Machine Learning Week 2 Assignment

Paraic O'Reilly: 19335497 Dataset: # id: 23-46-23

(a)

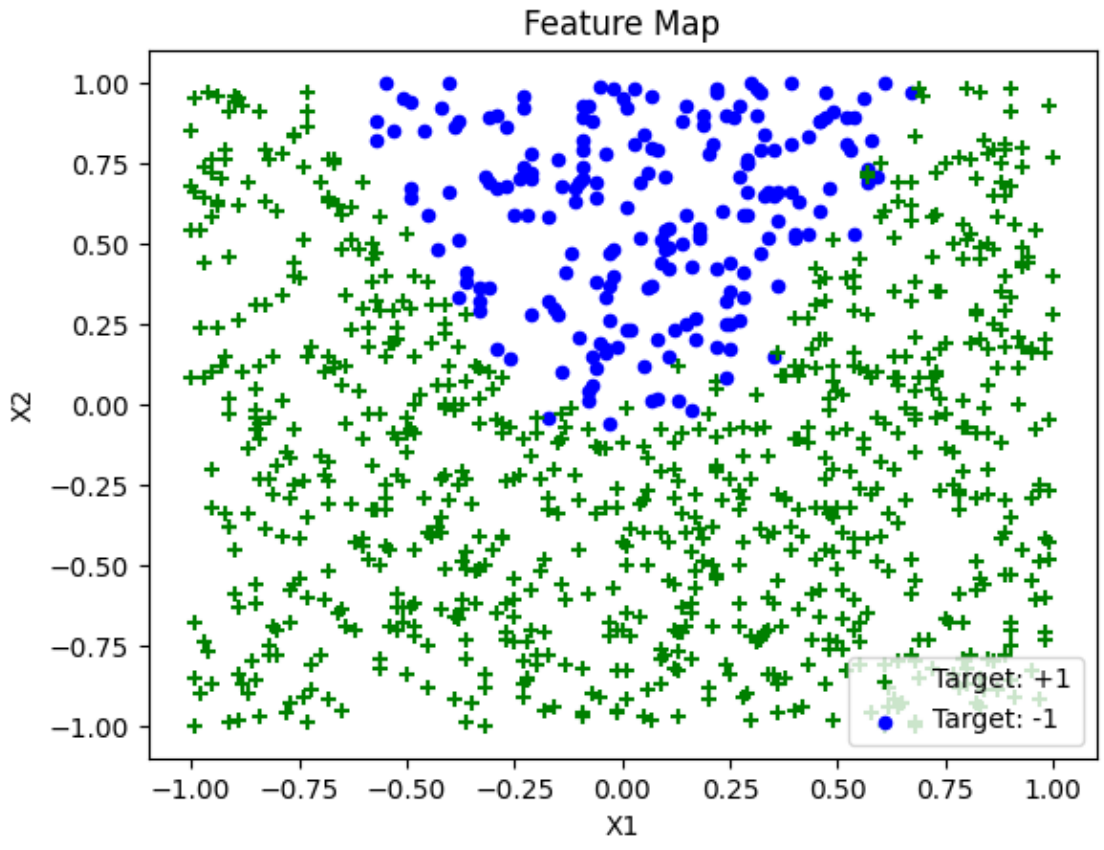
(i). Visualise the data you downloaded by placing a marker on a 2D plot for each pair of features values

A simple function is shown below which was used to map the data onto a 2D plot as specified. X1 is on the X axis and X2 is on the y axis. Each feature has a target variable of either +1 or -1. The function is show below:

The code iterates through X (which contains a column for each feature) and plots it relative to the target variable.

```
for i in range(len(X)-1):
    marker = markers[y.iloc[i]]
    color = colors[y.iloc[i]]
    marker_size = 40 if color == 'green' else 20
    ax.scatter(X[i, 0], X[i, 1], marker=marker,s=marker_size, color=color)
```

The +1s are represented by a green plus and the -1s are represented by a blue circle. The generated plot is shown below. The data consists of 796 positives and 203 negative values.



(ii) Give the logistic regression model for predictions and report the parameter values of the trained model. Discuss e.g. which feature has most influence on the prediction, which features cause the prediction to increase and which causes it to decrease.

The sklearn Logistic Regression takes two inputs:

- X the features
- y the target variable

The two features X1 and X2 were combined using a numpy function column stack into a variable X.

```
X1=df.iloc[:,0]
X2=df.iloc[:,1]
X=np.column_stack((X1,X2))
```

These were then inserted into the logistic regression model and fit to the training data.

