DECLARATION: I understand that this is an **individual** assessment and that collaboration is not permitted. I have read, understand and agree to abide by the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar. I understand that by returning this declaration with my work, I am agreeing with the above statement.

Machine Learning Week 2 Assignment - id:8–8-8

(a) Logistic Regression - Linear

i. Figure 1 shows the plotted data. The x and y axes are the values of the first and second features X_1 and X_2. The data points have different markers depending on target value y, with pink 'X' representing points with y = -1 and olive '+' representing points with y = 1

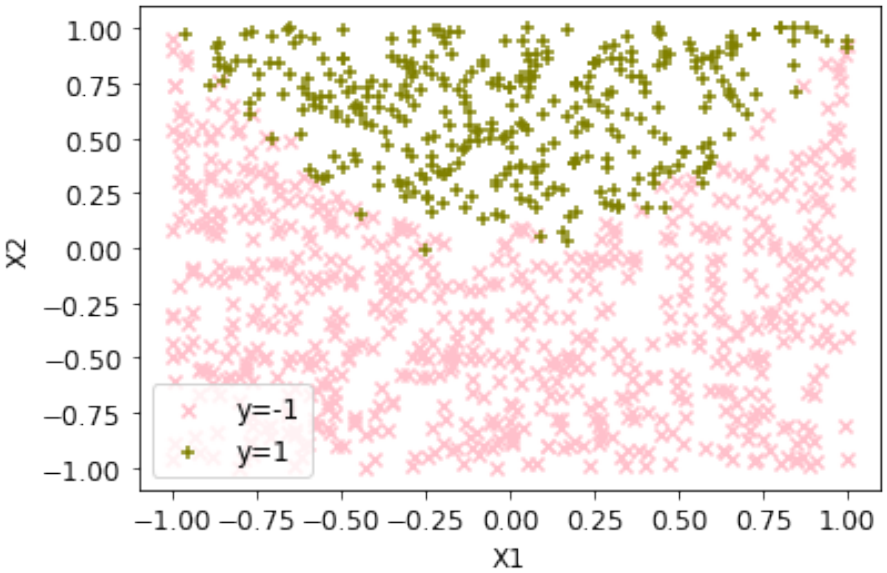

Figure 1: Data points plotted on X_1 and X_2 and differentiated by y

ii. The logistic regression model minimises the cost function $J(\theta) = \frac{1}{m}\sum_{i=1}^{m} log(1+e^{-y^{(i)}\theta^T x^{(i)}})$. The decision boundary is given by $y = sign(i + b_1X_1 + b_2X_2)$, where i is the intercept and $[b_1, b_2]$ are the coefficients. By plugging in the values resulting from training the LR model and printing the parameters we get $y = sign(-2.27 + 0.37X_1 + 6.25X_2)$. 0.37 and 6.25 are the coefficients of X_1 and X_2 respectively, and -2.27 is the intercept. The coefficients show that the LR model is influenced to a much greater extent by X_2 than by X_1. This happens because y is only 1 for $X_2 \leq 0$, and as X_2 further increases, y is almost certainly 1. On the flip side, X_1 doesn't give much information. Not only are y = 1 points distributed all throughout the range of X_1, but they are symmetrically distributed on each side of $X_1 = 0$. Thus, without accounting for the absolute value of X_1 or for X_1^2 there is little information to be gained from this parameter.

iii. Using the formula from (a) (ii), the decision boundary represents the values where $i + b_1X_1 + b_25X_2 = 0$. By rewriting this equation we obtain $X_2 = -\frac{i}{b_2} - \frac{b_1}{b_2}X_1$ with the intercept being $-\frac{i}{b_2}$ and the slope $-\frac{b_1}{b_2}$. Plugging in the numbers, the formula for the decision boundary is $X_2 = 0.36 - 0.06X_1$. Figure 2 shows the predictions along the actual values. In addition to the symbols in figure 1, predictions for $\hat{y} = -1$ are shown as red squares, and $\hat{y} = 1$ are shown as yellow diamonds. The decision boundary is also shown as a black line. It can be seen how all predictions for $\hat{y} = 1$ are concentrated in one side of X_2, while X_1 has barely

any impact on the decision boundary, as could be predicted from the coefficients.

