Machine Learning Assignment Week 4

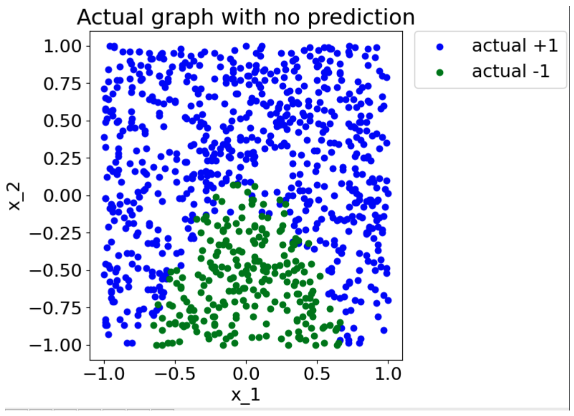
Name: Masanari Doi

Student number: 19313167

(i)

(a)

(i)



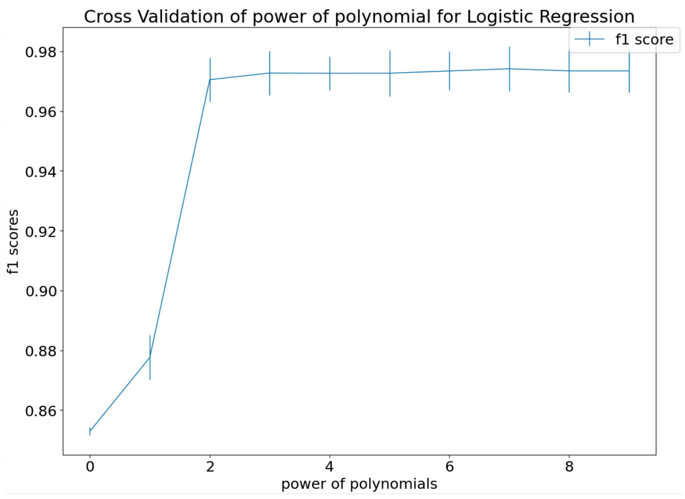
The above figure shows plots of dataset 1 without any predictions.

```
polyPowers = range(0, 10)
for polyPower in polyPowers:
        poly = PolynomialFeatures(polyPower)
        xPoly = poly.fit_transform(x)
        xPolyTrain = poly.fit_transform(xTrain)
        xPolyTest = poly.fit_transform(xTest)
        model = LogisticRegression(penalty = "l2").fit(xPolyTrain, yTrain)
        yPred = model.predict(xPolyTest)
        score = cross_val_score(model, xPoly, y, cv=5, scoring="f1")
```

model = LogisticRegression(penalty = “l2”).fit(xPolyTrain, yTrain) is used to train a model with polynomial features. Consequently, score = cross_val_score(model, xPoly, y, cv=5, scoring="f1") is used to get f1 scores using cross-validation = 5 (in this code, cv=5 represents so). In order to find the maximum order of polynomials to use, for polyPower in polyPowers is used to make a for-loop, with polyPowers = range(0,10).

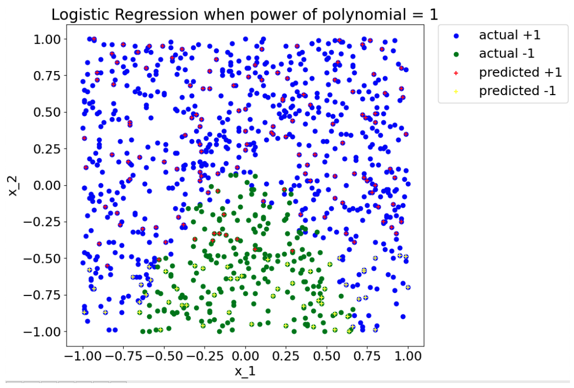
```
mean_error.append(np.array(score).mean())
std_error.append(np.array(score).std())
```

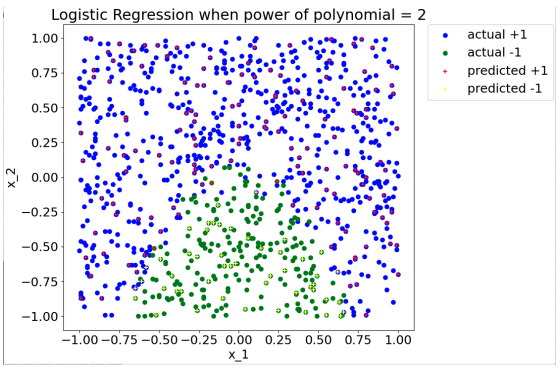
Following that, mean_error.append… is used to make mean of the f1 scores which is made in the first code, and then, std_error.append…. is used to make the standard deviation of the f1 scores.

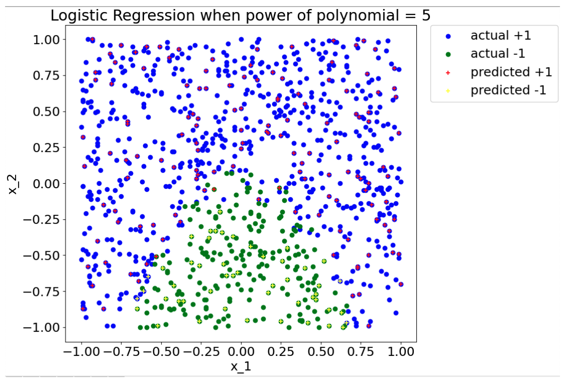


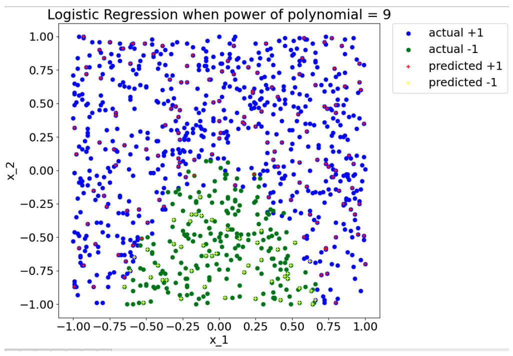
The graph above illustrates how f1 scores change in accordance with the power of polynomials. A linear line is connected to the mean of the score and the vertical line shows the standard deviation of the f1 scores. It can be seen that after the power of polynomials becomes 2, the f1 scores look almost the same. In addition, the code generated the output below which also shows that after 2, there is not much of a difference in f1 scores.

```
when power of polynomial =  1
f1 score =  [0.88       0.88571429 0.88275862 0.8641115  0.87544484]
when power of polynomial =  2
f1 score =  [0.96727273 0.97491039 0.98220641 0.96085409 0.96727273]
when power of polynomial =  3
f1 score =  [0.97101449 0.97857143 0.98220641 0.96085409 0.97101449]
when power of polynomial =  5
f1 score =  [0.96       0.98220641 0.9787234  0.97142857 0.97101449]
when power of polynomial =  8
f1 score =  [0.96376812 0.9858156  0.97526502 0.97142857 0.97101449]
when power of polynomial =  9
f1 score =  [0.96376812 0.9858156  0.97526502 0.97142857 0.97101449]
```









