coefficients: [-0.00503 -0.30697]

Here are the predicted values for SVM with C = 1

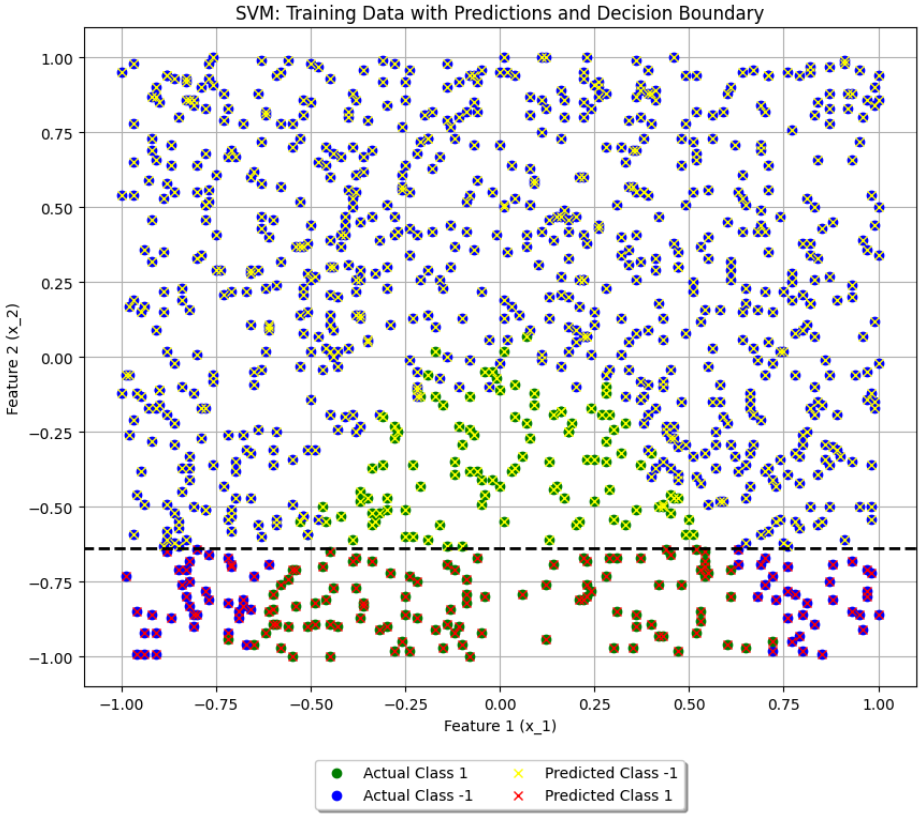

Figure 5. Predicted values for SVM with C = 1

Performance metrics:
Accuracy for C=1: 82.26%
Confusion Matrix:

	Predicted Negative	Predicted Positive
Actual Negative	718	69
Actual Positive	108	103

C = 100

Model parameters for SVM with C = 100:
Intercept: -0.746866
Coefficients: [5.387343 -1.170829]

