Assignment 2 – Logistic Regression & SVC

Student number: 20324412

Module: CS7CS4/CSU44061

Part a

Part i. Data

The data was read in as below.

```
#Data

data = pd.read_csv('week2.txt',)

data.head()

data.reset_index(inplace=True) #reset the index

data.columns = ['X1', 'X2', 'y']  #rename the columns
```

The resultant data is of the form in table 1. It includes two features X_1, X_2 and a target variable y.

```
#Display top 5 rows

#data.head()
```

	X1	X2	y
0	-1.00	0.83	1
1	0.99	0.22	1
2	0.76	0.85	-1
3	0.78	0.59	1
4	0.02	0.91	-1

Table 1. Data including two features X_1, X_2 and a target variable y.

Data inspection

The class balance for the target variable y is shown in table 2. Class 1 (y = 1) has significantly more observations than Class 2 (y = -1).

y	1	-1
Number of observations	637	362
% of Observations	63.76%	36.24%

Table 2. Class Balance

This was determined using the following code;

```
df2.y.value_counts()
```

Features

The features were extracted for future model fitting

```
#Extract Features

X1 = data.iloc[:,0]

X2 = data.iloc[:,1]

X = np.column_stack((X1,X2))

y= data.iloc[:,2]
```

A copy of the original dataframe was made for question 1 so that the original dataframe could be used in future sections;

```
df = data.copy()
```

Data Visualisation.

A visualisation of the data is shown in figure 1. The class imbalance is apparent, there is significantly more green observations (y = -1) than red observations (y = 1). The code to implement the visualisation is as below. The df.loc command was used to condition on the target variable y in order to color code the observations according to their value of the target variable y, either $(y = 1, y = -1)$

```
#Code to visualise data and the colour code according to the value of the
target variable y

plt.scatter(df.loc[df['y'] == 1, 'X1'], df.loc[df['y'] == 1, 'X2'], marker
= '+', c = 'g')

plt.scatter(df.loc[df['y'] == -1, 'X1'], df.loc[df['y'] == -1, 'X2'],
marker = '+', c = 'r')

plt.xlabel('X1')

plt.ylabel('X2')

plt.title('Data & Logistic Regression model')

plt.legend(['training data, y=1','training data, y=-1'], fancybox=True,
framealpha=1)

plt.show()
```

Figure 1. Visualisation of the data

Part ii. Logistic Regression Model

The logistic regression function from the sklearn library was used to train a logistic regression model on the data. The code used is as follows;

```
# Logistic regression model

log_reg_model = LogisticRegression(penalty= 'none', solver= 'lbfgs')

log_reg_model.fit(X, y)


#Output

log_reg_model.intercept_

array([1.75827784])


log_reg_model.coef_

array([[ 0.20438028, -5.44379509]])
```

Model Parameters

As above, the resultant model parameters are as follows;

$\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

$\theta^T x = 1.758 + 0.2014\, x_1 - 5.444 x_2$

Whereby θ_0 is the intercept and θ_1, θ_2 are the slope parameters.

These can be interpreted as follows;

$y = +1$ when $1.758 + 0.2014\, x_1 - 5.444 x_2 > 0$

$y = -1$ when $1.758 + 0.2014\, x_1 - 5.444 x_2 < 0$

Part iii. Prediction

The trained logistic regression classifier was then used to predict the target values in

the training data. This was achieved using the following code;

```
#Predictions

predictions = log_reg_model.predict(X)
```

The predictions were then added to the dataframe as below

```
df['preds'] = predictions
```

Decision boundary

The decision boundary of the logistic regression model is the set of all points x that satisfy;

$$P(y = 1\,|x) = P(y = 0|x) = \frac{1}{2}$$

Given

$$P(y = 1\,|x) = \frac{1}{1 + e^{-\boldsymbol{\theta}^t x_+}}$$

Where $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2)$

Therefore the decision boundary can be derived as follows;

$$\frac{1}{1 + e^{-\boldsymbol{\theta}^t x_+}} = \frac{1}{2}$$

$$1 + e^{-\theta^t x_+} = 2$$

$$\therefore\ e^{-\theta^t x_+} = 1 \text{ and so } \theta^t x_+ = 0$$

Now we have that

$$\boldsymbol{\theta}^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$$

Rearranging to determine the decision boundary to plot we have;

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

The code to implement the decision boundary is as follows;

```
decision_boundary = (-log_reg_model.coef_[0,0]/log_reg_model.coef_[0,1])*X1
- log_reg_model.intercept_[0]/log_reg_model.coef_[0,1]
```

Plot of data, predictions and decision boundary

The function *plot_data_preds_db* was used to plot the data, predictions and decision boundary is as below. The resultant plot is shown in figure 2.

```
#Function

def plot_data_preds_db (df, log_reg_model):

    'Plot data, logistic regression predictions and decision boundary'

    #Decision boundary

    decision_boundary = (-
log_reg_model.coef_[0,0]/log_reg_model.coef_[0,1])*X1 -
log_reg_model.intercept_[0]/log_reg_model.coef_[0,1]
```

```python
    #Plot of Predictions

    plt.scatter(df.loc[df['preds'] == 1, 'X1'], df.loc[df['preds'] == 1, 'X2'], marker = 'o', facecolors='none', edgecolors= 'k')

    plt.scatter(df.loc[df['preds'] == -1, 'X1'], df.loc[df['preds'] == -1, 'X2'], marker = 'o',facecolors='none', edgecolors= 'y')

    #Plot of Training Data

    plt.scatter(df.loc[df['y'] == 1, 'X1'], df.loc[df['y'] == 1, 'X2'], marker = '+', c = 'g')

    plt.scatter(df.loc[df['y'] == -1, 'X1'], df.loc[df['y'] == -1, 'X2'], marker = '+', c = 'r')

    #Plot decision boundary

    plt.plot(X1, decision_boundary, linewidth = '4')

    #Labels

    plt.xlabel('X1')

    plt.ylabel('X2')

    plt.title('Data & Logistic Regression model')

    plt.legend(['decision boundary', 'predictions, y = 1', 'predictions, y = -1', 'training data, y = 1','training data, y = -1'], fancybox=True, framealpha=1, bbox_to_anchor=(1.04,1), loc="upper left") #:D

    plt.show()


#Implement function

plot_data_preds_db (df, log_reg_model)
```

