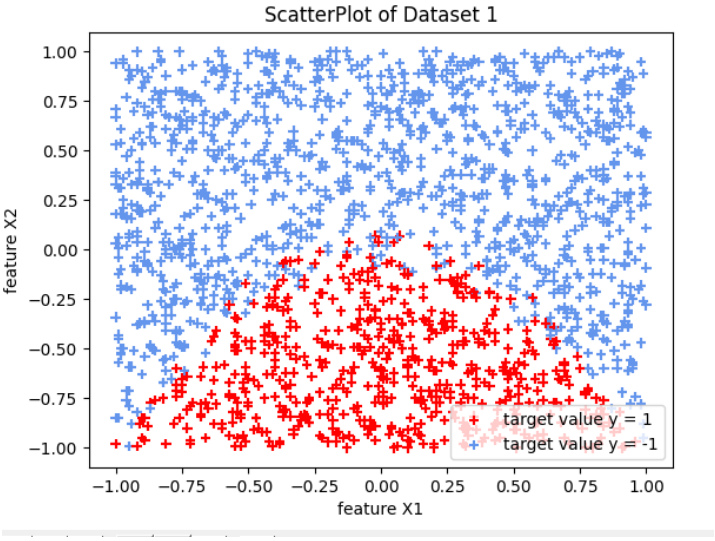
Machine Learning Assignment #4

Alex Kennedy *17328638*

First Dataset: # id:12--12--12-0, Second Dataset: # id:12--12-12-0

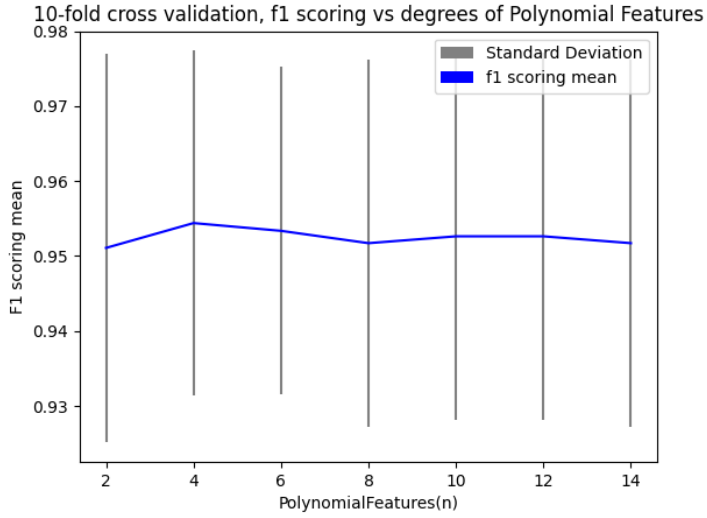
(i)(a) Train Logistic Regression Classifier with L2 penalty use cross validation to select the maximum order of polynomial features and C value.

First Dataset Plot:



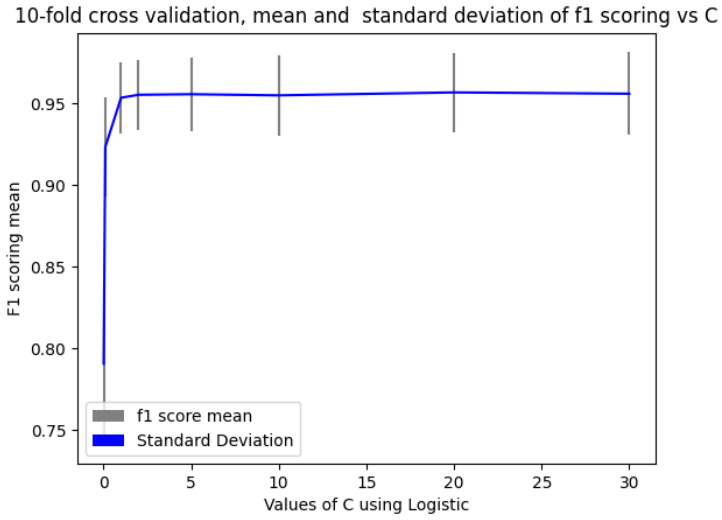
Using Cross-Validation:

I used cross validation using different orders of polynomials. For every value of degree, I create a model using Logistic Regression and perform 10-fold cross validation on the model and get the mean and standard deviation of the f1 score of the model. This is in the ***fold10_cVals()*** and ***fold10_polyFeat()*** functions. Here is the error bar plot I generated of values of 2 to 14, incrementing by 2. I use a baseline C = 10 here (will refine the C value when best polyFeature is found).



