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

CSU44061 Machine Learning
Lab 1

Adriana Hrabowych, 19304296

October 10, 2023

1 Part A: Logistic Regression Classifier

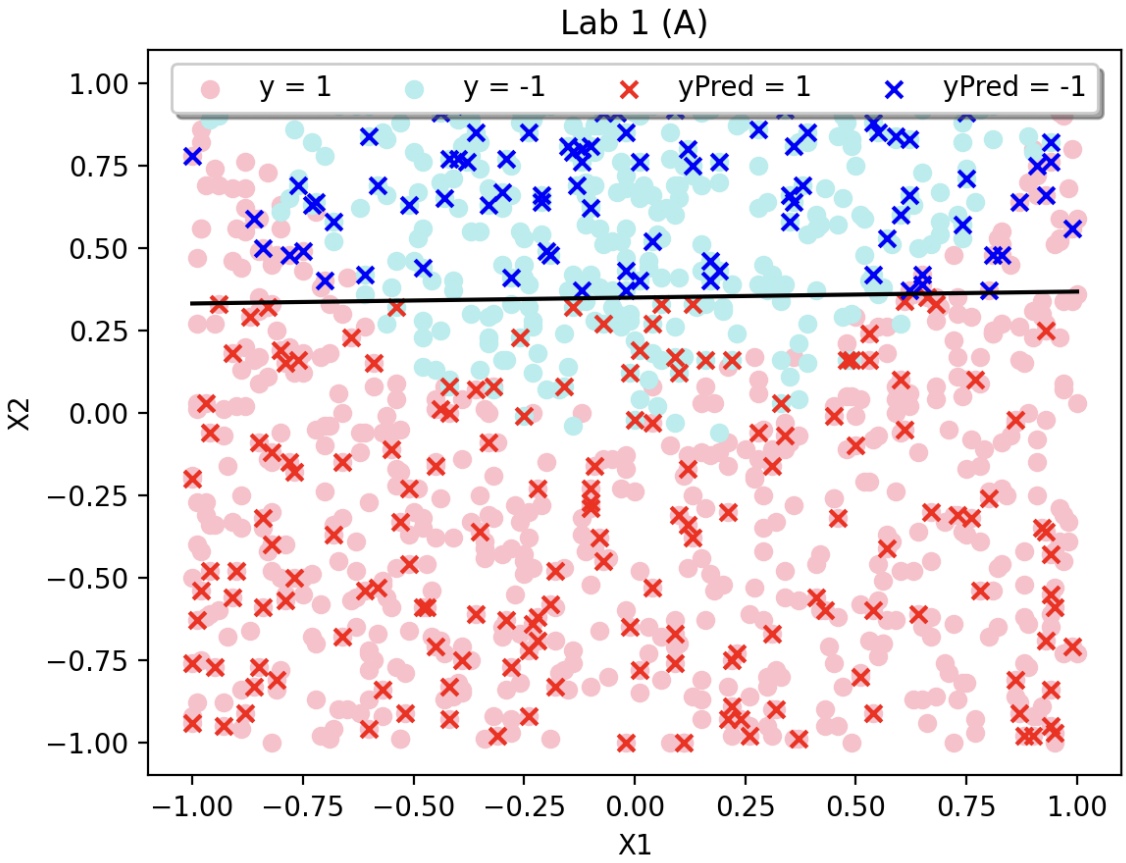

Figure 1: A scatter plot graph visualizing part A of the lab. Dataset used is id:13-13–13

1.1 A(i)

In A(i) the raw data is read in and then visualized on the graph seen in figure one using circular markers. Each data point is placed depending on the value of its two features, with the X-axis corresponding to X1

and the Y-axis corresponding to X2. The color of the marker depends on the target value y, if the marker is red then y=1 and if blue then y=-1.

1.2 A(ii)

The data is then randomly split with 25% being used for testing our models and 75% to be used for training our models. The ratio 25-75 was chosen as it is the default for the sklearn method and is typical for test-train splits.
A logistic regression model is then trained using the training data. The default sklearn parameters are used to define this model as they follow the usual standard for the parameters. The equation the model uses to predict the y values is $\text{sign}(\theta^T x)$, where $\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2$. The x's represent the two input features X1 and X2 while the thetas, bar theta 0 which is the intercept, are the parameters the model defines once trained.
Feature X2 has the greater influence on the prediction as when looking at the absolute values of the models coefficients, X2's is larger meaning it skews the prediction more. Each time the code is run the values change slightly as the data is split differently, but in one example the coefficients were:

$$\theta_1 = 0.1148 \qquad \theta_2 = -4.5772 \qquad intercept = 1.5772$$

From this we can see that X2's coefficient has much more influence and skews the results towards being negative and X1's coefficient, while small, skews the data positively.