Here are the predicted values for SVM with C = 100

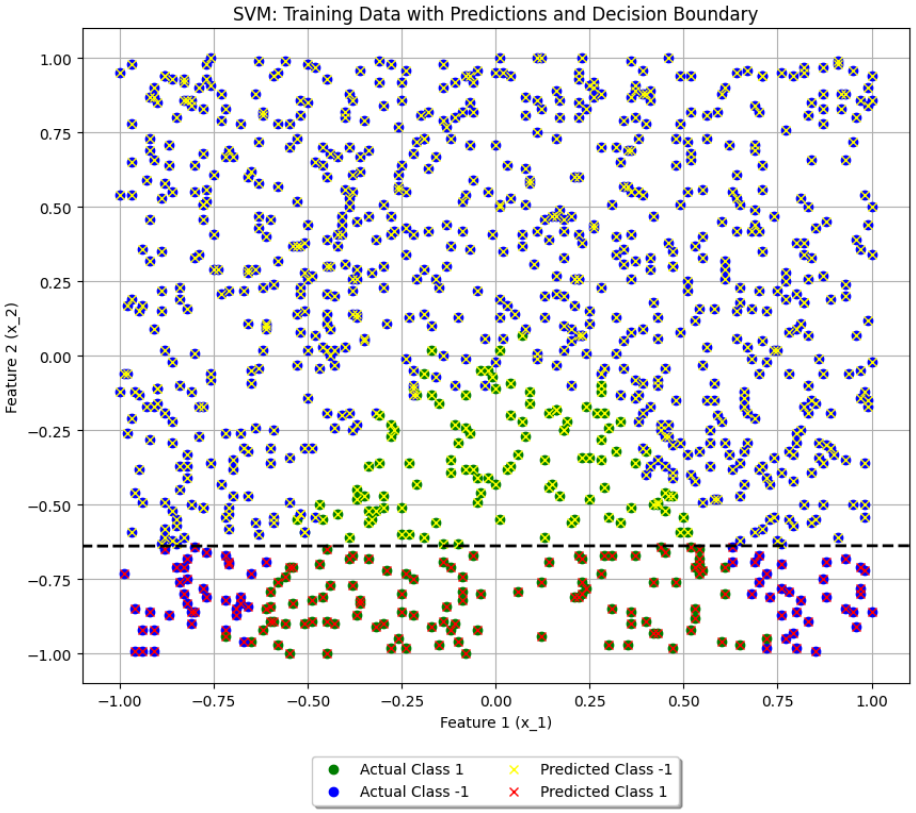

Figure 6: Predicted values for SVM with C = 100

Performance metrics:
Accuracy for C = 100: 82.26%
Confusion Matrix:

	Predicted Negative	Predicted Positive
Actual Negative	718	69
Actual Positive	108	103

How changing C affects the SVM (iii)

To give a concise overview of the changes, here is the direct effect on the model as C increases from 0.001 to 1 to 100:

- Intercept changes from -0.381 to -0.742 to -0.747
- Coefficient for feature 1 changes from -0.00503 to 0.0000876 to 0.000539
- Coefficient for feature 2 changes from -0.307 to -1.161 to -1.171

When C is very small (0.001 in this case), the SVM prioritises having a large margin over correctly classifying all points. This can lead to a situation where the model simply predicts the majority class for all points, which is what we're seeing here.

When C was set to 0.001, it predicted every class to be negative, since this was the majority class and we have quite an imbalance dataset. Due to the imbalance dataset, the accuracy of the model was still reasonable. Given that all predictions are negative, we don't have a visible decision boundary in this case with our model. This all happens because the C parameter in SVM is a regularisation parameter that controls the trade-off between achieving a low training error and a low testing error, and as mentioned above, having a low C encourages a larger margin, even if it misclassifies more points.

However, as we increased C to 1 and 100, we observed significant improvements in the model's performance. The accuracy increased to 82.26% for both C=1 and C=100. The model began to distinguish between classes, as evidenced by the confusion matrices showing predictions for both positive and negative classes - whereas when C was 0.001, it only predicted negative classes. The classification reports show improved precision, recall, and F1-scores for both classes, with the model achieving a balance between correctly identifying the majority class (-1) and the minority class (1).

The similarity in results between C=1 and C=100 suggests that the model reached a point of diminishing returns, where further increasing C didn't significantly change the model's behaviour. This indicates that a C value around 1 might be optimal for this dataset, providing a good balance between margin size and classification accuracy.

Comparative discussion of SVM vs Logistic Regression model from Part A (iv)

Accuracy: Very similar results were achieved by the Logistic Regression model (82.46%) and the SVM models with C = {1, 100} (82.26%), with the Logistic Regression model outperforming by a slight margin. The SVM model with a C = 0.001 performed significantly worse (78.86%) on accuracy.