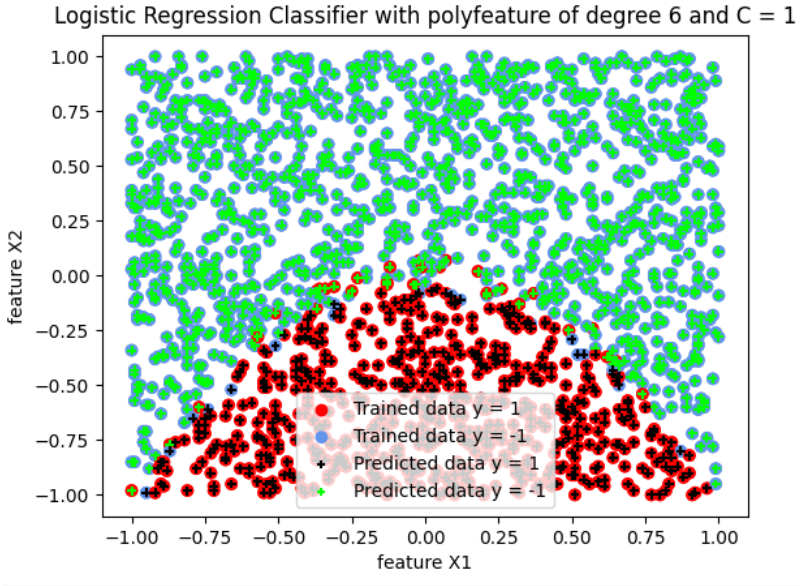
From the plot it seems that a good value for the Polynomial Feature is 6. This is due to it having quite a good f1 score while also having the lowest standard deviation. When calculating a good C value, I then used the polyFeatures value of 6. The C values I used were

C = (0.01, 0.1, 1, 2, 5, 10, 20, 30). Here is the 10-fold cross validation plot.



From the plot it looks like C = 1 might be the optimal value to use as the mean of the f1 score is quite high and the standard deviation of the scores is the lowest out of all of them. After C = 1 the standard deviation begins to increase again.

After finding both of these values. I then plotted the logistic regression classifier predictions against the actual data target values.



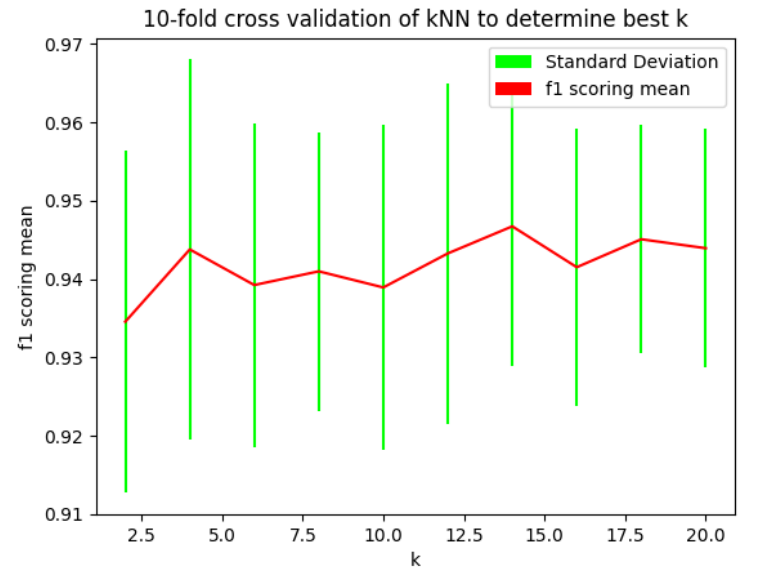
The Logistic Regression Classifier does fit the actual data quite well and so I believe the values I chose q (Polynomial Features degree) and C are were quite suitable.

(i)(b)

For kNN, I split the dataset into 2 parts, the test data and the training data. I split it into 20% test data and 80% training data. The test data is not used in the kNN Classifier. Again, I used cross validation to select the best value for k. using the ***getBestK()*** function The values for k I used were

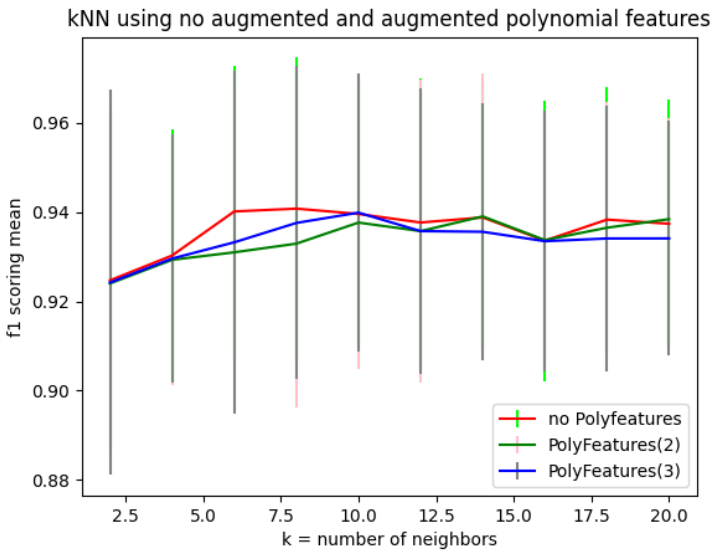
k = (2, 4, 6, 8, 10, 12, 14, 16, 18, 20).

Here is the cross-validation plot using f1 scoring.