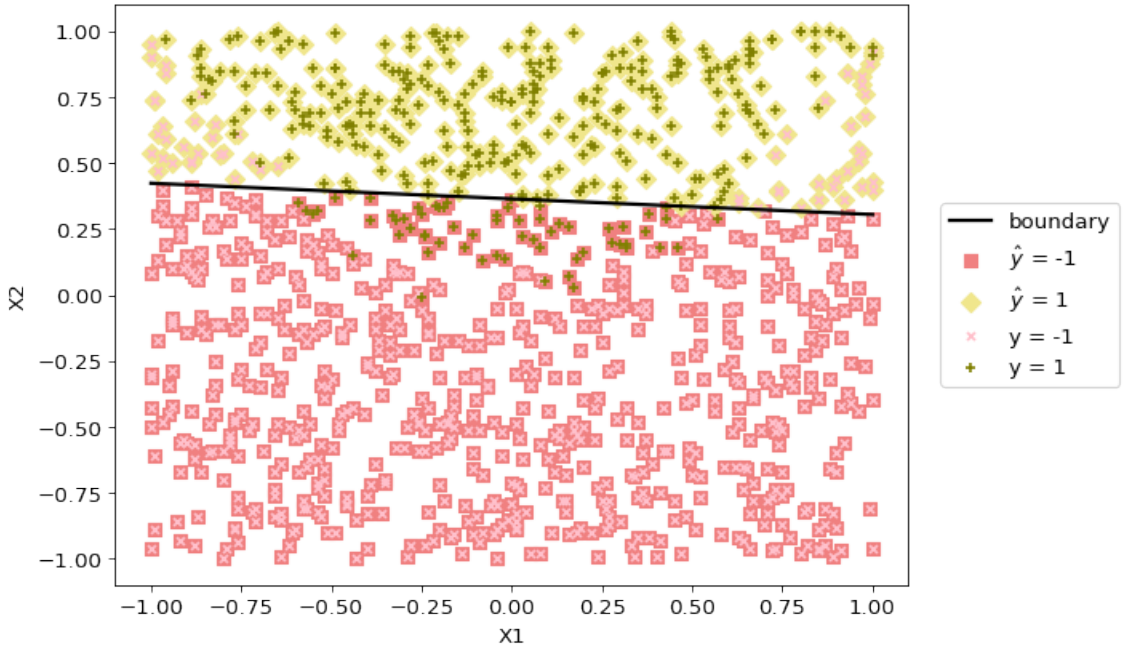

Figure 2: Data points plotted on X_1 and X_2, differentiated by y, alongside the predicted value $\hat{y}$ for each data point.

iv. The model seems to accurately predict most of the target values. However, given that the negative data points are not evenly distributed across the X_1 range and instead peak around $X_1 = 0$ and thin out as $|X|$ approaches 1, any linear model can only ever get so close to predicting the correct target values.

(b) SVM

i. SVM uses a hinge loss function which comes with a regularisation penalty. The model minimises the cost function $J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[max(0, 1 - y^{(i)}\theta^T x^{(i)}\right] + \frac{\theta^T\theta}{C}$
Table 1 shows the intercept and coefficients for the three various settings of C. Just like in (a), i is the intercept and $[b_1, b_2]$ are the coefficients.

	i	b_1	b_2
C = 0.001	-0.23	0.03	0.48
C = 1	-0.73	0.11	2.01
C = 100	-0.74	0.11	2.04

Table 1. *Resulting parameters from SVM classifer for three settings of hyperparameter C*

ii. Figures 3, 4, and 5 shows the predicted target values and actual target values plotted in the same format as that of 2.