1.3 A(iii)

The model is then given the test features and outputs the predicted y values. These predicted y's are then plotted as seen in figure one using x markers. The color of these markers have the same meaning as the raw data colors, just brighter for easier legibility. In addition a decision boundary is added to the graph in black. The decision boundary was calculated using the intercept and the feature coefficients. This information was fitted to a linear equation ax + (intercept / θ_2) = 0. The intercept was divided by θ_2 in order to translate the intercept from being for the model itself to reflecting the decision boundaries intercept. The slope of the line, a, was calculated with the equation $-\theta_1/\theta_2$.

1.4 A(iii)

The accuracy of the model is not perfect, the mean squared error of the model with these parameters equaled 0.528. As seen in figure one, the raw y values and the predicted y values differ quite dramatically. Though the general area of the red x's and blue x's match where the corresponding circles are the shape of the division between the two differs. Whereas the raw data has a curved division, the predictions have a straight line division.

2 Part B: Linear SVM Classifier

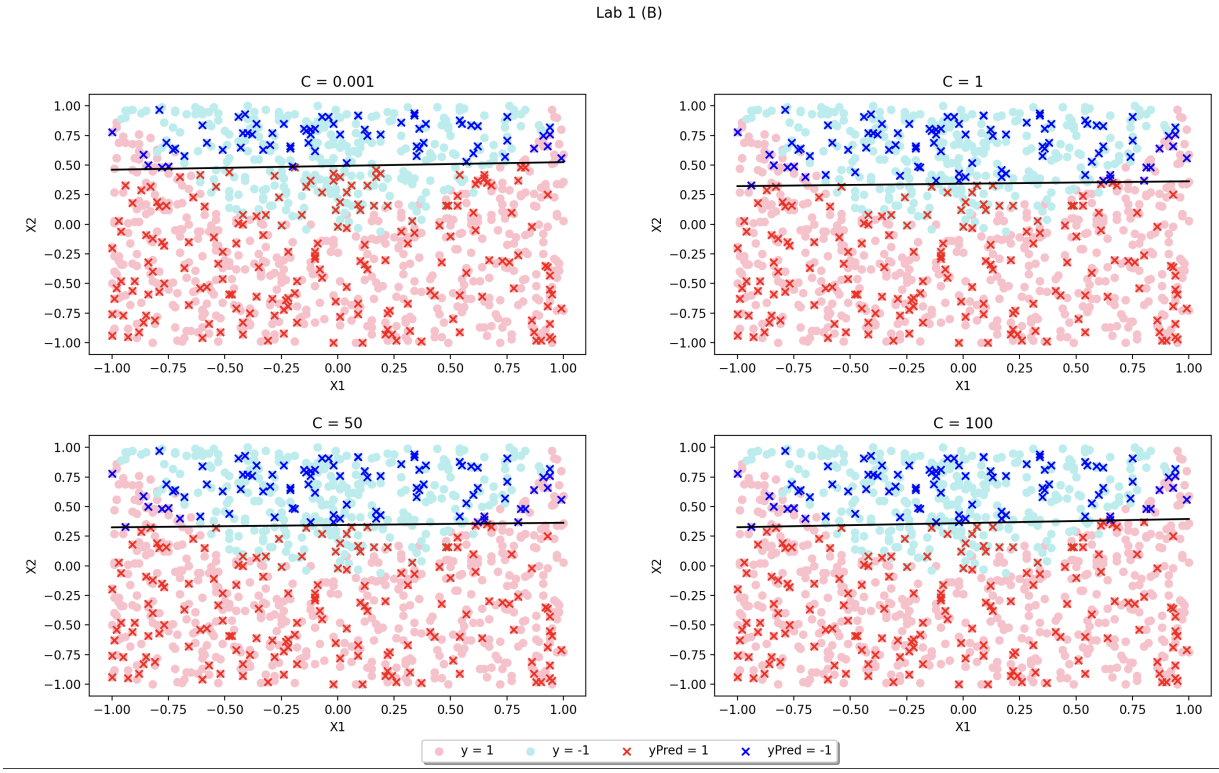

Figure 2: A figure containing 4 scatter plots visualizing part B of the lab. Dataset used is id:13-13–13