Precision and Recall: Precision and Recall offer more insight into how the models handle class imbalance. The Logistic Regression showed high precision (0.87) and recall (0.91) for the negative and majority class, resulting in a solid F1 score of 0.89. However, for class 1, precision (0.61) dropped and so did recall (0.49), resulting in an F1 score of 0.54.

For the SVM models with C = {1, 100}, precision (0.6), recall (0.49) and F1 scores (0.54) were almost identical to the Logistic Regression model. However the SVM model with C = 0.001 only predicted the majority class resulting in recalls and precisions of 0, a complete failure for the minority class.

Sensitivity to hyperparameters: The Logistic Regression model was able to achieve the highest accuracy without any need for hyperparameter tuning, which is highly favourable than needing to find the right C for the SVM.

Interpretability: Logistic Regression is more interpretable with its coefficients clearly showing how each feature impacts the prediction, making it easier to understand. In contrast, SVM is harder to interpret, as it focuses on maximising the margin between classes, and tuning parameters like C adds complexity, which makes it less clear how the

features directly influence predictions. The Logistic Regression approach has a clear advantage here.

Overall, the linear regression model is clearly preferable due to the reasons stated above in our comparison with SVM models.

Part C

Extended Features Linear Regression (LREF) Model (i)

Intercept: -0.6689621037333673
Coefficients: [0.02443751 -5.36917466 -8.24225874 -0.0719438]

LREF predictions and visualisation (ii)

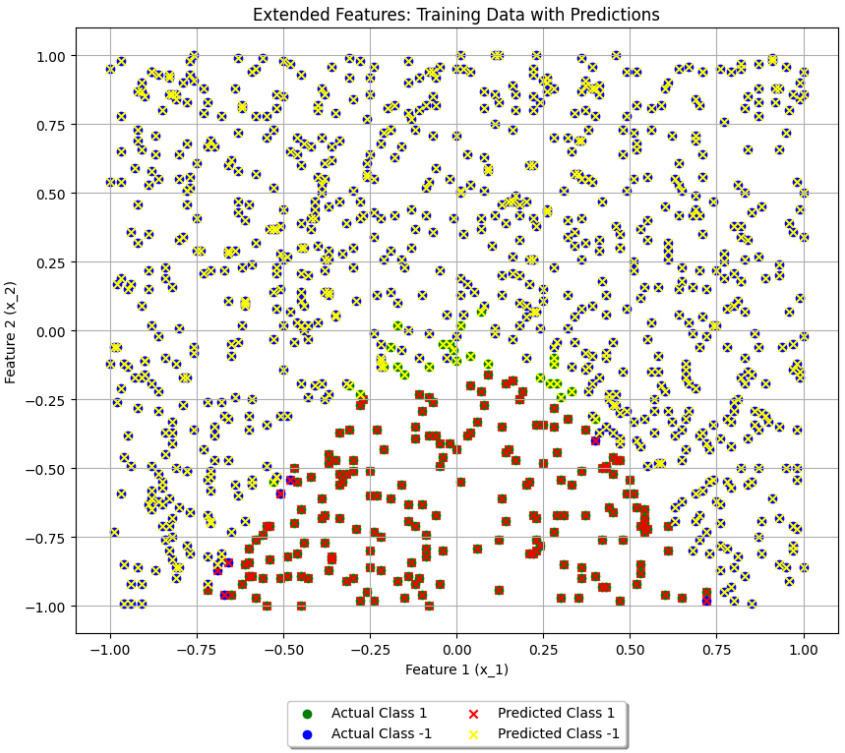

Figure 7. LREF predictions with original data (2 features only)

As can be seen from the above figure, the predictions of the LREF are considerably more accurate than previous models we trained. This new model is able to effectively capture the non-linear nature of the data.

Whereas the previous LR models which were only trained on two features resulted in a linear decision boundary - the SVM we used was a Linear SVM too which could only produce a linear decision boundary. This new model, LREF, with extended data allows for a quadratic decision boundary, modelling the data much more accurately and improving model performance.