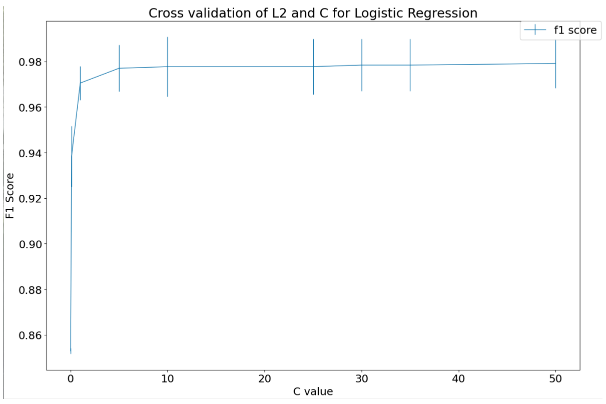
Moreover, the 4 graphs above show the predictions on the data with power of polynomials = 1, 2, 5, 9. These illustrate that from 1 to 2, the predictions become better but from 2 to 9, these predictions are not improved. Therefore, according to the graphs and the output from the console, it can be said that the maximum order of polynomials to use is 2.

(ii)

```
Ci_range = [0.01, 0.1, 1, 5, 10, 25, 30, 35, 50]
   for Ci in Ci_range:
       poly = PolynomialFeatures(2)
       model = LogisticRegression(penalty = "l2", C = Ci).fit(xPolyTrain, yTrain)
```

In order to decide the weight C given to the penalty in the cost function, it is almost the same way as selecting the maximum order of polynomials except for the codes above. poly =

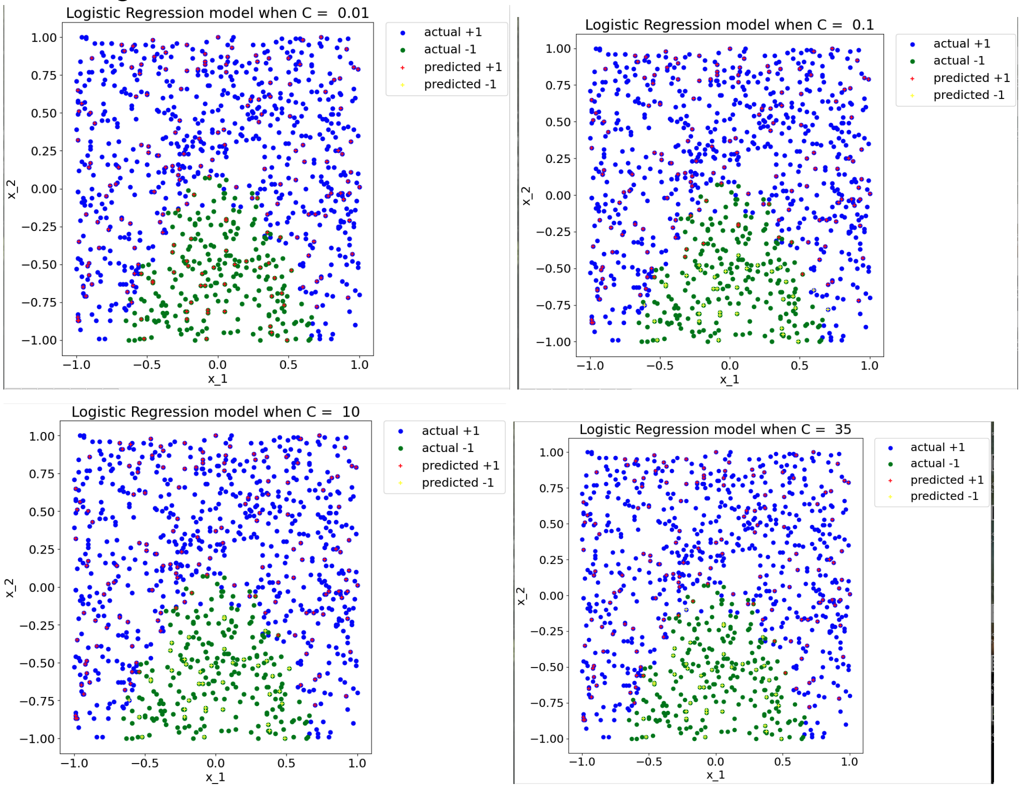
PolynomialFeatures(2) shows that for the training it is using the power of 2 obtained by With the range of Ci of [0.01, 0.1, 1, 5, 10, 25, 30, 35, 50], the figure below was generated.



It can clearly be seen that after C = 10, fi scores look the same. For the detailed f1 scores, the output below shows the scores with the C values.

```
when c = 0,01 and power of polynomial = 2
f1 score =  [0.85276074 0.85276074 0.85538462 0.85185185 0.85185185]
when c = 0,1 and power of polynomial = 2
f1 score =  [0.95       0.95804196 0.92783505 0.93197279 0.92361111]
when c = 1, and power of polynomial = 2
f1 score =  [0.96727273 0.97491039 0.98220641 0.96085409 0.96727273]
when c = 5, and power of polynomial = 2
f1 score =  [0.96028881 0.98932384 0.98220641 0.97122302 0.98194946]
when c = 10, and power of polynomial = 2
f1 score =  [0.96028881 0.98932384 0.99280576 0.96402878 0.98194946]
when c = 35, and power of polynomial = 2
f1 score =  [0.96402878 0.98932384 0.99280576 0.96750903 0.97826087]
```

There is a fact that when C = 10, the f1 scores look the highest and $C \geq 10$, almost all the f1 scores are the same. Therefore, C = 10 is chosen in this assignment. The figures below also show that from C = 0.01 to 10, the predictions are improving because too small C produces prediction error, and that $C \geq 10$, the huge difference can not be seen.