The following code was used to generate the linear model.

```
#a-(ii)
from sklearn.linear_model import LogisticRegression

# Initialize and train the logistic regression model
model = LogisticRegression(penalty='none', solver='lbfgs')
model.fit(X, y)
```

The following code was used to get the model parameters.

```
# Getting the coefficients (parameters) of the trained model
coef_feature1, coef_feature2 = model.coef_.flatten()
intercept = model.intercept_[0]
print('intercept = %f, slopes = (%f, %f)' % (model.intercept_, *model.coef_[0]))
```

Parameter	Result
θ_0	2.31
θ_1	-0.15
θ_2	-3.68

The magnitude of X1 and X2 is representative of their influence on prediction. X2 is considerably more influential on predictions than X1. Both parameters have a negative relationship with the target variable, so large values will decrease the probability of predicting the class.

(iii) Now use the trained logistic regression classifier to predict the target values in the training data. data. Add these predictions to the 2D plot you generated in part (i).

The following code snippet was used to generate predictions from the training data:

```
predictions = model.predict(X)
```

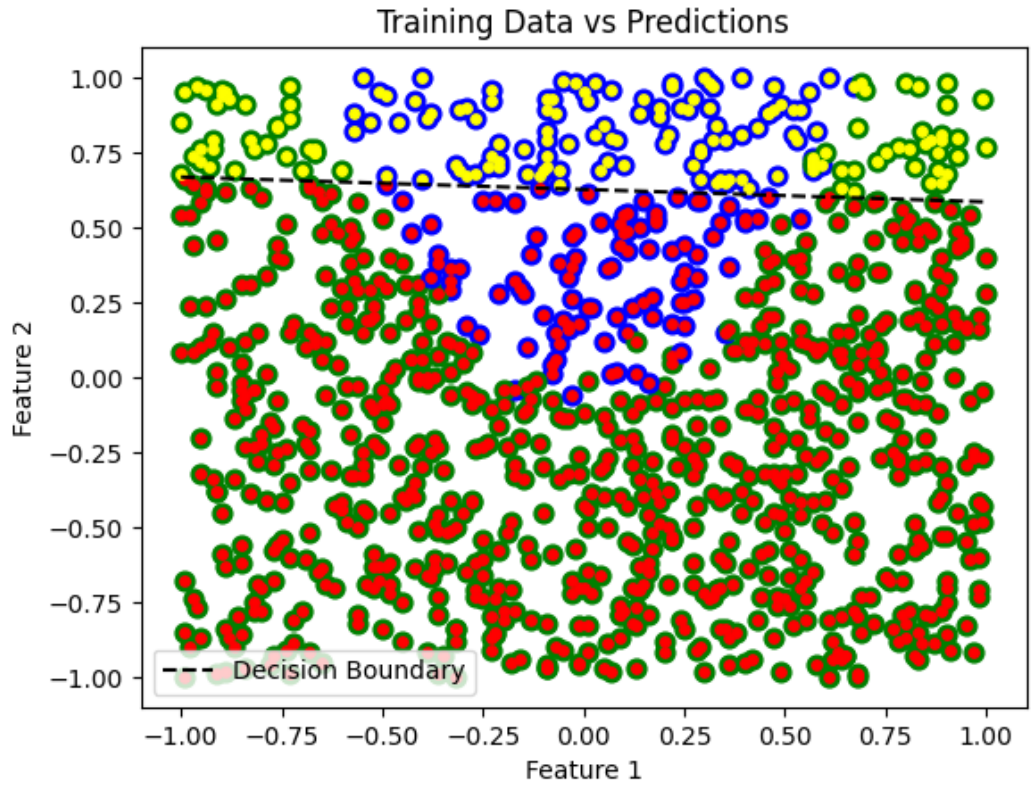
These predictions were then plotted onto the 2D plot from part a using the same plot function as before.

The decision boundary was calculated using the parameters from part (ii). The parameters were inputted into the following equation of the line.

$$y = \frac{x\theta_1 + \theta_0}{\theta_2}$$

Below in Figure X, we can see the updated 2D plot. Green (+1) and blue (-1) dots represent the target values. Overlaid on top of the target values we have the model predictions. Red dots represent the model predicting +1 and yellow dots represent the model predicting -1. Therefore, a correct prediction is represented by a green circle with a red dot inside or a blue circle with a yellow dot inside.

The decision boundary is represented by the dotted black line. This clearly divides the plot into two parts. Everything below the boundary represents a prediction of +1 and everything above represents a prediction of -1.



