Question (a)

My downloaded dataset for this assignment is identified by # id:4–4-4.

Part (i)

I visualized the downloaded data on a 2D plot for each pair of feature values using matplotlib in python. I read the data using code from 1.1 in the appendix and used 1.2 in the appendix so that I could plot X2 vs X1 with different markers and colours depending on the target value.

```
X1, X2, X, y, y_train = read_data()
plt.scatter(X1[y == 1], X2[y == 1], color='green', marker="+")
plt.scatter(X1[y == -1], X2[y == -1], color='red', marker="o")
```

The x-axis contains the value of feature 1 (which is labelled X1), and the y-axis contains the value of feature 2 (which is labelled X2). We can note from the plot that the data is not linearly separable meaning that the data containing -1's and +1's cannot be separated by a straight line.

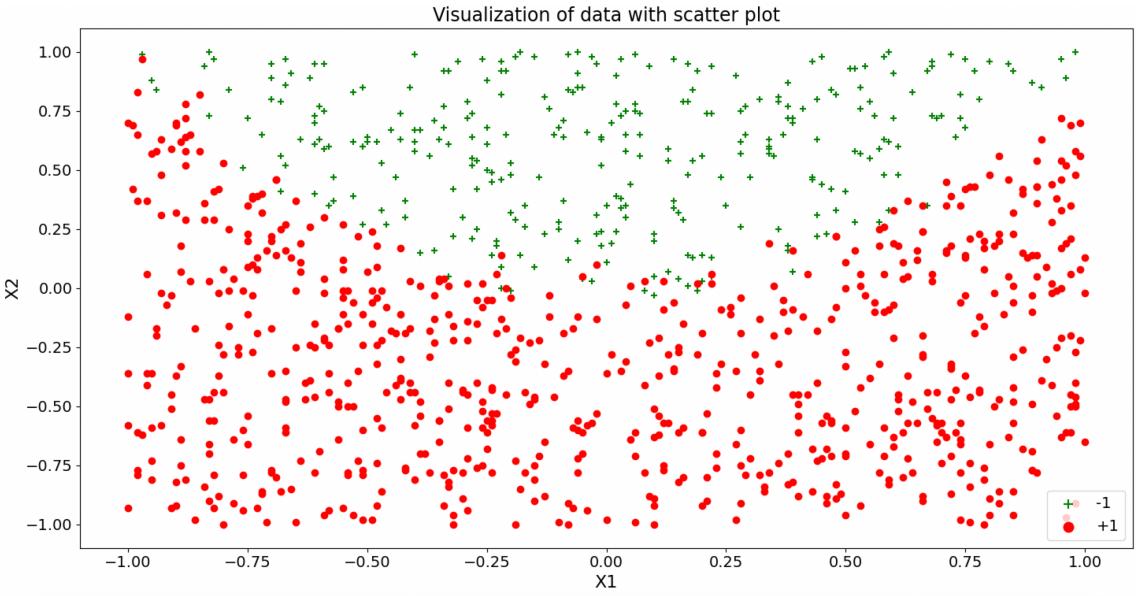

Figure 1: A scatterplot visualizing the data on a 2D plot for each pair of feature values. When the target value is +1, the marker is a green "+" and when the target value is -1, the marker is a red "o", as mentioned in the legend.

Part (ii)

I trained a logistic regression classifier on the data using sklearn in python. I used code from 1.2 in the appendix to train a logistic regression classifier on the data using LogisticRegression from sklearn.linear_model. I used parameters solver='lbfgs' and penalty='none' to ensure that the model has no penalty.

```
model = LogisticRegression(solver='lbfgs', penalty="none")
model.fit(X, y_train)
```

This gave me a logistic regression model with parameters:

$$\theta_0 + \theta_1 x1 + \theta_2 x2 = -2.05262479 + 0.11265394 * x1 + 5.75706825 * x2$$

We can conclude from this model that the feature X2 has the most influence on the prediction. This is because θ_1 is almost 0, meaning feature X1 has very little influence on the prediction. Meanwhile θ_2 is comparatively much bigger at around 5.7, and therefore X2 has the biggest influence on the prediction. We can also analyze that since θ_0 is negative and relatively big, it will influence the prediction y to decrease. Furthermore since θ_1 is positive but almost 0, feature X1 will somewhat increase the prediction y but not by much. Finally, since θ_2 is positive and comparatively large, feature X2 will notably increase the prediction by a significant amount.