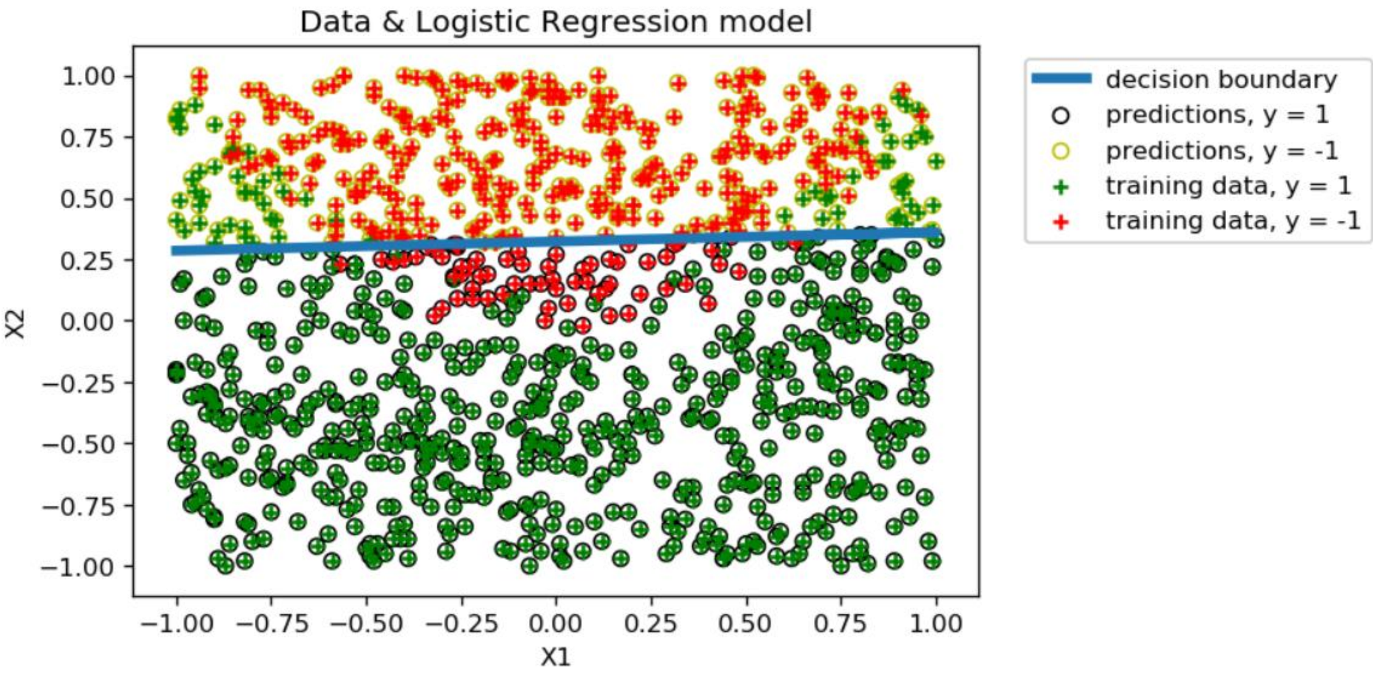

Figure 3. Training data, predictions and the decision boundary obtained via logistic regression.

Part 4 Comment

The predictions are plotted against the training data as shown in Figure 3. As can be seen, the two classes (y = 1, y = -1) within the data are not linearly separable, and therefore it is not possible to fit a logistic model that is 100% accurate, i.e that is able to fully separate the two classes. As a result several observations are misclassified on either side of the decision boundary. Correctly classified observations for class 1 (y = 1) are evident by green markers (true class, y = 1) surrounded by black circles (predicted y = 1), while misclassified observations in this class are those that are green markers (true class, y = 1), surrounded by yellow circles (predicted y = -1). On the other hand, correctly classified observations for class 2 (y = -1) are evident by red markers (true class, y = -1) surrounded by yellow circles (predicted y = 1), while misclassified observations in this class are those that are red markers (true class, y = -1), surrounded by black circles (predicted y = 1). However the model is correct in that it correctly classifies more observations as the dominant class, y = 1.

Part b – Support Vector Machine

Part i. Model parameters

Linear SVM classifiers were trained for a wide range of values of the penalty parameter C ranging from 0.0005 to 1000 and the resultant model parameters are shown in table 4. The results are discussed in b(iv).

Model parameters

	C	theta0	theta1	theta2
0	0.0005	0.142473	0.005147	-0.298434
1	0.0010	0.194007	0.010342	-0.480687
2	0.0050	0.293734	0.035715	-0.971433
3	0.0100	0.346585	0.051939	-1.179001
4	0.0500	0.465665	0.070106	-1.541634
5	0.1000	0.500913	0.073693	-1.639150
6	0.5000	0.540140	0.075964	-1.745865
7	1.0000	0.546026	0.076112	-1.761765
8	5.0000	0.551070	0.076186	-1.775186
9	10.0000	0.551718	0.076196	-1.776904
10	50.0000	0.551554	0.075381	-1.776814
11	100.0000	0.590483	0.038071	-1.780726
12	500.0000	0.833970	0.301772	-2.040766
13	1000.0000	1.060432	0.145159	-0.714983

Table 4. SVC model parameters for a range of values of the penalty term C

The following code was implemented to train the SVC models for a range of values of C and determine the various model parameters as shown in table 4;

```
#Function to implement SVC for a range of parameters

def svc_range_c(X, y, c_test):

    '''Implement Support Vector Classification (SVC)

    for  a range of values of the penalty term C '''

    #Setup

    df_results = []


    #Loop through c parameters and implement SVC

    for c_param in c_test:

        model = LinearSVC(C= c_param).fit(X, y)
```

```
        #Dictionarry of values

        d = {

            'C' : c_param,

        'theta0': model.intercept_[0],

        'theta1' :  model.coef_[0,0],

        'theta2' :  model.coef_[0,1] ,

        }


        df_results.append(d)

    #Return dataframe of results - model parameters for a range of penalty
terms C

    df_svc_results = pd.DataFrame(df_results)


    return df_svc_results
```

The following code was used to implement the function and return the dataframe, i.e table 4. A range of C values ranging from 0.0005 to 1000 were tested.

```
#C parameter values

c_test = np.geomspace(0.001, 1000, num = 7)

c_test = np.concatenate((c_test, c_test/2))

c_test = np.sort(c_test)


#Dataframe of results

df_svc_res = svc_range_c(X, y, c_test)
```

Part ii. Plot predictions

A subset of these predictions, i.e for C increasing on a logarithmic scale, are plotted in Figure 4 along with decision boundary. Similarly to part two, the decision boundary is given by;

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

The code used to plot the data, predictions and decision boundary is as below;