In previous models, we only had two coefficients but we now have four. It can also be seen that the our third new feature, which relates to the squared x1 feature, has quite a large magnitude (-8.24) and thus

Comparison to baseline predictor (iii)

The performance of the Linear Regression model trained with extended features (LREF) is considerably stronger than a baseline predictor across all metrics.

Accuracy: The accuracy of the LREF is 96.69% whereas the accuracy of the baseline predictor is 78.86% (majority class).

Precision and Recall: Of course the precision and recall of the baseline predictor is 0 for the minority class, whereas the LREF boasts an F1 score of 0.98 for the negative/ majority class and 0.92 for the positive/ minority class.

Clearly the LREF significantly outperformed the baseline predictor.

LREF Decision Boundary

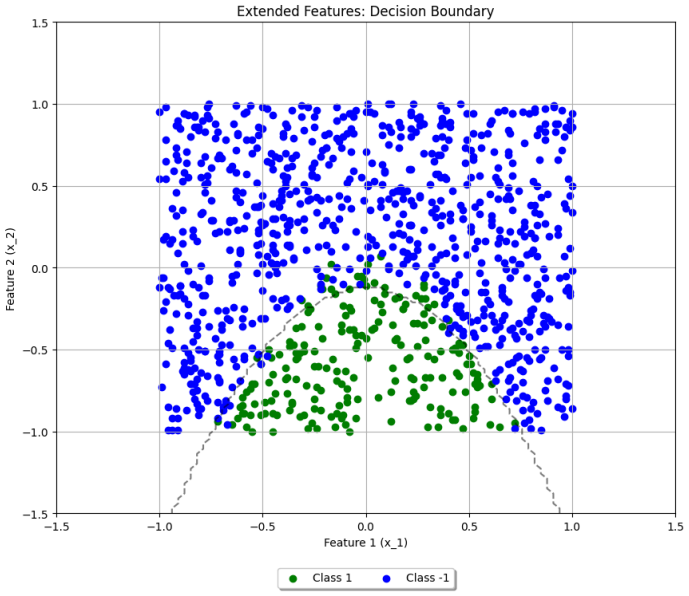

Figure 9. Decision Boundary for LREF

As can be seen from Figure 9, our decision boundary for the LREF is a quadratic line. We found this using a numerical approach rather than directly solving a quadratic equation. Specifically, we create a mesh grid covering the range of both original features, extend this grid with squared terms, and then use the classifier to predict the class for each point. The contour function is used to plot the line where the probability of it being either class is 0.5 effectively showing the decision boundary.

Appendix A

Code

All my code is contained in a single main.py file:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.svm import LinearSVC

DATA_PATH: str = "./data/week2.csv"


def read_data(data_path: str = DATA_PATH) -> pd.DataFrame:
    return pd.read_csv(data_path)


def parse_data(df: pd.DataFrame) -> (np.ndarray, np.ndarray, np.ndarray):
    X1 = df.iloc[:, 0]
    X2 = df.iloc[:, 1]
    y = df.iloc[:, 2]
    return X1, X2, y


def plot_data(X1: np.ndarray, X2: np.ndarray, y: np.ndarray, markers: dict,
title: str, file_name: str) -> None:
    plt.figure(figsize=(8, 6))  # Increase figure size for better layout
    for xi1, xi2, yi in zip(X1, X2, y):
        label = f"Class {yi}" if f"Class {yi}" not in
plt.gca().get_legend_handles_labels()[1] else ""
        plt.plot(xi1, xi2, markers[yi]['marker'], color=markers[yi]['color'],
label=label)

    plt.xlabel('Feature 1 (x_1)')
    plt.ylabel('Feature 2 (x_2)')
    plt.title(title)
    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), fancybox=True,
shadow=True, ncol=2)
    plt.grid(True)

    plt.tight_layout()  # Adjust the layout so everything fits well
    plt.savefig(file_name, bbox_inches='tight')

def visualize_data(X1: np.ndarray, X2: np.ndarray, y: np.ndarray) -> None:
    markers = {
        1: {'marker': '+', 'color': 'green'},
        -1: {'marker': 'o', 'color': 'blue'}
    }
    plot_data(X1, X2, y, markers, 'Training Data', 'training_data_plot.png')
```

