Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

CS7CS4 Machine Learning
Week 2 Assignment

Boris Flesch
20300025

October 22, 2020

MSc Computer Science, Intelligent Systems

1 Downloaded Dataset

id:1-1-1

2 Questions

(a) (i) Read data from CSV file (code taken from the assignment sheet) and visualise it using Matplotlib (Figure 1).

```
df = pd.read_csv("week2.csv", comment="#")
X1 = df.iloc[:,0]
X2 = df.iloc[:,1]
X = np.column_stack((X1,X2))
y = df.iloc[:,2]

X_m1 = X[np.where(y == -1)]
X_p1 = X[np.where(y == 1)]
plt.scatter(X_m1[:, 0], X_m1[:, 1], c='r', marker='+',
↪  label="y = -1")
plt.scatter(X_p1[:, 0], X_p1[:, 1], c='b', marker='+',
↪  label="y = 1")
```

Note: instead of creating X_m1 and X_p1 variables, it would have been possible to scatter all points in a single line as following:

```
plt.scatter(X1, X2, c=np.where(y == -1, 'r', 'b'),
↪  marker='+')
```

However this method does not allow to plot the label of each different color (i.e. what red crosses and blue crosses refer to).

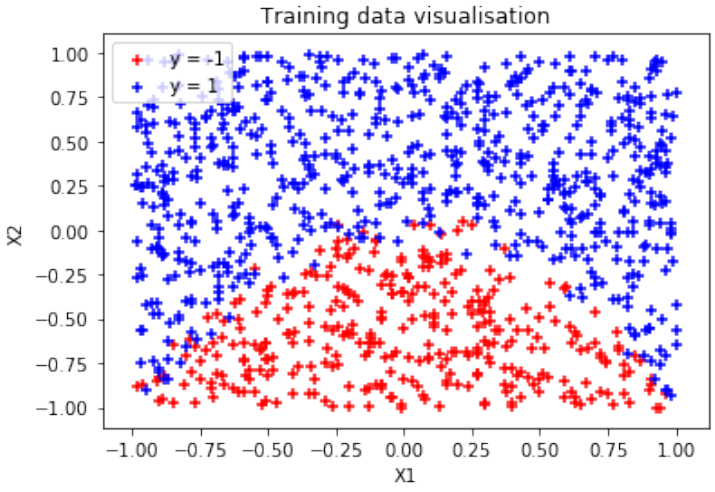

Figure 1

(ii) Train a logistic regression classifier on the data:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression().fit(X, y)
theta = [model.intercept_[0], model.coef_[0][0],
↪  model.coef_[0][1]]
```

$$\theta = \begin{bmatrix} 1.6713466864527504 \\ -0.03177179625806771 \\ 5.045678045815783 \end{bmatrix} \tag{1}$$

An issue with this model is that it is of dimension 2 and will therefore define a line to separate outputs. However, our training data is not linearly separable as seen on Figure 1.

With the parameter values obtained, another issue is the neglect of feature $X1$ due to the value of θ_1 which is close to zero. This indicates that the prediction will essentially rely on the value of feature $X2$. Considering $\theta_1 = 0$, we roughly obtain :

$$sign(\theta^T X) = +1 \tag{2}$$
$$\Leftrightarrow \quad 5.04 \cdot X2 > -1.67 \tag{3}$$
$$\Leftrightarrow \quad X2 > -0.33 \tag{4}$$

(iii) Figure 2 below shows logistic regression classifier predictions and its decision boundary over the training data. Our model is $h_\theta(x) = sign(\theta^T X)$, the decision boundary can therefore be defined by the equation $\theta^T X = 0$, so that all +1 predictions appear above this line and all -1 predictions appear below it. To plot this decision boundary on the graph with X_1 and X_2 respectively on x- and y-axis, we have to express X_2 as a function of X_1:

$$\theta^T X = 0 \tag{5}$$
$$\Leftrightarrow \quad \theta_0 + \theta_1 X_1 + \theta_2 X_2 = 0 \tag{6}$$
$$\Leftrightarrow \quad X_2 = \frac{-1}{\theta_2}(\theta_0 + \theta_1 X_1) \tag{7}$$

This can be plotted using the following code:

```
# Training data
plt.scatter(X_m1[:, 0], X_m1[:, 1], c='r', marker='+', s=500,
↪  label="Training y = -1")
plt.scatter(X_p1[:, 0], X_p1[:, 1], c='b', marker='+', s=500,
↪  label="Training y = 1")