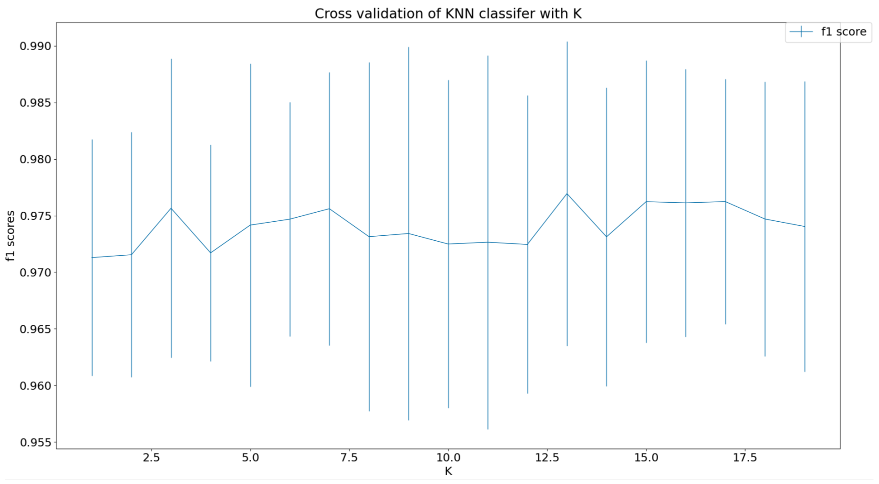


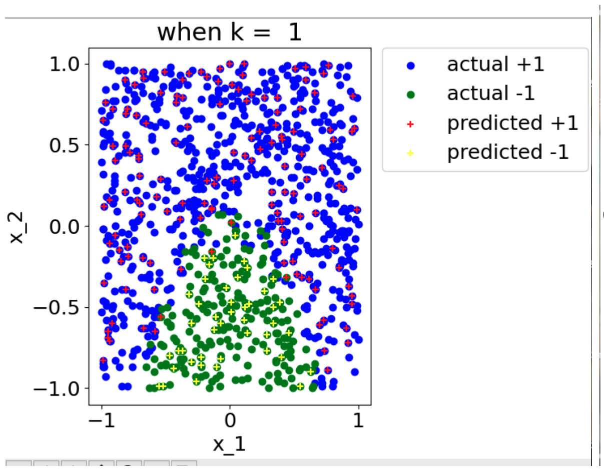
(b)

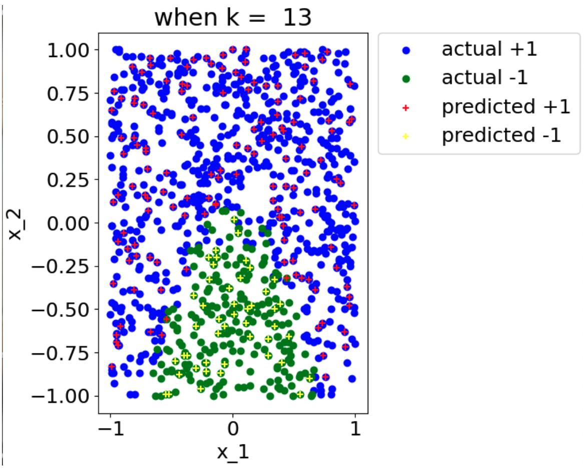
```
xTrain, xTest, yTrain, xTest1, xTest2, yTest = splitTrainAndTest(x, y)
  k_range = range(1, 20)
  mean_error=[]
  std_error=[]
  for k in k_range :
     # train a kNN classifier on the data
     model = KNeighborsClassifier(n_neighbors=k, weights='uniform').fit(xTrain, yTrain)
     # getting model score with using cross-validation = 5
     score = cross_val_score(model, x, y, cv=5, scoring="f1")

     print("when k = ", k)
     print("f1 score = ", score)
     print("mean = ", score.mean())

     # getting a prediction plot
     yPred = model.predict(xTest)
     getPredictionPlot(x1, x2, y, xTest1, xTest2, yPred, 0, 0, k)
     mean_error.append(np.array(score).mean())
     std_error.append(np.array(score).std())
```

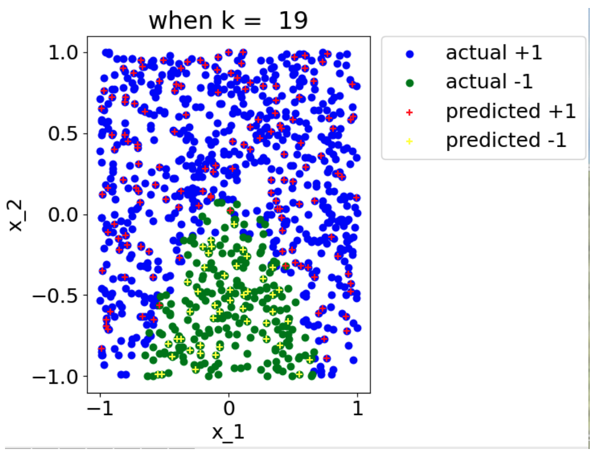
In order to select k, firstly data is split into 20% for training and 80% for testing in the function I made, called splitTrainAndTest(x,y). Secondly, with the k_range from 1 to 20, model = KNeighborsClassifier(n_neighbors=k, weights='uniform').fit(xTrain, yTrain) is used to make a model of KNeighborsClassifier as training data can be directly used to make predictions . In the for-loop, score = cross_val_score(model, x, y, cv=5, scoring="f1") is used to generate f1 score with cross-validation which should be used because too increased k would result in under-fitting and too small k can be the cause of over-fitting. The below shows the output of the f1 scores, the figure of Cross validation of KNN and some figures of predictions made by this classifier.

```
when k =  1
f1 score =  [0.96774194 0.96774194 0.98916968 0.95744681 0.97435897]
mean =  0.9712918657855214
when k =  2
f1 score =  [0.97058824 0.97122302 0.98540146 0.95272727 0.97777778]
mean =  0.9715435534471831
when k =  9
f1 score =  [0.95683453 0.98571429 0.99641577 0.95340502 0.97472924]
mean =  0.9734197696992217
when k =  12
f1 score =  [0.95620438 0.97841727 0.99280576 0.96028881 0.97454545]
mean =  0.9724523328708983
when k =  13
f1 score =  [0.96028881 0.98571429 0.99641577 0.96402878 0.97826087]
mean =  0.9769417023063
when k =  14
f1 score =  [0.96       0.97841727 0.99641577 0.96376812 0.96703297]
mean =  0.9731268239542731
```









As above output and figures, when k = 13, the f1 score is the highest, and there is no prediction error although k = 1 and k = 19 have a tiny prediction error. k = 13 does not seem to be under-fitting or over-fitting.