#Code to plot SV

```
def svc_plot_preds_range_c(data, c_test):

    '''Implement Support Vector Classification (SVC)

    for a range of values of the penatly term C. Plot the resultant SVC
predictions and decision boundaries against the data  '''


    #Param setup

    fig = plt.figure(figsize=(15, 10))

    count = 0

    data_results = []


    #Features

    X1 = data.iloc[:,0]

    X2 = data.iloc[:,1]

    X = np.column_stack((X1,X2))

    y= data.iloc[:,2]


    #Loop through c parameters and implement SVC

    for c_param in c_test:

        count +=1

        model = LinearSVC(C = c_param).fit(X, y)

        #Predictions

        predictions = model.predict(X)

        data['preds'] = predictions
```

```
        #Plot

        plt.subplot(3, 3, count)


        plt.scatter(data.loc[data['preds'] == 1, 'X1'], data.loc[data['preds'] == 1, 'X2'], marker = 'o', facecolors='none', edgecolors= 'k')

        plt.scatter(data.loc[data['preds'] == -1, 'X1'], data.loc[data['preds'] == -1, 'X2'], marker = 'o',facecolors='none', edgecolors= 'y')

        #Truth

        plt.scatter(data.loc[data['y'] == 1, 'X1'], data.loc[data['y'] == 1, 'X2'], marker = '+', c = 'g')

        plt.scatter(data.loc[data['y'] == -1, 'X1'], data.loc[data['y'] == -1, 'X2'], marker = '+', c = 'r')


        #Decision boundary

        decision_boundary = (-model.coef_[0,0]/model.coef_[0,1])*X1 - model.intercept_[0]/model.coef_[0,1]

        plt.plot(X1, decision_boundary, linewidth = '4')

        #Labels

        plt.title('SVM, C = %.3f' %c_param)

        plt.xlabel('X1')

        plt.ylabel('X2')


    plt.legend(['decision boundary', 'predictions, y = 1', 'predictions, y = -1', 'training data, y = 1','training data, y = -1'], fancybox=True, framealpha=1, bbox_to_anchor=(1.04,1), loc="upper left") #:D

    plt.show()
```

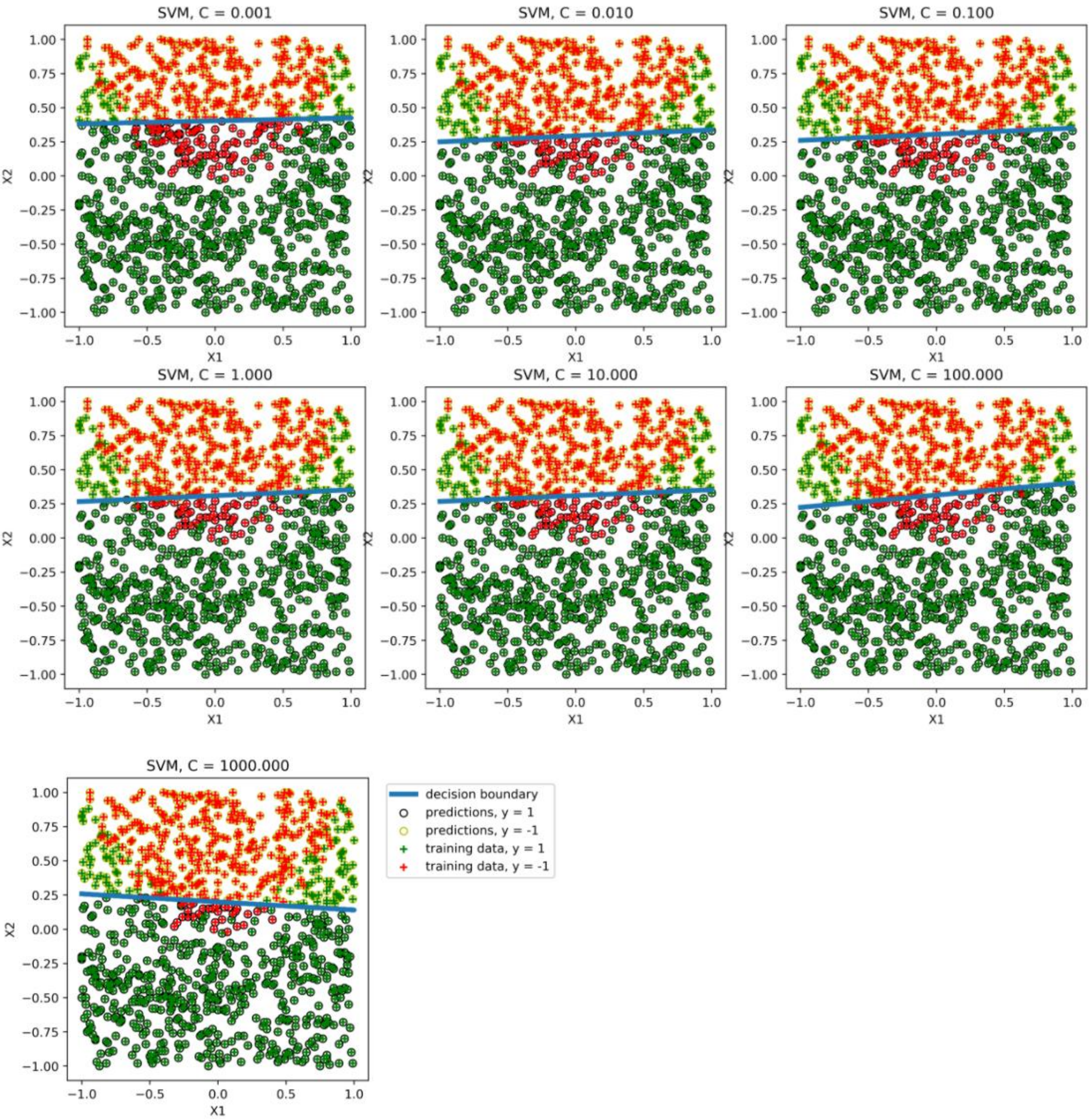

Figure 4. Support Vector Machine Classification for varying levels of the parameter C.

Part iii Discuss Results - Impact of changing C on model parameters & predictions

The hyper parameter C allows the influence of the penalty term to be controlled as given by equation I.

$$J(\vartheta) = \frac{1}{m}\sum_{i=1}^{m} \max\left(0,\ 1 - y^{(i)}\theta^T x^{(i)}\right) + \frac{\vartheta^T\vartheta}{C} \qquad (1)$$

It's value has a significant effect on the model parameters. Decreasing C gives the penalty more importance. This results in SVM choosing smaller model parameters θ, as seen in Table 4, shown again below. It leads to higher bias but lower variance given the smaller model parameters and possibly underfitting of the data. On the other hand, increasing C gives the penalty less importance, resulting in larger model parameters.

As C increases from 0.0005 to 1000 a steadying increase in the model parameters is observed. For example, for C = 0.005, the model is as follows

$C = 0.0005$

$0.142 + 0.00514\, x_1 - 5.444 x_2$

While for C = 500, the model is;

$0.834 + 0.302\, x_1 - 2.041 x_2$

I.e a significant increase in the model parameters is observed. This was the case until C = 1000, whereby the model failed to converge and yielding unreliable results.

