Machine Learning Assignment 2

Thomas Kelly - 16323455

October 15, 2020

The first line of my data set is identified with a first line: id:16–32–16

1 Part (a)

1.1 (i) Visualise the data on a scatter plot

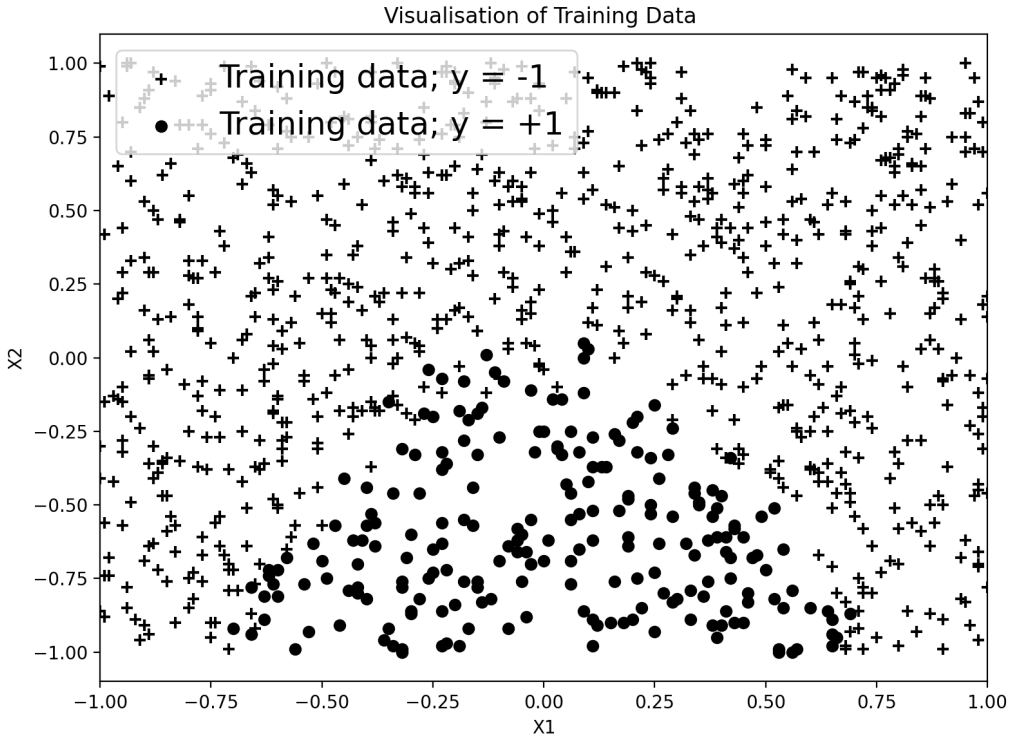

Figure 1: Visualisation of Training Data

The data was firstly read and formatted as a data frame with the help of the pandas library. Utilising the .groupby function, the input data for X1, X2 was then grouped based on the corresponding value of the y column. The resulting grouping shows training data where y = +1 with '+' points and where y = -1 represented by dots.

1.2 (a) (ii) Train a logistic regression classifier, reporting the parameter values of the trained model

The logistic model was trained using sklearn, reporting the following parameter values:

- $\theta_0 = -2.159$
- $\theta_1 = 0.2269$
- $\theta_2 = -3.288$

These values then define the expression for $y = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2$, which is later used to classify new data.

- $y = -2.159 + 0.2269x_1 - 3.88x_2$

1.3 (a) (iii) Use the trained classifier to predict the target values in the training data and show decision boundary