```
def visualize_linear_classifier_predictions(X1: np.ndarray, X2: np.ndarray,
y: np.ndarray, predictions: np.ndarray, clf: LogisticRegression, filename:
str) -> None:
    plt.figure()

    # Original data markers
    markers = {
        1: {'marker': 'o', 'color': 'green', 'label': 'Actual Class 1'},
        -1: {'marker': 'o', 'color': 'blue', 'label': 'Actual Class -1'}
    }
    for xi1, xi2, yi in zip(X1, X2, y):
        plt.plot(xi1, xi2, markers[yi]['marker'], color=markers[yi]['color'],
label=markers[yi]['label'] if markers[yi]['label'] not in
plt.gca().get_legend_handles_labels()[1] else "")

    # Predicted data markers
    pred_markers = {
        1: {'marker': 'x', 'color': 'red', 'label': 'Predicted Class 1'},
        -1: {'marker': 's', 'color': 'purple', 'label': 'Predicted Class -1'}
    }
    for xi1, xi2, pred in zip(X1, X2, predictions):
        plt.plot(xi1, xi2, pred_markers[pred]['marker'],
color=pred_markers[pred]['color'], label=pred_markers[pred]['label'] if
pred_markers[pred]['label'] not in plt.gca().get_legend_handles_labels()[1]
else "")

    # Plot decision boundary
    xmin, xmax = X1.min() - 0.1, X1.max() + 0.1
    ymin, ymax = X2.min() - 0.1, X2.max() + 0.1
    xx, yy = np.meshgrid(np.linspace(xmin, xmax, 200), np.linspace(ymin, ymax,
200))
    grid = np.c_[xx.ravel(), yy.ravel()]
    probs = clf.predict_proba(grid)[:, 1].reshape(xx.shape)
    plt.contour(xx, yy, probs, levels=[0.5], linewidths=2, colors='black',
linestyles='dashed')


    plt.xlabel('Feature 1 (x_1)')
    plt.ylabel('Feature 2 (x_2)')
    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.1), fancybox=True,
shadow=True, ncol=2)
    plt.grid(True)
    plt.title('Training Data with Predictions and Decision Boundary')
    plt.savefig(filename, bbox_inches='tight')


def train_logistic_regression(X: np.ndarray, y: np.ndarray) ->
LogisticRegression:
    clf = LogisticRegression()
    clf.fit(X, y)
    return clf
```

```
def train_svm(X: np.ndarray, y: np.ndarray, C: float) -> LinearSVC:
    clf = LinearSVC(C=C, max_iter=10000)
    clf.fit(X, y)
    return clf


def visualize_svm_predictions(X1: np.ndarray, X2: np.ndarray, y: np.ndarray,
predictions: np.ndarray, clf: LinearSVC, filename: str) -> None:
    plt.figure(figsize=(10, 8))

    # Original data markers
    markers = {
        1: {'marker': 'o', 'color': 'green', 'label': 'Actual Class 1'},
        -1: {'marker': 'o', 'color': 'blue', 'label': 'Actual Class -1'}
    }
    for xi1, xi2, yi in zip(X1, X2, y):
        plt.plot(xi1, xi2, markers[yi]['marker'], color=markers[yi]['color'],
label=markers[yi]['label'] if markers[yi]['label'] not in
plt.gca().get_legend_handles_labels()[1] else "")

    # Predicted data markers
    pred_markers = {
        1: {'marker': 'x', 'color': 'red', 'label': 'Predicted Class 1'},
        -1: {'marker': 'x', 'color': 'yellow', 'label': 'Predicted Class -1'}
    }
    for xi1, xi2, pred in zip(X1, X2, predictions):
        plt.plot(xi1, xi2, pred_markers[pred]['marker'],
color=pred_markers[pred]['color'], label=pred_markers[pred]['label'] if
pred_markers[pred]['label'] not in plt.gca().get_legend_handles_labels()[1]
else "")