# Predictions
X_pred_m1 = X[np.where(model.predict(X) == -1)]
X_pred_p1 = X[np.where(model.predict(X) == 1)]
plt.scatter(X_pred_m1[:, 0], X_pred_m1[:, 1], marker='o',
↪  facecolor='none', edgecolor='r', s=500, label="Prediction
↪  y = -1")
```

```
    plt.scatter(X_pred_p1[:, 0], X_pred_p1[:, 1], marker='o',
    ↪  facecolor='none', edgecolor='b', s=500, label="Prediction
    ↪  y = 1")

    # Decision boundary
    plt.plot(X1, -1/theta[2] * (theta[0] + theta[1]*X1),
    ↪  linewidth=5, c='g', label='Decision boundary')
```

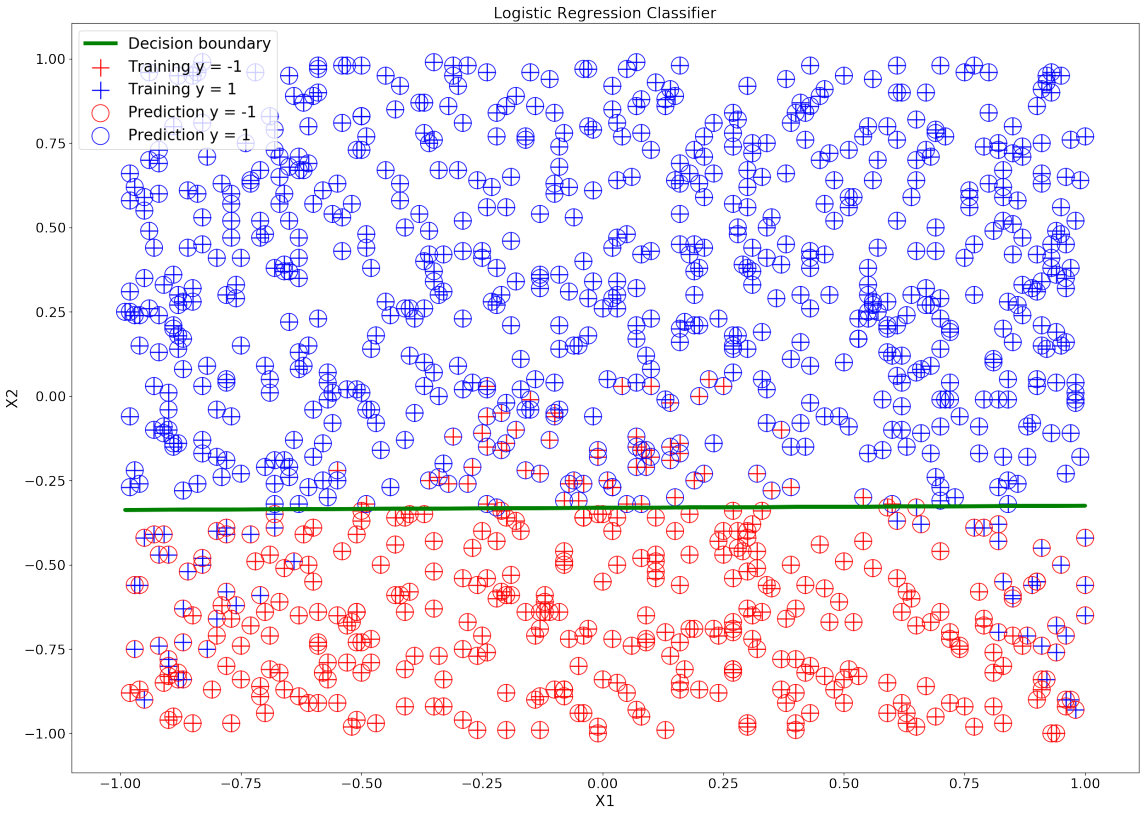

Figure 2

This representation of predictions over training data is quite interesting as it allow us to easily distinguish correct and missed predictions: basically, when a cross is not of the same color as its surrounding circle, the prediction failed. The color then allow us to determine if the missed prediction is a false positive or false negative.

(iv) The trained logistic regression classifier gives a straight decision boundary line close to -0.33 on y-axis, as expected and explained in (a)(ii). We can therefore see that the predictions obtained are very approximate. Predictions that are very distant from the decision boundary are quite accurate, unlike the ones close to the decision boundary and in the edges. The linear shape of the prediction model does not follow the quadratic behaviour of the data: this is **under-fitting**.

(b) (i) The LinearSVC function from sklearn allows us to easily train SVM classifiers given a C value as following:

```
    from sklearn.svm import LinearSVC
    Cs = [0.001, 1, 1000]
```

```
    for Ci in Cs:
        model = LinearSVC(C=Ci).fit(X, y)
        theta = [model.intercept_[0], model.coef_[0][0],
        ↪  model.coef_[0][1]]
        print("C = %.3f"%Ci, "\n theta =", theta)
```

The following parameter values have been obtained (and will be discussed in (b)(iii)):

- For $C = 0.001$ (see Figure 3):

$$\theta = \begin{bmatrix} 0.2189386198207775 \\ 0.007272583598406376 \\ 0.48180974288823664 \end{bmatrix} \tag{8}$$

- For $C = 1$ (see Figure 4):

$$\theta = \begin{bmatrix} 0.5899032183960152 \\ -0.01362192025379305 \\ 1.8196765208475205 \end{bmatrix} \tag{9}$$

- For $C = 1000$ (see Figure 5):

$$\theta = \begin{bmatrix} -0.060980250342034914 \\ -0.6821202341260354 \\ 2.5697912854521707 \end{bmatrix} \tag{10}$$

(ii) To plot the data, we can use a code similar to the one used in (a)(iii), which gives the following figures:

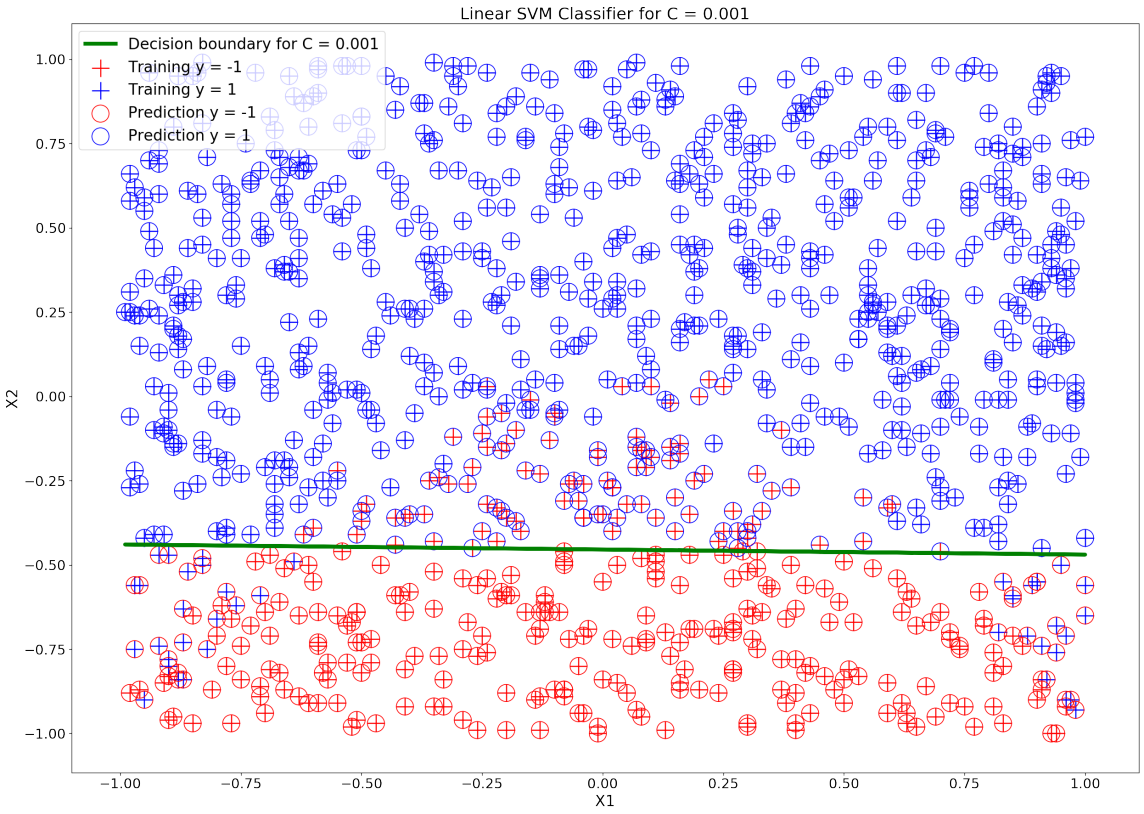

Figure 3