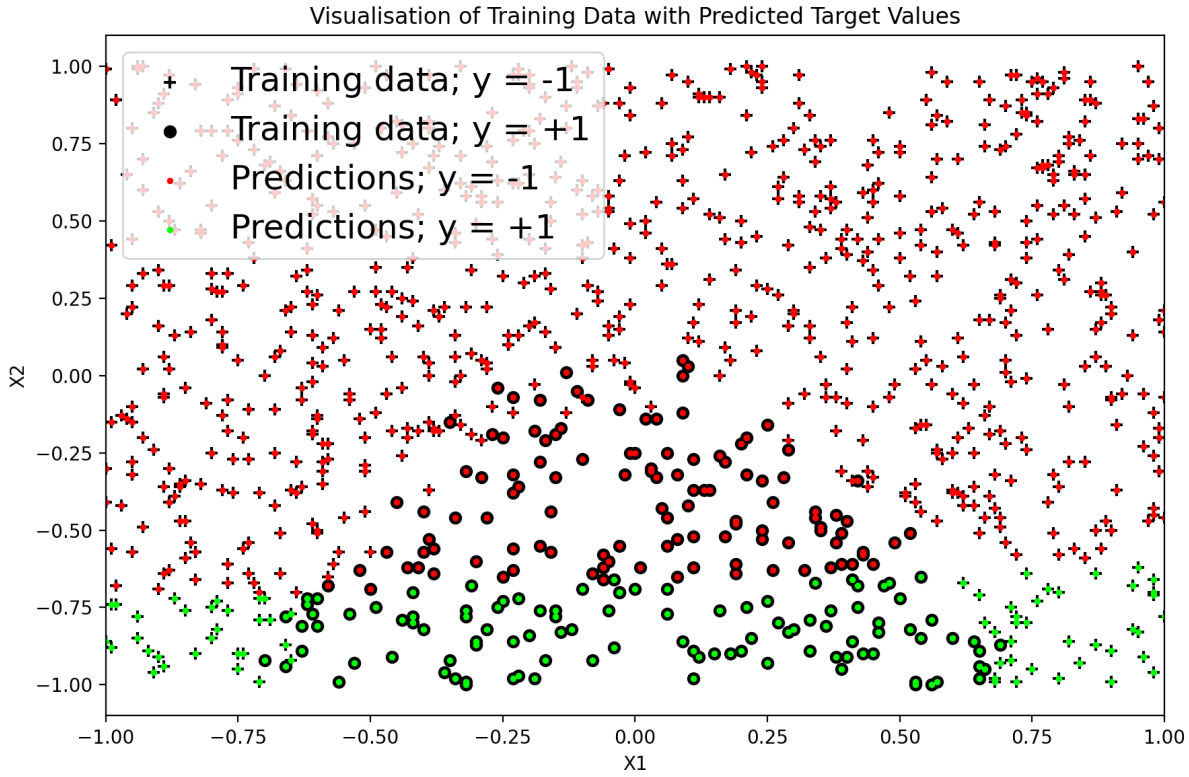

Figure 2: Visualisation of data with predicted target values

The trained model for logistic regression is used to predict the classes (y) of input data. As can be seen in Figure 2, the model does not correctly classify the data. The trained model predicts the two respective classes in red and green above, which evidently mismatches the shape of the training data which it is superimposed upon.

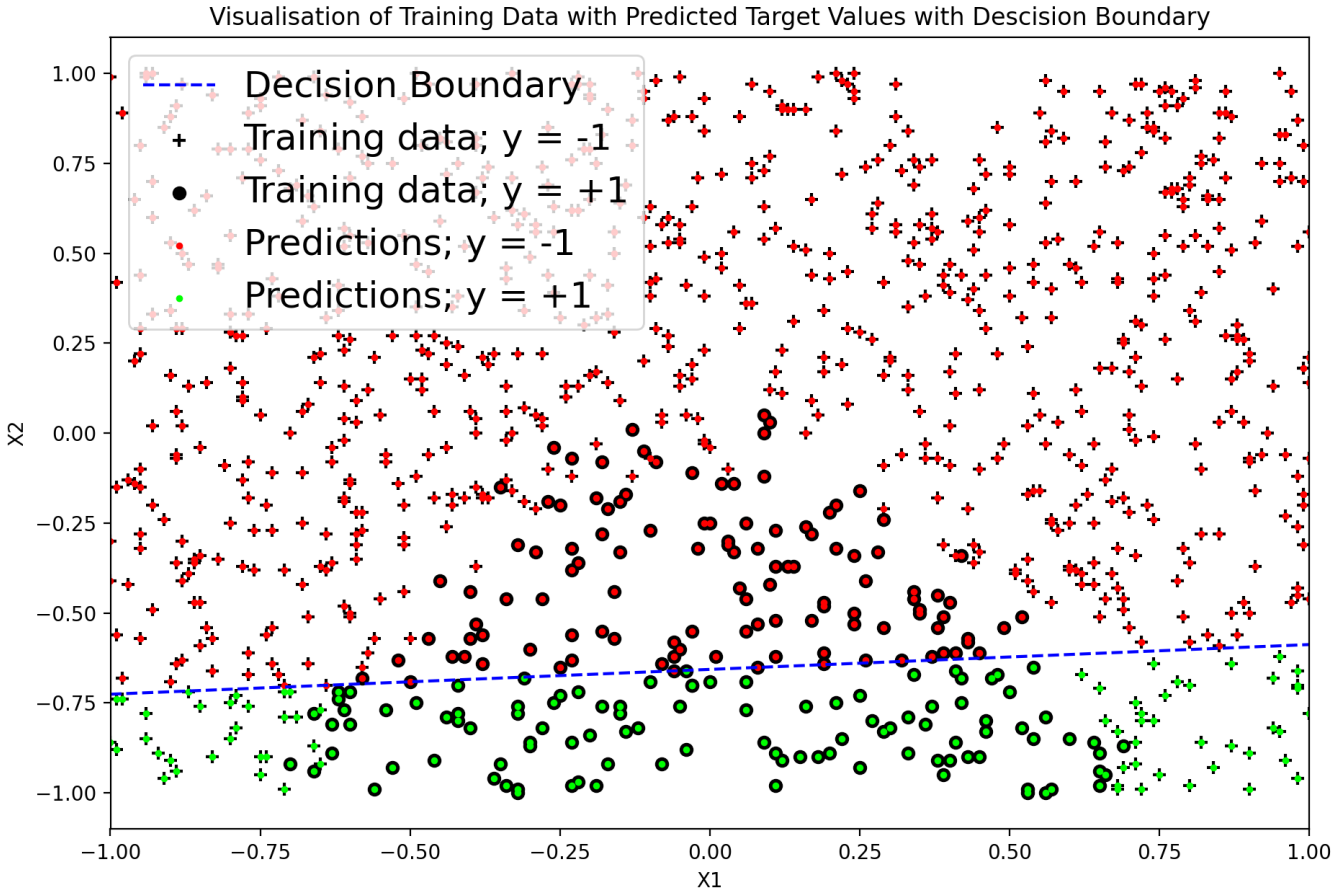

Figure 3: Visualisation of data with predicted target values, along with decision boundary

A decision boundary was fit to the plot - as described in lectures, the equation for this plot was derived as follows;

- $\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$ ...where y can be thought of as being neither class

Using the figures reported by the model for each theta, this leaves us with unknowns for x_1 and x_2. Re-organising the above expression for x_2 :

- $x_2 = \frac{(-\theta_1 x_1 - \theta_0)}{\theta_2}$

- By subbing in $x_1 = 0$ and similarly for $x_2 = 0$, we can discover two points on this line.
- Two points: (0, -θ_0/θ_2), $(-\theta_0/\theta_1, 0)$
- Using these points, a slope can be calculated $= -\frac{\theta_0\theta_1}{\theta_0\theta_2} = 0.069$
- ...along with a y-intercept $= -\frac{\theta_0}{\theta_2} = -0.6566$
- The line equation can then be written; $y = 0.069x - 0.6566$, which defines the decision boundary.