(iv) Briefly comment on how the predictions and the training data compare

The updated 2D plot gives a clear visualisation of the model's performance. As it is a simple linear model, the decision boundary is a straight line that divides the predictions of the two classes. Although this provides a reasonably high accuracy (83%), the model is underfitting. We can see three areas on the graph where the model is predicting wrong. The upper middle section (red on blue), and the two top corners (yellow on green) are predicting incorrectly. This is because a linear model cannot fully capture this data.

We can place a numerical value to this visualisation by calculating the accuracy of the model. The accuracy represents the total correct predictions over the total number of predictions. The calculated accuracy score for the model was 83%. This was calculated using the accuracy_score function from sklearn. This coincides with our visual representation of the predictions as only 3 small sections near the decision boundary were incorrect.

(b) Use sklearn to train linear SVM classifiers on your data

(i) Train linear SVM classifiers for a wide range of values of the penalty parameter C e.g. C = 0.001, C = 1, C = 100. Give the SVM model for predictions and report the parameter values of each trained model.

To create an SVM classifier the following code was used:

```
SVCmodel = LinearSVC(C=c).fit(X, y)
```

The SVM classifiers trained used four values, C=.01, C=1,C=50 , C=100, C=1000.

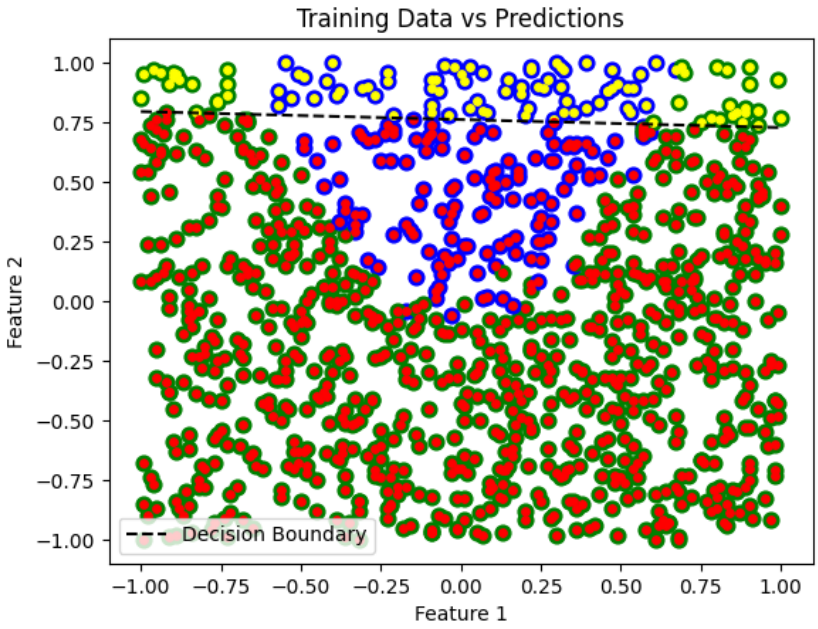
The table below shows the parameters for each model. The parameters were calculated using the same method for part a.

C	θ_0	θ_1	θ_2
.01	0.58	-0.03	-0.77
1	0.76	-0.05	-1.21
50	0.76	-0.05	-1.22
100	0.74	-0.04	-1.23
1000	0.91	-0.65	-1.16

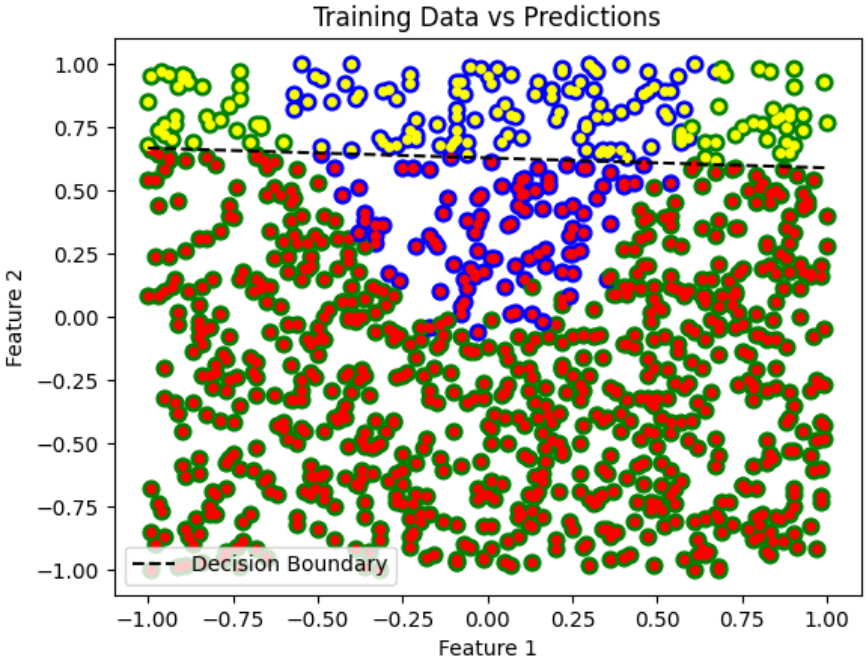
(ii) Use each of these trained classifiers to predict the target values in the training data. Plot these predictions and the actual target values from the data, together with the classifier decision.