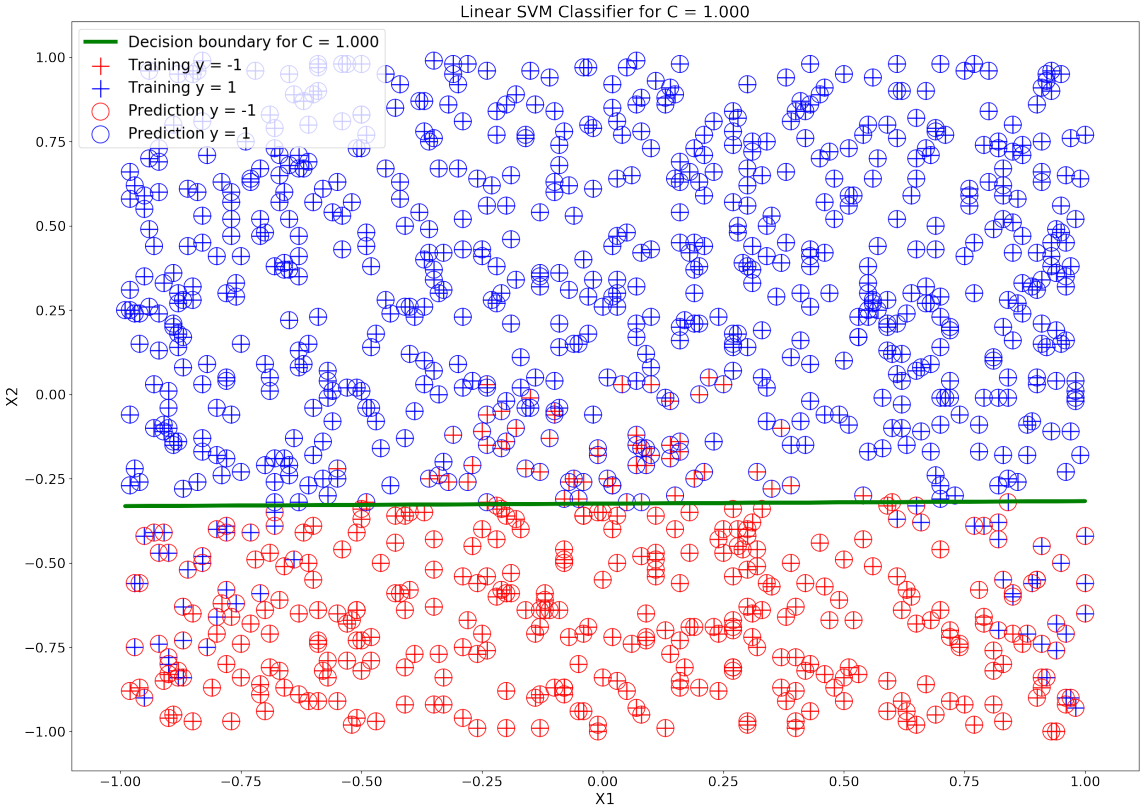

Figure 4

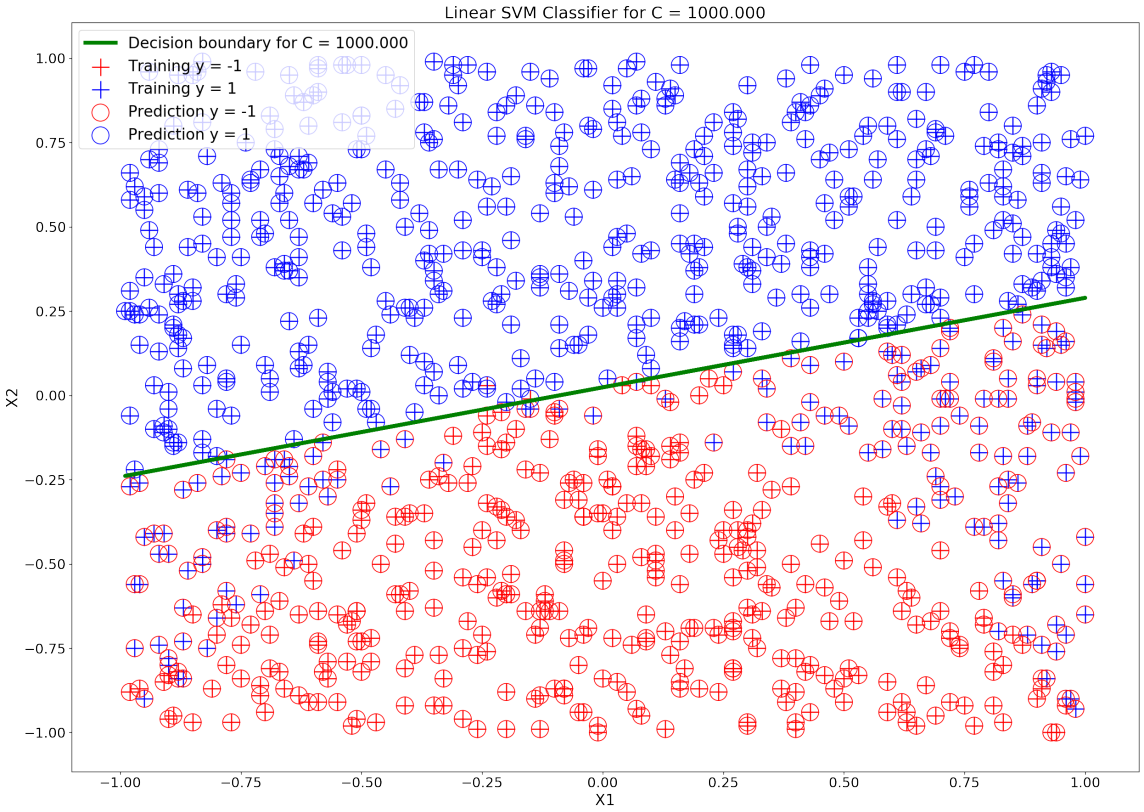

Figure 5

(iii) Changing the weighting parameter C influences the impact of the penalty on the hinge-loss function of the SVM model.

This penalty should only penalise large values of θ; however with a too small C value, θ will systematically get penalised, leading to an inaccurate model. That is why we obtain so small parameter values for $C = 0.001$ (especially for θ_3) compared to $C = 1$ and $C = 1000$.

On the contrary, a too big value (i.e. 1000) can lead to an insignificant penalty and therefore some difficulties for the cost function to converge (i.e. *ConvergenceWarning* generated by sklearn).

This explains why the trained model for $C = 1$ (i.e. not an extreme value) is the most accurate; although it remains a linear model and can not fit well the training data especially in the edges.

(c) (i) Generating additional features and training a logistic regression model can be done with the following Python code:

```
X3 = np.power(X1, 2)
X4 = np.power(X2, 2)
X = np.column_stack((X1,X2,X3,X4))

model = LogisticRegression().fit(X, y)
theta = np.array([model.intercept_[0], model.coef_[0][0],
 ↪  model.coef_[0][1], model.coef_[0][2], model.coef_[0][3]])
```

The trained logistic regression classifier gives the following parameter values:

$$\theta = \begin{bmatrix} 0.29322948 \\ -0.09209744 \\ 6.72896399 \\ 6.48226436 \\ -0.84717209 \end{bmatrix} \tag{11}$$

This 4-dimension model generated thanks to feature engineering allows us to obtain a curved prediction boundary rather than the straight lines obtained until now. In addition to that, the additional feature $X3$ (i.e. $X1^2$) balances the use of feature $X1$ in our prediction as it has been systematically neglected in our previous models. Thus all features are now nearly equally used in this new logistic regression model.

(ii) The plot showing predicted values and training data can be seen on Figure 6. Adding the square of each initial feature leads to a more reliable model that fits well the training data and therefore produces more accurate predictions.

(iii) Our training data is composed of 335 outputs -1 and 663 outputs 1. Considering a baseline predictor that always predicts the most common class (i.e. 1 in this case) is therefore a good idea as it gives an accuracy of 66.4% on the training data. Its parameter values are the following:

$$\theta = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{12}$$

The decision boundary of this baseline model is also shown on Figure 6.

To evaluate and compare these models, we can compare their respective cost function value after calculating them using Python:

```
def calcCost(X, y, theta):
    m = X.shape[0]
    Xtmp = np.insert(X, 0, 1, axis=1).T
    cost = 1/m * sum(np.log(1 + np.exp(-y * np.dot(theta,
    ↪  Xtmp)))/np.log(2))
    return cost
```

Note: this cost function value calculator automatically adds feature $X_0 = 1$ to the given X data in order to calculate the cost function value properly. That prevents from altering the original value of X in the rest of the program.

The value of the cost function for the logistic regression model and the baseline model are the following:

$$J(\theta) = 0.179947 \tag{13}$$
$$J(\theta_{baseline}) = 0.936212 \tag{14}$$

Although our baseline model does not seem to be totally inaccurate (as it predicts the most common value of our training data), we can see that $J(\theta_{baseline}) > J(\theta)$, thus confirming the better accuracy of the logistic regression model.

(iv) It is possible to use the same principle as in (a)(iii) to obtain the decision boundary equation for this logistic regression model. Starting from $\theta^T X = 0$, we have to get X_2 as a function of the other features. Furthermore features X_3 and X_4 can be substituted by X_1^2 and X_2^2 respectively. This gives us the following equation:

$$\theta^T X = 0 \tag{15}$$
$$\Leftrightarrow \quad \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3 + \theta_4 X_4 = 0 \tag{16}$$
$$\Leftrightarrow \quad X_2 = -\frac{1}{\theta_2}(\theta_0 + \theta_1 X_1 + \theta_3 X_3 + \theta_4 X_4) \tag{17}$$
$$\Leftrightarrow \quad X_2 = -\frac{1}{\theta_2}(\theta_0 + \theta_1 X_1 + \theta_3 X_1^2 + \theta_4 X_2^2) \tag{18}$$
$$\Leftrightarrow \quad X_2 = -\frac{\theta_3}{\theta_2} X_1^2 - \frac{\theta_1}{\theta_2} X_1 - \frac{\theta_0}{\theta_2} - \frac{\theta_4}{\theta_2} X_2^2 \tag{19}$$
$$\Leftrightarrow \quad X_2 = ax^2 + bx + c \tag{20}$$

With $a = -\frac{\theta_3}{\theta_2}$, $b = -\frac{\theta_1}{\theta_2}$ and $c = -\frac{\theta_0}{\theta_2} - \frac{\theta_4}{\theta_2} X_2^2$.

The following Python code has been written to plot the decision boundaries of the logistic regression model and the baseline model:

```
X1_sorted = np.sort(X1)
X2_sorted = np.sort(X2)
a = -theta[3]/theta[2]
b = -theta[1]/theta[2]
c = -theta[0]/theta[2] - theta[4] / theta[2] *
↪  np.power(X2_sorted, 2)
boundary = a * np.power(X1_sorted, 2) + b * X1_sorted + c

plt.plot(X1_sorted, boundary, linewidth=5, c='g',
↪  label="Decision boundary")
plt.axhline(-baselineTheta[0], c='y', linewidth=5,
↪  label='Baseline model')
```

Note: sorting $X1$ and $X2$ features has been required to avoid lines crossing from one side of the curve to the other on the figure.

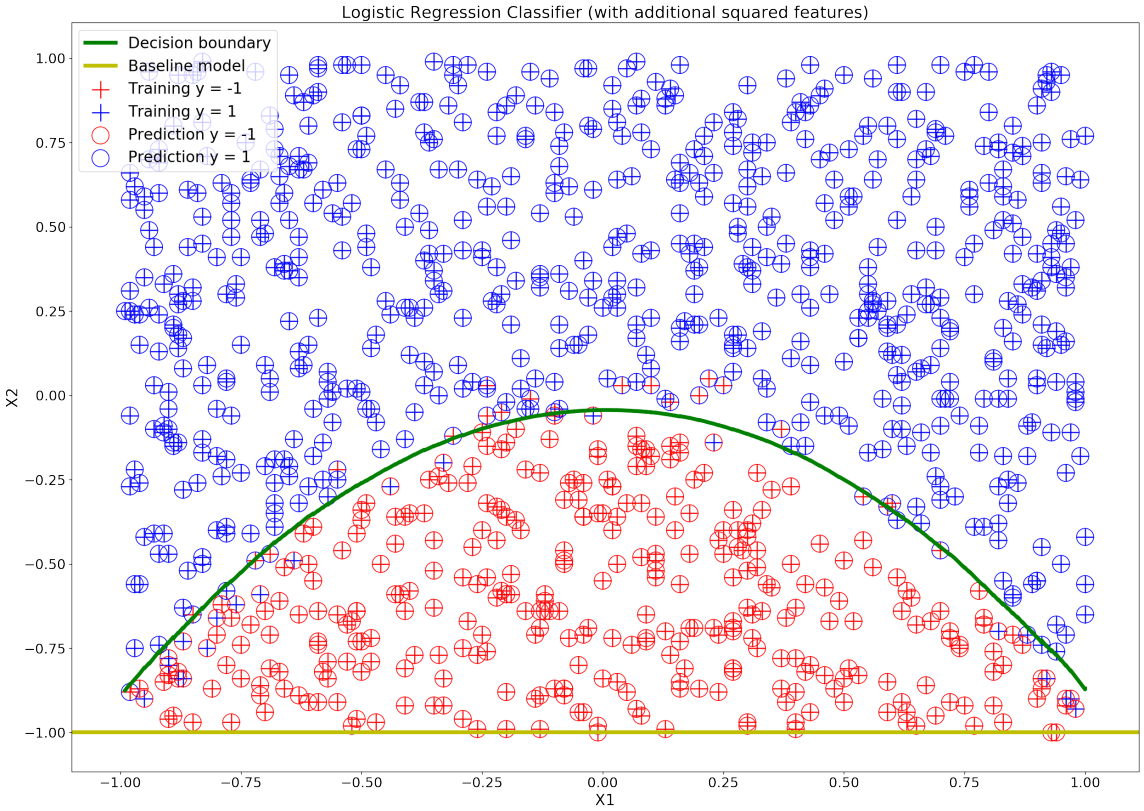

Figure 6

A Appendix

A.1 Python Code