1.4 (a) (iv) Comment of how predictions and training data compare

As is evident, the classifier seems to classify the data incorrectly in many cases. This is the case because the two classes of training data (where y = -1 y = +1), are not linearly separable and we do not have enough features to correctly distinguish the data. It is clear in this case that with more features, we might be able to attain a more accurate model - and from examining the graph a quadratic model might be more suitable.

2 Part (b)

2.1 (b) (i) Train SVM classifiers for different values of C, reporting model parameters

The LinearSVC function was utilised to train the model. Values of $C = [0.01, 0.1, 1, 100, 1000]$ were used and associated parameters are displayed below

C	0.01	0.1	1	100	1000
Θ_0	-0.5757	-0.7356	-0.77	-0.7569	-0.8567
Θ_1	0.0326	0.0423	0.0436	0.04969	0.2122
Θ_2	-0.7811	-1.167	-1.257	-1.319	-1.995

Figure 4: Table comparing the parameters of models using different C values

2.2 (b) (ii) Use Trained Classifiers to predict target values, plot together with training data

Scatter plots were generated for training data and the target predictions. The decision boundary for each classifier is also included.

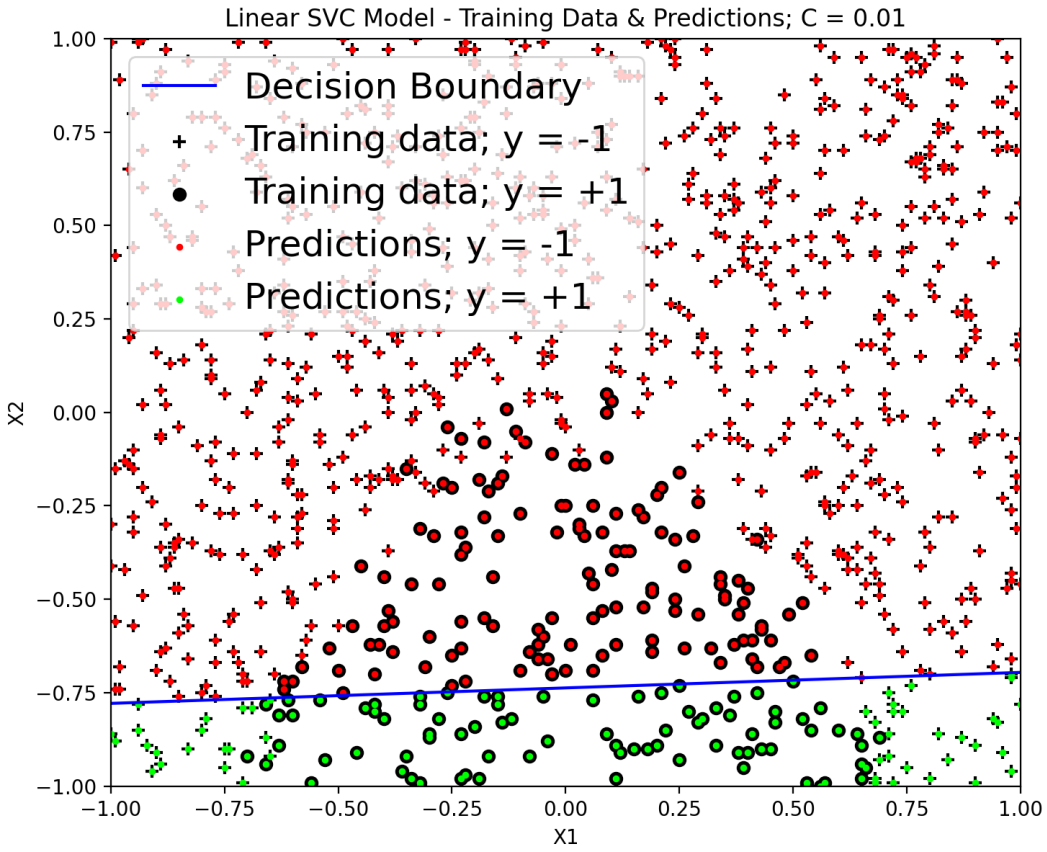

Figure 5: SVM classifier for C = 0.01

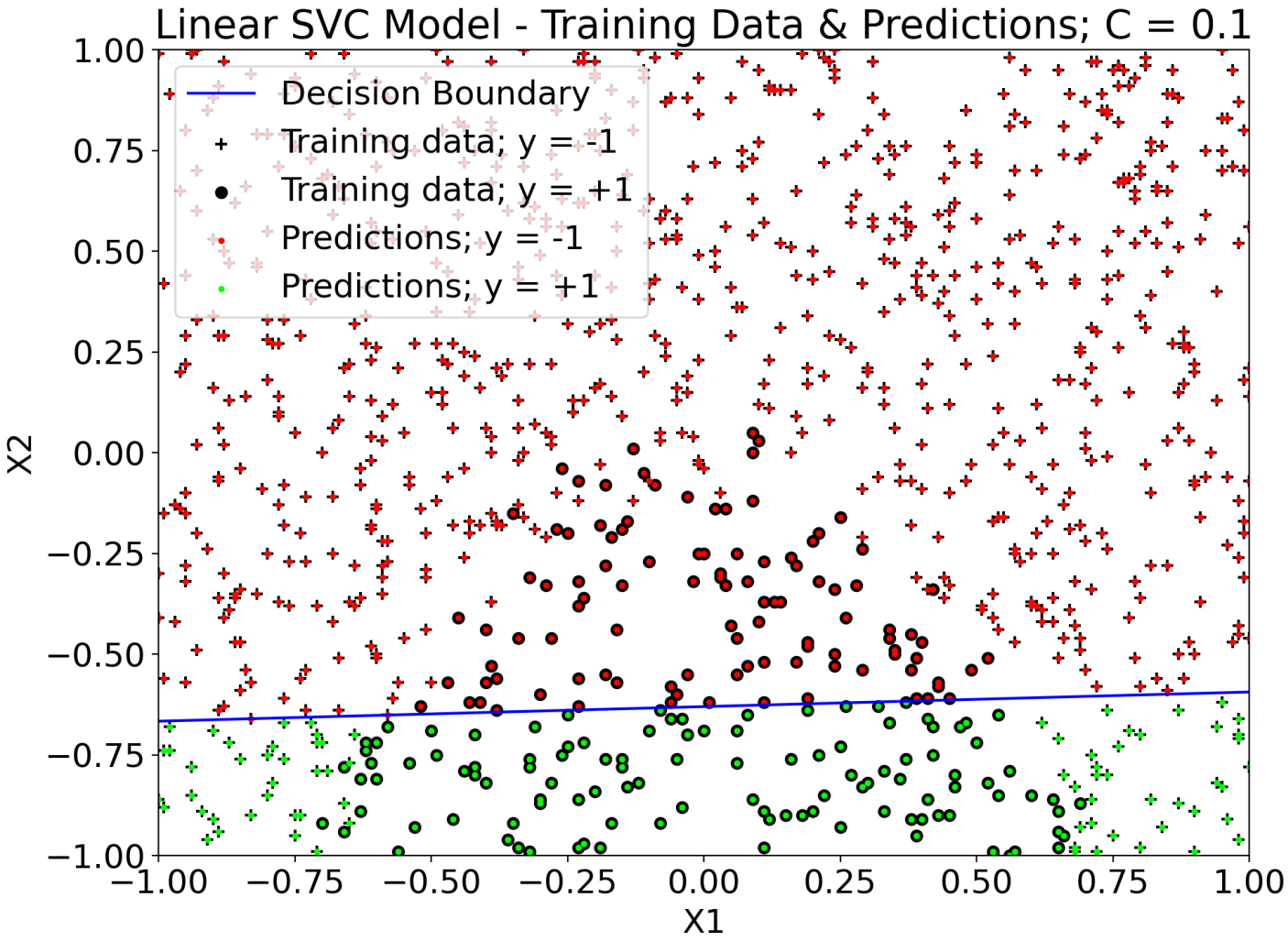

Figure 6: SVM classifier for C = 0.1

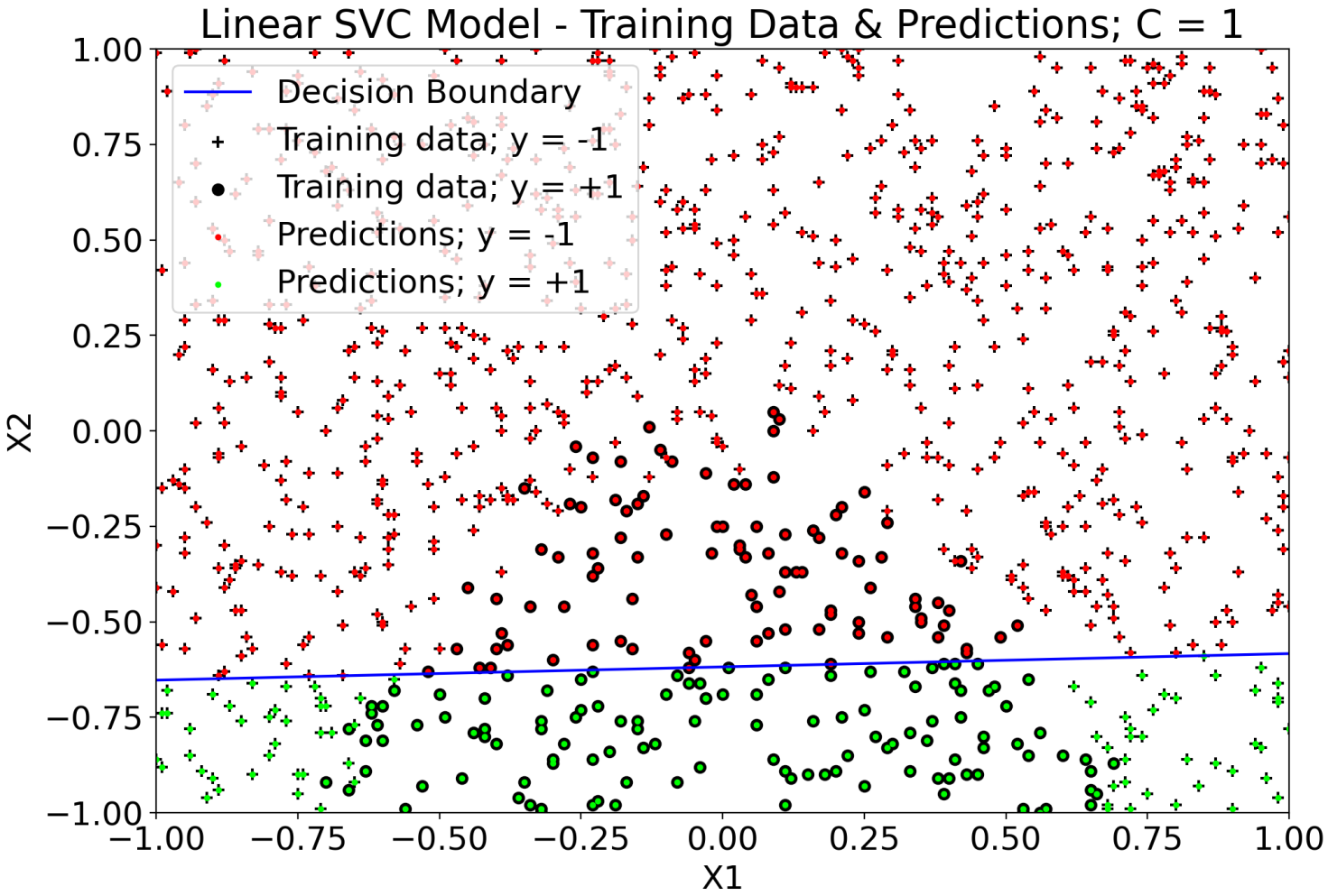

Figure 7: SVM classifier for C = 1

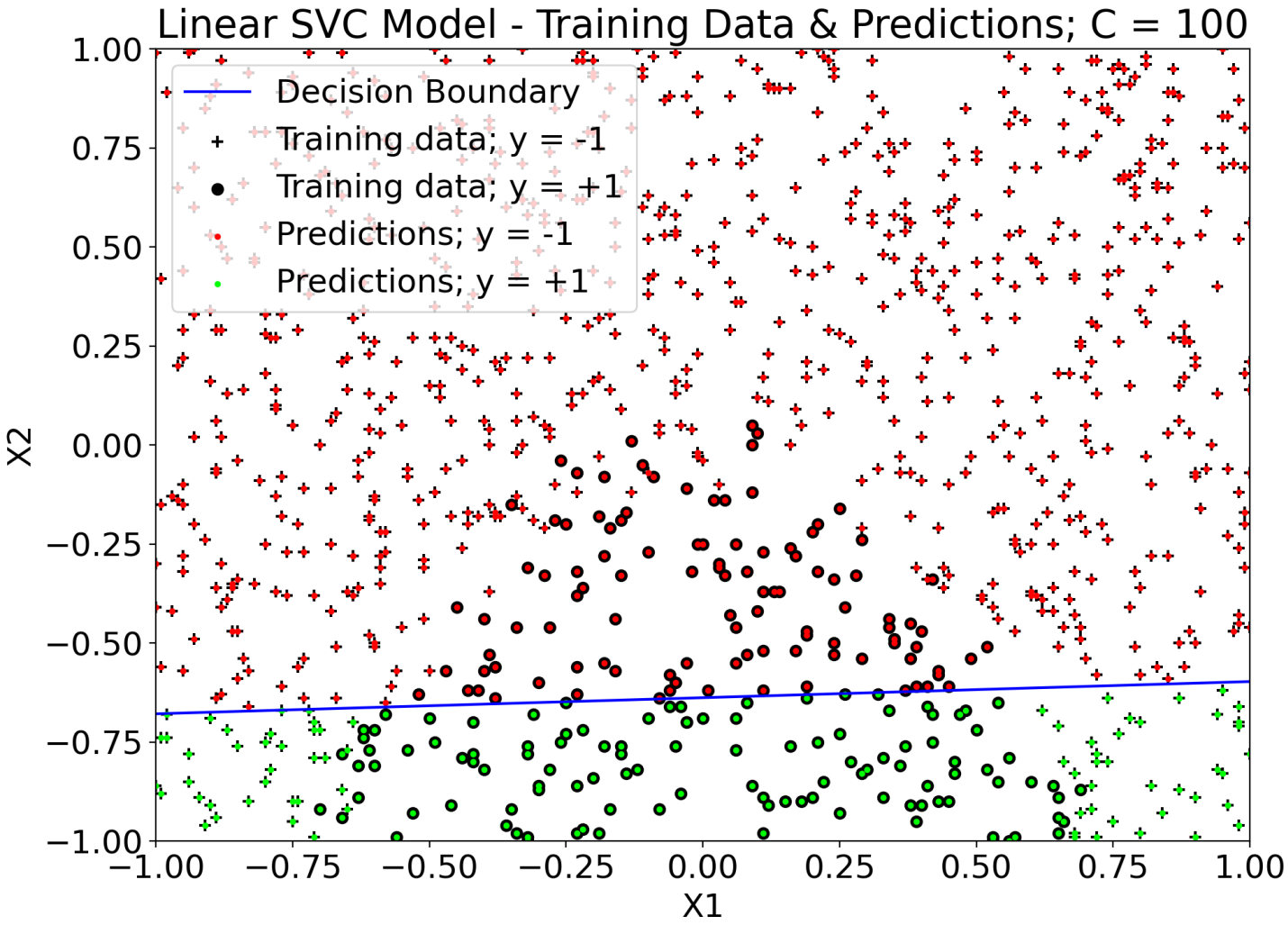

Figure 8: SVM classifier for C = 100

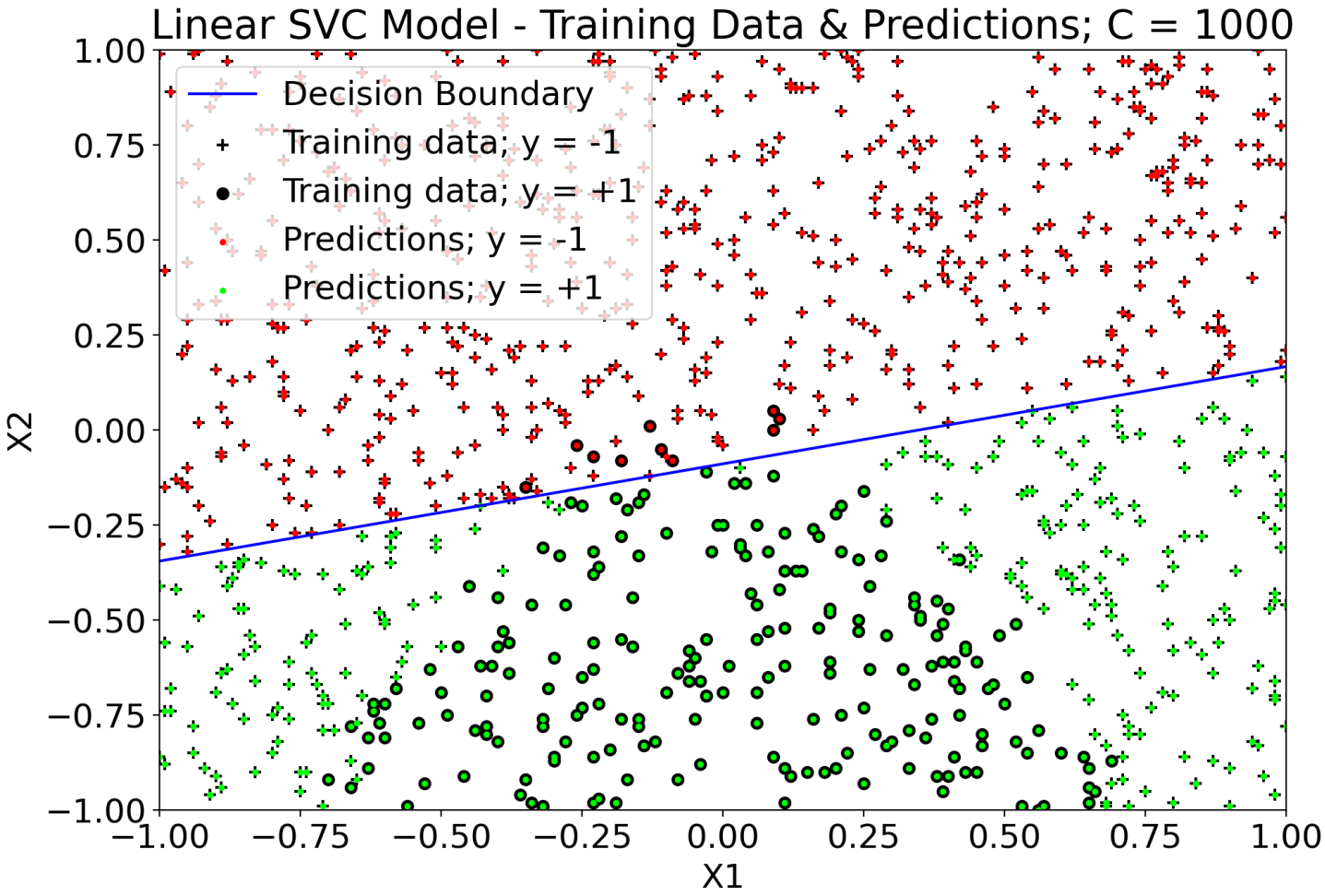

Figure 9: SVM classifier for C = 1000

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