Part (iii)

I used the trained logistic regression classifier to predict the target values in the data and then added these predictions to the plot from part (i) using code from 1.4 in the appendix. First, We need to do cross-validation to see how effective our model is for new data. We can use the "hold out" method by splitting our data into 80% training data to train our model, and 20% test data to evaluate prediction performance. We can do this repeatedly in a loop, and choose the iteration with the lowest mean squared error (MSE). The mean squared error measures the average squared difference between the predicted values and the real values.

```
for i in range(iterations):
    Xtrain, Xtest, ytrain, ytest = train_test_split(x, y,
    test_size=test_size)
    model = LogisticRegression().fit(Xtrain, ytrain)
    ypred = np.sign(model.predict(Xtest))

    if i == 0 or mean_squared_error(ytest, ypred) < temporaryLowestMSE:
        temporaryLowestMSE = mean_squared_error(ytest, ypred)
```

We then have to compare our model to a baseline predictor to see whether the prediction error we got on unseen test data is good or not. I did this with a dummy classifier that makes predictions that ignore the input features, and I was able to get a confusion matrix and classification report for both the more complicated model we trained, and also the trivial baseline classifier.

```
dummy =
DummyClassifier(strategy="most_frequent").fit(XtrainWithLowestMSE,
ytrainWithLowestMSE)
ydummy = dummy.predict(XtestWithLowestMSE)
print(confusion_matrix(ytestWithLowestMSE, ypredWithLowestMSE))
print(confusion_matrix(ytestWithLowestMSE, ydummy))
# Results      Complex model | Trivial model
# [[TN][FP]  =  [[130  10]  | [[153   0]
#  [FN][TP]] =   [10  50]]  |  [47   0]]
# Accuracy =       0.9      |    0.765
print(classification_report(ytestWithLowestMSE, ypredWithLowestMSE))
print(classification_report(ytestWithLowestMSE, ydummy))
# Complex model we trained weighted average precision =  0.88
# Trivial model weighted average precision = 0.46
```

We can see that our more complex model is more precise than the dummy model (since $0.88 > 0.46$) which is what we want. Also, using the formula for accuracy (TN+TP/TN+TP+FN+FP), the accuracy of our trained model is more than the dummy model (since $0.9 > 0.765$). This means that our model is more effective than the trivial dummy classifier, which in this case always predicts '-1', meaning our model fits it's purpose in predicting target values well. Finally to plot the data, we can plot the data from (i) as well as the new predictions. We differentiate between the two using different markers and colours as stated in the legend. We plot the decision boundary by rearranging the equation from part (ii) and expressing it in terms of y (i.e. X2):

$$y = -((\theta_0 + x_i * \theta_1)/\theta_2)$$

```
y_val = [-1 * ((theta_zero + x * theta_one) / theta_two) for x in X1]
plt.plot(X1, y_val, linewidth=2, color='black')
```

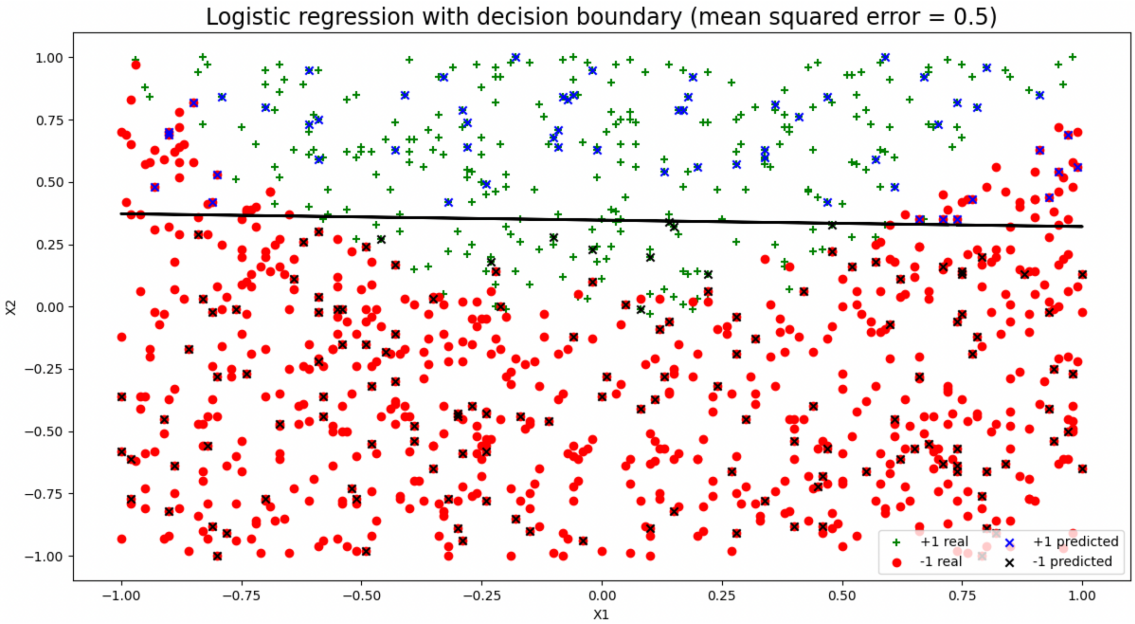

Figure 2: A scatterplot visualizing the data on a 2D plot for each pair of feature values, as well as predictions from the trained logistic regression classifier.