The same plotting function was used to plot the predictions vs the training data for each of the models.

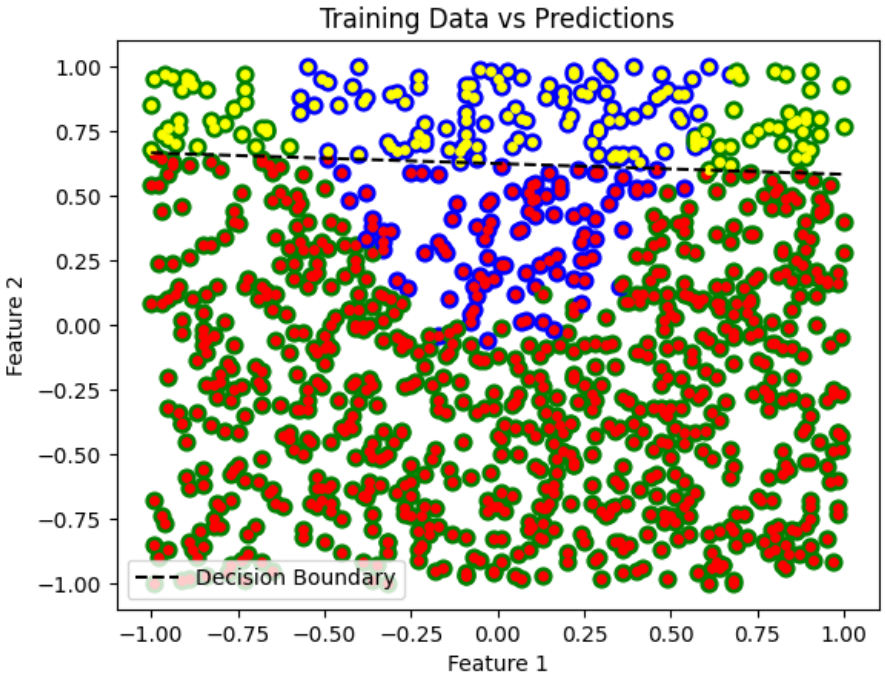
C=0.01, Accuracy: 0.828829



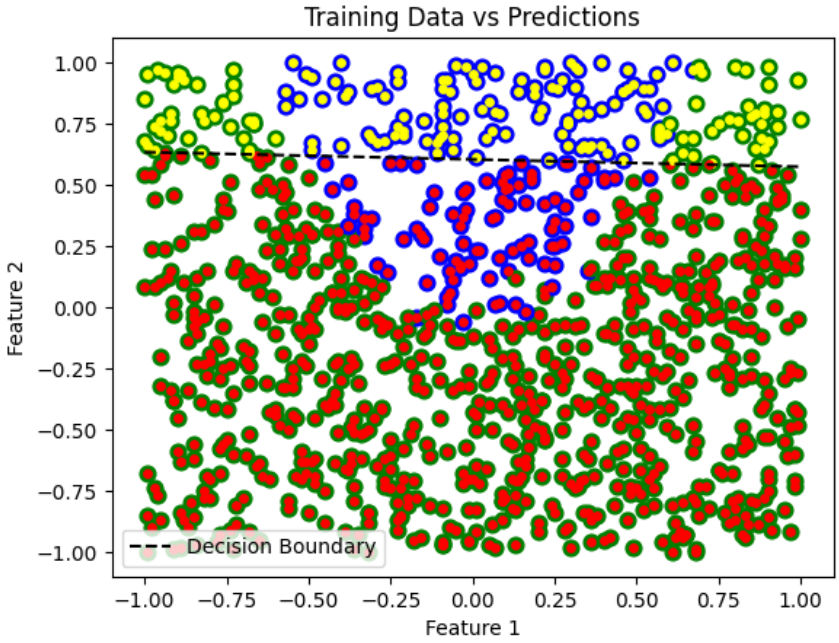
C=1, Accuracy = 0.835%



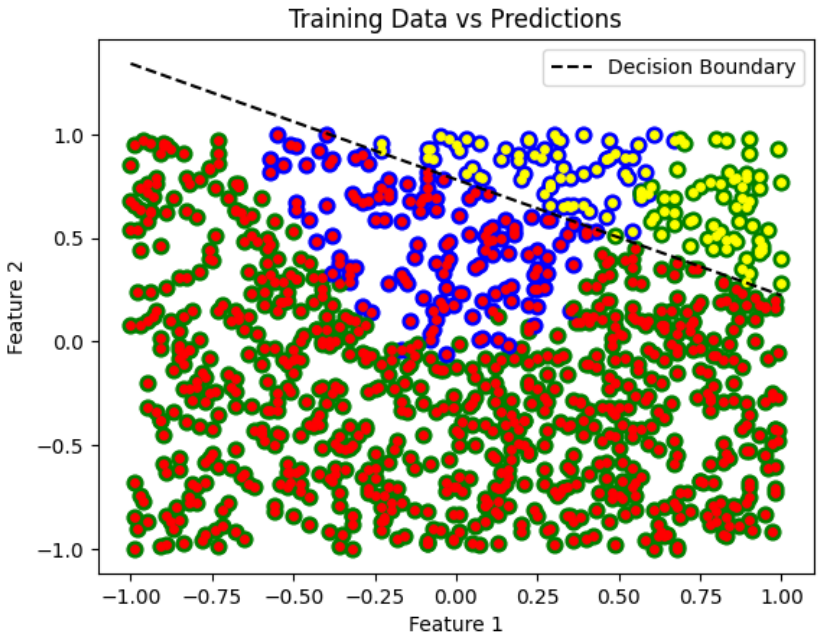
C=50, Accuracy = 0.835%



C=100, Accuracy= 0.832%



C=1000, Accuracy = 80%



(iii) What is the impact on the model parameters of changing C, and why ? What is the impact on the SVM predictions?

Changing the value of C, affects the fit of the model through regularisation. A smaller value of C can helps combat overfitting by reducing the model complexity. A larger value of C will fit to the training data more closely but may result in overfitting.

A larger value of C will have more fitted and adjusted parameters whereas a smaller value will lead to more simple parameters.

(iv) How do the SVM model parameters and predictions compare to those of the logistic regression model in part a

The values for C which gave the highest accuracy were 1 and 50 which both resulted in the same accuracy as the original model, 83%.