Examining the parameter values for different values of C, a general decrease in each parameter $\theta_0, \theta_1, and \theta_2$ is observed as C increases. The loss function for LinearSVC is as follows:

- $\mathrm{J}(\theta) = \frac{1}{m}\Sigma max(0, 1 - y^i \theta^T x^i) + \frac{\theta^T \theta}{C}$

For smaller values of C, the penalty ($\frac{\theta^T \theta}{C}$) becomes larger and plays a more important role in the loss function. The loss function will attempt to minimise it by making the parameters θ_j closer to 0. As

the value for C increases, this penalty becomes less important and would explain why we see the figures trending away from 0 as C increases.

C is a basically weighting parameter that influences how much we want to avoid misclassifying data. For higher values of C, the model will be more conservative about misclassifying values. Similarly with a low C, the model will be more willing to misclassify some data. When data is linearly separable, a high C value influences how strict the model is about how many instances of the classes in question are separated correctly.
This is visible particularly in the case where C = 1000 (Figure 9), where the model has decided it would rather misclassify more crosses as green (more y = -1 as y = +1).

3 Part (c)

3.1 (c) (i) Create additional features by adding the square of each feature.

Taking initial features X1 and X2 and squaring them both respectively produce new features X3 and X4. The model was trained using these 4 features, reporting the following parameters while C = 1:

- θ_0 = -0.184
- θ_1 = -0.0175
- θ_2 = -3.573
- θ_3 = -6.204
- θ_4 = -0.259

3.2 (c) (ii) Use the trained classifier to predict new target values.