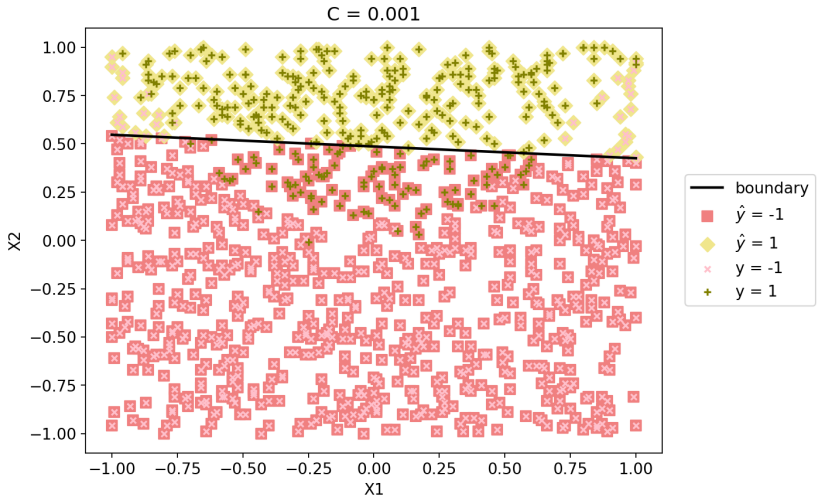

Figure 3: SVM prediction results for C = 0.001

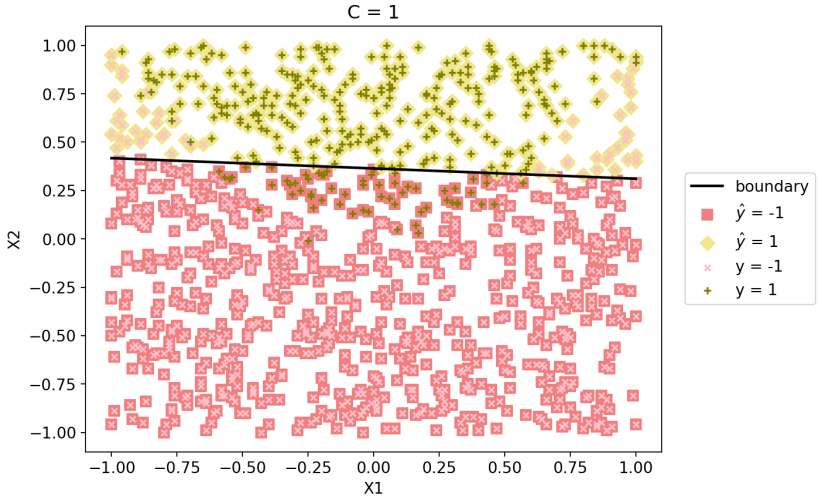

Figure 4: SVM prediction results for C = 1

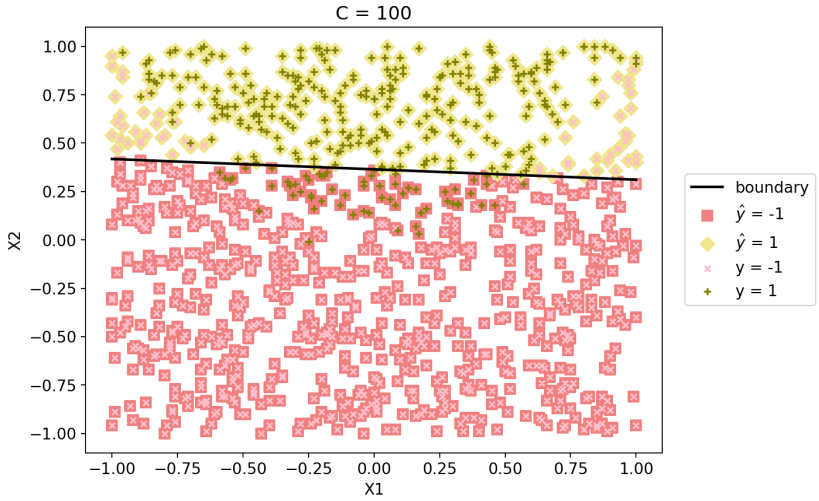

Figure 5: SVM prediction results for C = 100

iii. Tables 1 and 2 both show that there is very little variation betwen the model trained with $C = 1$ or $C = 100$. For $C = 0.001$ there is a slight visual difference which stems from both coefficients being set to very small values. This is because C is inversely proportional to the strength of regularisation. Higher regularisation means that the model attempts to be more generalisable in order to avoid overfitting. In practice, this results in smaller parameter values for smaller C. Table 2 shows that the F1 score and accuracy of SVM with C = 0.001 is lower than that of higher C. However, this result is based on testing on the same set that was used to train the model. A higher C will lead in less misclassification in the training set, but it is entirely possible that, since it is less generalisable, it could perform worse on a new test set. Indeed the -1 data points do not appear to be purposely skewed towards the right. Thus, the model with C = 0.001 which has a flatter line could potentially outperform the models with higher C, which end up having a slanted decision boundary that is designed to better fit this specfic set of random points.