All the other values of C resulted in slight decreases in accuracy (1%-4%). This was due to some slight overfitting which was visible in the graphs. The slight slant of the decision boundary was less accurate than the original model.

Increasing C did not result in higher accuracy as the linear model was failing to capture some dimensionality in the data. As C increased accuracy would have continued do drop off accuracy slightly.

Larger and smaller values also caused some convergence issues which appeared as warnings when running the model.

(c)

(i) Now create two additional features by adding the square of each feature (i.e. giving four features in total). Train a logistic regression classifier. Give the model and the trained parameter values.

Two additional features were added to the original column stack of features. The square of X1 and X2 were added using the following code snippet:

```
X1=df.iloc[:,0]
X2=df.iloc[:,1]
X3=X1**2
X4=X2**2
Xe = np.column_stack((X1, X2, X3, X4))
```

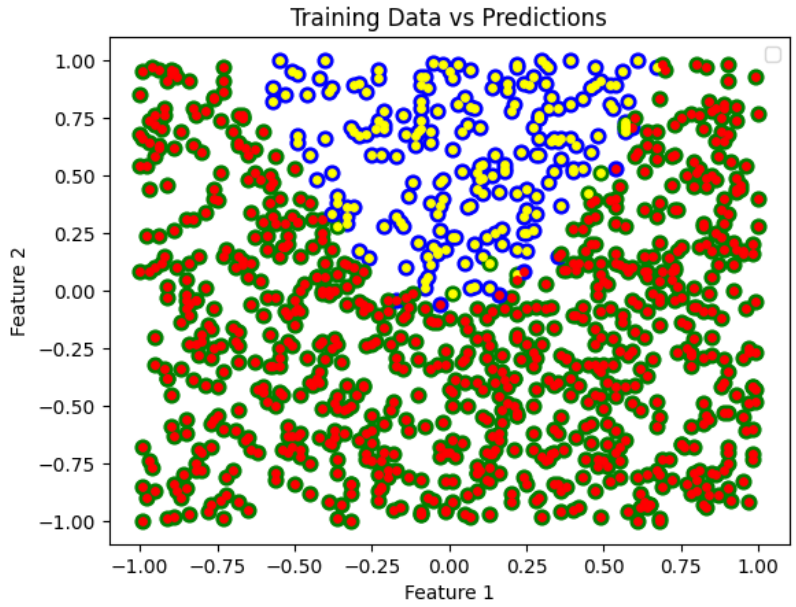
θ_0	θ_1	θ_2	θ_3	θ_4
-1.25	-1.04	-28.06	67.67	-1.25