    # Plot decision boundary
    xmin, xmax = X1.min() - 0.1, X1.max() + 0.1
    ymin, ymax = X2.min() - 0.1, X2.max() + 0.1
    xx, yy = np.meshgrid(np.linspace(xmin, xmax, 200), np.linspace(ymin, ymax,
200))
    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black',
linestyles='dashed')

    plt.xlabel('Feature 1 (x_1)')
    plt.ylabel('Feature 2 (x_2)')
    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.1), fancybox=True,
shadow=True, ncol=2)
    plt.grid(True)
    plt.title('SVM: Training Data with Predictions and Decision Boundary')
    plt.savefig(filename, bbox_inches='tight')
    plt.close()

def part_a(X, X1, X2, y):
```

```
    # ########### Part A ############

    # (a)(i) Visualize the data
    visualize_data(X1, X2, y)

    # (a)(ii) Train logistic regression model
    clf = train_logistic_regression(X, y)
    predictions = clf.predict(X)
    parameters = np.concatenate((clf.intercept_, clf.coef_.flatten()))
    print(f"Model parameters: {parameters}")
    print(f"Intercept: {clf.intercept_[0]}")
    print(f"Coefficients: {clf.coef_[0]}")

    # Discuss feature influence
    coef = clf.coef_[0]
    most_influential_feature = 'Feature 1' if abs(coef[0]) > abs(coef[1]) else
'Feature 2'
    print(f"The most influential feature is {most_influential_feature}")
    print(f"Feature 1 coefficient ({coef[0]}): {'increases' if coef[0]>0 else
'decreases'} the prediction")
    print(f"Feature 2 coefficient ({coef[1]}): {'increases' if coef[1]>0 else
'decreases'} the prediction")

    # (a)(iii) Add predictions to the plot
    visualize_linear_classifier_predictions(X1, X2, y, predictions, clf,
"visualize_linear_classifier_predictions.png")

    # (a)(iv) Comment on predictions vs training data

    report = classification_report(y, predictions)
    conf_matrix = confusion_matrix(y, predictions)
    print(f"Classification report:\n{report}")
    print(f"\nConfusion matrix:\n{conf_matrix}")

    correct = (predictions == y).sum()
    total = len(y)
    accuracy = correct / total
    print(f"Accuracy on training data: {accuracy*100:.2f}%")


def part_b(X, X1, X2, y):
    # Train and analyze SVMs for different values of C
    C_values = [0.001, 1, 100]
    for C in C_values:
        print(f"Training SVM with C={C}")
        clf = train_svm(X, y, C)
        predictions = clf.predict(X)

        # Print model parameters
        parameters = np.concatenate((clf.intercept_, clf.coef_.flatten()))
        print(f"Model parameters for SVM with C={C}: {parameters}")
        print(f"Intercept: {clf.intercept_[0]}")
        print(f"Coefficients: {clf.coef_[0]}")
```

```
        # Accuracy
        accuracy = (predictions == y).mean() * 100
        print(f"Accuracy for C={C}: {accuracy:.2f}%")

        # Other metrics
        report = classification_report(y, predictions)
        conf_matrix = confusion_matrix(y, predictions)
        print(f"Classification report:\n{report}")
        print(f"\nConfusion matrix:\n{conf_matrix}")

        # Visualise the predictions on top of the base data
        visualize_svm_predictions(X1, X2, y, predictions, clf,
f"svm_predictions_C_{C}.png")

        print("\n\n ------------------- \n\n")


def create_additional_features(X) -> np.ndarray:
    X_squared = X ** 2
    return np.hstack((X, X_squared))


def part_c(X, X1, X2, y):
    # (i) Create additional features and train logistic regression
    X_extended = create_additional_features(X)
    clf = train_logistic_regression(X_extended, y)

    print("Logistic Regression with Extended Features:")
    print(f"Intercept: {clf.intercept_[0]}")
    print(f"Coefficients: {clf.coef_[0]}")