Impact on the SVM predictions

As stated above, smaller values of C results in smaller model parameters, higher bias, lower variance and possibly underfitting of the data. While larger values of C results in larger model parameters, lower bias, higher variance and potential overfitting of the data. This is observed when comparing the model predictions in Figure 4, for say $C = 0.01$ and $C = 100$ which are re-plotted in Figure 5 below for ease of visualisation.

Considering C = 0.001 it could be argued that the model is under fitting the data. The decision boundary does not separate the two data classes sufficiently well and as a result a significant number of observations are misclassified, particularly for class 2 (y = -1) which are misclassified as class 1 (y = 1). This occurrence of misclassification could also be viewed as overfitting to the dominant class i.e class 1 (y = 1). The smaller model parameters are also evident in that the slope of the decision boundary is negligible, i.e ~ 0.

For C = 100, the model correctly classifies more observations than that of C = 0.001. Evidently there are fewer misclassified observations for class 2 (y = -1) below the decision boundary. However it could be argued that the model is overfitting the data. It is a less simple model to that of C = 0.001 in that the decision boundary is less neutral, it has a greater slope and perhaps it would not fit well to unseen test data.

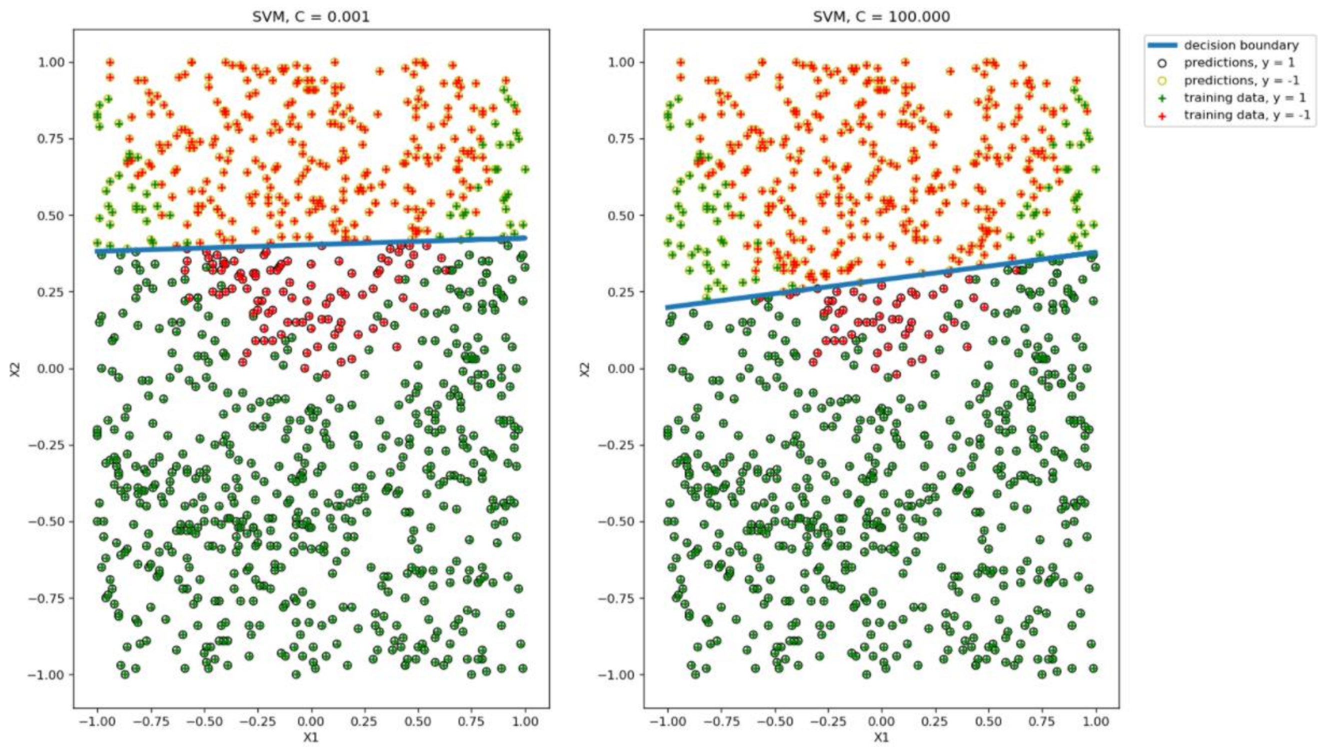

Figure 5. Support Vector Machine Classification for C = 0.001, 100.

Part c – Logistic Regression & Additional Features

Part i. Additional features

Two additional features were created by adding the square of each feature as below.

```
#Code to create additional features

df2 = data.copy()

df2['X1_sq'] = df2['X1']**2

df2['X2_sq'] = df2['X2']**2

#Extract Features

X1_sq = df2.iloc[:,3]

X2_sq = df2.iloc[:,4]

X_v2 = np.column_stack((X,X1_sq, X2_sq))
```

The resultant data frame is given by table 3;

	X1	X2	y	X1_sq	X2_sq
0	-1.00	0.83	1	1.0000	0.6889
1	0.99	0.22	1	0.9801	0.0484
2	0.76	0.85	-1	0.5776	0.7225
3	0.78	0.59	1	0.6084	0.3481
4	0.02	0.91	-1	0.0004	0.8281

Table 3. Dataframe of original data and the additional squared features.

Logistic Regression Classifier

A logistic regression classifier was then trained on the data as below.

```
#Code to train logistic regression classifier

log_reg_model2 = LogisticRegression(penalty= 'none', solver= 'lbfgs')

log_reg_model2.fit(X_v2, y)

#Model parameters

log_reg_model2.intercept_

array([0.20221889])

log_reg_model2.coef_

array([[  0.15519415, -19.32105785,  18.53330391,   1.3851579 ]])
```

The model parameters are as follows;

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

$$\theta^T x = 0.202 + 0.1552\, x_1 - 19.321 x_2 + 18.533 x_1^2 + 1.385 x_2^2$$

Part ii. Predictions & Plot

The trained logistic regression classifier was then used to predict the target values in the training data. This was achieved using the following code;

```
#Predictions

predictions2 = log_reg_model2.predict(X_v2)
```

The predictions were then added to the dataframe as below

```
df['preds'] = predictions2
```