Part (iv)

Using intercept [-2.05262479] and slope [0.11265394, 5.75706825], we can approximate that $(-2.05 + 5.76 * x_1 > 0)$, therefore $(X2 > 0.355)$. Although this is an approximation, we can see in the plot that the decision boundary is close to 0.355 on the y-axis. Therefore the model trained from the training data is roughly correct but not exactly. Also, since the predictions are linearly shaped with the straight line, it does not match the quadratic trained data meaning this shows signs of underfitting.

Question (b)

Part (i)

I trained linear SVM classifiers on the data for C=[0.001, 1, 100] using the LinearSVC function in sklearn using the code in 1.5 of the appendix, with a max iterations of 10,000 so that we can be sure it doesn't fail to converge for values like C=100.

```
for C in [0.001, 1, 100]:
    model = LinearSVC(C=C, max_iter=10000).fit(X, y_train)
```

For C = 0.001, the SVM model and the parameter values are:

$$\theta_0 + \theta_1 x1 + \theta_2 x2 = -0.25716789 + 0.0006304 * x1 + 0.45816137 * x2$$

For C = 1, the SVM model and the parameter values are:

$$\theta_0 + \theta_1 x1 + \theta_2 x2 = -0.67564508 + 0.02089856 * x1 + 1.89978293 * x2$$

For C = 100, the SVM model and the parameter values are:

$$\theta_0 + \theta_1 x1 + \theta_2 x2 = -0.68429681 + 0.0217945 * x1 + 1.9237112 * x2$$

Part (ii)

I predicted the target values in the training data by using the trained classifiers, and plotted these predictions with the actual target values using code from 1.6 in the appendix. Once again, I completed cross validation by repeatedly using the "hold out" method and getting the model with the lowest mean squared error.

```
lowestMSE, modelWithLowestMSE, XtrainWithLowestMSE,
XtestWithLowestMSE, ytrainWithLowestMSE, ytestWithLowestMSE,
ypredWithLowestMSE =
cross_validate_with_hold_out_method_and_multiple_splits(
    5, X, y, 0.2, get_C_from_loop_number[i], 10000)
```

Also, I compared the model with a baseline predictor to see if the prediction error on unseen test data is good or not.

```
dummy =
DummyClassifier(strategy="most_frequent").fit(XtrainWithLowestMSE,
ytrainWithLowestMSE)
ydummy = dummy.predict(XtestWithLowestMSE)
print(confusion_matrix(ytestWithLowestMSE, ypredWithLowestMSE))
print(confusion_matrix(ytestWithLowestMSE, ydummy))
# C=0.001      Complex model | Trivial model
# [[TN][FP]  =  [[141   3]  | [[144   0]
#  [FN][TP]] =   [22  34]]  |  [56   0]]
# Accuracy =       0.875    |    0.72
# C=1          Complex model | Trivial model
# [[TN][FP]  =  [[132  12]  | [[144   0]
#  [FN][TP]] =   [11  45]]  |  [56   0]]
# Accuracy =       0.885    |    0.72
# C=100        Complex model | Trivial model
# [[TN][FP]  =  [[131  15]  | [[146   0]
```

```
# [FN][TP]] =      [9   45]]   |   [54    0]]
# Accuracy =         0.88        |      0.73
print(classification_report(ytestWithLowestMSE, ypredWithLowestMSE))
print(classification_report(ytestWithLowestMSE, ydummy))
# Weighted average precision:
# C=0.001 Complex model = 0.88, Trivial = 0.52
# C=1 Complex model =  0.90, Trivial model = 0.59
# C=100 Complex model =  0.89, Trivial = 0.53
```

To plot the classifier decision boundary, we once again rearrange the equation from part (a) (ii), expressing it in terms of y (i.e. X2): $y = -((\theta_0 + x_i * \theta_1)/\theta_2$

