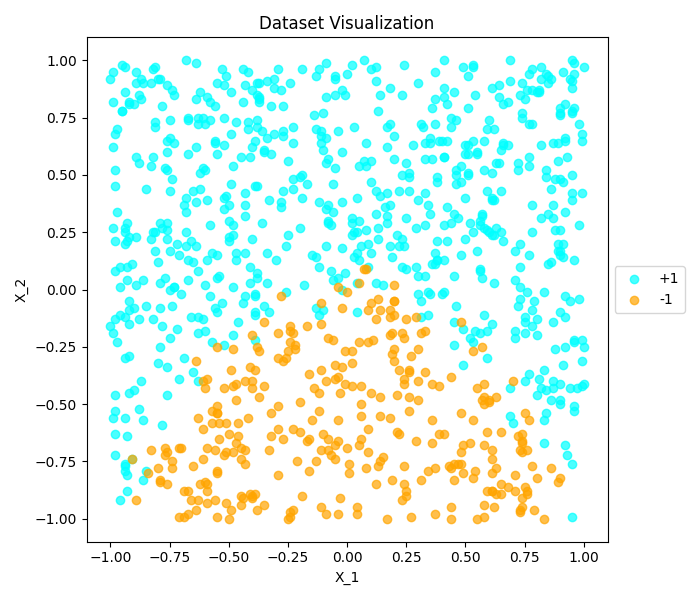
**CS7CS4 Machine Learning**

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

**Part A**

**(i) Visualization of Original Data**

The provided dataset consists of two numerical features (dubbed and ) and a binary target label (that was -1 or +1). The features were plotted into a scatter plot, with the X-axis representing the feature and the Y-axis representing the 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 was using matplotlib’s scatter function by providing it the given features, and is present under the [plot\_given\_data()](#plot_given_data) function.

**(ii) Training Logistic Regression Model**

The Logistic Regression model was trained using the given features and , which computes the probability of a point using the sigmoid function as +1 when and -1 when , where represents the linear combination of the features and .

After fitting, the feature coefficients that were produced were and , while the intercept is . 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:

* is negative but small, which implies that increasing feature would slightly decrease the probability of predicting +1.
* is positive and larger, which implies that increasing feature would increase the probability of predicting +1.
* As such, since , feature would have the most influence on prediction.

This analysis aligns with how the data is structured in [Figure 1](#fig1), where the data points appear to be vertically segregated into two distinct regions.

The code segment that produced this was using sklearn’s LogisticRegression() function with default parameters due to less features, and was fitted using the given data as input and target weights, and is present under the [train\_log\_regr()](#train_log_regr) and [make\_predictions()](#make_predictions) functions.

**(iii) Plotting the Logistic Regression Model’s Predictions**

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)](#a2), 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)](#a2): by setting (classifiers are undecided), which then allows us to solve for , which helps us define the decision boundary i.e. line that separates the +1 and -1 classes on the graph shown in [Figure 2](#fig2).

The code segment that produced this was using matplotlib’s scatter function with the given features and predicted values, and is present under the [plot\_log\_regr\_predictions()](#plot_log_regr_predictions) function.

**(iv) Analysis of Model’s Predictions**

From [Figure 2](#fig2) 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 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) Training Support Vector Machines on different C Values.**

I trained a total of 7 SVM classifiers on the same dataset, with the weighting parameter having values of 0.001, 0.01, 0.1, 1, 10, 100, 1000, to properly examine the behavior.