Plot of data & predictions

The function *plot_data_preds* was used to plot the data, predictions and decision boundary as below. The resultant plot is shown in figure 6.

```
#Function

def plot_data_preds_db (df, log_reg_model, model_name, preds_col):

    'Plot data, logistic regression predictions and decision boundary'


    #Decision boundary

    decision_boundary = (-
log_reg_model.coef_[0,0]/log_reg_model.coef_[0,1])*X1 -
log_reg_model.intercept_[0]/log_reg_model.coef_[0,1]


    #Plot of Predictions

    plt.scatter(df.loc[df['preds'] == 1, 'X1'], df.loc[df['preds'] == 1,
'X2'], marker = 'o', facecolors='none', edgecolors= 'k')

    plt.scatter(df.loc[df['preds'] == -1, 'X1'], df.loc[df['preds'] == -1,
'X2'], marker = 'o',facecolors='none', edgecolors= 'y')

    #Plot of Training Data

    plt.scatter(df.loc[df['y'] == 1, 'X1'], df.loc[df['y'] == 1, 'X2'],
marker = '+', c = 'g')
```

```
    plt.scatter(df.loc[df['y'] == -1, 'X1'], df.loc[df['y'] == -1, 'X2'],
marker = '+', c = 'r')


    #Labels

    plt.xlabel('X1')

    plt.ylabel('X2')

    plt.title('Data & {}'.format(model_name))

    plt.legend(['predictions, y = 1', 'predictions, y = -1', 'training
data, y = 1','training data, y = -1'], fancybox=True, framealpha=1,
bbox_to_anchor=(1.04,1), loc="upper left") #:D

    plt.show()


#Implement function

preds_col = 'preds'

model_name = 'Logistic Regression model w/ Squared Featues'

plot_data_preds(df2, log_reg_model2, model_name, preds_col)
```

Comment

The resultant model provides a very good fit of the data as shown in Figure 6. The majority of the observations are correctly classified. It is interesting to note the effect of including the quadratic terms. The decision boundary is no longer linear and is able to match the non-linearity present in the class distribution of the data. However it could be argued that the model is overfitting to the training data and perhaps would not fit well to test data.

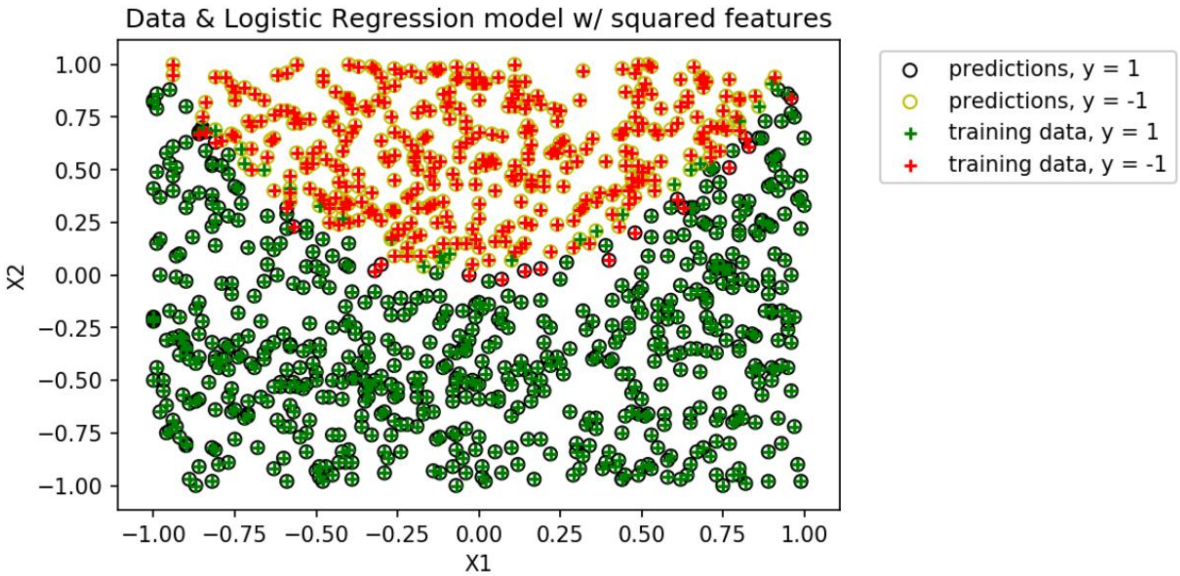

Figure 6. Training data and predictions obtained via logistic regression including squared features.

Part iii. Baseline model

A baseline model was fit to the data and the results are shown in figure 7. The baseline model in question was chosen as one that predicts all observations to come from the dominant class, y = 1. Again the class balance, shown in table 2 is;

y	1	-1
Number of observations	637	362
% of Observations	63.76%	36.24%

Table 2. Class Balance

Thus the predictions were simply determined as follows;

```
df2['preds_baseline'] = np.ones(len(y))
```

Again the function, plot_data_preds was used to plot the data and predictions, implemented as follows;

```
#Implement

preds_col_bl = 'preds_baseline'

model_name2 = 'Baseline model'

plot_data_preds(df2, log_reg_model2, model_name2, preds_col_bl)
```

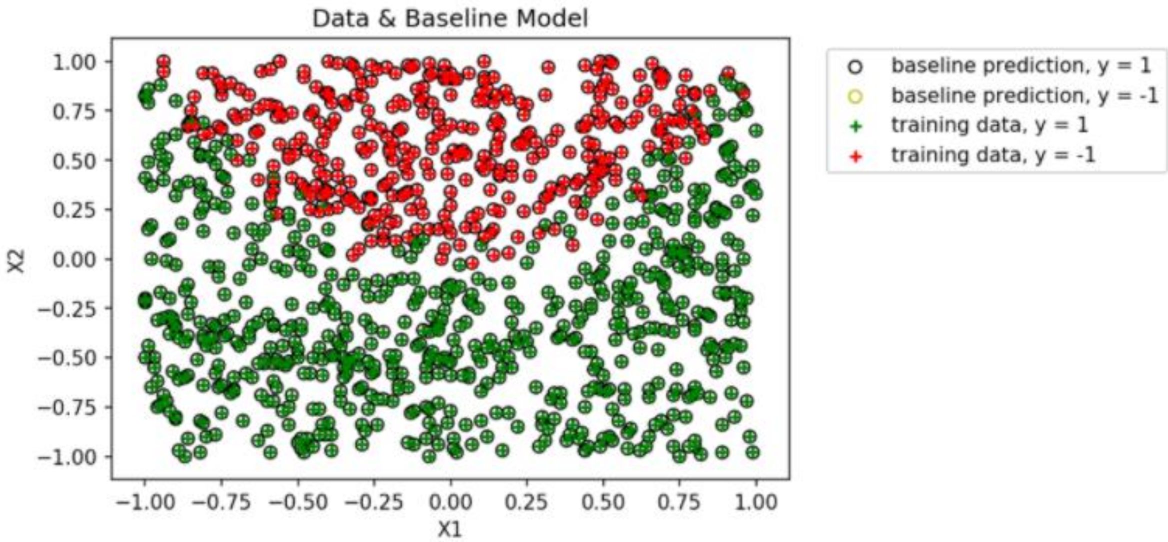

Figure 7. Training data and predictions obtained via a baseline model

Comparison

Comparing the baseline model in Figure 7 to the logistic regression model in Figure 6, it is evident that the logistic regression model provides a much better fit of the data. However in the case of the baseline model, although it is underfitting the data, it is correct 63.76% of the time, in that it correctly classifies 637 of the observations as class 1 (y =1). For a very simple model, this is not the worst result in that it is significantly better than chance (50%). Furthermore the model however it is undeniable that it provides a better fit to the data and is able to capture the non-linear decision bound. It misclassifies very few observations. However it is arguable that it is overfitting to the training data and perhaps would not fit well to test data.