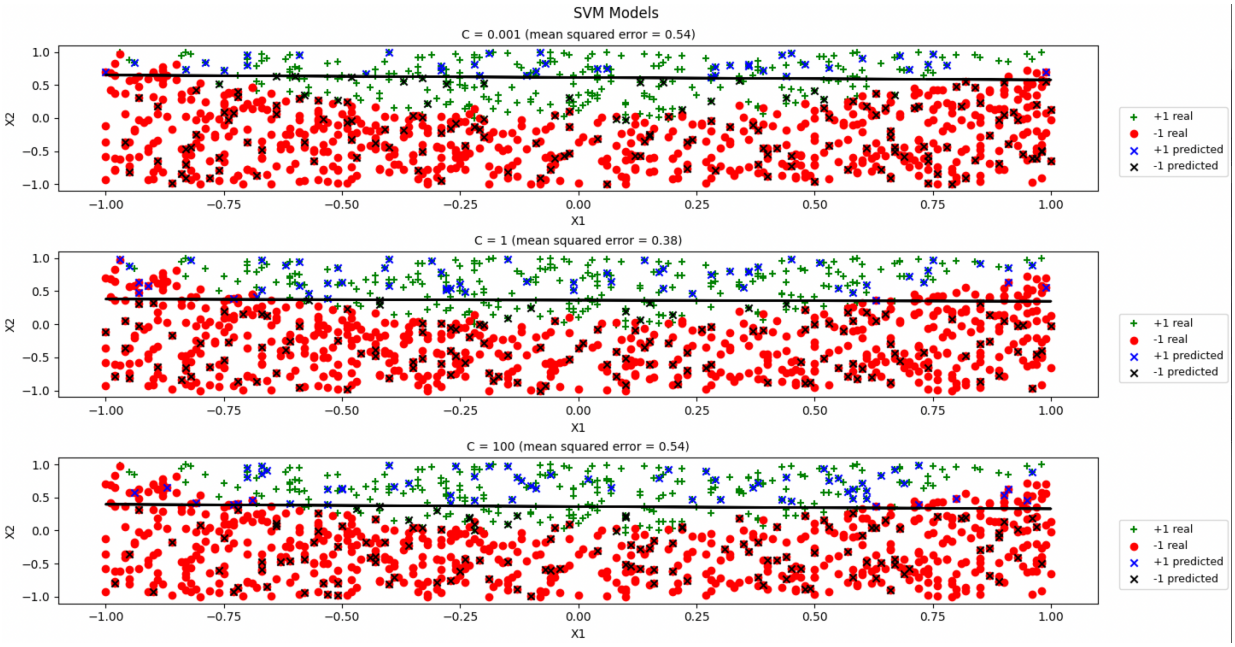

Figure 3: A scatterplot visualizing the data on a 2D plot for each pair of feature values, as well as predictions from the trained SVM models

Part (iii)

A change in C will influence the amount of penalty we apply on the hinge loss function in the SVM model. When C=100, we can see that the parameter values are comparatively big however this means that θ will get penalized very slightly leading to the cost function failing to converge. When C=0.001, we can see that the parameter values are small. This is because C is too small, and θ will get penalized heavily leading to an incorrect model. Therefore, a value such as C=1 is ideal as it is not too low or high. In terms of the impact on the SVM predictions, we see that the lowest mean squared error occurs when C=1 as well. This is because the model is able to predict the target values more accurately since C is not too high or low (and therefore the parameters are not too big or small).

Part (iv)

Comparing the parameters and predictions using the SVM model to logistic regression in (a), we can see that the parameter values for θ_0, θ_1, and θ_2 are much bigger for logistic regression than using the SVM models. This is because we use fairly small values for C which applies a big penalty on the hinge loss function in the SVM models and the parameters are smaller for the SVM model. As for predictions, we can see using

our baseline predictor comparison from part (b) (ii) that C=1 has the best accuracy compared to C=0.001 and C=100 (since $0.885 > 0.875$ and $0.885 > 0.88$) as well as the best weighted average precision (since $0.90 > 0.88$ and $0.90 > 0.89$). Comparing with the logistic regression from part (a) (iii), the SVM model using C=1 is fairly similar in terms of accuracy (since $0.885 \approx 0.9$) and also the weighted average precision (since $0.90 \approx 0.88$). The main differentiator is the mean squared error, where the SVM model using C=1 has a MSE of 0.38 compared to the logistic regression's MSE of 0.5. This means that the SVM model using C=1 is more accurate and makes less mistakes in it's predictions.

Question (c)

Part (i)

I trained a logistic regression classifier with 2 new features using code from 1.7 in the appendix. I created 2 new features by squaring X1 and X2.

```
X1 = np.array(X1)
X2 = np.array(X2)
X1_squared = np.square(X1)
X2_squared = np.square(X2)
X = np.column_stack((X1, X2, X1_squared, X2_squared))
model = LogisticRegression(solver='lbfgs', penalty="none")
model.fit(X, y_train)
```

The model and the trained parameter values are:

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 =$$

$$0.04168601 + 0.0.60008691 * x_1 + 25.16175 * x_2 - 24.62745109 * x_1^2 + 0.38876443 * x_2^2$$

Part (ii)

I used the trained logistic classifier to predict target values in the training data. I then plotted the predictions with the actual target values using code from 1.8 in the appendix. I completed cross validation by repeatedly using the "hold out" method and getting the model with the lowest mean squared error, using the same method shown in part (b) (ii). I also compared with a baseline predictor which is discussed in part (iii).