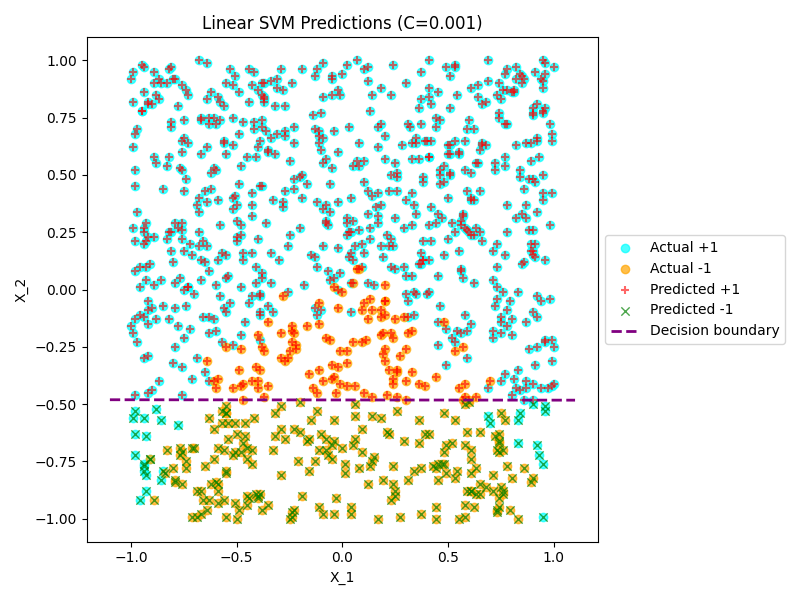
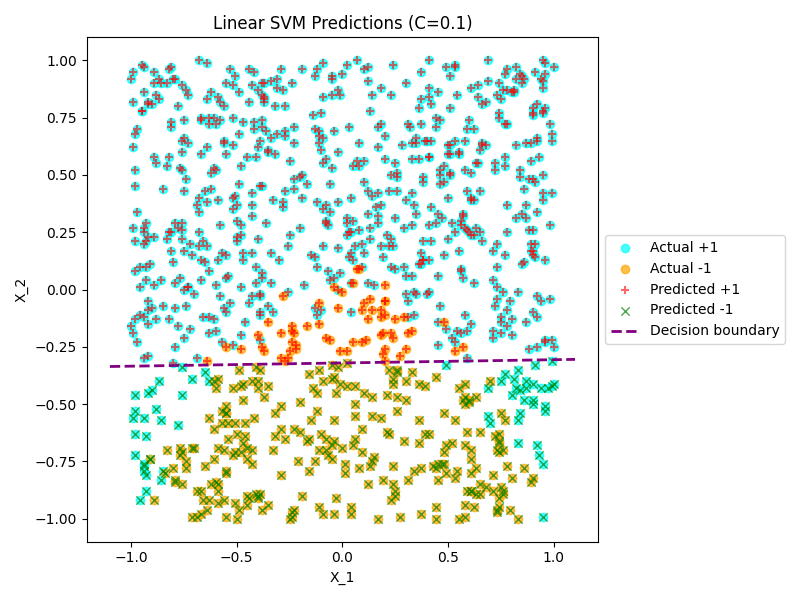
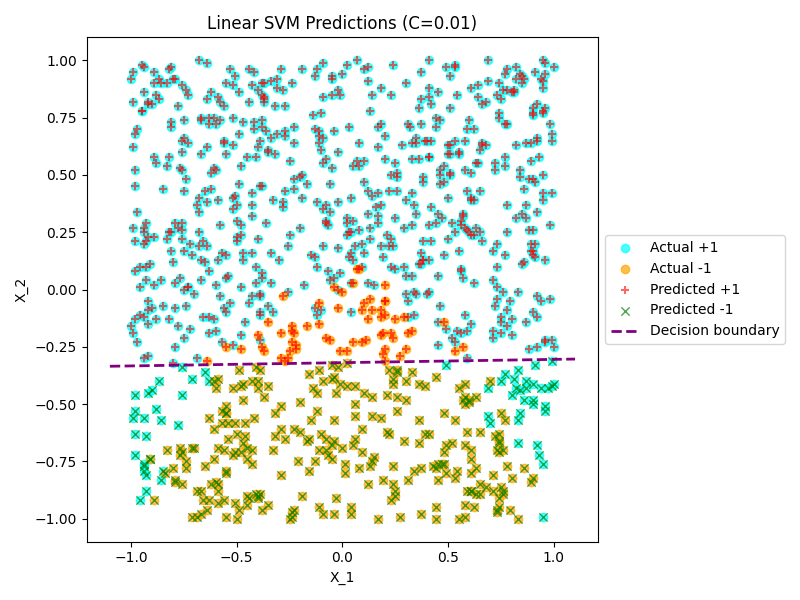
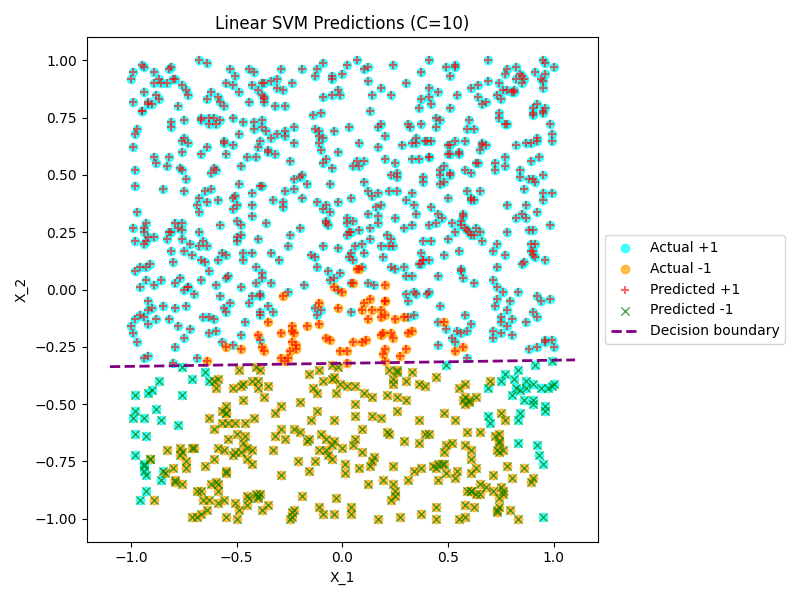
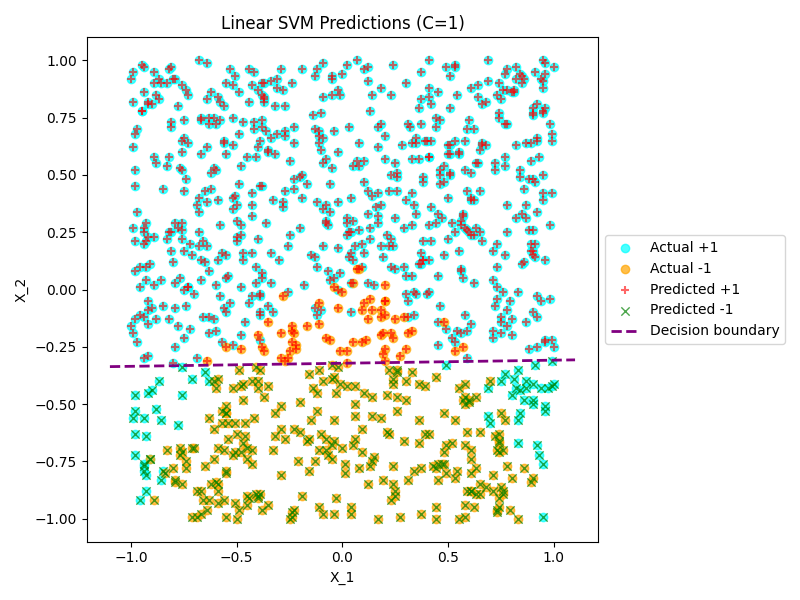
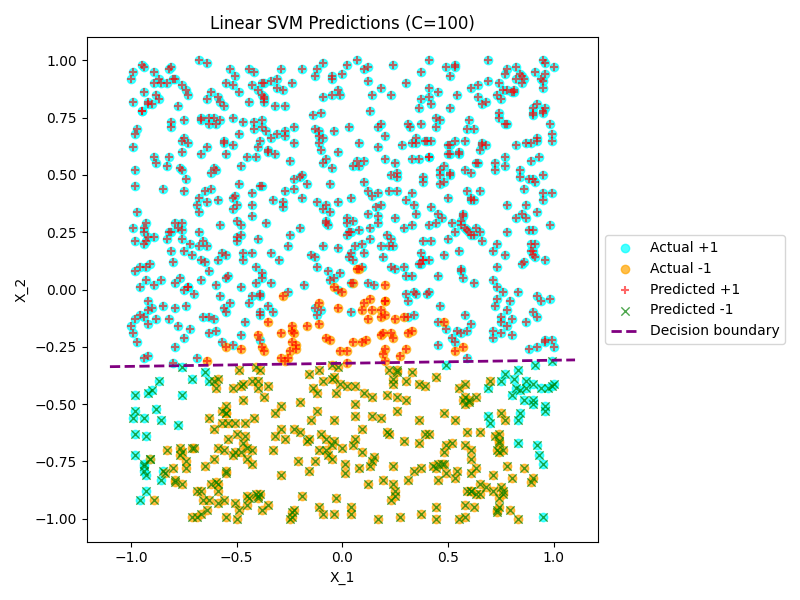
The decision function for just two features (and ) can be computed as (conventional notation), where the weights learned by the SVM are and , while represents the intercept or bias, and each SVM uses this to predict the class of the data point based on the sign of the calculated . The resulting parameters for all SVMs are as follows:

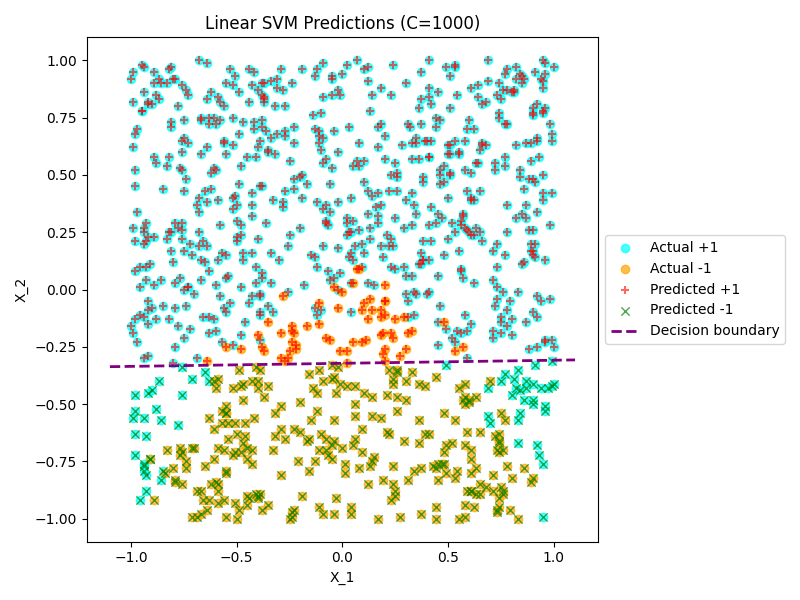
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C |  |  | bias | accuracy |
| 0.001 | 0.00040 | 0.48143 | 0.23188 | 0.85686 |
| 0.01 | -0.01701 | 1.19233 | 0.38027 | 0.87888 |
| 0.1 | -0.02430 | 1.72052 | 0.55092 | 0.87888 |
| 1 | -0.02498 | 1.86583 | 0.59995 | 0.87788 |
| 10 | -0.02524 | 1.88392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