Part 4 – Quadratic Decision Boundary

As in part a, the decision boundary of the logistic regression model is the set of all points x that satisfy;

$$P(y = 1\,|x) = P(y = 0|x) = \frac{1}{2}$$

Given

$$P(y = 1\,|x) = \frac{1}{1 + e^{-\boldsymbol{\theta}^t x_+}}$$

Where $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2, \theta_3, \theta_4)$

Therefore the decision boundary can be derived as follows;

$$\frac{1}{1 + e^{-\boldsymbol{\theta}^t x_+}} = \frac{1}{2}$$

$$1 + e^{-\theta^t x_+} = 2$$

$\therefore\ e^{-\theta^t x_+} = 1$ and so $\theta^t x_+ = 0$

Now we have that

$$\boldsymbol{\theta}^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 = 0$$

Rearranging to determine the decision boundary to plot we have;

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

To determine the curve, each value of x_1 was plugged into the right hand side giving;

$$\theta_2 x_2 + \theta_4 x_2^2 = -c$$

This was then solved using the quadratic equation formula, where the aforementioned could be written as;

$$\theta_2 x_2 + a x_2^2 + c = 0$$

And solving it using the quadratic formula, i.e

$$x_2 = -b \pm \frac{\sqrt{b^2 - 4ac}}{2a}$$

This was achieved using the following code. A function *get_roots* was written to solve the above equation and determine the roots and a function *get_quadratic_dec_bound* was used to implement this for each value of x1

#Function – get_roots

```
def get_roots(c, log_reg_model):

    #
```

```
    a = log_reg_model.coef_[0,3]

    b = log_reg_model.coef_[0,1]

    dis = (b**2) - (4 * a*c)


    # find two results

    root1 = (-b-cmath.sqrt(dis))/(2 * a)

    root2 = (-b + cmath.sqrt(dis))/(2 * a)


    #Choose appropriate root

    if ((root1 < 1.1) and (root1 > -0.15)):

        root = root1

    else:

        root = root2


    return root


#Function - get_quadratic_dec_bound

def get_quadratic_dec_bound(X1, log_reg_model2):

    ''' Get decision boundary from logistic regression model with quadratic
terms'''

    #Set up

    dec_boundary2 = []


    #Loop through all x values and determine corresponding x2 value

    for xx1 in X1:

        c = log_reg_model2.intercept_ + log_reg_model2.coef_[0,0]*xx1 +
log_reg_model2.coef_[0,2]*xx1**2

        xx2 = get_roots(c, log_reg_model2)

        dec_boundary2.append(xx2)

return dec_boundary2

#Implement
```

```
dec_boundary2 = get_quadratic_dec_bound(X1, log_reg_model2)
```

Decision boundary plot

The decision boundary along with the predictions were then plotted using the function *plot_data_preds_dbII*. The resultant plot is shown in figure 8. The fact that the model is now a quadratic function due to the quadratic terms is apparent. The resultant model provides a very good fit of the data. The decision boundary is no longer linear and is able to match the non-linearity present in the class distribution of the data. Again it is arguable that the model is overfitting to the training data and perhaps would not fit well to unseen test data.

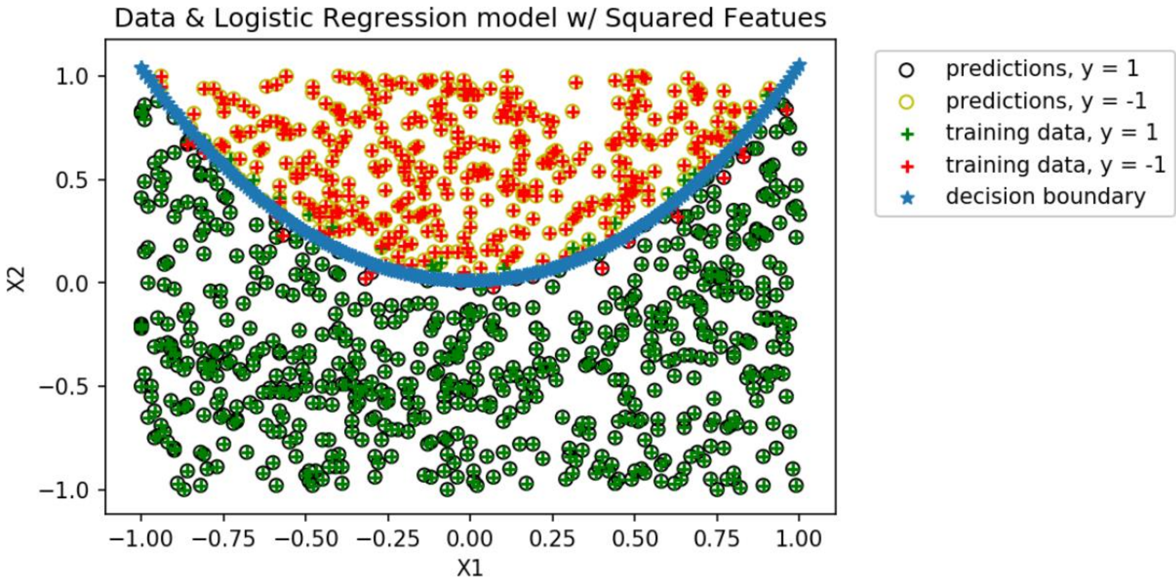

Figure 8. Training data, predictions and decision boundary obtained via logistic regression including squared features.