2.1 B(i)

In part B(i), four SVM Classifiers were trained on the same test and training data as each other and the model in the previous part. The equation these models uses to predict the y values is sign($\theta^T x$), where $\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2$. The x's represent the two input features X1 and X2 while the $\theta_1 and \theta_2$ are the parameters the model defines once trained. θ_0 represents the model intercept.
The only difference between the four models is that each one has a different value for C, either 0.001, 1, 50, and 100. One run of the code gave the following coefficients and intercepts:

C= 0.001: $\theta_1 = 0.0084$ $\theta_2 = -0.3906$ $intercept = 0.1806$

C= 1: $\theta_1 = 0.0478$ $\theta_2 = -1.7132$ $intercept = 0.5753$

C= 50: $\theta_1 = 0.0394$ $\theta_2 = -1.7238$ $intercept = 0.5898$

C= 100: $\theta_1 = 0.0638$ $\theta_2 = -1.6036$ $intercept = 0.5729$

2.2 B(ii)

Each model is plotted on its own graph for legibility. Each graph has the same raw data, plotted in the same way as in part A. The predicted Y values are then again, for each model and graph, plotted using red and blue x markers. Red signifies Y = 1 and blue signifies Y = -1. A decision boundary is also plotted for each model using the same equation as in part A.

2.3 B(iii)

In SVM models, C defines the priority of minimizing errors during training. A low C value allows a larger margin while a high C value has a lower tolerance for training error. The trend for the model coefficients

is that the larger C becomes, the larger θ_1, θ_2, and the intercept become. This results in differences in the accuracy of predictions, with these coefficients and parameters the mean squared error of each model is given as C=0.001 has an accuracy of 0.64, C=1 has one of 0.528, C=50 has 0.528, and C=100 has 0.544.
From these accuracies it is clear the importance of choosing a good C value for a model, having a C too small and having a C too big can ruin the accuracy of a model. Finding the C that balances the margins and tolerance for training errors is an important step in defining your model. For this example a C between 1 and 50 provided the best accuracy.

2.4 B(iv)

The most noticeable difference between the SVM Classifiers and the Logistic Regression Classifier is the size of the coefficients and intercepts. The intercept for our Logistic Regression Classifier was 1.5772 while the SVM models has intercepts ranging between 0.1 and 0.6. The former is nearly three times the size of the latter, which is a large gap. The coefficients for our Logistic Regression Classifier were 0.1148 and -4.5772 for X1 and X2 respectively. While the SVM models had coefficients for X1 being ranging between 0.008 and 0.04 and for X2 being between -0.3 and -1.7. Once again the Logistic Regression model had larger numbers than the SVM models.
Though these differences did not result in a large change in accuracy, the Logistic Regression model has an mean squared error value of 0.528 which is the same as the SVM models with C=1 and C=50. The other two models are not that far away either with error of C=0.001 being 0.64 and C=100 being 0.544.