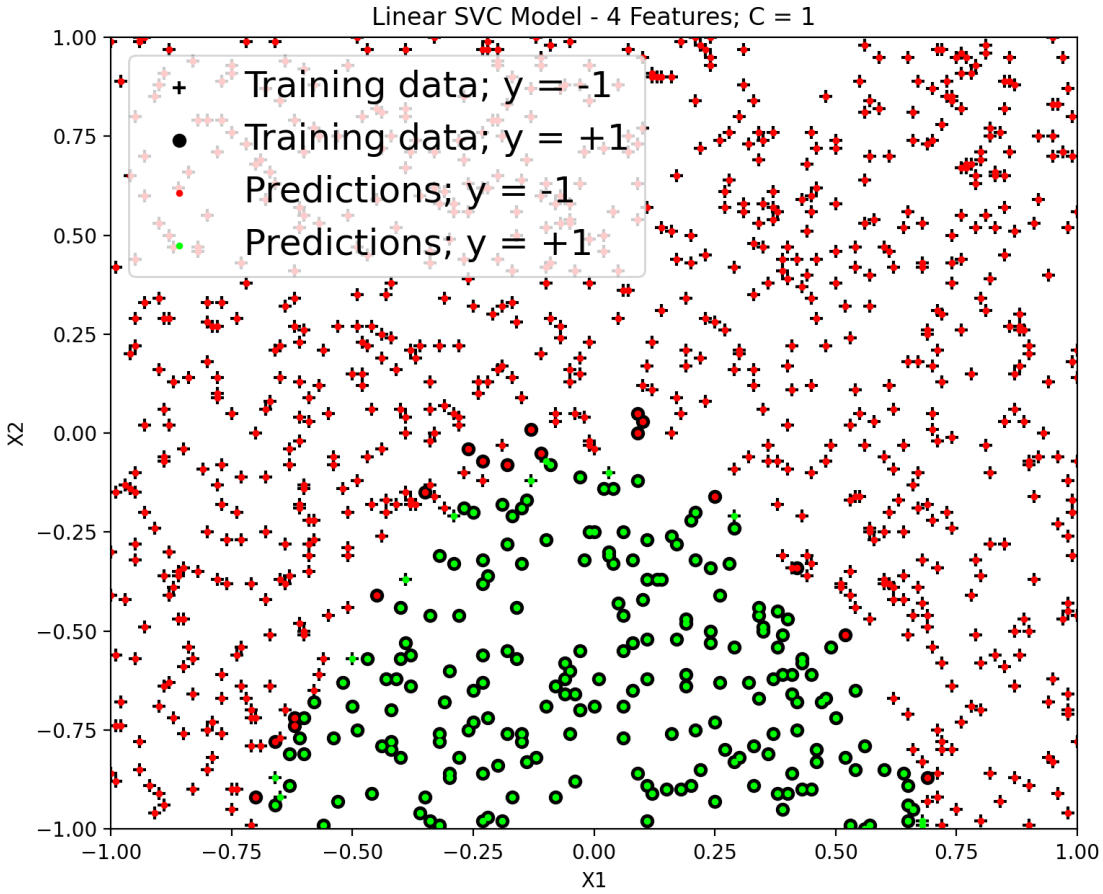

Figure 10: 4 feature logistic regression model predictions

Trained with additional features, the model is able to more correctly predict the target values. The addition of the extra features allows the data to be linearly separated in a higher dimension and the model can trained to differentiate the classes more accurately.

3.3 (c) (iii) Compare the performance of this classifier with that of a baseline classifier.

A baseline model defined as specified predicts the most common class from the training data. Choosing error rate as a metric to compare the classifiers, the error rate of such a baseline model can be calculated for the given data as follows:

A model that predicts the most common output of a data-set, simply returns the most common 'y' value of that data, this will be the mode of the data.
If we count the values it classifies incorrectly in the training data and divide that by the total values, we can calculate an error rate.

- $(total\_values - mode\_count)/total\_values = \frac{998-781}{998} = 0.217 = 21.7\%$ error
- Utilising sklearn's metrics tools to obtain a value for accuracy, $error\_rate = 1 - accuracy = 0.031 = 3.1\%$ error

Comparing these two values, we can safely surmise that the trained SVM model behaves with a much higher accuracy than the baseline model.

3.4 (c) (iv) Plot the decision boundary of this classifier

To plot the decision boundary, we allow $y = \theta^T x = 0$. This leads to the following expression:

- $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 = 0$

Realising that our equation here will be quadratic, I decided to exclude x_4 so the problem became quadratic.

Because $x_3 = x_1^2$ we can rewrite the expression:

- $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 = 0$
- $x_2 = -\frac{\theta_3}{\theta_2} x_1^2 - \frac{\theta_1}{\theta_2} x_1 - \frac{\theta_0}{\theta_2}$ ... re-arranging

Passing in the calculated parameter values here a quadratic expression is obtained and plot over the rest of the data.

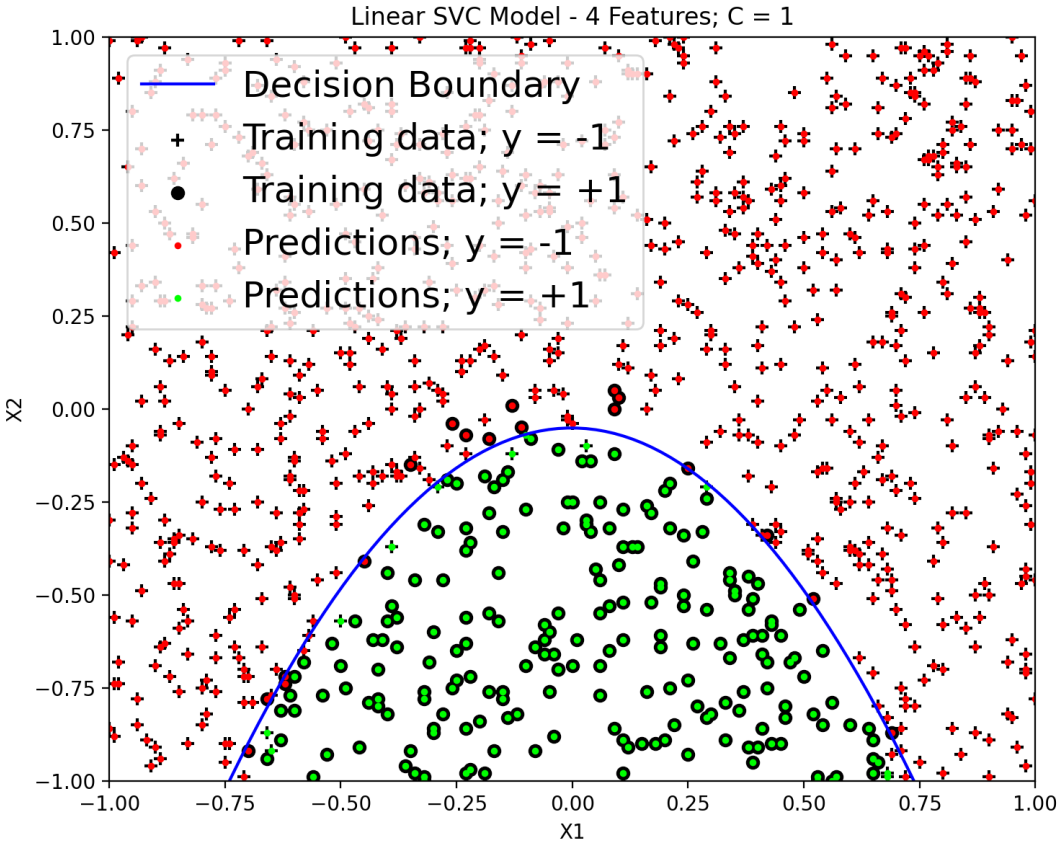

Figure 11: Decision boundary on 4 feature model

4 Code

4.1 Part A - *w2_part_a.py* - Logistic Regression

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


df = pd.read_csv("week2.csv")
```

```
X1 = df.iloc[:, 0]
X2 = df.iloc[:, 1]
X = np.column_stack((X1, X2))
y = df.iloc[:, 2]