(c)

In order to calculate the confusion matrices, firstly, the data is split for training and testing. 80% of the data is used for training and the rest is for testing. Consequently, a model is made, and then the function confusion_matrix() is used to see confusion metrics. Take logistic Regression, for example, as the code below, logistic regression model is made and then confusion_matrix(yTest, yPred) is used.

```
# split the data for training and testing.
   xTrain, xTest, yTrain, xTest1, xTest2, yTest = splitTrainAndTest(x, y)

   # getting confusion matrix for Logistic Regression
   poly = PolynomialFeatures(p)
   xPoly = poly.fit_transform(x)
   xPolyTrain = poly.fit_transform(xTrain)
   xPolyTest = poly.fit_transform(xTest)
   model = LogisticRegression(penalty = "l2", C=c).fit(xPolyTrain, yTrain)
   yPred = model.predict(xPolyTest)
   print("confusion matrix for Logistic Regression with power of polynomial = " +
str(p))
   print(confusion_matrix(yTest, yPred))
   print(classification_report(yTest, yPred))
```

The output of confusion_matrix() should be as below.
TP is True positive. FN is False Negative. FP is False Positive. TN is True Negative.

[TP   FN]
[FP TN]

Accuracy = TN + TP / (TN + TP + FN + FP)
True Positive Rate = TP/ (TP + FN)
False Positive Rate = FP / (TP + FP)
Precision = TP / (TP + FP)
Output of confusion_metrix() for Logistic Regression, kNN classifier and baseline classifiers that always predict the most frequent class in the training data and one that makes random predictions as below.
Confusion matrix for Logistic Regression with power of polynomial = 2 (this 2 is determined in (i) in (a))and c = 25(this 25 is determined in (ii) in (a)).

```
[ 53   2]          Accuracy  = 131 + 53 / (131 + 53 + 2 + 1) = 0.98
[ 1 131]           True Positive Rate = 53/ (53 + 2) = 0.96
                   False Positive Rate = 1 / (53 + 1) = 0.02
                   Precision = 53 / (53 + 1) = 0.98
```

confusion matrix for KNN classifier when k = 13

```
[ 54   1]          Accuracy  = 0.98
[ 3 129]           True Positive Rate = 0.98
                   False Positive Rate = 0.05
                   Precision = 0.95
```

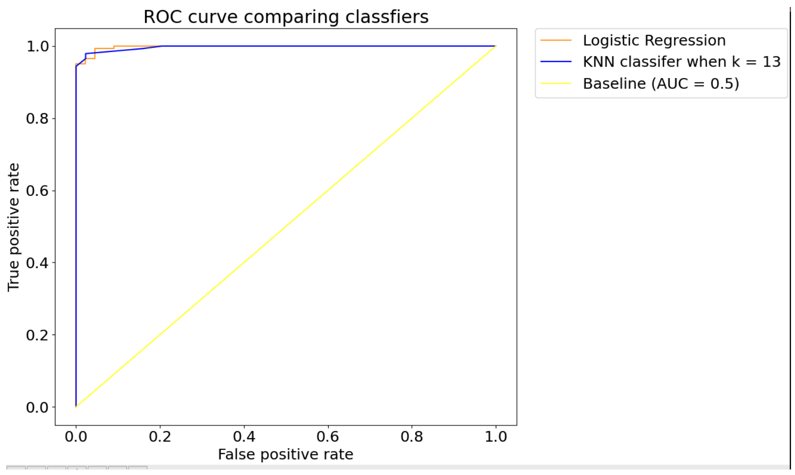
confusion matrix for baseline classifiers that makes random predictions

```
[ 29   26]         Accuracy  = 0.51
[ 65   67]         True Positive Rate = 0.53
                   False Positive Rate = 0.69
                   Precision = 0.31
```

(d)

In order to plot the ROC curves for trained Logistic Regression, kNN classifiers and baseline classifiers, firstly, the data is split for training and testing. 80% of the data is used for training and the rest is for testing. Consequently, a model is made, and then the function model.decision_function is used to generate confidence decision probability. Finally, for x-axis, false positive rate value is generated and for y-axis, true positive rate value is also generated and then they are plotted in a graph. Take Logistic Regression for example, model = LogisticRegression(penalty = "l2", C = c).fit(xPolyTrain, yTrain) is used to train, and then, fpr, tpr, _ = roc_curve(yTest ,model.decision_function(xPolyTest)) is used to make x-axis and y-axis. The ROC curve is below the code.

```
#  split the data for training and testing.
   xTrain, xTest, yTrain, xTest1, xTest2, yTest = splitTrainAndTest(x, y)

   #  getting ROC curve for logisticRegression
   poly = PolynomialFeatures(p)
   xPoly = poly.fit_transform(x)
   xPolyTrain = poly.fit_transform(xTrain)
   xPolyTest = poly.fit_transform(xTest)
   model = LogisticRegression(penalty = "l2", C = c).fit(xPolyTrain, yTrain)
   fpr, tpr, _ = roc_curve(yTest ,model.decision_function(xPolyTest)) # confidence
decision probability page 15 of evaluation
   plt.plot(fpr,tpr, color = "orange" ,label = "Logistic Regression")
```



(e)

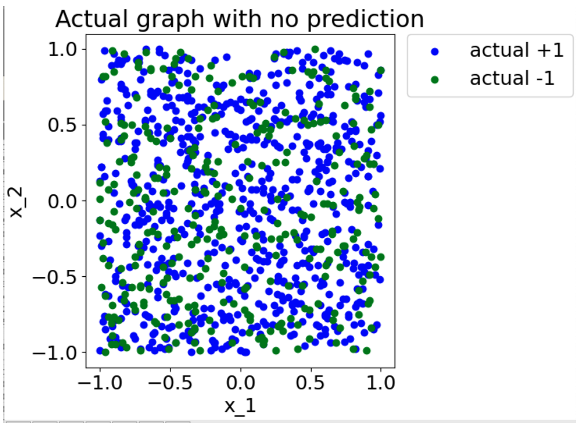
According to (c), it would be difficult to decide which is better, Logistic Regression or kNN classifier because their Accuracy, True Positive Rate, False Positive Rate and Precision are almost the same and all of them show that both of Logistic and kNN are accurate. Moreover, according to (d), from the ROC curve, although the Logistic Regression comes slightly closer to top-left corner, their behaviours are also almost the same and this graph shows they are accurate because both of them come close to the top-left corner where true positive rate is high and false positive rate is low. In this result, Logistic Regression shows it is slightly better, but it can not be denied that kNN classifier gives better results than Logistic Regression for the different cases or data.