iv. Table 2 shows the F1 score and accuracy of the various classifiers, including the three settings of the SVM, and the LR. It can be seen that for very small C, predictions are worse than for LR. When C increases, predictions improve but only up to C = 1. There is no difference between the SVM with C = 1 and C = 100. Moreover, there is no difference between these two and the LR.

		F1	accuracy
baseline		0.5410	0.6727
LR	Linear	0.8817	0.8819
	Quadratic	0.9709	0.9709
SVM	C = 0.001	0.8615	0.8659
	C = 1	0.8817	0.8819
	C = 100	0.8817	0.8819

Table 2. *Performance measurements of various classifiers*

(c) Logistic Regression - Quadratic

i. The formula for the decision boundary with the square of each feature becomes $y = sign(i + b_1X_1 + b_2X_2 + b_3X_1^2 + b_4X_2^2$ where i is the intercept and $[b_1, b_2, b_3, b_4]$ are the coefficients. Training the model, printing the parameters and plugging in the values results in $y = -1.49 + 0.26X_1 + 29.49X_2 - 26.26X_1^2 - 1.10X_2^2$. As hypothesised at (a) (iv), X_1^2 is very useful in predicting the target value, shown by the high coefficient $b_3 = -26.26$

ii. Figure 6 shows the result of the LR classifier using the square of each feature in addition to the two features. It manages to correctly classify almost every data point because the shape of the decision boundary is a much better fit for the distribution of -1 points than the linear boundaries generated at (a) and (b). Table 2 shows the performance measures of the LR with a quadratic decision boundary in contrast to the ones of the linear models. It signifcantly outperforms every other moedel, achieving F1 and accuracy scores of 0.97. This shows how important it is to design a model that correctly captures the distribution of the data.