```python
#!/usr/bin/env python
# coding: utf-8

# CS7CS4/CSU44061 Machine Learning
# Week 2 Assignment
# Boris Flesch (20300025)
#
# Downloaded dataset
# id:1-1-1

# Read data
import numpy as np
import pandas as pd
df = pd.read_csv("week2.csv", comment="#")
print(df.head())
X1 = df.iloc[:,0]
X2 = df.iloc[:,1]
X = np.column_stack((X1,X2))
y = df.iloc[:,2]

# (a)(i)
import matplotlib.pyplot as plt

X_m1 = X[np.where(y == -1)]
X_p1 = X[np.where(y == 1)]

plt.scatter(X_m1[:, 0], X_m1[:, 1], c='r', marker='+', label="y = -1")
plt.scatter(X_p1[:, 0], X_p1[:, 1], c='b', marker='+', label="y = 1")
plt.title("Training data visualisation")
plt.legend()
plt.xlabel("X1")
plt.ylabel("X2")
plt.show()

# (a)(ii) Use sklearn to train a logistic regression classifier on the
↪  data and report the parameter values of the trai
from sklearn.linear_model import LogisticRegression
model = LogisticRegression().fit(X, y)
theta = [model.intercept_[0], model.coef_[0][0], model.coef_[0][1]]
print("theta = ", theta)

# (a)(iii)
plt.figure(figsize=(25,18), dpi=120)
plt.rcParams.update({'font.size': 18})
```

```
44
45  # Training data
46  plt.scatter(X_m1[:, 0], X_m1[:, 1], c='r', marker='+', s=500,
    ↪  label="Training y = -1")
47  plt.scatter(X_p1[:, 0], X_p1[:, 1], c='b', marker='+', s=500,
    ↪  label="Training y = 1")
48
49  # Predictions
50  X_pred_m1 = X[np.where(model.predict(X) == -1)]
51  X_pred_p1 = X[np.where(model.predict(X) == 1)]
52  plt.scatter(X_pred_m1[:, 0], X_pred_m1[:, 1], marker='o',
    ↪  facecolor='none', edgecolor='r', s=500, label="Prediction y = -1")
53  plt.scatter(X_pred_p1[:, 0], X_pred_p1[:, 1], marker='o',
    ↪  facecolor='none', edgecolor='b', s=500, label="Prediction y = 1")
54
55  # Decision boundary
56  plt.plot(X1, -1/theta[2] * (theta[0] + theta[1]*X1), linewidth=5, c='g',
    ↪  label='Decision boundary')
57
58  plt.legend(scatterpoints=1, fontsize=20)
59  plt.title("Logistic Regression Classifier", fontsize=20)
60  plt.xlabel("X1", fontsize=20)
61  plt.ylabel("X2", fontsize=20)
62  plt.show()
63
64  # (b)(i)
65  from sklearn.svm import LinearSVC
66  Cs = [0.001, 1, 1000]
67
68  for Ci in Cs:
69      model = LinearSVC(C=Ci).fit(X, y)
70      theta = [model.intercept_[0], model.coef_[0][0], model.coef_[0][1]]
71      print("C = %.3f"%Ci, "\n theta =", theta)
72
73      plt.figure(figsize=(25,18), dpi=120)
74      plt.rcParams.update({'font.size': 18})
75
76      # Training data
77      plt.scatter(X_m1[:, 0], X_m1[:, 1], c='r', marker='+', s=500,
        ↪  label="Training y = -1")
78      plt.scatter(X_p1[:, 0], X_p1[:, 1], c='b', marker='+', s=500,
        ↪  label="Training y = 1")
79
80      # Predictions
81      X_pred_m1 = X[np.where(model.predict(X) == -1)]
82      X_pred_p1 = X[np.where(model.predict(X) == 1)]
```