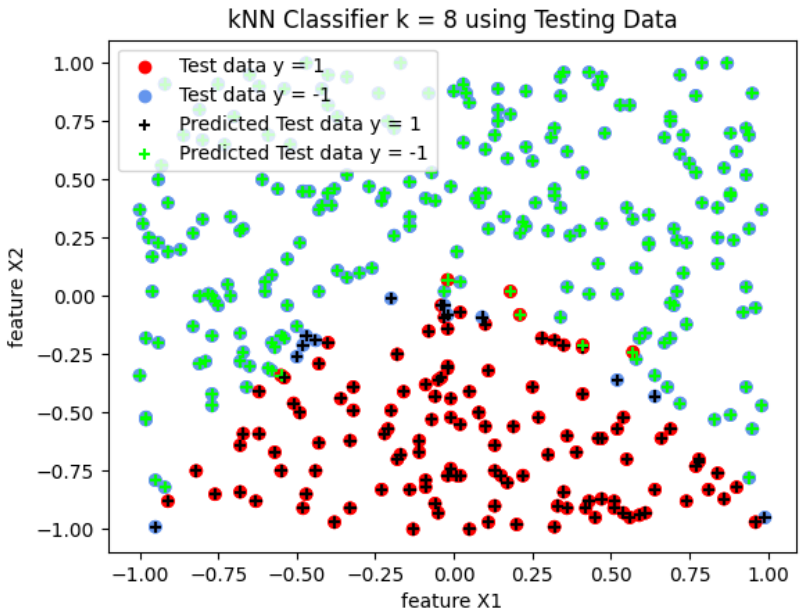
From the plot, k = 8 seems like a decent value as the standard deviation is low as well as the f1 score being reasonably high. A low k value will tend to lead to over-fitting, while a high k value will tend to lead to under-fitting, so a mid-range k value seems to be a good fit. I will use k = 8 in my kNN Classifier.

I also created Polynomial features of degree 2 and 3 to see if it is worth augmenting the features using ***getBestKPoly()*** function. Here is the plot for that.



It seems like it doesn't affect the outcome too much, so I would suggest that it is not worth to do as it would require a lot more computation power which could slow down systems, especially on very large datasets.

Here is the plot for the kNN Classifier with k = 8 on the testing data.



As one can see, the Classifier does seem to predict the target values of the actual dataset very well, so the value I chose for k seems to be quite suitable.

(i)(c)

Using sklearns confusion matrix api, it makes it quite east to generate the confusion matrices for both Logistic Regression and kNN. The baseline classifier I used was sklearn's DummyClassifier as it has a parameter value which will predict the most frequent class. Here are the confusion matrices:

```
Confusion Matrix for Logistic Regression Classifier
[[1161   18]
 [  31  556]]
Confusion Matrix for kNN
[[217   2]
 [ 10 125]]
Confusion Matrix for Dummy Classifier
[[219   0]
 [135   0]]
```

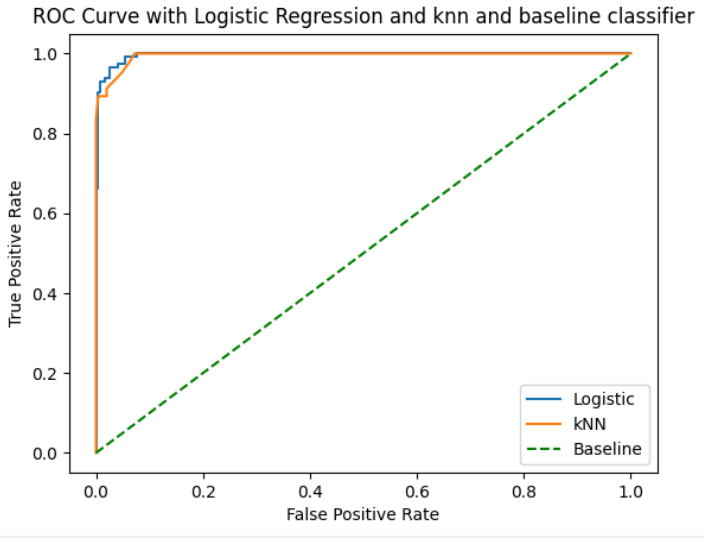
Logistic Regression Classifier has an accuracy of 97.23%

kNN has an accuracy of 96.6%

Dummy Classifier has an accuracy of 61.87%

(i)(d)

The Baseline classifier points generate a straight line going through the origin. Here are the ROC curves:

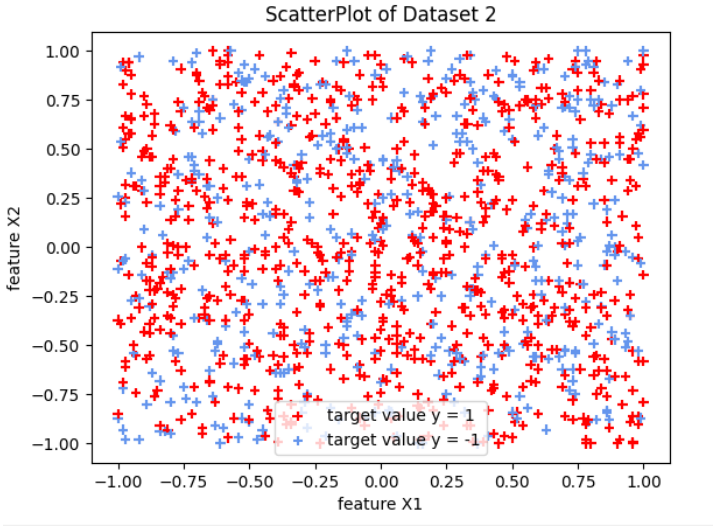


(i)(e)