3 Part C: Logistic Regressor with Four Features

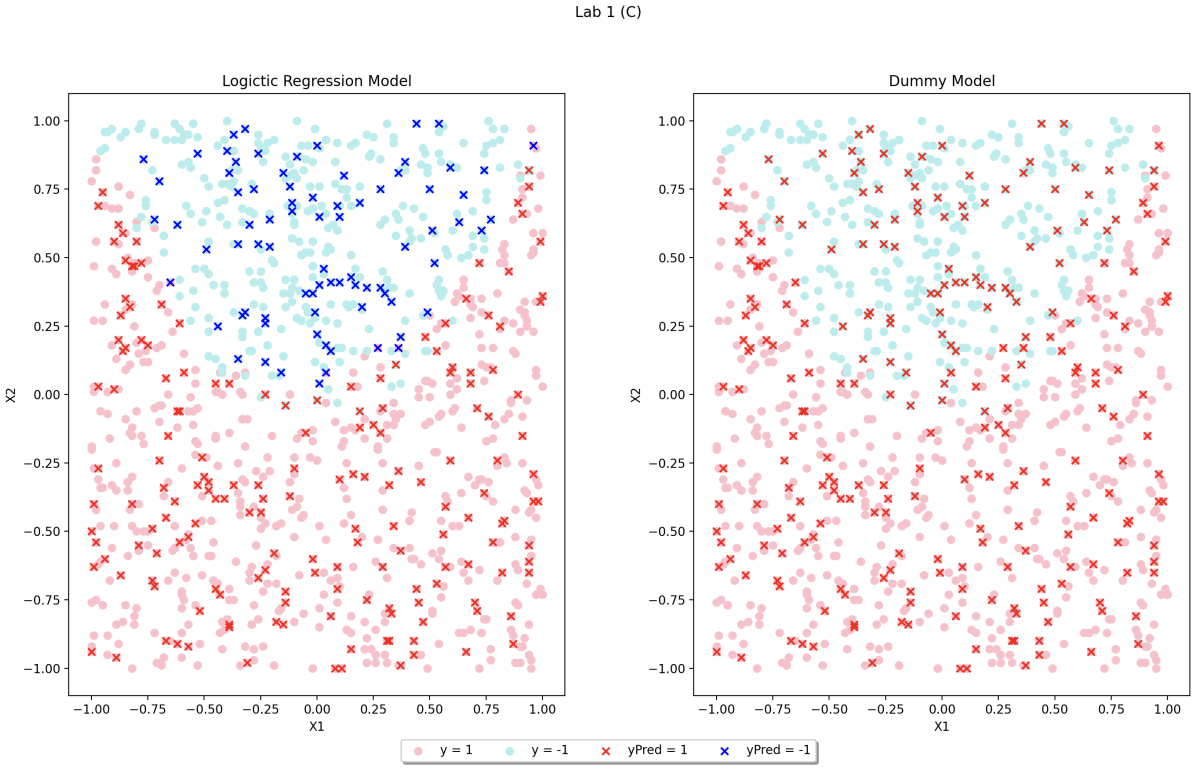

Figure 3: A figure containing 2 scatter plots visualizing part C of the lab. Dataset used is id:13-13–13

3.1 C(i)

We create the two additional features by squaring both X1 and X2 and giving them to the model in addition to the original X1 and X2. The equation this model now uses to predict the y values is $sign(\theta^T x)$, where

$\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2$. θ_0 represents the intercept, and the other thetas are the coefficients for their corresponding X's.
In one run of the code the coefficients for the data were $\theta_0 = 0.1961, \theta_1 = 0.1584, \theta_2 = -5.9691, \theta_3 = 6.163$,and $\theta_4 = -0.7811$.

3.2 C(ii)

In figure 3, the graph on the left is a visualization of this 4-feature Logistic Regression model. Each data point is placed depending on the value of the first two features, with the X-axis corresponding to X1 and the Y-axis corresponding to X2. The color of the marker depends on the target value y, if the marker is red then y=1 and if blue then y=-1.
The predictions made by this model are much more accurate than the ones in part A and B, the boundary between the red and blue predicted value are now a curve like in the raw data. This is due to the face that the two additional features added changed the decision boundary from a linear equation to a quadratic equation. The mean squared error given by this model was 0.096, which is incredibly low compared to the 2-feature Logistic Regression and SVM models.

3.3 C(iii)

In figure 3, the graph on the left is a visualization of a 4-feature 'Dummy' model. Each data point is placed depending on the value of the first two features, with the X-axis corresponding to X1 and the Y-axis corresponding to X2. The color of the marker depends on the target value y, if the marker is red then y=1 and if blue then y=-1.
This dummy model always predicts whatever the most common y-value was during training, which in this case was y=1 so every predicted mark is red. The mean squared error for this baseline is 1.472, which is much worse than the four-feature Logistic Regression model and every other model we've had in this lab. This is too be expected however as the model is very simplistic.