The code segment that produced this was using sklearn’s LinearSVC() function with different C values and max iterations increased to 10000 to increase optimization, and was fitted using the given data as input and target weights, and is present under the [train\_linear\_svm()](#train_linear_svm) function.

**(ii) Plotting the SVM Model’s Predictions**

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*



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

The code segment that produced this was using matplotlib’s scatter function with the given features and predicted values, and is present under the [plot\_svm\_predictions()](#plot_svm_prediction) function.

**(iii) Analysis of Model’s Predictions and Impact of changing C**

The weighting parameter is used in the cost function , where represents the regularization, that describes an inverse relationship where bigger makes the penalty less important, and as a result SVM prioritizes correctly classifying points, whereas a smaller would increase the regularization and result in a wider margin for potential misclassifications.

My interpretation of the table in [Part B (i)](#b1) and [Figures 3](#fig3)-[9](#fig9) is as follows:

* Smaller 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 to be lower than that of the other SVM models with higher values.
* Higher values ranging from 1 to 1000 display much less variance in the values of weights and stronger biases than when , 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 has very little effect on how the SVM model computes its predictions.
* The decision boundary was computed using , which can be derived by setting (classifiers are undecided), which then allows us to solve for . The data points that appear close to the decision boundary reflect how the change in variance in the different values can affect the predicted labels.
* Similar to the Logistic Regression model in [Part A (ii)](#a2), is mostly negative and small, while is positive and larger, so increasing will decrease the probability of predicting +1, while increasing would increase the probability. Furthermore, so once again feature seems to have a greater influence on prediction.
* To conclude, 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) Comparison of Logistic Regression and SVM Models**

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 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 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) Training Logistic Regression Model with squared features**