Code to plot predictions and decision boundary as in Figure 8.

```
def plot_data_preds_dbII(df, log_reg_model, model_name, preds_col, decision_boundary):

    'Plot data, logistic regression predictions + dec_boundary'


    #Plot of Predictions

    plt.scatter(df.loc[df[preds_col] == 1, 'X1'], df.loc[df[preds_col] == 1, 'X2'], marker = 'o', facecolors='none', edgecolors= 'k')

    plt.scatter(df.loc[df[preds_col] == -1, 'X1'], df.loc[df[preds_col] == -1, 'X2'], marker = 'o',facecolors='none', edgecolors= 'y')
```

```
    #Plot of Training Data

    plt.scatter(df.loc[df['y'] == 1, 'X1'], df.loc[df['y'] == 1, 'X2'],
marker = '+', c = 'g')

    plt.scatter(df.loc[df['y'] == -1, 'X1'], df.loc[df['y'] == -1, 'X2'],
marker = '+', c = 'r')


    #Decision boundary

    X1 = df.iloc[:,0]

    plt.scatter(X1, decision_boundary, marker = '*', linewidth = '1')


    #Labels

    plt.xlabel('X1')

    plt.ylabel('X2')

    plt.title('Data & {}'.format(model_name))

    plt.legend(['predictions, y = 1', 'predictions, y = -1', 'training
data, y = 1','training data, y = -1', 'decision boundary'], fancybox=True,
framealpha=1, bbox_to_anchor=(1.04,1), loc="upper left") #:D

    plt.show()


#Implement

preds_col = 'preds'

model_name = 'Logistic Regression model w/ Squared Featues'

plot_data_preds_dbII(df2, log_reg_model2, model_name, preds_col,
dec_boundary2)
```

Appendix Code

```
#Week 2 Assignment - Logistic Regression & SVC


#Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
import cmath


#****** Part a - Data & Logistic Regression
*********************************


#***************************
#Part i Data + Visualisation
#Data
data = pd.read_csv('week2.txt',)
data.head()
data.reset_index(inplace=True)
data.columns = ['X1', 'X2', 'y']


#Class balance
data.y.value_counts()


#Make a copy
df = data.copy()
df.head()


#Extract Features
X1 = df.iloc[:,0]
```

```
X2 = df.iloc[:,1]

X = np.column_stack((X1,X2))

y= df.iloc[:,2]


#Plot data and color code the observations according to their value of the target variable y

plt.scatter(df.loc[df['y'] == 1, 'X1'], df.loc[df['y'] == 1, 'X2'], marker = '+', c = 'g')

plt.scatter(df.loc[df['y'] == -1, 'X1'], df.loc[df['y'] == -1, 'X2'], marker = '+', c = 'r')

plt.xlabel('X1')

plt.ylabel('X2')

plt.title('Data & Logistic Regression model')

plt.legend(['training data, y=1','training data, y=-1'], fancybox=True, framealpha=1) #:D

plt.show()


#**** Part b Logistic Regression Model  *********************************


#Part ii Logistic Regression Model


log_reg_model = LogisticRegression(penalty= 'none',solver= 'lbfgs')

log_reg_model.fit(X, y)

log_reg_model.intercept_

log_reg_model.coef_


#*********************************

#Part iii Predictions


predictions = log_reg_model.predict(X)

df['preds'] = predictions

df.head()
```

```
#Test decision boundary

decision_boundary = log_reg_model.intercept_[0] +
log_reg_model.coef_[0,0]*X1 + log_reg_model.coef_[0,1]*X2



#Function to plot data, predictions, decision boundary

def plot_data_preds_db (df, log_reg_model):

    'Plot data, logistic regression predictions and decision boundary'



    #Decision boundary

    decision_boundary = (-
log_reg_model.coef_[0,0]/log_reg_model.coef_[0,1])*X1 -
log_reg_model.intercept_[0]/log_reg_model.coef_[0,1]



    #Plot of Predictions

    plt.scatter(df.loc[df['preds'] == 1, 'X1'], df.loc[df['preds'] == 1,
'X2'], marker = 'o', facecolors='none', edgecolors= 'k')

    plt.scatter(df.loc[df['preds'] == -1, 'X1'], df.loc[df['preds'] == -1,
'X2'], marker = 'o',facecolors='none', edgecolors= 'y')

    #Plot of Training Data

    plt.scatter(df.loc[df['y'] == 1, 'X1'], df.loc[df['y'] == 1, 'X2'],
marker = '+', c = 'g')

    plt.scatter(df.loc[df['y'] == -1, 'X1'], df.loc[df['y'] == -1, 'X2'],
marker = '+', c = 'r')

    #Plot decision boundary

    plt.plot(X1, decision_boundary, linewidth = '4')

    #Labels

    plt.xlabel('X1')

    plt.ylabel('X2')

    plt.title('Data & Logistic Regression model')

    plt.legend(['decision boundary', 'predictions, y = 1', 'predictions, y
= -1', 'training data, y = 1','training data, y = -1'], fancybox=True,
framealpha=1, bbox_to_anchor=(1.04,1), loc="upper left") #:D

    plt.show()
```

```
#Implement
plot_data_preds_db (df, log_reg_model)


#**************** Part b  - SVC ************************************

#SVC - test
model = LinearSVC(C=1).fit(X, y)
model.intercept_
model.coef_

#Part (i) Test range of values of C parameter

#Range of C
c_test = np.geomspace(0.001, 1000, num = 7)
c_test = np.concatenate((c_test, c_test/2))
c_test = np.sort(c_test)

#Test SVC for range of values of C
def svc_range_c(X, y, c_test):
    '''Implement Support Vector Classification (SVC)
    for a range of values of the penatly term C '''

    #Setup
    df_results = []

    #Loop through c parameters and implement SVC
    for c_param in c_test:
```

```
        model = LinearSVC(C= c_param).fit(X, y)

        #Dictionarry of values
        d = {
            'C' : c_param,
        'theta0': model.intercept_[0],
        'theta1' :  model.coef_[0,0],
        'theta2' :  model.coef_[0,1] ,
        }

        df_results.append(d)

    #Return dataframe of results - model parameters for a range of penalty
terms C
    df_svc_results = pd.DataFrame(df_results) #Results;

    return df_svc_results

#Implement & get dataframe
df_svc_res = svc_range_c(X, y, c_test)

#*******************************************************
#Part (ii) Plot Data, Predictions & Decision Boundary

#Function to plot range
def svc_plot_range_c(data, c_test, plot_dim):
    '''Implement Support Vector Classification (SVC)
    for a range of values of the penatly term C. Plot the resultant SVC
predictions and decision boundaries against the data  '''

    #Param setup
```

```
    fig = plt.figure(figsize=(15, 10))

    count = 0

    data_results = []


    #Features

    X1 = data.iloc[:,0]

    X2 = data.iloc[:,1]

    X = np.column_stack((X1,X2))

    y= data.iloc[:,2]

    #Loop through c parameters and implement SVC

    for c_param in c_test:

        count +=1

        model = LinearSVC(C = c_param).fit(X, y)

        #Predictions

        predictions = model.predict(X)

        data['preds'] = predictions


        #Plot

        plt.subplot(plot_dim[0], plot_dim[1], count)


        plt.scatter(data.loc[data['preds'] == 1, 'X1'],
data.loc[data['preds'] == 1, 'X2'], marker = 'o', facecolors='none',
edgecolors= 'k')

        plt.scatter(data.loc[data['preds'] == -1, 'X1'],
data.loc[data['preds'] == -1, 'X2'], marker = 'o',facecolors='none',
edgecolors= 'y')

        #Truth

        plt.scatter(data.loc[data['y'] == 1, 'X1'], data.loc[data['y'] ==
1, 'X2'], marker = '+', c = 'g')

        plt.scatter(data.loc[data['y'] == -1, 'X1'], data.loc[data['y'] ==
-1, 'X2'], marker = '+', c = 'r')


        #Decision boundary
```

```
        decision_boundary = (-model.coef_[0,0]/model.coef_[0,1])*X1 -
model.intercept_[0]/model.coef_[0,1]

        plt.plot(X1, decision_boundary, linewidth = '4')

        #Labels

        plt.title('SVM, C = %.3f' %c_param)

        plt.xlabel('X1')

        plt.ylabel('X2')


    plt.legend(['decision boundary', 'predictions, y = 1', 'predictions, y
= -1', 'training data, y = 1','training data, y = -1'], fancybox=True,
framealpha=1, bbox_to_anchor=(1.04,1), loc="upper left") #:D

    plt.show()


#Implement function

c_test = np.geomspace(0.001, 1000, num = 7) # Range of c values

plot_dim = [3,3] #plot dimensions

svc_plot_range_c(data, c_test, plot_dim) #implement


#Implement - focus on two values of C

c_test = [0.001, 100]

plot_dim = [1,2]

svc_plot_range_c(data, c_test, plot_dim)


#********** Part c  - Logistic Regression + Additional Features ********

#Part (i)

#Data

df2 = data.copy()

df2['X1_sq'] = df2['X1']**2

df2['X2_sq'] = df2['X2']**2

df2.head()


#Features
```

```
X1_sq = df2.iloc[:,3]

X2_sq = df2.iloc[:,4]

X_v2 = np.column_stack((X,X1_sq, X2_sq))


#Log reg model

log_reg_model2 = LogisticRegression(penalty= 'none',solver= 'lbfgs')

log_reg_model2.fit(X_v2, y)

log_reg_model2.intercept_

log_reg_model2.coef_


#**********************

#Part ii - Predicitions

predictions2  = log_reg_model2.predict(X_v2)

preds_col = 'preds'

df2[preds_col] = predictions2


#Function to plot predictions

def plot_data_preds(df, log_reg_model, model_name, preds_col):

    'Plot data, logistic regression predictions'


    #Plot of Predictions

    plt.scatter(df.loc[df[preds_col] == 1, 'X1'], df.loc[df[preds_col] ==
1, 'X2'], marker = 'o', facecolors='none', edgecolors= 'k')

    plt.scatter(df.loc[df[preds_col] == -1, 'X1'], df.loc[df[preds_col] ==
-1, 'X2'], marker = 'o',facecolors='none', edgecolors= 'y')

    #Plot of Training Data

    plt.scatter(df.loc[df['y'] == 1, 'X1'], df.loc[df['y'] == 1, 'X2'],
marker = '+', c = 'g')

    plt.scatter(df.loc[df['y'] == -1, 'X1'], df.loc[df['y'] == -1, 'X2'],
marker = '+', c = 'r')


    #Labels
```

```
    plt.xlabel('X1')

    plt.ylabel('X2')

    plt.title('Data & {}'.format(model_name))

    plt.legend(['predictions, y = 1', 'predictions, y = -1', 'training data, y = 1','training data, y = -1'], fancybox=True, framealpha=1, bbox_to_anchor=(1.04,1), loc="upper left") #:D

    plt.show()


#Implement

preds_col = 'preds'

model_name = 'Logistic Regression model w/ Squared Featues'

plot_data_preds(df2, log_reg_model2, model_name, preds_col)


#**************************************************************
```