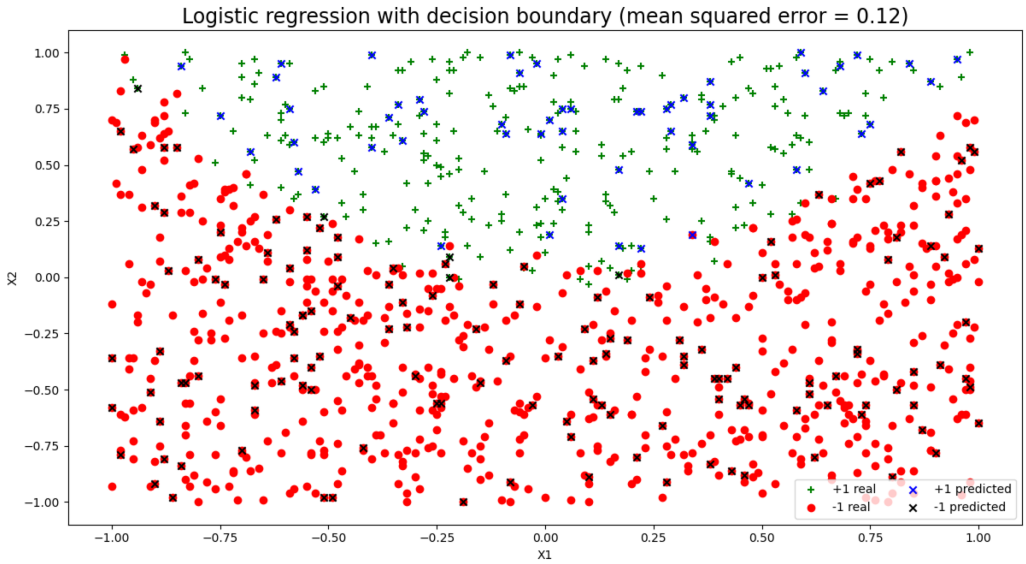

Figure 4: A scatterplot visualizing the data on a 2D plot for each pair of feature values, as well as predictions from the trained logistic regression model with 4 features

These predictions are more accurate than those in part (a) and (b) because the mean squared error is considerably lower at 0.12, compared to 0.38 using C=1 in (b) and 0.5 in (a). The mean squared error measures the average squared difference between the predicted values and the real values. This means the model makes less mistakes and more accurate predictions.

Part (iii)

I compared the model with a baseline predictor that predicts the most common class to see if the prediction error on unseen test data is good or not.

```
dummy = DummyClassifier(strategy="most_frequent").fit(XtrainWithLowestMSE,
ytrainWithLowestMSE)
    ydummy = dummy.predict(XtestWithLowestMSE)
    print(confusion_matrix(ytestWithLowestMSE, ypredWithLowestMSE))
    print(confusion_matrix(ytestWithLowestMSE, ydummy))
    # Results      Complex model | Trivial model
    # [[TN][FP]  =  [[143   0]  | [[143   0]
    #  [FN][TP]] =   [4   53]]  |  [57   0]]
    # Accuracy =       0.98     |    0.715
    print(classification_report(ytestWithLowestMSE, ypredWithLowestMSE))
    print(classification_report(ytestWithLowestMSE, ydummy))
    # Complex model we trained weighted average precision = 0.98
    # Trivial model weighted average precision = 0.51
```

We can see that our model has an accuracy and weighted average precision of 0.98, compared to the baseline predictor having an accuracy of 0.715 and weighted average precision of 0.51. This means that our model is more accurate than a dummy classifier that makes predictions based on the most common class. We can conclude from this that the quality of predictions from our model is high, and the effort of training our model is worth it compared to a dummy model.

Part (iv)

I plotted the decision boundary which was a quadratic. The general formula for a quadratic equation is $ax^2 + bx + c = 0$. I rearranged the formula $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 = 0$ to make it in terms of y (i.e. X2). This resulted in:

$$y = -((\theta_3/\theta_2) * x_1^2) - ((\theta_1/\theta_2) * x_1) - (\theta_0/\theta_2) - ((\theta_4/\theta_2) * x_2^2)$$

Hence, we can rearrange to get $a = -(\theta_3/\theta_2)$, $b = (\theta_1/\theta_2)$, $c = -(\theta_0/\theta_2) - (\theta_4/\theta_2 * x_2^2)$

```
    sorted_X1 = np.sort(X1)
    sorted_X2 = np.sort(X2)
    a = (-1 * (theta_three/theta_two))
    b = (-1 * (theta_one/theta_two))
    c = (-1 * (theta_zero/theta_two)) + ((-1 * (theta_four/theta_two)) *
    np.power(sorted_X2, 2))
    quadratic_boundary = a * np.power(sorted_X1, 2) + b * sorted_X1 + c
    plt.plot(sorted_X1, quadratic_boundary, linewidth=2, color='black')
```