# (a) (i) Group input data based on y = -1 and y = +1
dataframe = pd.DataFrame({'column1': X1, 'column2': X2, 'column3': y})
minusDF = []
postivieDF = []

for grouped_y, grouping_x in dataframe.groupby(["column3"]):
    if grouped_y == -1:
        minusDF = grouping_x  # X rows that have y = -1
    else:
        positiveDF = grouping_x  #  X rows that have y = +1

# (a) (ii)
logRegressionModel = LogisticRegression()
logRegressionModel.fit(X, y)


# (a) (iii)
predictions = logRegressionModel.predict(X)

prediction_dataframe = pd.DataFrame(
    {'column1': X1, 'column2': X2, 'column3': predictions})
minus_pred_DF = []
positive_pred_DF = []

for grouped_y, grouping_x in prediction_dataframe.groupby(["column3"]):
    if grouped_y == -1:
        minus_pred_DF = grouping_x  # X rows that have y = -1
    else:
        positive_pred_DF = grouping_x  #  X rows that have y = +1

slope = -(logRegressionModel.coef_[0][0]*logRegressionModel.intercept_)/(
    logRegressionModel.intercept_*logRegressionModel.coef_[0][1])
y_intercept = -logRegressionModel.intercept_ / logRegressionModel.coef_[0][1]
decision_boundary_X = np.linspace(-1, 1, 2)
decision_boundary_Y = slope*decision_boundary_X + y_intercept
```

4.2 Part B - *w2_part_c.py* - SVM Classifier

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

```
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC

df = pd.read_csv("week2.csv")

X1 = df.iloc[:, 0]
X2 = df.iloc[:, 1]
X = np.column_stack((X1, X2))
y = df.iloc[:, 2]

dataframe = pd.DataFrame({'column1': X1, 'column2': X2, 'column3': y})
minusDF = [0]
postivieDF = [0]

for grouped_y, grouping_x in dataframe.groupby(["column3"]):
    if grouped_y == -1:
        minusDF = grouping_x  # X rows that have y = -1
    else:
        positiveDF = grouping_x  #  X rows that have y = +1


for i, C in enumerate([0.01, 0.1, 1, 100, 1000]):
    svmModel = LinearSVC(C=C)
    svmModel.fit(X, y)

    predictions = svmModel.predict(X)
    prediction_dataframe = pd.DataFrame(
        {'column1': X1, 'column2': X2, 'column3': predictions})

    minus_pred_DF = []
    positive_pred_DF = []

    for grouped_y, grouping_x in prediction_dataframe.groupby(["column3"]):
        if grouped_y == -1:
            minus_pred_DF = grouping_x  # X rows that have y = -1
        else:
            positive_pred_DF = grouping_x  #  X rows that have y = +1

    slope = -(svmModel.intercept_ *
              svmModel.coef_[0][0]) / (svmModel.intercept_ * svmModel.coef_[0][1])
    y_intercept = -svmModel.intercept_ / svmModel.coef_[0][1]

    decision_boundary_X = np.linspace(-1, 1, 2)
    decision_boundary_Y = slope*decision_boundary_X + y_intercept
```

4.3 Part C - $w2\_part\_c$ - Adding Additional features

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from scipy import stats
from sklearn.metrics import accuracy_score


df = pd.read_csv("week2.csv")

X1 = df.iloc[:, 0]
X2 = df.iloc[:, 1]
y = df.iloc[:, 2]

# (c) (i) Create new features
X3 = X1 * X1
X4 = X2 * X2

X = np.column_stack((X1, X2, X3, X4))
dataframe = pd.DataFrame(
    {'column1': X1, 'column2': X2, 'column4': X3, 'column5': X4, 'column3': y})
minusDF = []
postivieDF = []

for grouped_y, grouping_x in dataframe.groupby(["column3"]):
    if grouped_y == -1:
        minusDF = grouping_x  # X rows that have y = -1
    else:
        positiveDF = grouping_x  #  X rows that have y = +1

for i, C in enumerate([1]):
    # (c) (ii) Train SVM model
    svmModel = LinearSVC(C=C)
    svmModel.fit(X, y)

    predictions = svmModel.predict(X)
    prediction_dataframe = pd.DataFrame(
        {'column1': X1, 'column2': X2, 'column4': X3, 'column5': X4, 'column3': predictions}

    minus_pred_DF = []
    positive_pred_DF = []

    for grouped_y, grouping_x in prediction_dataframe.groupby(["column3"]):
```

```
    if grouped_y == -1:
        minus_pred_DF = grouping_x  # X rows that have y = -1
    else:
        positive_pred_DF = grouping_x  #  X rows that have y = +1

# (c) (iii) Get errors for baseline and svm model
def baseline_model_error():
    mode_count = stats.mode(y).count  #  Model that predicts mode output
    total_n = len(y)
    error_rate = (total_n - mode_count) / total_n
    return error_rate
print(baseline_model_error())

svm_accuracy = accuracy_score(y, predictions)
svm_error_rate = 1 - svm_accuracy
print(svm_error_rate)

# (c) (iv) Calculate a decision boundary
decision_boundary_X = np.linspace(-1, 1, 100)
decision_b_y = (-svmModel.coef_[0][2]/svmModel.coef_[0][1])*(decision_boundary_X**2)
- (svmModel.coef_[0][0]/svmModel.coef_[0][1])
*decision_boundary_X -
(svmModel.intercept_/svmModel.coef_[0][1])
```