4 Appendix

```
# id:13-13--13

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.calibration import LinearSVC
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")

#A(i)
##Read in data
df = pd.read_csv("week2.csv")
X1=df.iloc[:,0]
X2=df.iloc[:,1]
X=np.column_stack((X1,X2))
y=df.iloc[:,2]

##Graph points
plt.scatter(X1[y == 1], X2[y == 1], color='pink', marker='o')
plt.scatter(X1[y == -1], X2[y == -1], color='paleturquoise', marker='o')

##Specifiy Graoph Details
plt.title("Lab 1 (A)")
plt.xlabel("X1")
plt.ylabel("X2")

#A(ii)
```

```
32
33 ##Train Log Model
34 X_train, X_test, y_train, y_test = train_test_split(X, y)
35 logreg = LogisticRegression()
36 logreg.fit(X_train, y_train)
37 ypred = logreg.predict(X_test)
38
39 ##Graph Predictions
40 plt.scatter(X_test[ypred == 1,0], X_test[ypred == 1,1], color='r', marker='x')
41 plt.scatter(X_test[ypred == -1,0], X_test[ypred == -1,1], color='b', marker='x')
42
43 #A(iii)
44
45 ##Create and plot decision boundary
46 cf = logreg.coef_[0]
47 print("Coefficiants for X1 and X2 of the Logistic Regression model: ")
48 print(cf[0])
49 print(cf[1])
50 a = -cf[0] / cf[1]
51 xx = np.linspace(-1, 1)
52 yy = a * xx - (logreg.intercept_[0]) / cf[1]
53 plt.plot(xx, yy, 'k-')
54 print("Intercept for the Logistic Regression model: ")
55 print(logreg.intercept_[0])
56
57 ##Add Legend and Plot
58 plt.legend(["y = 1", "y = -1", "yPred = 1", "yPred = -1"], ncol=4, loc='upper center',
       fancybox=True, shadow=True)
59 print("Simple Logictic Regressor Mean Squared Error: ")
60 print(mean_squared_error(y_test, ypred))
61 plt.show()
62
63
64 #B(i)
65 ##Create Models, Train, and predict
66 svc1 = LinearSVC(C=0.001)
67 svc2 = LinearSVC(C=1)
68 svc3 = LinearSVC(C=50)
69 svc4 = LinearSVC(C=100)
70
71 svc1.fit(X_train, y_train)
72 svc2.fit(X_train, y_train)
73 svc3.fit(X_train, y_train)
74 svc4.fit(X_train, y_train)
75
76 ypred1 = svc1.predict(X_test)
77 ypred2 = svc2.predict(X_test)
78 ypred3 = svc3.predict(X_test)
79 ypred4 = svc4.predict(X_test)
80
81 #B(ii)
82 ##Setup 4 graphs
83 fig, ax = plt.subplots(2, 2)
84 fig.suptitle("Lab 1 (B)")
85 fig.tight_layout(pad=1.5)
86
87 ##Graph data and add labels
88 ax[0, 0].scatter(X1[y == 1], X2[y == 1], color='pink', marker='o')
89 ax[0, 0].scatter(X1[y == -1], X2[y == -1], color='paleturquoise', marker='o')
90 ax[0, 0].set_title("C = 0.001")
91 ax[0, 0].set(xlabel='X1', ylabel='X2')
92
93 ax[0, 1].scatter(X1[y == 1], X2[y == 1], color='pink', marker='o')
94 ax[0, 1].scatter(X1[y == -1], X2[y == -1], color='paleturquoise', marker='o')
95 ax[0, 1].set_title("C = 1")
96 ax[0, 1].set(xlabel='X1', ylabel='X2')
97
98 ax[1, 0].scatter(X1[y == 1], X2[y == 1], color='pink', marker='o')
99 ax[1, 0].scatter(X1[y == -1], X2[y == -1], color='paleturquoise', marker='o')
```

```
ax[1, 0].set_title("C = 50")
ax[1, 0].set(xlabel='X1', ylabel='X2')

ax[1, 1].scatter(X1[y == 1], X2[y == 1], color='pink', marker='o')
ax[1, 1].scatter(X1[y == -1], X2[y == -1], color='paleturquoise', marker='o')
ax[1, 1].set_title("C = 100")
ax[1, 1].set(xlabel='X1', ylabel='X2')

##Add Predictions
ax[0, 0].scatter(X_test[ypred1 == 1,0], X_test[ypred1 == 1,1], color='r', marker='x')
ax[0, 0].scatter(X_test[ypred1 == -1,0], X_test[ypred1 == -1,1], color='b', marker='x')


ax[0, 1].scatter(X_test[ypred2 == 1,0], X_test[ypred2 == 1,1], color='r', marker='x')
ax[0, 1].scatter(X_test[ypred2 == -1,0], X_test[ypred2 == -1,1], color='b', marker='x')

ax[1, 0].scatter(X_test[ypred3 == 1,0], X_test[ypred3 == 1,1], color='r', marker='x')
ax[1, 0].scatter(X_test[ypred3 == -1,0], X_test[ypred3 == -1,1], color='b', marker='x')

ax[1, 1].scatter(X_test[ypred4 == 1,0], X_test[ypred4 == 1,1], color='r', marker='x')
ax[1, 1].scatter(X_test[ypred4 == -1,0], X_test[ypred4 == -1,1], color='b', marker='x')

##Add Decision Boundaries

cf = svc1.coef_[0]
a = -cf[0] / cf[1]
xx = np.linspace(-1, 1)
yy = a * xx - (svc1.intercept_[0]) / cf[1]
ax[0, 0].plot(xx, yy, 'k-')
print("Coefficiants for X1 and X2 of the C = 0.001: ")
print(cf[0])
print(cf[1])
print("Intercept for C = 0.001: ")
print(svc1.intercept_[0])

cf = svc2.coef_[0]
a = -cf[0] / cf[1]
xx = np.linspace(-1, 1)
yy = a * xx - (svc2.intercept_[0]) / cf[1]
ax[0, 1].plot(xx, yy, 'k-')
print("Coefficiants for X1 and X2 of the C = 1: ")
print(cf[0])
print(cf[1])
print("Intercept for C = 1: ")
print(svc2.intercept_[0])

cf = svc3.coef_[0]
a = -cf[0] / cf[1]
xx = np.linspace(-1, 1)
yy = a * xx - (svc3.intercept_[0]) / cf[1]
ax[1, 0].plot(xx, yy, 'k-')
print("Coefficiants for X1 and X2 of the C = 50: ")
print(cf[0])
print(cf[1])
print("Intercept for C = 50: ")
print(svc3.intercept_[0])

cf = svc4.coef_[0]
a = -cf[0] / cf[1]
xx = np.linspace(-1, 1)
yy = a * xx - (svc4.intercept_[0]) / cf[1]
ax[1, 1].plot(xx, yy, 'k-')
print("Coefficiants for X1 and X2 of the C = 100: ")
print(cf[0])
print(cf[1])
print("Intercept for C = 100: ")
print(svc4.intercept_[0])