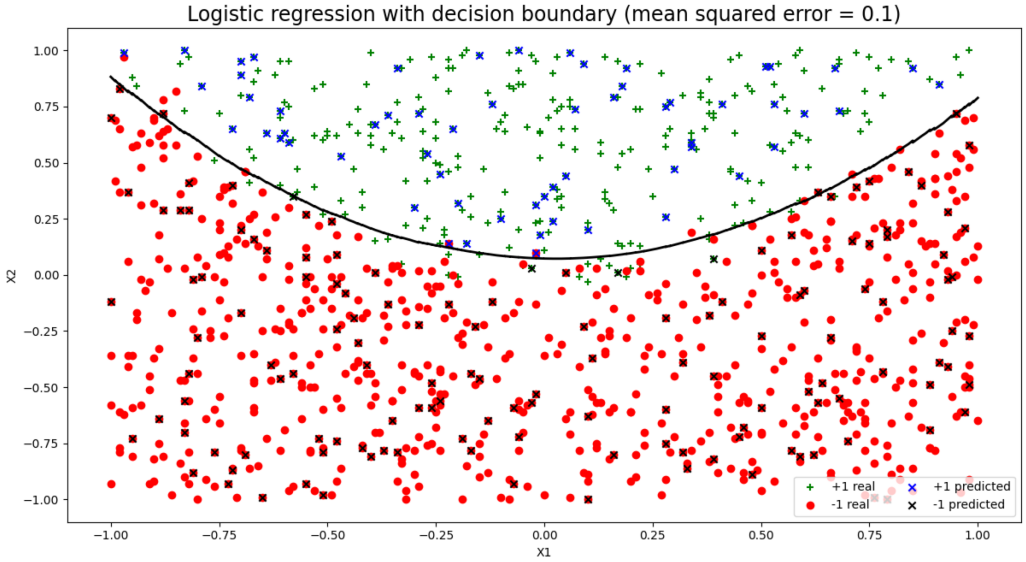

Figure 5: A scatterplot showing the classifier decision boundary for the predictions from the trained logistic regression model with 4 features

1 Appendix

1.1 Helper function to read data from CSV

```
import numpy as np
import pandas as pd

def read_data():
    df = pd.read_csv("week2.csv")
    X1 = df.iloc[:, 0]
    X2 = df.iloc[:, 1]
    X = np.column_stack((X1, X2))
    y = df.iloc[:, 2]
    ytrain = np.sign(y)
    return X1, X2, X, y, ytrain
```

1.2 Question (a) (i)

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from helper_functions import read_data
matplotlib.use('TkAgg')

def part_one():
    X1, X2, X, y, y_train = read_data()
    plt.rcParams['figure.constrained_layout.use'] = True
    plt.scatter(X1[y == 1], X2[y == 1], color='green', marker="+")
    plt.scatter(X1[y == -1], X2[y == -1], color='red', marker="o")
    plt.title("Visualization of data with scatter plot", fontsize=17)
    plt.ylabel("X2", fontsize=16)
    plt.xlabel("X1", fontsize=16)
    plt.legend(["-1", "+1"], loc='lower right', fontsize=14,
    markerscale=1.5)
    plt.xticks(np.arange(-1, 1.01, 0.25), fontsize=14)
    plt.yticks(np.arange(-1, 1.01, 0.25), fontsize=14)
    plt.show()
```

1.3 Question (a) (ii)

```
from sklearn.linear_model import LogisticRegression
from helper_functions import read_data