For a random classifier, it is a point on the 45 degree line in the ROC curve graph. Moreover, its accuracy is 0.51. Therefore, its AUC = 0.5 and it can be said that it is ideal that other curves' AUC is more than 0.5(random classifier).

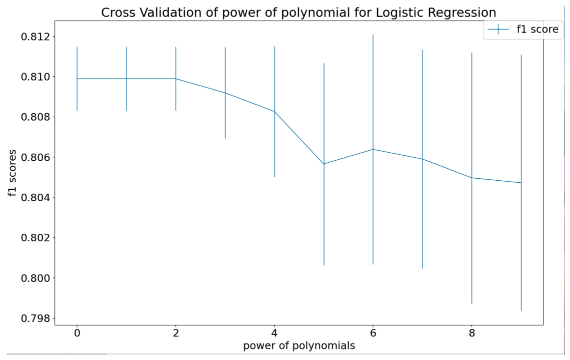
(ii)

(a) for (ii), the way of producing the output is the same, so only important outputs will be shown.

(i)



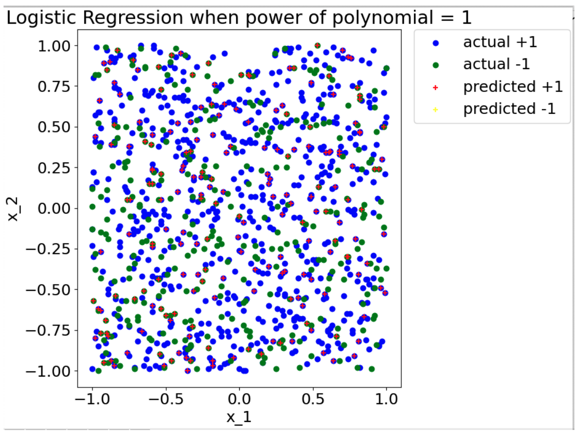
The above figure shows plots of dataset 2 without any predictions.

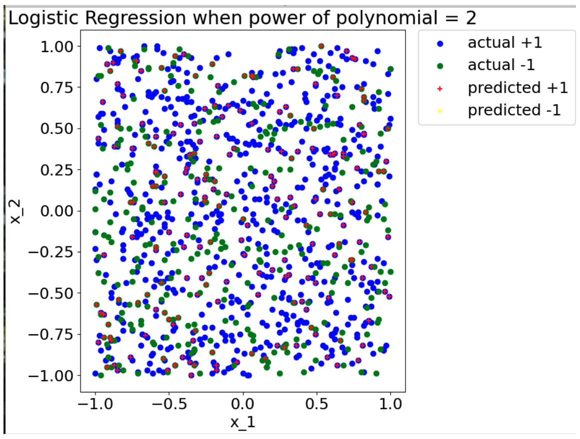


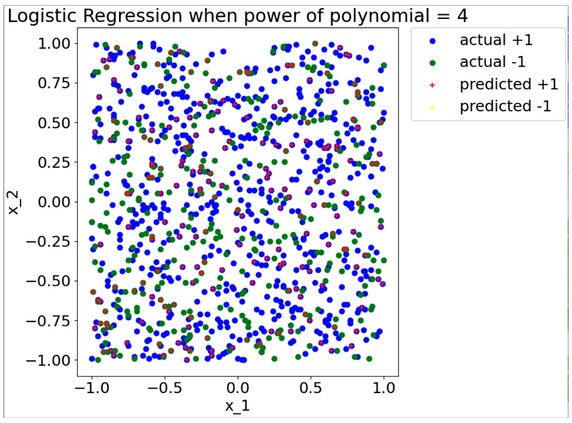
It can be seen that until power of polynomials = 2, the f1 scores are the same. The graph and the detailed outputs below show that after that of polynomials = 3, the f1 scores are decreasing.

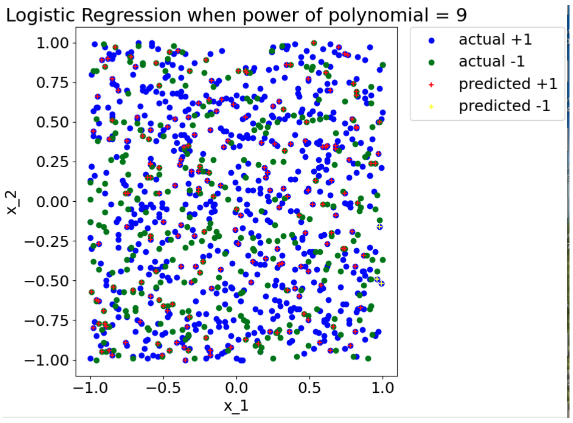
```
when power of polynomial =  1
f1 score =  [0.80938416 0.81176471 0.81176471 0.80825959 0.80825959]
when power of polynomial =  2
f1 score =  [0.80938416 0.81176471 0.81176471 0.80825959 0.80825959]
when power of polynomial =  4
f1 score =  [0.80235988 0.81176471 0.80825959 0.81065089 0.80825959]
when power of polynomial =  9
```

f1 score = [0.79761905 0.80825959 0.79640719 0.81065089 0.81065089]



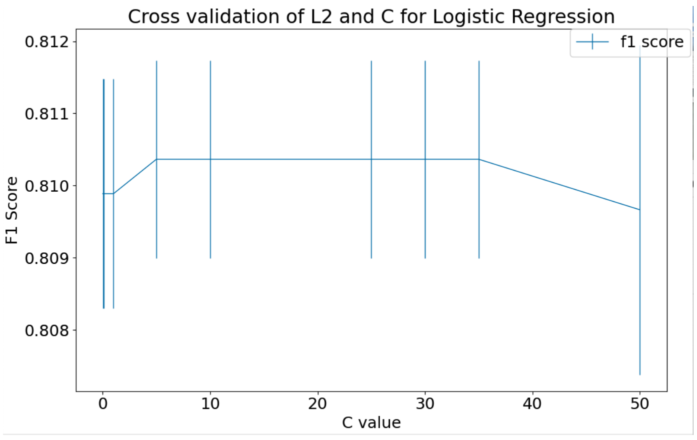






The 4 graphs above show the predictions on the data with power of polynomials = 1, 2, 4, 9. There should be a difference from power of polynomials = 2 to 4 & 9. However, it is difficult to see it because of noisy inputs.