```

```
fig.legend(["y = 1", "y = -1", "yPred = 1", "yPred = -1"], ncol=4, loc='lower center',
    fancybox=True, shadow=True)

print("SVC (C = 0.001) Mean Squared Error: ")
print(mean_squared_error(y_test, ypred1))
print("SVC (C = 1) Mean Squared Error: ")
print(mean_squared_error(y_test, ypred2))
print("SVC (C = 50) Mean Squared Error: ")
print(mean_squared_error(y_test, ypred3))
print("SVC (C = 100) Mean Squared Error: ")
print(mean_squared_error(y_test, ypred4))

plt.show()

#C(i)
##Create Graphs
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle("Lab 1 (C)")
fig.tight_layout(pad=1.5)

##Graph points
ax1.scatter(X1[y == 1], X2[y == 1], color='pink', marker='o')
ax1.scatter(X1[y == -1], X2[y == -1], color='paleturquoise', marker='o')
ax1.set_title("Logictic Regression Model")
ax1.set(xlabel='X1', ylabel='X2')

ax2.scatter(X1[y == 1], X2[y == 1], color='pink', marker='o')
ax2.scatter(X1[y == -1], X2[y == -1], color='paleturquoise', marker='o')
ax2.set_title("Dummy Model")
ax2.set(xlabel='X1', ylabel='X2')

##Add squares of features to X's
X1X = [x ** 2 for x in X1]
X2X = [x ** 2 for x in X2]
Xs=np.column_stack((X1,X2,X1X,X2X))

##Train Log Model
X_train, X_test, y_train, y_test = train_test_split(Xs, y)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
ypred = logreg.predict(X_test)

#C(ii)
##Graph Predictions
ax1.scatter(X_test[ypred == 1,0], X_test[ypred == 1,1], color='r', marker='x')
ax1.scatter(X_test[ypred == -1,0], X_test[ypred == -1,1], color='b', marker='x')

#C(iii)
dclf = DummyClassifier(strategy = 'most_frequent')
dclf.fit(X_train, y_train)
ydummy = dclf.predict(X_test)

ax2.scatter(X_test[ydummy == 1,0], X_test[ydummy == 1,1], color='r', marker='x')
ax2.scatter(X_test[ydummy == -1,0], X_test[ydummy == -1,1], color='b', marker='x')

fig.legend(["y = 1", "y = -1", "yPred = 1", "yPred = -1"], ncol=4, loc='lower center',
    fancybox=True, shadow=True)

cf = logreg.coef_[0]
print("Coefficiants for 4 feature Logistic Regression: ")
print(cf[0])
print(cf[1])
print(cf[2])
print(cf[3])
print("Intercept: ")
print(logreg.intercept_[0])

print("Logistic Regressor Mean Squared Error: ")
print(mean_squared_error(y_test, ypred))
```

```
235 print("Dummy Model Mean Squared Error: ")
236 print(mean_squared_error(y_test, ydummy))
237
238 plt.show()
```