def part_two():
    X1, X2, X, y, y_train = read_data()
    model = LogisticRegression(solver='lbfgs', penalty="none")
    model.fit(X, y_train)
    print("Intercept:", model.intercept_)
    print("Slope:", model.coef_)
```

1.4 Question (a) (iii)

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, confusion_matrix,
classification_report
from sklearn.dummy import DummyClassifier
import matplotlib
import matplotlib.pyplot as plt
from helper_functions import read_data

def part_three():
    X1, X2, X, y, y_train = read_data()
    lowestMSE, modelWithLowestMSE, XtrainWithLowestMSE,
    XtestWithLowestMSE, ytrainWithLowestMSE, ytestWithLowestMSE,
    ypredWithLowestMSE =
    cross_validate_with_hold_out_method_and_multiple_splits(
        5, X, y, 0.2)
```

```
    dummy =
    DummyClassifier(strategy="most_frequent").fit(XtrainWithLowestMSE,
    ytrainWithLowestMSE)
    ydummy = dummy.predict(XtestWithLowestMSE)
    print(confusion_matrix(ytestWithLowestMSE, ypredWithLowestMSE))
    print(confusion_matrix(ytestWithLowestMSE, ydummy))
    print(classification_report(ytestWithLowestMSE, ypredWithLowestMSE))
    print(classification_report(ytestWithLowestMSE, ydummy))
    XtestAsDataFrame = pd.DataFrame(XtestWithLowestMSE)
    X1TestData = XtestAsDataFrame[0]  # X1 as test data
    X2TestData = XtestAsDataFrame[1]  # X2 as test data
    plt.scatter(X1[y == 1], X2[y == 1], color='green', marker="+")
    plt.scatter(X1[y == -1], X2[y == -1], color='red', marker="o")
    # Plot the predictions
    plt.scatter(X1TestData[ypredWithLowestMSE == 1],
    X2TestData[ypredWithLowestMSE == 1], color='blue', marker="x")
    plt.scatter(X1TestData[ypredWithLowestMSE == -1],
    X2TestData[ypredWithLowestMSE == -1], color='black', marker="x")
    plt.title("Logistic regression with decision boundary (mean squared
    error = " + str(lowestMSE) + ")", fontsize=17, wrap=True)
    plt.ylabel("X2")
    plt.xlabel("X1")
    slopes = modelWithLowestMSE.coef_.T
    theta_zero = modelWithLowestMSE.intercept_
    theta_one = slopes[0]
    theta_two = slopes[1]
    y_val = [-1 * ((theta_zero + x * theta_one) / theta_two) for x in X1]
    plt.plot(X1, y_val, linewidth=2, color='black')
    plt.legend(["+1 real", "-1 real", "+1 predicted", "-1 predicted"],
    loc='lower right', ncol=2, fontsize=10)
    plt.show()


def cross_validate_with_hold_out_method_and_multiple_splits(iterations, x,
y, test_size):
    temporaryLowestMSE = 0
    modelWithLowestMSE = 0
    XtrainWithLowestMSE = 0
    XtestWithLowestMSE = 0
    ytrainWithLowestMSE = 0
    ytestWithLowestMSE = 0
    ypredWithLowestMSE = 0
    for i in range(iterations):
        Xtrain, Xtest, ytrain, ytest = train_test_split(x, y,
        test_size=test_size)
        model = LogisticRegression().fit(Xtrain, ytrain)
        ypred = np.sign(model.predict(Xtest))
        if i == 0 or mean_squared_error(ytest, ypred) < temporaryLowestMSE:
            # At this point, we are on a new iteration of training our
            model and have a new lowest MSE.
            # We can save all our data associated with this model.
            temporaryLowestMSE = mean_squared_error(ytest, ypred)
            modelWithLowestMSE = model
            XtrainWithLowestMSE = Xtrain
            XtestWithLowestMSE = Xtest
```

```
            ytrainWithLowestMSE = ytrain
            ytestWithLowestMSE = ytest
            ypredWithLowestMSE = ypred
    return temporaryLowestMSE, modelWithLowestMSE, XtrainWithLowestMSE,
    XtestWithLowestMSE, ytrainWithLowestMSE, ytestWithLowestMSE,
    ypredWithLowestMSE
```

1.5 Question (b) (i)

```
from sklearn.svm import LinearSVC
from helper_functions import read_data

def part_one():
    X1, X2, X, y, y_train = read_data()
    for C in [0.001, 1, 100]:
        model = LinearSVC(C=C, max_iter=10000).fit(X, y_train)
        print("C: ", C)
        print("Intercept: ", model.intercept_)
        print("Slope: ", model.coef_)
```

1.6 Question (b) (ii)

```
import pandas as pd
from sklearn.svm import LinearSVC
import matplotlib
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, confusion_matrix,
classification_report
from helper_functions import read_data
from sklearn.dummy import DummyClassifier
import numpy as np

def part_two():
    X1, X2, X, y, y_train = read_data()
    get_C_from_loop_number = {0: 0.001, 1: 1, 2: 100}
    for i in range(3):
        lowestMSE, modelWithLowestMSE, XtrainWithLowestMSE,
        XtestWithLowestMSE, ytrainWithLowestMSE, ytestWithLowestMSE,
        ypredWithLowestMSE =
        cross_validate_with_hold_out_method_and_multiple_splits(
            5, X, y, 0.2, get_C_from_loop_number[i], 10000)
        dummy =
        DummyClassifier(strategy="most_frequent").fit(XtrainWithLowestMSE,
        ytrainWithLowestMSE)
        ydummy = dummy.predict(XtestWithLowestMSE)
        print(confusion_matrix(ytestWithLowestMSE, ypredWithLowestMSE))
        print(confusion_matrix(ytestWithLowestMSE, ydummy))
        print(classification_report(ytestWithLowestMSE,
        ypredWithLowestMSE))
        print(classification_report(ytestWithLowestMSE, ydummy))
        XtestAsDataFrame = pd.DataFrame(XtestWithLowestMSE)
        X1TestData = XtestAsDataFrame[0]  # X1 as test data
        X2TestData = XtestAsDataFrame[1]  # X2 as test data
        plt.rcParams['figure.constrained_layout.use'] = True
        plt.suptitle("SVM Models")
```

```
        plt.subplot(311 + i)
        plt.title("C = " + str(get_C_from_loop_number[i]) + " (mean
        squared error = " + str(lowestMSE) + ")", fontsize=10)
        plt.scatter(X1[y == 1], X2[y == 1], color='green', marker="+")
        plt.scatter(X1[y == -1], X2[y == -1], color='red', marker="o")
        plt.scatter(X1TestData[ypredWithLowestMSE == 1],
        X2TestData[ypredWithLowestMSE == 1], color='blue', marker="x")
        plt.scatter(X1TestData[ypredWithLowestMSE == -1],
        X2TestData[ypredWithLowestMSE == -1], color='black',marker="x")
        plt.ylabel("X2")
        plt.xlabel("X1")
        slopes = modelWithLowestMSE.coef_.T
        theta_zero = modelWithLowestMSE.intercept_
        theta_one = slopes[0]
        theta_two = slopes[1]
        y_val = [-1 * ((theta_zero + x * theta_one) / theta_two) for x in
        X1]
        plt.plot(X1, y_val, linewidth=2, color='black')
        plt.legend(["+1 real", "-1 real", "+1 predicted", "-1 predicted"],
        loc=(1.02, 0.1), fontsize=9)
    plt.show()
```