#Part iii - Baseline Model

```
#Recheck classes

df2.y.value_counts()


#Preds Baseline

preds_col_bl = 'preds_baseline'

df2['preds_baseline'] = np.ones(len(y))


#Plot Baseline model - use plot_data_preds function

preds_col_bl = 'preds_baseline'

model_name2 = 'Baseline model'

plot_data_preds(df2, log_reg_model2, model_name2, preds_col_bl)



#**************************************************************
```

#Part iii - Quadratic Decision Boundary

```
def get_roots(c, log_reg_model):

    '''Get roots of quadratic equation '''

    #

    a = log_reg_model.coef_[0,3]

    b = log_reg_model.coef_[0,1]

    distance = (b**2) - (4 * a*c)


    # find two results

    root1 = (-b-cmath.sqrt(distance))/(2 * a)

    root2 = (-b + cmath.sqrt(distance))/(2 * a)


    #Choose appropriate root

    if ((root1 < 1.1) and (root1 > -0.15)):

        root = root1

    else:

        root = root2


    return root


def get_quadratic_dec_bound(X1, log_reg_model2):

    ''' Get decision boundary from logistic regression model with quadratic
terms'''

    #Set up

    dec_boundary2 = []


    #Loop through all x values and determine corresponding x2 value

    for xx1 in X1:

        c = log_reg_model2.intercept_ + log_reg_model2.coef_[0,0]*xx1 +
log_reg_model2.coef_[0,2]*xx1**2

        xx2 = get_roots(c, log_reg_model2)

        dec_boundary2.append(xx2)
```

```
    return dec_boundary2


#Implement

dec_boundary2 = get_quadratic_dec_bound(X1, log_reg_model2)


#Plot

def plot_data_preds_dbII(df, log_reg_model, model_name, preds_col, decision_boundary):

    'Plot data, logistic regression predictions + dec_boundary'


    #Plot of Predictions

    plt.scatter(df.loc[df[preds_col] == 1, 'X1'], df.loc[df[preds_col] == 1, 'X2'], marker = 'o', facecolors='none', edgecolors= 'k')

    plt.scatter(df.loc[df[preds_col] == -1, 'X1'], df.loc[df[preds_col] == -1, 'X2'], marker = 'o',facecolors='none', edgecolors= 'y')

    #Plot of Training Data

    plt.scatter(df.loc[df['y'] == 1, 'X1'], df.loc[df['y'] == 1, 'X2'], marker = '+', c = 'g')

    plt.scatter(df.loc[df['y'] == -1, 'X1'], df.loc[df['y'] == -1, 'X2'], marker = '+', c = 'r')


    #Decision boundary

    X1 = df.iloc[:,0]

    plt.scatter(X1, decision_boundary, marker = '*', linewidth = '1')


    #Labels

    plt.xlabel('X1')

    plt.ylabel('X2')

    plt.title('Data & {}'.format(model_name))

    plt.legend(['predictions, y = 1', 'predictions, y = -1', 'training data, y = 1','training data, y = -1', 'decision boundary'], fancybox=True, framealpha=1, bbox_to_anchor=(1.04,1), loc="upper left") #:D
```

```
    plt.show()


#Implement

preds_col = 'preds'

model_name = 'Logistic Regression model w/ Squared Featues'

plot_data_preds_dbII(df2, log_reg_model2, model_name, preds_col,
dec_boundary2)
```