(ii) Use the trained classifier to predict the target values in the training data. Plot these predictions and the actual target values from the data using the same style of plot as before i.e. using just the two original features as x and y axes. Compare and discuss. How do the predictions compare with those in parts (a) and (b)

The same function was used again from part A to plot the predictions vs target values.

From this graph we can see a considerable improvement in the performance. The original linear model was underfitting considerably. A straight line as the decision boundary was failing to capture the dimensionality of the data and did not have much better performance than a model that just predicted the most common class.

The updated model fits the data almost perfectly. The few errors on the graph are right on the decision boundary and are not due to the model underfitting as opposed to before when chunks of the graph were predicted incorrectly.



(iii) Compare the performance of the classifier against a reasonable baseline predictor, e.g. one that always predicts the most common class

The dataset contains 796 positive values and 203 negative values.

Therefore, a predictor that always predicted the most common class would be right 79% of the time. This is why it is multiple metrics should always be considered as although this model would be reasonably accurate, it is not actually fitting the data.

The final model has an accuracy of 98.5%. This is a considerable improvement over the baseline predictor. The mistakes (visible on the graph) are also split between both classes. This means the model is more reliable and informative then the baseline as the baseline would always be right for the common class and wrong for the other class.

The mistakes are most likely due to noise and trying to add new features or make any large changes would result in overfitting.

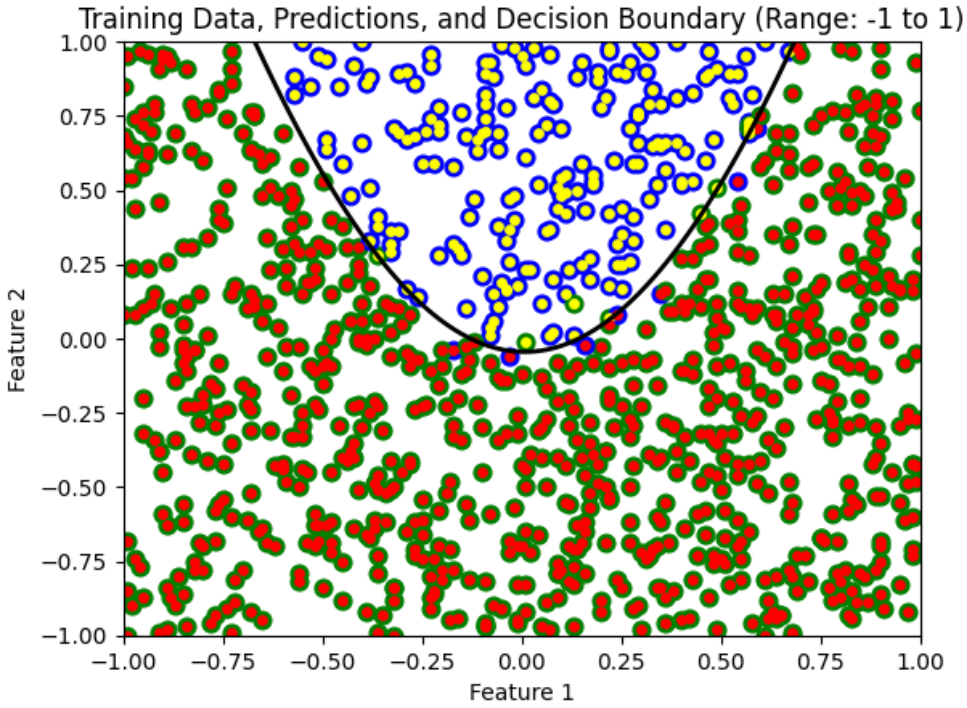
(iv) For a bonus try to plot the classifier decision boundary (nb: it can be derived from the model parameters but it's not a straight line anymore and you'll need to solve a quadratic equation).