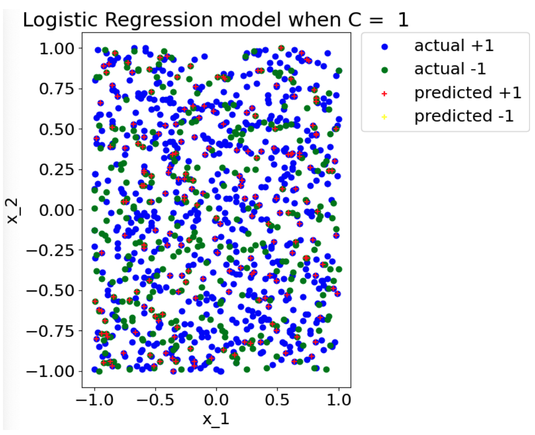
1 is chosen as it relatively has high f1 scores.

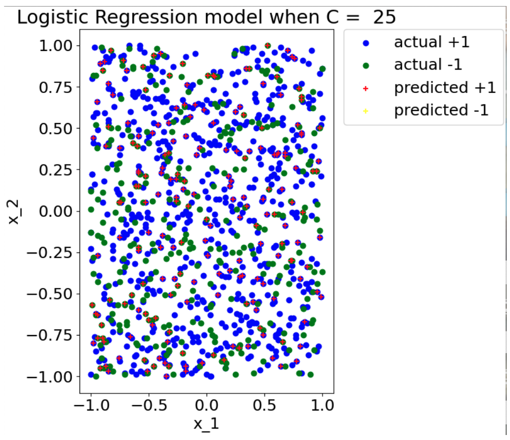


The figure above shows how F1 scores change in accordance with C. It can be seen that from C = 10 to C = 35, there is not much difference as Dataset 1 showed the same behaviour, but from C = 0.01 to C =10 and from C = 35 to C = 50, the F1 values are decreasing. As the output below shows, C = 10 seems to have high F1 scores, and choosing too small C produces an error, C = 10 is chosen.

when c = 0,01 and power of polynomial = 2
f1 score = [0.80938416 0.81176471 0.81176471 0.80825959 0.80825959]
when c = 1, and power of polynomial = 2
f1 score = [0.80938416 0.81176471 0.81176471 0.80825959 0.80825959]

```
when c = 5, and power of polynomial = 2
f1 score =  [0.80938416 0.81176471 0.81176471 0.80825959 0.81065089]
when c = 10, and power of polynomial = 2
f1 score =  [0.80938416 0.81176471 0.81176471 0.80825959 0.81065089]
when c = 35, and power of polynomial = 2
f1 score =  [0.80938416 0.81176471 0.81176471 0.80825959 0.81065089]
when c = 50, and power of polynomial = 2
f1 score =  [0.80588235 0.81176471 0.81176471 0.80825959 0.81065089]
```



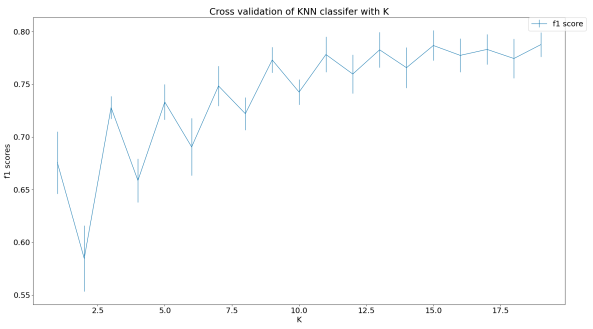


As with the figures of predictions above, these 2 figures above also look noisy.

(b)

The output below are f1 scores and mean of them.

```
when k =  1
f1 score =  [0.69144981 0.7        0.61764706 0.68571429 0.68327402]
mean =  0.6756170360033045
when k =  4
f1 score =  [0.65891473 0.63779528 0.65625    0.6969697  0.64341085]
mean =  0.6586681107911194
when k =  5
f1 score =  [0.73648649 0.71947195 0.74247492 0.75747508 0.70989761]
mean =  0.7331612088094291
when k =  10
f1 score =  [0.73154362 0.74342105 0.72972973 0.76315789 0.74496644]
mean =  0.7425637488424488
when k =  13
f1 score =  [0.76971609 0.76012461 0.79750779 0.80487805 0.78125   ]
mean =  0.7826953071724915
when k =  19
f1 score =  [0.7788162  0.77777778 0.80120482 0.80239521 0.77777778]
mean =  0.7875943567580899
```



From the outputs and the graph above, it can be seen that as k increases, f1 scores increase. Therefore, because the k should be the highest in k_range from 1 to 20, it is 20.

(c)

Confusion matrix for Logistic Regression with power of polynomial = 2 (this 2 is determined in (i) in (a))and c = 25(this 25 is determined in (ii) in (a)).

```
[ 0    66]          Accuracy  = 0.67
[ 0   137]          True Positive Rate = 0
                   False Positive Rate = N/A
                   Precision = N/A
```

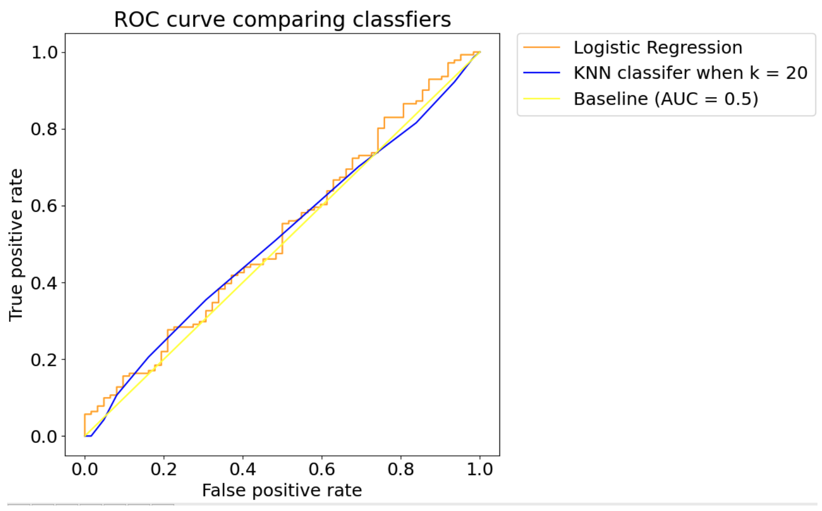
confusion matrix for KNN classifier when k = 13

```
[ 6    60]          Accuracy  = 0.65
[ 12 125]           True Positive Rate = 0.09
                   False Positive Rate = 0.6
                   Precision = 0.33
```

confusion matrix for baseline classifiers that makes random predictions

```
[ 34   32]          Accuracy  = 0.51
[ 68   69]          True Positive Rate = 0.52
                   False Positive Rate = 0.6
                   Precision = 0.33
```

(d)



(e)

Because the data is noisy, their confusion matrices show that they are not so accurate. Moreover, their ROC curve is close to the baseline, which also shows that they are not accurate enough to use. Furthermore, in both confusion matrices and ROC curves, they behave almost the same. Therefore, it can be true that it is difficult to choose which classifier is better.