From the confusion matrices and the ROC curves, the Logistic Regression does perform the best out of all the classifiers. kNN is also better than the random classifier. From the confusion matrices, we can calculate accuracy of True Positive and True Negative and the Logistic Regression Classifier has the highest accuracy. From the ROC curve, we can see that the Logistic Regression does push into the top a bit more than the kNN Classifier. The difference in performance of the Logistic and kNN is quite small though, both of them do a great job at classification, for that reason I wouldn't recommend one over the other. If you wanted your classifier to be as accurate as possible then the Logistic Classifier would be marginally better to use.

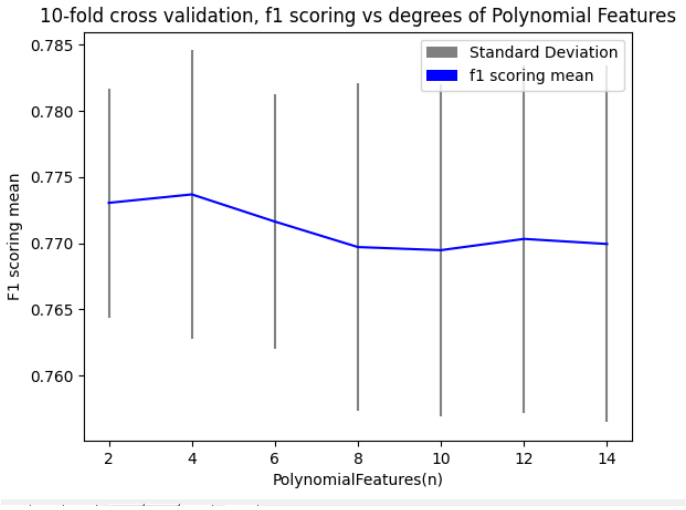
Using the second Dataset

Here will we plot the same things we did for the first dataset, however this one has a lot of noise in it as you can see from the scatter plot.



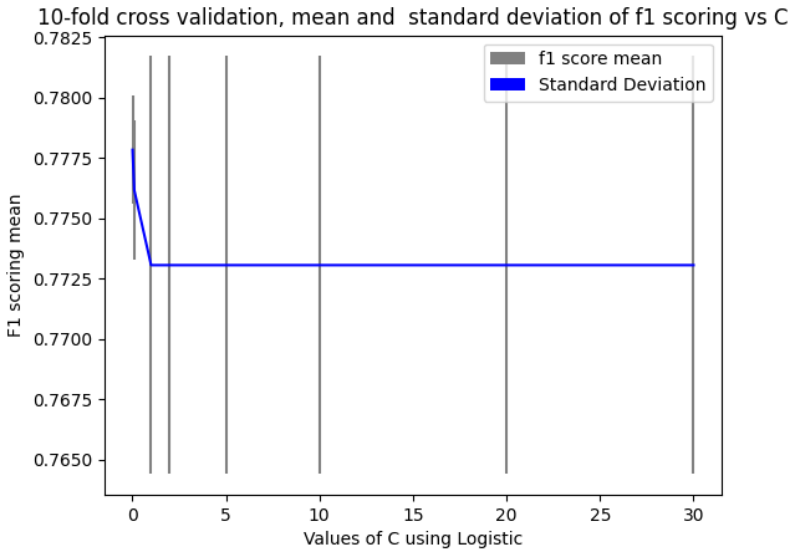
(ii)(a)

I used the same values for polynomial features q (with C = 1) to cross validate on a linear regression classifier, here is the plot.



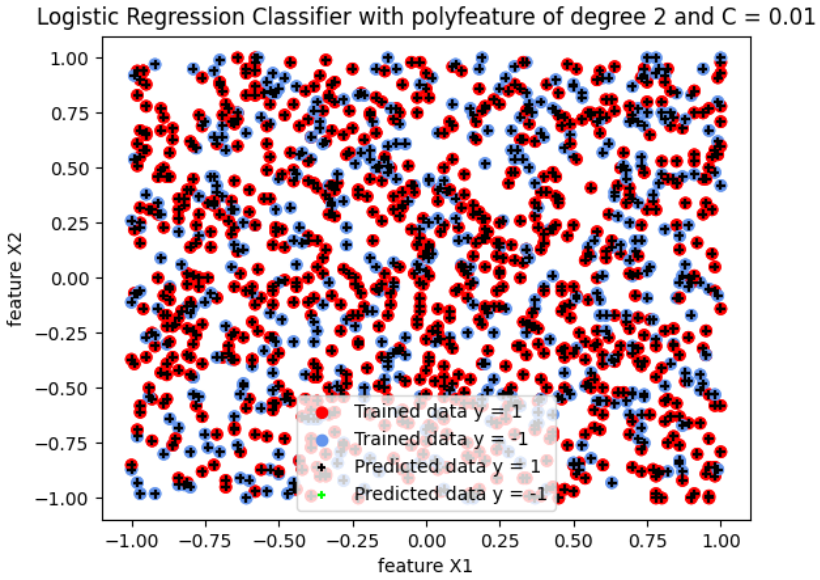
The best value I see here is to pick q = 2. This is because it has the lowest standard deviation out of all other values of q. This could be due to the fact that the data is noisy, increasing the features will in turn, most likely increasing standard deviation. So q = 2 is the value we will use for finding C.