    # (ii) Predict and plot
    predictions = clf.predict(X_extended)
    visualize_extended_predictions(X1, X2, y, predictions, clf,
"extended_predictions.png")

    # (iii) Compare with baseline
    compare_with_baseline(y, predictions)

    # (iv) Plot decision boundary (bonus)
    plot_decision_boundary(X1, X2, y, clf, "extended_decision_boundary.png")


def visualize_extended_predictions(X1, X2, y, predictions, clf, filename):
    plt.figure(figsize=(10, 8))

    # Plot original data
    plt.scatter(X1[y == 1], X2[y == 1], c='green', marker='o', label='Actual
Class 1')
    plt.scatter(X1[y == -1], X2[y == -1], c='blue', marker='o', label='Actual
Class -1')
```

```
    # Plot predictions
    plt.scatter(X1[predictions == 1], X2[predictions == 1], c='red',
marker='x', label='Predicted Class 1')
    plt.scatter(X1[predictions == -1], X2[predictions == -1], c='yellow',
marker='x', label='Predicted Class -1')

    plt.xlabel('Feature 1 (x_1)')
    plt.ylabel('Feature 2 (x_2)')
    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.1), fancybox=True,
shadow=True, ncol=2)
    plt.grid(True)
    plt.title('Extended Features: Training Data with Predictions')
    plt.savefig(filename, bbox_inches='tight')
    plt.close()


def compare_with_baseline(y, predictions):
    # Baseline: always predict the most common class
    most_common_class = np.sign(np.sum(y))
    baseline_predictions = np.full_like(y, most_common_class)

    print("\nComparison with Baseline:")
    print("Extended Logistic Regression:")
    print(classification_report(y, predictions, zero_division=0))
    print("\nBaseline (Most Common Class):")
    print(classification_report(y, baseline_predictions, zero_division=0))

    # Accuracy
    accuracy = (predictions == y).mean() * 100
    baseline_accuracy = (baseline_predictions == y).mean() * 100
    print(f"\nAccuracy for Extended Logistic Regression: {accuracy:.2f}%")
    print(f"Accuracy for Baseline: {baseline_accuracy:.2f}%")


def plot_decision_boundary(X1, X2, y, clf, filename):
    plt.figure(figsize=(10, 8))

    # Plot original data
    plt.scatter(X1[y == 1], X2[y == 1], c='green', marker='o', label='Class
1')
    plt.scatter(X1[y == -1], X2[y == -1], c='blue', marker='o', label='Class
-1')

    # Create a grid of points
    xx, yy = np.meshgrid(np.linspace(X1.min() - 0.5, X1.max() + 0.5, 100),
                         np.linspace(X2.min() - 0.5, X2.max() + 0.5, 100))

    # Create extended features for the grid
    grid = np.c_[xx.ravel(), yy.ravel()]
    grid_extended = create_additional_features(grid)

    # Get predictions for the grid
    Z = clf.predict(grid_extended)
```

```
    Z = Z.reshape(xx.shape)

    # Plot decision boundary
    plt.contour(xx, yy, Z, colors='k', levels=[0], alpha=0.5,
linestyles=['--'])

    plt.xlabel('Feature 1 (x_1)')
    plt.ylabel('Feature 2 (x_2)')
    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.1), fancybox=True,
shadow=True, ncol=2)
    plt.grid(True)
    plt.title('Extended Features: Decision Boundary')
    plt.savefig(filename, bbox_inches='tight')
    plt.close()


def main():
    df = read_data(DATA_PATH)
    X1, X2, y = parse_data(df)
    X = np.column_stack((X1, X2))

    # Part A
    # part_a(X, X1, X2, y)

    # Part B
    # part_b(X, X1, X2, y)

    # Part C
    part_c(X, X1, X2, y)

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