Appendix

```
from cProfile import label
from calendar import c
from re import X
from statistics import mean
# from statistics import LinearRegression
from tkinter import Y
from turtle import color
from xml.etree.ElementInclude import include
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
from sklearn.dummy import DummyClassifier


df1 = pd.read_csv('/Users/doimasanari/Desktop/ML/week4/codes/#
id:12-24-12-1,,.csv')

df2 = pd.read_csv('/Users/doimasanari/Desktop/ML/week4/codes/# id:12--12-12-1.csv')

data1X = df1.iloc[:, 0:2]
data1X1 = df1.iloc[:, 0]
data1X2 = df1.iloc[:, 1]
data1Y = df1.iloc[:, 2]

data2X = df2.iloc[:, 0:2]
data2X1 = df2.iloc[:, 0]
data2X2 = df2.iloc[:, 1]
data2Y = df2.iloc[:, 2]

# print(data1X)
# print(data1X1)
# print(data1X2)
# print(data1Y)
# print(data2X1)
# print(data2X2)
# print(data2Y)

def getANormalGraph(x1, x2, y, onlyThisGraph):
   # the graph below is an actual graph
   plt.figure()
   plt.rc('font', size=18)
   plt.rcParams["figure.constrained_layout.use"] = True
   plt.scatter(x1[y>0], x2[y>0], color="blue", label="actual +1")
   plt.scatter(x1[y<0], x2[y<0], color="green",label="actual -1")
   plt.xlabel("x_1")
   plt.ylabel("x_2")
   if onlyThisGraph == True:
       plt.legend(bbox_to_anchor=(1.04, 1), borderaxespad=0)
       plt.title("Actual graph with no prediction")
       plt.show()

def getPredictionPlot(x1, x2, y, xTest1, xTest2, yPred, polyPower, Ci, k):
```

```python
    getANormalGraph(x1, x2, y, False) # combine actual data with predicted data
    plt.scatter(xTest1[yPred>0], xTest2[yPred>0], color="red", marker="+", label =
"predicted +1")
    plt.scatter(xTest1[yPred<0], xTest2[yPred<0], color="yellow", marker="+", label
= "predicted -1")
    plt.xlabel("x_1")
    plt.ylabel("x_2")
    if Ci == 0 and k == 0:
        plt.title("Logistic Regression when power of polynomial = " +
str(polyPower))
    elif k == 0 and polyPower == 0:
        plt.title("Logistic Regression model when C =  " + str(Ci))
    else:
        plt.title("when k =  " + str(k))
    plt.legend(bbox_to_anchor=(1.04, 1), borderaxespad=0)

def splitTrainAndTest(x, y):
    xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.2) # split the
data for training and testing.
    xTest1 = xTest.iloc[:, 0]
    xTest2 = xTest.iloc[:, 1]
    return xTrain, xTest, yTrain, xTest1, xTest2, yTest


def Qa(x, x1, x2, y, isData1):

    # a(i)
    mean_error=[]
    std_error=[]

    polyPowers = range(0, 10)
    # split the data into train and test
    xTrain, xTest, yTrain, xTest1, xTest2, yTest = splitTrainAndTest(x, y)

    # getting a logisticRegression model with polynomial features and serch for the
maximum order of polynomial to use
    for polyPower in polyPowers:

        #  getttig a model with polynomial features
        poly = PolynomialFeatures(polyPower)
        xPoly = poly.fit_transform(x)
        xPolyTrain = poly.fit_transform(xTrain)
        xPolyTest = poly.fit_transform(xTest)
        model = LogisticRegression(penalty = "l2").fit(xPolyTrain, yTrain)
        yPred = model.predict(xPolyTest)
        getPredictionPlot(x1, x2, y, xTest1, xTest2, yPred, polyPower, 0, 0)

        # getting a f1 score with cross validation = 5
        score = cross_val_score(model, xPoly, y, cv=5, scoring="f1")
        print("when power of polynomial = ", polyPower)
        print("f1 score = ", score)

        # score.append(model.score(xPoly, y))
        # error.append(model.score(xPoly, y))
        # error.append(mean_squared_error(y, yPred))

        mean_error.append(np.array(score).mean())
        std_error.append(np.array(score).std())

    plt.figure()
    plt.rc('font', size=18)
    plt.rcParams["figure.constrained_layout.use"] = True
    plt.errorbar(polyPowers, mean_error, yerr=std_error, linewidth = 1, label = "f1
score")
    plt.legend(bbox_to_anchor=(1.04, 1), borderaxespad=0)
    plt.title("Cross Validation of power of polynomial for Logistic Regression")
    plt.xlabel("power of polynomials")
```

```
   plt.ylabel("f1 scores")
   plt.show()

   # a (ii) looking for the weight C given to the penalty in the cost function
   print("(ii)")
   mean_error=[]
   std_error=[]
   Ci_range = [0.01, 0.1, 1, 5, 10, 25, 30, 35, 50]
   for Ci in Ci_range:
       # polyPowers = range(2)
       # for polyPower in polyPowers:
       if isData1 == True:
           poly = PolynomialFeatures(2)
       else:
           poly = PolynomialFeatures(1)
       xPoly = poly.fit_transform(x)
       xPolyTrain = poly.fit_transform(xTrain)
       xPolyTest = poly.fit_transform(xTest)
       model = LogisticRegression(penalty = "l2", C = Ci).fit(xPolyTrain, yTrain)
       yPred = model.predict(xPolyTest)
       getPredictionPlot(x1, x2, y, xTest1, xTest2, yPred, 0, Ci, 0)

       #  getting f1 score
       score = cross_val_score(model, xPoly, y, cv=5, scoring="f1")
       print("when c = %d, and power of polynomial = %d" % (Ci, 2))
       print("f1 score = ", score)
       mean_error.append(np.array(score).mean())
       std_error.append(np.array(score).std())

   plt.figure()
   plt.rc("font", size=18)
   plt.rcParams["figure.constrained_layout.use"] = True
   plt.errorbar(Ci_range, mean_error, yerr=std_error, linewidth=1, label = "f1
score")
   plt.xlabel("C value")
   plt.ylabel("F1 Score")
   plt.title("Cross validation of L2 and C for Logistic Regression")