Here is the cross-validation plot for the optimal C Value to use:



The classifier starts to perform quite poorly very quickly as C increases as shown by the decrease in f1 score and standard deviation. For that reason we will pick the lowest C value we decided to use which is C = 0.01.

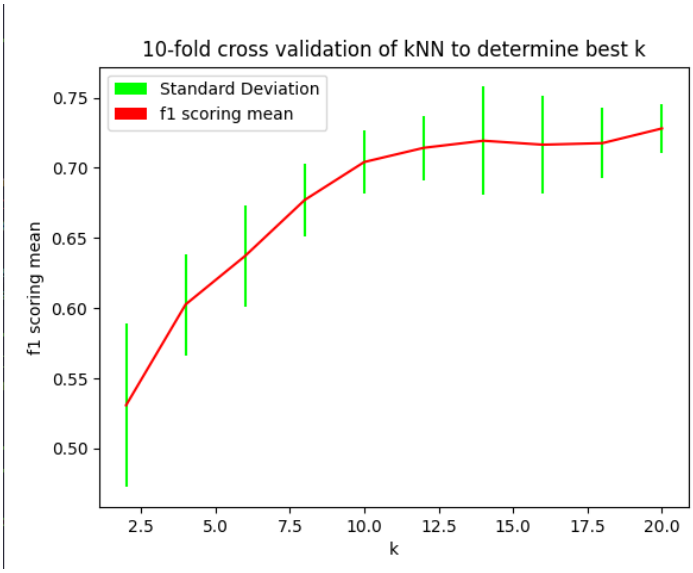
After finding the best values to use, here is the logistic regression classifier plot on the noisy dataset:



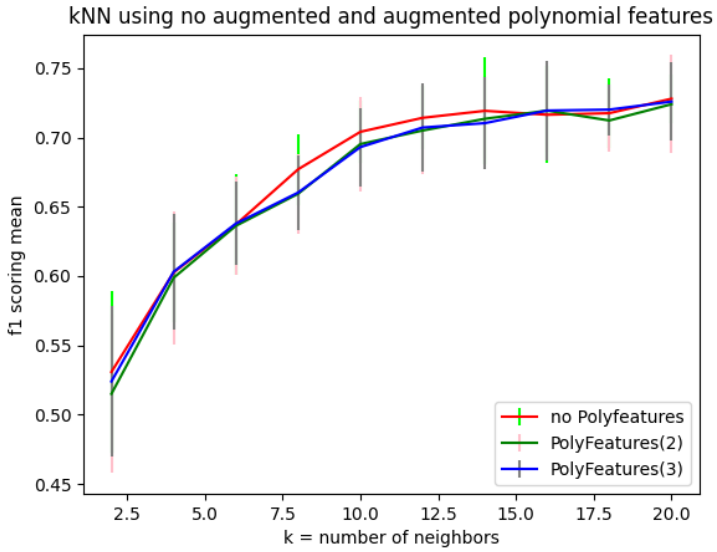
As you can see, The model did not predict any target values = -1 as there are no lime coloured points. This is because the data itself is not uniform, and so the logistic regression has trouble making accurate predictions.

(ii)(b)

Now ill train a kNN classifier on the data. First ill use cross-validation to determine the best value of k to use.