With the new features, we must solve an equation with the 5 parameters from our model:

$$\theta_0 + \theta_1 X1\, \theta_2 X2 + \theta_3 X1^2 + \theta_4 X2^2$$

This takes the form of a quadratic equation found through the following code snippet:

```
def decision_boundary(X1, X2, model):
    b0 = model.intercept_[0]
    b1, b2, b3, b4 = model.coef_[0]
    return b0 + b1*X1 + b2*X2 + b3*X1**2 + b4*X2**2
```



Appendix

*Jupytr notebook was used *

```
#load in the data
import pandas as pd
df = pd.read_csv("week2.csv")
```

```
import numpy as np

#Store data in X and y
X1=df.iloc[:,0]
X2=df.iloc[:,1]
X=np.column_stack((X1,X2))

plus = 0
minus = 0
y=df.iloc[:,2]
for i in range(len(y)):
    if(y[i]==-1):
        minus = minus + 1
    if(y[i]==+1):
        plus = plus + 1

print(plus)
print(minus)

baseline = (796/(203+796))*100
print(baseline)
```

```
#A (i) - Visualise the data on a 2D plot
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

# Define marker types based on the target values (+1 or -1)
markers = {1: '+', -1: 'o'}
colors = {1: 'green', -1: 'blue'}

legend_handles = []
legend_labels = []

for i in range(len(X)-1):
    marker = markers[y.iloc[i]]
    color = colors[y.iloc[i]]
    marker_size = 40 if color == 'green' else 20
    ax.scatter(X[i, 0], X[i, 1], marker=marker,s=marker_size, color=color)
```

```
# Customize the plot
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_title('Feature Map')

legend_handles.append(ax.scatter([], [], marker='+', color='green', s=40))
legend_labels.append('Target: +1')

legend_handles.append(ax.scatter([], [], marker='o', color='blue', s=20))
legend_labels.append('Target: -1')


leg = ax.legend(legend_handles, legend_labels, loc='lower right')

# Show the plot
plt.show()
```

```
#a-(ii)
from sklearn.linear_model import LogisticRegression

# Initialize and train the logistic regression model
model = LogisticRegression(penalty='none', solver='lbfgs')
model.fit(X, y)

# Getting the coefficients (parameters) of the trained model
coef_feature1, coef_feature2 = model.coef_.flatten()
intercept = model.intercept_[0]
print('intercept = %f, slopes = (%f, %f)' % (model.intercept_,
*model.coef_[0]))


print("Logistic Regression Model:")
print(f"Prediction = 1 / (1 + exp(-({coef_feature1:.2f}*Feature1 +
{coef_feature2:.2f}*Feature2 + {intercept:.2f})))")
print("\nCoefficients (Parameters):")
print(f"Feature 1 Coefficient: {coef_feature1:.2f}")
print(f"Feature 2 Coefficient: {coef_feature2:.2f}")
print(f"Intercept: {intercept:.2f}")
```

```
rom sklearn.metrics import accuracy_score
#make the predictions and check accuracy


predictions = model.predict(X)


accuracy = accuracy_score(y, predictions)
print('accuracy = %f' % (accuracy))
```

```
# set up decision boundary function
m0, m1, c = model.coef_[0][0], model.coef_[0][1], model.intercept_
boundary_f = lambda x: -((m0 * x) + c) / m1
```

```
def plot(X,y,predictions,boundary_f):
 fig, ax = plt.subplots()

 # Define marker types based on the target values (+1 or -1)
 markers = {1: 'o', -1: 'o'}
 colors = {1: 'green', -1: 'blue'}

 # Plot the training data points
 for i in range(len(X)):
    marker = markers[y.iloc[i]]
    color = colors[y.iloc[i]]
    ax.scatter(X[i, 0], X[i, 1], marker=marker, s=60, color=color, )

 # Plot the decision boundary
 x_values = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
 y_values = boundary_f(x_values)
 ax.plot(x_values, y_values, color='black', linestyle='--', label='Decision
Boundary')

 # Plot the predictions
 for i in range(len(X)):
    marker = markers[predictions[i]]
    color = 'red' if predictions[i] == 1 else 'yellow'
    ax.scatter(X[i, 0], X[i, 1], marker=marker, s=15, color=color,)