   plt.legend(bbox_to_anchor=(1.04, 1), borderaxespad=0)
   plt.show()


def Qb(x, x1, x2, y):

   xTrain, xTest, yTrain, xTest1, xTest2, yTest = splitTrainAndTest(x, y)
   k_range = range(1, 20)
   mean_error=[]
   std_error=[]
   print("Qb")
   for k in k_range :
       # train a kNN classifier on the data
       model = KNeighborsClassifier(n_neighbors=k, weights='uniform').fit(xTrain,
yTrain)
       # getting model score with using cross-validation = 5
       score = cross_val_score(model, x, y, cv=5, scoring="f1")

       print("when k = ", k)
       print("f1 score = ", score)
       print("mean = ", score.mean())

       # getting a prediction plot
       yPred = model.predict(xTest)
       getPredictionPlot(x1, x2, y, xTest1, xTest2, yPred, 0, 0, k)
       mean_error.append(np.array(score).mean())
       std_error.append(np.array(score).std())

   plt.figure()
   plt.rc('font', size=18)
   plt.rcParams["figure.constrained_layout.use"] = True
   plt.errorbar(k_range, mean_error, yerr=std_error, linewidth = 1, label = "f1 
score")
   plt.xlabel("K")
   plt.ylabel("f1 scores")
   plt.title("Cross validation of KNN classifer with K")
   plt.legend(bbox_to_anchor=(1.04, 1), borderaxespad=0)
   plt.show()


def Qc(x, x1, x2, y, p, k, c):

   # split the data for training and testing.
   xTrain, xTest, yTrain, xTest1, xTest2, yTest = splitTrainAndTest(x, y)

   # getting confusion matrix for Logistic Regression
   poly = PolynomialFeatures(p)
   xPoly = poly.fit_transform(x)
   xPolyTrain = poly.fit_transform(xTrain)
   xPolyTest = poly.fit_transform(xTest)
   model = LogisticRegression(penalty = "l2", C=c).fit(xPolyTrain, yTrain)
   yPred = model.predict(xPolyTest)
   print("confusion matrix for Logistic Regression with power of polynomial = " + 
str(p))
   print(confusion_matrix(yTest, yPred))
   print(classification_report(yTest, yPred))


   # getting confusion matrix for kNN classifier
   model = KNeighborsClassifier(n_neighbors=k, weights='uniform').fit(xTrain, 
yTrain)
   yPred = model.predict(xTest)
   print(confusion_matrix(yTest, yPred))
   print(classification_report(yTest, yPred))
   print("confusion matrix for KNN classifer when k = " + str(k))
   print(confusion_matrix(yTest, yPred))
   print(classification_report(yTest, yPred))

   # # getting confusion matrix for baseline classifier that always predicts the 
most frequent class in the training data
   # dummy = DummyClassifier(strategy="most_frequent").fit(xTrain, yTrain)
   # ydummy = dummy.predict(xTest)
   # print("confusion matrix for baseline classifiers that always predicts the most 
frequent class")
   # print(confusion_matrix(yTest, ydummy))
   # print(classification_report(yTest, ydummy))

   # getting confusion matrix for baseline classifier that makes random predictions
   dummy = DummyClassifier(strategy="uniform").fit(xTrain, yTrain)
   ydummy = dummy.predict(xTest)
   print("confusion matrix for baseline classifiers that makes random predictions")
   print(confusion_matrix(yTest, ydummy))
   print(classification_report(yTest, ydummy))


def Qd(x, x1, x2, y, p, k, c):

   #  split the data for training and testing.
   xTrain, xTest, yTrain, xTest1, xTest2, yTest = splitTrainAndTest(x, y)

   #  getting ROC curve for logisticRegression
   poly = PolynomialFeatures(p)
   xPoly = poly.fit_transform(x)
   xPolyTrain = poly.fit_transform(xTrain)
   xPolyTest = poly.fit_transform(xTest)
   model = LogisticRegression(penalty = "l2", C = c).fit(xPolyTrain, yTrain)
   fpr, tpr, _ = roc_curve(yTest ,model.decision_function(xPolyTest)) # confidence 
decision probability page 15 of evaluation
   plt.plot(fpr,tpr, color = "orange" ,label = "Logistic Regression")
   # plt.plot([0, 1], [0, 1], color='green',linestyle='--')

   #  getting ROC curve for kNN classifier
   model = KNeighborsClassifier(n_neighbors=k, weights='uniform').fit(xTrain, 
yTrain)
   fpr, tpr, _ = roc_curve(yTest ,model.predict_proba(xTest)[:, 1]) # confidence 
decision probability page 15 of evaluation
   plt.plot(fpr,tpr, color = "blue" ,label = "KNN classifer when k = " + str(k))

   # getting ROC curve for Dummy classifer that makes random predictions
   model = DummyClassifier(strategy='uniform').fit(xTrain, yTrain) # expected one 
of ('most_frequent', 'stratified', 'uniform', 'constant', 'prior')
   fpr, tpr, _ = roc_curve(yTest ,model.predict_proba(xTest)[:, 1]) # confidence 
decision probability page 15 of evaluation
   plt.plot(fpr,tpr, color = "yellow" ,label = "Baseline (AUC = 0.5)")
   plt.xlabel("False positive rate") # named it true positive and false negative
   plt.ylabel("True positive rate") # True positive we predict label +1 and correct 
label is +1, false positive we predict label +1 and correct label is -1
   plt.title("ROC curve comparing classfiers")
   plt.legend(bbox_to_anchor=(1.04, 1), borderaxespad=0)
   plt.show()


# get a graph with just a plain data
getANormalGraph(data1X1, data1X2, data1Y, True)
getANormalGraph(data2X1, data2X2, data2Y, True)

# Qa
Qa(data1X, data1X1, data1X2, data1Y, True)
Qa(data2X, data2X1, data2X2, data2Y, False)

# Qb
Qb(data1X, data1X1, data1X2, data1Y)
Qb(data2X, data2X1, data2X2, data2Y)

# Qc
Qc(data1X, data1X1, data1X2, data1Y, 2, 13, 10)
Qc(data2X, data2X1, data2X2, data2Y, 2, 20, 25)

# Qd
Qd(data1X, data1X1, data1X2, data1Y, 2, 13, 10)
Qd(data2X, data2X1, data2X2, data2Y, 2, 20, 25)
```