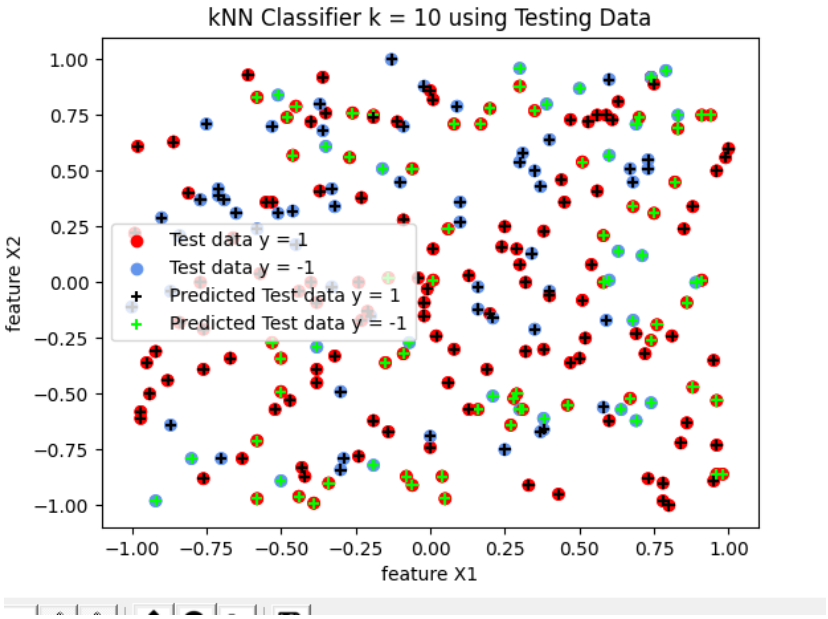


We know that picking a large value of k can lead to overfitting so although k = 20 does have the smallest deviation it is best to avoid it. A good mid-range value is k = 10 as it has a small standard deviation and still has a decently high f1 score, so we will pick that as our optimal value. I also augmented the features with polynomials to see if that would improve things, here is that plot.



In a similar situation to the first dataset, it seems that augmenting the polynomial features does not do all that much and so we will not augment our test and training data with polyfeatures.

Here is the plot of the kNN Classifier with k = 10 on the second dataset:



In contrast to the Logistic Regression Classifier, the kNN classifier does actually match some of its predictions with the actual data. Granted it is not entirely accurate at all, but still an interesting observation none the less.

(ii)(c)

```
Confusion Matrix for Logistic Regression Classifier
[[  0 445]
 [  0 779]]
Confusion Matrix for kNN
[[ 38  52]
 [ 45 110]]
Confusion Matrix for Dummy Classifier
[[  0  90]
 [  0 155]]
```

Logistic Regression Classifier has an accuracy of 63.6%

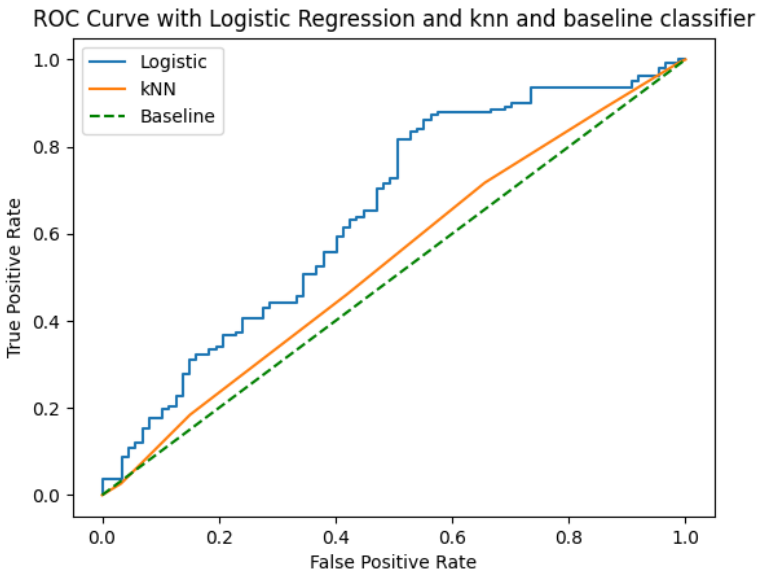
kNN has an accuracy of 64%

Dummy Classifier has an accuracy of 63.3%

We know that the accuracy rating is not exactly a good metric here as both Logistic regression and the dummy classifier are just predicting the most frequent class. The kNN on the other hand is at least predicting some of the True Positive and True Negative Values.

(ii)(d)

Here are the ROC curves for Logistic Regression and kNN in relation to the baseline classifier:



(ii)(e)

From the ROC curves, it seems that Logistic Regression once again performs better than the other classifiers. Which is interesting as Logistic on this dataset does not pick up any True Positive Values, it just sets all values to the most frequent class which is target value -1. kNN also performs very similar to the baseline according to the ROC curve. In this instance however, I would use kNN over Logistic in a noisy set like this as it does have the highest accuracy from the confusion matrices, but it also actually can correctly predict at least some of the less frequent class which can be seen from the confusion matrices.