 # Customize the plot
 ax.set_xlabel('Feature 1')
 ax.set_ylabel('Feature 2')
 ax.set_title('Training Data vs Predictions')
 ax.legend()
```

```
plot(X,y,predictions,boundary_f)
```

```
from sklearn.svm import LinearSVC
def svc(X,y,c):
 SVCmodel = LinearSVC(C=c).fit(X, y)

 predictions = SVCmodel.predict(X)
 accuracy = accuracy_score(y, predictions)
 print("Paramater ")
 print('accuracy = %f' % (accuracy))

 coef_feature1, coef_feature2 = SVCmodel.coef_.flatten()
```

```
 intercept = SVCmodel.intercept_[0]
 m0, m1, c = SVCmodel.coef_[0][0], SVCmodel.coef_[0][1], SVCmodel.intercept_
 boundary_f = lambda x: -((m0 * x) + c) / m1

 print(f"Feature 1 Coefficient: {coef_feature1:.2f}")
 print(f"Feature 2 Coefficient: {coef_feature2:.2f}")
 print(f"Intercept: {intercept:.2f}\n")


 plot(X,y,predictions,boundary_f)
```

```
svc(X,y,0.001)
#svc(X,y,1)
#svc(X,y,10)
#svc(X,y,1000)
```

```
X1=df.iloc[:,0]
X2=df.iloc[:,1]
X3=X1**2
X4=X2**2
Xe = np.column_stack((X1, X2, X3, X4))


ye=df.iloc[:,2]

# Initialize and train the logistic regression model
model2 = LogisticRegression(penalty='none', solver='lbfgs')
model2.fit(Xe, ye)

coef_feature1, coef_feature2,coef_feature3, coef_feature4 =
model2.coef_.flatten()
intercept = model2.intercept_[0]




print("\nCoefficients (Parameters):")
print(f"Feature 1 Coefficient: {coef_feature1:.2f}")
print(f"Feature 2 Coefficient: {coef_feature2:.2f}")
print(f"Feature 3 Coefficient: {coef_feature3:.2f}")
print(f"Feature 4 Coefficient: {coef_feature4:.2f}")
print(f"Intercept: {intercept:.2f}")

predictions2 = model2.predict(Xe)
accuracy2 = accuracy_score(y, predictions2)
print('accuracy = %f' % (accuracy2))
```

```
fig, ax = plt.subplots()

# Define marker types based on the target values (+1 or -1)
markers = {1: 'o', -1: 'o'}
colors = {1: 'green', -1: 'blue'}

for i in range(len(X)):
    marker = markers[y.iloc[i]]
    color = colors[y.iloc[i]]
    ax.scatter(X[i, 0], X[i, 1], marker=marker, s=60, color=color, )



markersp = {1: '+', -1: '+'}
for i in range(len(X)):
    marker = markers[predictions2[i]]
    color = 'red' if predictions2[i] == 1 else 'yellow'
    ax.scatter(X[i, 0], X[i, 1], marker=marker, s=15, color=color,)




# Customize the plot
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_title('Training Data vs Predictions')
ax.legend()

# Show the plot
plt.show()
```

```
# Define a function to calculate the decision boundary
def decision_boundary(X1, X2, model):
    b0 = model.intercept_[0]
    b1, b2, b3, b4 = model.coef_[0]
    return b0 + b1*X1 + b2*X2 + b3*X1**2 + b4*X2**2

# Create a meshgrid for the feature space within the range of -1 to 1
xx1, xx2 = np.meshgrid(np.linspace(-1, 1, 400), np.linspace(-1, 1, 400))
# Calculate the decision boundary values for the meshgrid points
Z = decision_boundary(xx1.ravel(), xx2.ravel(), model2)
Z = Z.reshape(xx1.shape)

# Plot the decision boundary, training data, and predictions
fig, ax = plt.subplots()

# Define marker types based on the target values (+1 or -1)
markers = {1: 'o', -1: 'o'}
```

```
colors = {1: 'green', -1: 'blue'}

for i in range(len(X)):
    marker = markers[y.iloc[i]]
    color = colors[y.iloc[i]]
    ax.scatter(X[i, 0], X[i, 1], marker=marker, s=60, color=color, )



markersp = {1: '+', -1: '+'}
for i in range(len(X)):
    marker = markers[predictions2[i]]
    color = 'red' if predictions2[i] == 1 else 'yellow'
    ax.scatter(X[i, 0], X[i, 1], marker=marker, s=15, color=color,)

# Plot decision boundary
ax.contour(xx1, xx2, Z, levels=[0], linewidths=2, colors='black')

# Customize the plot
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_title('Training Data, Predictions, and Decision Boundary (Range: -1 to
1)')
plt.xlim([-1, 1])
plt.ylim([-1, 1])
plt.show()
```