```
83      plt.scatter(X_pred_m1[:, 0], X_pred_m1[:, 1], marker='o',
        ↪  facecolor='none', edgecolor='r', s=500, label="Prediction y =
        ↪  -1")
84      plt.scatter(X_pred_p1[:, 0], X_pred_p1[:, 1], marker='o',
        ↪  facecolor='none', edgecolor='b', s=500, label="Prediction y = 1")
85
86      # Decision boundary
87      plt.plot(X1, -1/theta[2] * (theta[0] + theta[1]*X1), linewidth=5,
        ↪  c='g', label="Decision boundary for C = %.3f"%Ci)
88
89      plt.legend(scatterpoints=1, fontsize=20)
90      plt.title("Linear SVM Classifier for C = %.3f"%Ci)
91      plt.xlabel("X1", fontsize=20)
92      plt.ylabel("X2", fontsize=20)
93      plt.show()
94
95  #(c)(i)
96  # Additional features
97  X3 = np.power(X1, 2)
98  X4 = np.power(X2, 2)
99  X = np.column_stack((X1,X2,X3,X4))
100
101 model = LogisticRegression().fit(X, y)
102 theta = np.array([model.intercept_[0], model.coef_[0][0],
    ↪  model.coef_[0][1], model.coef_[0][2], model.coef_[0][3]])
103 print("theta =", theta)
104
105 #(c)(ii)
106 plt.figure(figsize=(25,18), dpi=120)
107 plt.rcParams.update({'font.size': 18})
108
109 # Training data
110 plt.scatter(X_m1[:, 0], X_m1[:, 1], c='r', marker='+', s=500,
    ↪  label="Training y = -1")
111 plt.scatter(X_p1[:, 0], X_p1[:, 1], c='b', marker='+', s=500,
    ↪  label="Training y = 1")
112
113 # Predictions
114 X_pred_m1 = X[np.where(model.predict(X) == -1)]
115 X_pred_p1 = X[np.where(model.predict(X) == 1)]
116 plt.scatter(X_pred_m1[:, 0], X_pred_m1[:, 1], marker='o',
    ↪  facecolor='none', edgecolor='r', s=500, label="Prediction y = -1")
117 plt.scatter(X_pred_p1[:, 0], X_pred_p1[:, 1], marker='o',
    ↪  facecolor='none', edgecolor='b', s=500, label="Prediction y = 1")
118
119 # (c)(iii)
120 def calcCost(X, y, theta):
121     """
```

```
122     Calculates the cost function value
123 
124     :param X: Features X (without X_0 = 1 for each row)
125     :param y: Outputs y
126     :param theta: Theta parameter values
127     :return: Cost function
128     """
129     m = X.shape[0]
130     Xtmp = np.insert(X, 0, 1, axis=1).T
131     cost = 1/m * sum(np.log(1 + np.exp(-y * np.dot(theta,
        ↪  Xtmp)))/np.log(2))
132     return cost
133 
134 # Cost function for the logistic regression model
135 print("J(theta) = %f\n"%calcCost(X, y, theta))
136 
137 # Baseline model
138 (values,counts) = np.unique(y, return_counts=True)
139 mostCommonValue = values[np.argmax(counts)]  # prints the most frequent
    ↪  element
140 baselineTheta = [mostCommonValue, 0, 0, 0, 0]
141 print("theta_baseline =", baselineTheta)
142 print("J(theta_baseline) = %f"%calcCost(X, y, baselineTheta))
143 
144 # (c)(iv)
145 # Decision boundary
146 X1_sorted = np.sort(X1)
147 X2_sorted = np.sort(X2)
148 a = -theta[3]/theta[2]
149 b = -theta[1]/theta[2]
150 c = -theta[0]/theta[2] - theta[4] / theta[2] * np.power(X2_sorted, 2)
151 boundary = a * np.power(X1_sorted, 2) + b * X1_sorted + c
152 
153 plt.plot(X1_sorted, boundary, linewidth=5, c='g', label="Decision
    ↪  boundary")
154 plt.axhline(-baselineTheta[0], c='y', linewidth=5, label='Baseline
    ↪  model')
155 plt.legend(scatterpoints=1, fontsize=20)
156 plt.title("Logistic Regression Classifier (with additional squared
    ↪  features)")
157 plt.xlabel("X1", fontsize=20)
158 plt.ylabel("X2", fontsize=20)
159 plt.show()
```