Code Appendix

```
import numpy as np
import math
import pandas as pd
import matplotlib.pyplot as mpl
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
from sklearn.linear_model import Ridge
import random
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.dummy import DummyClassifier
from sklearn.metrics import roc_curve
from sklearn.neighbors import KNeighborsRegressor

def readFiles():
    out1 = None
    out2 = None
    with open("week4.txt") as contained:
        for line in contained:
            line = line.strip()
            if line.startswith('# id') and out1 is None:
                out1 = open("week4_first.txt", 'w')
                out1.write(line +'\n')
            elif line.startswith('# id') and out2 is None:
                out2 = open('week4_second.txt', 'w')
                out2.write(line + '\n')

            elif out1 is not None and out2 is None:
                out1.write(line + '\n')

            elif out2 is not None:
                out2.write(line+'\n')

def splitData(filename):
    df = pd.read_csv(filename, comment = '#')
    X1 = df.iloc[:,0]
    X2 = df.iloc[:,1]
    targetVal = df.iloc[:,2]
    X = np.column_stack((X1,X2))
```

```
    return X1, X2, targetVal, X

def selectClassifier(fit, regModel, cVal, X, targetVal):
    if regModel == "Logistic":
        model = LogisticRegression(penalty='l2', solver='lbfgs', C = cVal, max_it
er=100000)
    model = model.fit(fit, targetVal)
    return model

def getPoly(degree, X, targetVal):
    poly = PolynomialFeatures(degree)
    fit = poly.fit_transform(X)
    return fit

def kCrossValid(X, targetVal, cVal, k, regModel, degree):
    model = selectClassifier(X, "Logistic", cVal, X, targetVal)
    polyFit = getPoly(degree, X, targetVal)
    scores = cross_val_score(model, polyFit, targetVal, cv = 10, scoring = 'f1')
    return scores.mean(), scores.var(), scores.std()

def kCrossValidPoly(model, X, targetVal, polyValue, k):
    scores = cross_val_score(model, X, targetVal, cv=10, scoring='f1')
    return scores.mean(), scores.var(), scores.std()

def fold10_cVals(X, targetVal, cVals, regModel, degree):
    meanArray = np.array([0]*len(cVals), dtype=float)
    stdArray = np.array([0]*len(cVals), dtype=float)
    i = 0
    for i in range(len(cVals)):
        mean, var, std = kCrossValid(X, targetVal, cVals[i], 10, "Logistic", degr
ee)
        meanArray[i] = mean
        stdArray[i] = std
    return meanArray, stdArray

def fold10_polyFeat(model, X, targetVal, polyValue):
    mean, var, std = kCrossValidPoly(model, X, targetVal, polyValue, 10)
    return mean, std

def getBestK(kVals, XTrain, TargetValTrain):
    meanArray = np.array([0] * len(kVals), dtype= float)
    stdArray = np.array([0] * len(kVals), dtype=float)
    for i in range(len(kVals)):
        model = KNeighborsClassifier(n_neighbors = kVals[i], weights = 'uniform')
.fit(XTrain, TargetValTrain)
        scores = cross_val_score(model, XTrain, TargetValTrain, cv = 10, scoring=
'f1')
        meanArray[i] = scores.mean()
```

```
        stdArray[i] = scores.std()
    return meanArray, stdArray

def getBestKAugmentPoly(kVals, XTrain, TargetValTrain,n):
    meanArray = np.array([0] * len(kVals), dtype= float)
    stdArray = np.array([0] * len(kVals), dtype=float)
    poly = PolynomialFeatures(n)
    polyFit = poly.fit_transform(XTrain)
    for i in range(len(kVals)):
        model = KNeighborsClassifier(n_neighbors = kVals[i], weights = 'uniform')
#.fit(polyFit, TargetValTrain)
        scores = cross_val_score(model, polyFit, TargetValTrain, cv = 10, scoring
='f1')
        meanArray[i] = scores.mean()
        stdArray[i] = scores.std()
    return meanArray, stdArray

def plot_error(cVals, mean, std, colors, lineColor, title, xLabel, yLabel):
    mpl.errorbar(cVals, mean, yerr=std, ecolor=colors, color = lineColor)
    mpl.title(title)
    mpl.xlabel(xLabel)
    mpl.ylabel(yLabel)

def first_plot(X1, X2, targetVal, title):
    pointColours = ['red'] * len(targetVal)
    for i in range(len(pointColours)):
        if targetVal[i] == -1:
            pointColours[i] = 'cornflowerblue'
    for labels in ["target value y = 1", "target value y = -1"]:
        mpl.scatter(X1, X2, c = pointColours, marker = '+', label = labels)
    mpl.title(title)
    mpl.xlabel("feature X1")
    mpl.ylabel("feature X2")
    mpl.legend()
    mpl.gca()
    legend = mpl.legend()
    legend.legendHandles[0].set_color('red')
    mpl.show()

def second_plot(X1, X2, targetVal, title):
    pointColours = ['red'] * len(targetVal)
    for i in range(len(pointColours)):
        if targetVal[i] == -1:
            pointColours[i] = 'cornflowerblue'