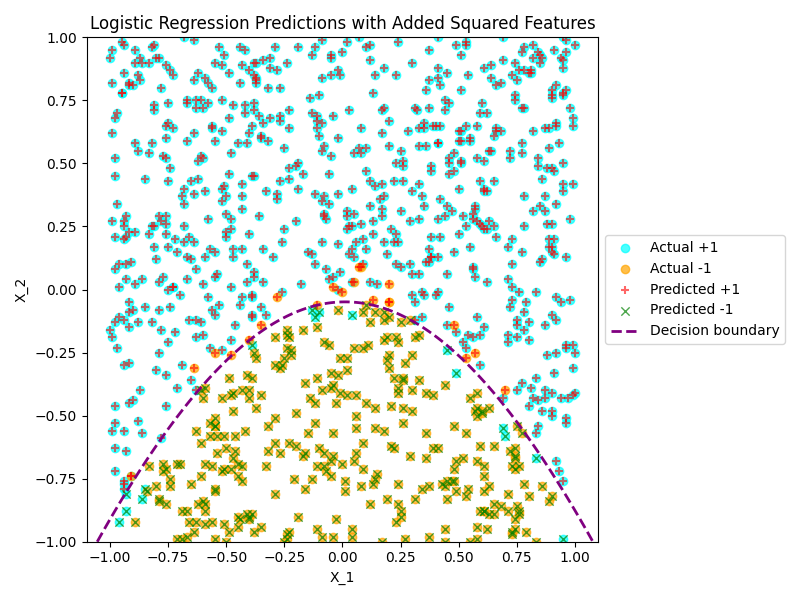
The Logistic Regression model was trained with two additional features that were the squares of the original input features and to allow the model to learn using a quadratic combination of the features as

After fitting, the feature coefficients that were produced were , , and while the intercept is . 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 was using sklearn’s LogisticRegression() function, and was fitted using the given data combined with the squared features as input, and the target weights, and is present under the [train\_log\_regr\_with\_sq()](#train_log_regr_with_sq) function.

**(ii) Plotting the new Logistic Regression Model’s Predictions**

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 was using matplotlib’s scatter function with the given features and predicted values, and is present under the [plot\_sq\_predictions()](#plot_sq_predictions) function.

The interpretation of the coefficients mentioned in [Part C (i)](#c1) is as follows:

* is negative but small, which implies that increasing feature would slightly decrease the probability of predicting +1.
* is positive and larger, which implies that increasing feature would increase the probability of predicting +1.
* is positive and larger, which implies that increasing feature would increase the probability of predicting +1.
* is negative but small, which implies that increasing feature would slightly decrease the probability of predicting +1.
* Based on the absolute value of the coefficients, feature would have the most influence on prediction, followed by

The key difference that the quadratic function introduced for the model to be more refined than the ones in Parts [A](#a2) and [B](#b1) 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. Across all the different models, feature consistently exhibited the strongest influence on classification, driving the decision boundary primarily along the vertical axis.

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 and , and other points form the larger region of +1 labels.
* Points that are farther from the central point of are more likely to be classified as +1, which is reflected in the graph where the given data points at and are indeed labelled as +1.
* Points that are farther from the central point of are more likely to be classified as -1, which is reflected in the graph where the given data points at are mostly -1.

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

**(iii) Comparison of Classifier against a Baseline Predictor**

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, as we already experience how the model is able to linearly separate the data into two distinct regions and make predictions, but introducing non-linear features into the Logistic Regression model allows us to demonstrate how powerful it is to distinctively evaluate the decision boundary and make predictions.

This code segment that produced this was by simply calculating the total number of points in the +1 and -1 classes in the given data and then determining which of them is more commonly found and thus computing the accuracy of that set of prediction labels over the entire data set, which gave a simple accuracy evaluation. This is all present in the [baseline\_accuracy()](#baseline_accuracy) function.

**(iv) Describing the Decision Boundary**

Using the formula stated in [Part C (i)](#c1):, the decision boundary can be derived by setting . We can solve for using the quadratic formula which allows us to solve for . That would turn our equation into , which would give . In code, this is achieved by developing a 2D mesh grid of the given data points, reshaping the results to match the mesh, using matplotlib’s decision\_function() to evaluate the model’s raw output before applying the sigmoid function, and plotting that as a contour.

This code segment is already present in [plot\_sq\_predictions()](#plot_sq_predictions) function, which was evaluated using the aforementioned methodology, and the decision boundary has been displayed in [Figure 10](#fig10) as a dashed purple curve.

**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: X2 = -(theta0 + theta1\*X1)/theta2

    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: X2 = -(bias + weight\_X1\*X1)/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', linestyles='--', 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")

    # eoad 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