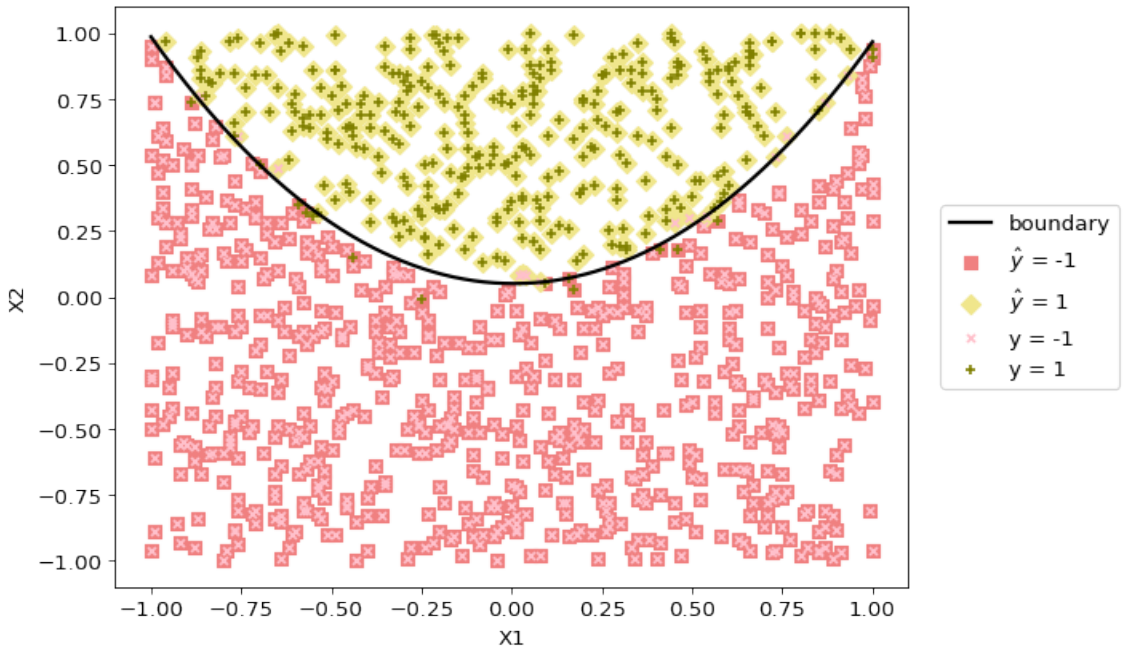

Figure 6: LR prediction with a quadratic decision boundary

iii. Table 2 shows the performance measures of the baseline classifier which always predicts $\hat{y} = -1$ in contrast to the other classifiers. While the accuracy score seems pretty high, at 67%, this is just a result of around 67% of points being -1. The F1 score however, which includes measures of precsion and recall, is only 0.54. This is much smaler than even the linear models, and it is just over half the F1 score of the best performing classifier, the LR with a quadratic decision boundary.

iv. Using the formula from (c) (i), the decision boundary represents the boundary of $i + b_1X_1 + b_2X_2 + b_3X_1^2 + b_4X_2^2 = 0$. Solving the quadratic equation:

$$b_4X_2^2 + b_2X_2 + i + b_1X_1 + b_3X_1^2 = 0$$

$$X_2 = \frac{-b_2 + \sqrt{b_2^2 - 4b_4(i + X_1(b_1 + b_3X_1))}}{2b_4}$$

Appendix - Code

```
#id:8--8-8
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
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error,
     accuracy_score, f1_score,recall_score, precision_score
import matplotlib.colors as mcolors
from matplotlib.pyplot import figure
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
import matplotlib.colors as mcolors
from matplotlib.pyplot import figure
import matplotlib.colors as mcolors
from matplotlib.pyplot import figure

#reading and plotting data
df = pd.read_csv("MLAss1Dataset.csv",header=None)
X1=df.iloc[:,0]
X2=df.iloc[:,1]
X=np.column_stack((X1,X2))
y=df.iloc[:,2]
plt.rc('font',size=12)
plt.scatter(df.loc[df.loc[:,2]==-1,0],df.loc[df.loc[:,2]==-1,1],marker='x',color='pink')
plt.scatter(df.loc[df.loc[:,2]==1,0],df.loc[df.loc[:,2]==1,1],marker='+',color='olive')
plt.xlabel("X1")
plt.ylabel("X2")
plt.legend(["y=-1","y=1"])
plt.show()

#Logistic regression - linear boundary
#training model
lr = LogisticRegression(penalty = 'none', solver='lbfgs')
lr.fit(X,y)
print('Logistic Regression')
#parameters
print('Coefficients: ',lr.coef_.T)
print('Intercept: ',lr.intercept_[0])
#performance measures
y_pred = lr.predict(X)
print('F-measure', f1_score(y, y_pred, average='weighted'))
print('Accuracy', accuracy_score(y, y_pred))
#boundary function
b1, b2 = lr.coef_.T
b = -lr.intercept_[0]/b2
m = -b1/b2
xd = np.array([-1,1])
yd = m*xd+b
#plotting predictions, actual values, boundary
plt.rc('font',size=12)
figure(figsize=(8, 6), dpi=80)
plt.scatter(X1[y_pred==-1],X2[y_pred==-1],marker='s',color='lightcoral',s=60)
plt.scatter(X1[y_pred==1],X2[y_pred==1],marker='D',color='khaki',s=60)
plt.scatter(X1[y==-1],X2[y==-1],marker='x',color='pink',s=20)
plt.scatter(X1[y==1],X2[y==1],marker='+',color='olive',s=30)
```

```
plt.plot(xd,yd,'k', lw=2, ls='-')
plt.xlabel("X1")
plt.ylabel("X2")
plt.legend(["boundary","$\^y$ = -1","$\^y$ = 1","y = -1","y = 1"],bbox_to_anchor=(1.04, 0.5),
            loc="center left", borderaxespad=0)
plt.show()

#SVM
f = plt.figure()
plt.rc('font',size=12)
f, axes = plt.subplots(ncols=3,figsize=(27,6),dpi=160)
print('SVM')
for i,c in enumerate([0.001, 1, 100]):
  lsvc = LinearSVC(C=c,max_iter=10000).fit(X,y)
  print('C = ',c)
  #parameters
  print('Coefficients: ',lsvc.coef_.T)
  print('Intercept: ',lsvc.intercept_[0])
  #performance measures
  y_pred = lsvc.predict(X)
  print('F-measure', f1_score(y, y_pred, average='weighted'))
  print('Accuracy', accuracy_score(y, y_pred))
  #boundary function
  b1, b2 = lsvc.coef_.T
  b = -lsvc.intercept_[0]/b2
  m = -b1/b2
  xd = np.array([-1,1])
  yd = m*xd+b
  #subplot for each set of predictions, actual values, and boundary
  axes[i].set_title("C = "+repr(c))
  axes[i].scatter(X1[y_pred==-1],X2[y_pred==-1],marker='s',color='lightcoral',s=60)
  axes[i].scatter(X1[y_pred==1],X2[y_pred==1],marker='D',color='khaki',s=60)
  axes[i].scatter(X1[y==-1],X2[y==-1],marker='x',color='pink',s=20)
  axes[i].scatter(X1[y==1],X2[y==1],marker='+',color='olive',s=30)
  axes[i].plot(xd,yd,'k', lw=2, ls='-')
  plt.setp(axes[i],xlabel="X1")
  plt.setp(axes[i],ylabel="X2")
plt.legend(["boundary","$\^y$ = -1","$\^y$ = 1","y = -1","y = 1"],bbox_to_anchor=(1.04, 0.5),
            loc="center left", borderaxespad=0)
plt.show()

#Logistic Regression - quadratic boundary
X = df.loc[:,[0,1]]
Xq =pd.concat([X, X**2], axis=1)
lr2 = LogisticRegression(penalty = 'none', solver='lbfgs')
lr2.fit(Xq,y)
print('Logistic Regression 2')
#parameters
print('Coefficients: ',lr2.coef_.T)
print('Intercept: ',lr2.intercept_[0])
#performance measures
y_pred = lr2.predict(Xq)
print('F-measure', f1_score(y, y_pred, average='weighted'))
print('Accuracy', accuracy_score(y, y_pred))
#boundary function
b1, b2, b3, b4 = lr2.coef_.T
i = lr2.intercept_[0]
xd = np.linspace(-1, 1, 1000)
```

```
yd = (-b2+np.sqrt(b2**2-4*b4*(i+xd*(b1+b3*xd))))/(2*b4)
plt.rc('font',size=12)
figure(figsize=(8, 6), dpi=80)
#plotting predictions, actual values, boundary
plt.scatter(X1[y_pred==-1],X2[y_pred==-1],marker='s',color='lightcoral',s=60)
plt.scatter(X1[y_pred==1],X2[y_pred==1],marker='D',color='khaki',s=60)
plt.scatter(X1[y==-1],X2[y==-1],marker='x',color='pink',s=20)
plt.scatter(X1[y==1],X2[y==1],marker='+',color='olive',s=30)
plt.plot(xd,yd,'k', lw=2, ls='-')
plt.xlabel("X1")
plt.ylabel("X2")
plt.legend(["boundary","$\^y$ = -1","$\^y$ = 1","y = -1","y = 1"],bbox_to_anchor=(1.04, 0.5),
            loc="center left", borderaxespad=0)
plt.show()

#Baseline model performance
y_pred = [-1]*len(X)
print('Baseline Model')
print('F-measure', f1_score(y, y_pred, average='weighted'))
print('Accuracy', accuracy_score(y, y_pred))
```