1.7 Question (c) (i)

```
from helper_functions import read_data
from sklearn.linear_model import LogisticRegression
import numpy as np

def part_one():
    X1, X2, X, y, y_train = read_data()
    X1 = np.array(X1)
    X2 = np.array(X2)
    X1_squared = np.square(X1)
    X2_squared = np.square(X2)
    X = np.column_stack((X1, X2, X1_squared, X2_squared))
    model = LogisticRegression(solver='lbfgs', penalty="none")
    model.fit(X, y_train)
    print("Intercept:", model.intercept_)
    print("Slope:", model.coef_)
```

1.8 Question (c) (ii)

```
from helper_functions import read_data
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error, confusion_matrix,
classification_report
from sklearn.model_selection import train_test_split
from sklearn.dummy import DummyClassifier
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
matplotlib.use('TkAgg')

def part_two():
    X1, X2, X, y, y_train = read_data()
```

```
X1 = np.array(X1)
X2 = np.array(X2)
X1_squared = np.square(X1)
X2_squared = np.square(X2)
X = np.column_stack((X1, X2, X1_squared, X2_squared))
lowestMSE, modelWithLowestMSE, XtrainWithLowestMSE,
XtestWithLowestMSE, ytrainWithLowestMSE, ytestWithLowestMSE,
ypredWithLowestMSE =
cross_validate_with_hold_out_method_and_multiple_splits(
    5, X, y, 0.2)
dummy =
DummyClassifier(strategy="most_frequent").fit(XtrainWithLowestMSE,
ytrainWithLowestMSE)
ydummy = dummy.predict(XtestWithLowestMSE)
print(confusion_matrix(ytestWithLowestMSE, ypredWithLowestMSE))
print(confusion_matrix(ytestWithLowestMSE, ydummy))
print(classification_report(ytestWithLowestMSE, ypredWithLowestMSE))
print(classification_report(ytestWithLowestMSE, ydummy))
XtestAsDataFrame = pd.DataFrame(XtestWithLowestMSE)
X1TestData = XtestAsDataFrame[0]  # X1 as test data
X2TestData = XtestAsDataFrame[1]  # X2 as test data
plt.scatter(X1[y == 1], X2[y == 1], color='green', marker="+")
plt.scatter(X1[y == -1], X2[y == -1], color='red', marker="o")
plt.scatter(X1TestData[ypredWithLowestMSE == 1],
X2TestData[ypredWithLowestMSE == 1], color='blue', marker="x")
plt.scatter(X1TestData[ypredWithLowestMSE == -1],
X2TestData[ypredWithLowestMSE == -1], color='black', marker="x")
plt.title("Logistic regression with decision boundary (mean squared
error = " + str(lowestMSE) + ")", fontsize=17, wrap=True)
plt.ylabel("X2")
plt.xlabel("X1")
slopes = modelWithLowestMSE.coef_.T
theta_zero = modelWithLowestMSE.intercept_
theta_one = slopes[0]
theta_two = slopes[1]
theta_three = slopes[2]
theta_four = slopes[3]
sorted_X1 = np.sort(X1)
sorted_X2 = np.sort(X2)
a = (-1 * (theta_three/theta_two))
b = (-1 * (theta_one/theta_two))
c = (-1 * (theta_zero/theta_two)) + ((-1 * (theta_four/theta_two)) *
np.power(sorted_X2, 2))
quadratic_boundary = a * np.power(sorted_X1, 2) + b * sorted_X1 + c
plt.plot(sorted_X1, quadratic_boundary, linewidth=2, color='black')
plt.legend(["+1 real", "-1 real", "+1 predicted", "-1 predicted"],
loc='lower right', ncol=2, fontsize=10)
plt.show()
```