    for labels in ["target value y = 1", "target value y = -1"]:
        mpl.scatter(X1, X2, c = pointColours, marker = '+', label = labels)
    mpl.title(title)
    mpl.xlabel("feature X1")
```

```
    mpl.ylabel("feature X2")
    mpl.legend()
    mpl.gca()
    legend = mpl.legend()
    legend.legendHandles[1].set_color('cornflowerblue')
    mpl.show()

def classifier(yPredict, color1, color2):
    predictColours = [color1] * len(yPredict)
    for i in range(len(yPredict)):
        if(yPredict[i] == -1):
            predictColours[i] = color2
    return predictColours


def main():
    readFiles()
    X1, X2, firstTargetVal, firstX = splitData('week4_first.txt')
    X3, X4, secondTargetVal, secondX = splitData('week4_second.txt')
    first_plot(X1, X2, firstTargetVal, "ScatterPlot of Dataset 1")
    cVals = np.array([0.01, 0.1, 1, 2, 5, 10, 20, 30])
    degrees = np.array([2,4,6,8,10,12,14])
#USING THE FIRST DATASET HERE, WILL SPECIFY WHEN USING THE SECOND DATASET
    meanArrayC, stdArrayC = fold10_cVals(firstX, firstTargetVal, cVals, 'Logistic
', 6)
    polyMeanArray = np.array([0]*len(degrees), dtype=float)
    polyStdArray = np.array([0]*len(degrees), dtype=float)
    for n in range(len(degrees)):
        fit = getPoly(degrees[n], firstX, firstTargetVal)
        model = selectClassifier(fit, "Logistic", 1, firstX, firstTargetVal)
        mean, std = fold10_polyFeat(model, fit, firstTargetVal, degrees[n])
        polyMeanArray[n] = mean
        polyStdArray[n] = std
    print(polyStdArray)
    plot_error(cVals, meanArrayC, stdArrayC,"grey","blue", "10-
fold cross validation, mean and  standard deviation of f1 scoring vs C", "Values
of C using Logistic", "F1 scoring mean")
    leg_point0 = mpl.Rectangle((0,0), 1, 1, fc = 'grey')
    leg_point1 = mpl.Rectangle((0,0), 1,1 , fc = 'blue')
    mpl.legend([leg_point0,leg_point1], ["f1 score mean", "Standard Deviation"])
    mpl.show()

    plot_error(degrees, polyMeanArray, polyStdArray,"grey","blue", "10-
fold cross validation, f1 scoring vs degrees of Polynomial Features", "Polynomial
Features(n)", "F1 scoring mean")
    leg_point0 = mpl.Rectangle((0,0), 1, 1, fc = 'grey')
    leg_point1 = mpl.Rectangle((0,0), 1,1 , fc = 'blue')
```

```
    mpl.legend([leg_point0,leg_point1], ["Standard Deviation", "f1 scoring mean"]
)
    mpl.show()

#Logistic Regression Classifier using best polyfeature and C Value.
    bestPolyFit = getPoly(6, firstX, firstTargetVal)
    model = LogisticRegression(penalty='l2', solver='lbfgs', C = 1)
    model.fit(bestPolyFit, firstTargetVal)
    ypred = model.predict(bestPolyFit)
    ax = mpl.subplot()
    trainColor = classifier(firstTargetVal, 'red', 'cornflowerblue')
    predictColour = classifier(ypred, 'black', 'lime')
    for labels in ["Trained data y = 1", "Trained data y = -1"]:
        ax.scatter(X1, X2, c = trainColor, label = labels)
    for labels in ["Predicted data y = 1", "Predicted data y = -1"]:
        ax.scatter(X1, X2, c = predictColour, marker = '+', label = labels, s= 5*
*2)
    mpl.xlabel("feature X1")
    mpl.ylabel("feature X2")
    mpl.title("Logistic Regression Classifier with polyfeature of degree 6 and C
= 1")
    mpl.legend(loc = 'lower center')
    ax = mpl.gca()
    leg = ax.get_legend()
    leg.legendHandles[0].set_color('red')
    leg.legendHandles[2].set_color('black')
    leg.legendHandles[3].set_color('lime')
    mpl.show()


    #nearest neighbors
    firstXTrain, firstXTest, firstTargetValTrain, firstTargetValTest = train_test
_split(firstX, firstTargetVal, test_size=0.2)
    kVals = np.array([2, 4, 6, 8, 10, 12, 14, 16, 18, 20])
    meanArrayK, stdArrayK = getBestK(kVals, firstXTrain, firstTargetValTrain)
    print(stdArrayK)
    meanPolyK, stdPolyK = getBestKAugmentPoly(kVals, firstXTrain, firstTargetValT
rain, 2)
    meanPolyK1, stdPolyK1 = getBestKAugmentPoly(kVals, firstXTrain, firstTargetVa
lTrain, 3)

    plot_error(kVals, meanArrayK, stdArrayK, "lime", 'red', "10-
fold cross validation of kNN to determine best k", "k", "f1 scoring mean" )
    leg_point0 = mpl.Rectangle((0,0), 1, 1, fc = 'lime')
    leg_point1 = mpl.Rectangle((0,0), 1,1 , fc = 'red')
    mpl.legend([leg_point0,leg_point1], ["Standard Deviation", "f1 scoring mean"]
)
    mpl.show()
```

```
    plot_error(kVals, meanArrayK, stdArrayK, "lime", 'red', None, None, None )
    plot_error(kVals, meanPolyK1, stdPolyK1, "pink", 'green', None, None, None )
    plot_error(kVals, meanPolyK, stdPolyK, "grey", 'blue', None, None, None )
    mpl.title("kNN using no augmented and augmented polynomial features")
    mpl.xlabel("k = number of neighbors")
    mpl.ylabel("f1 scoring mean")
    mpl.legend(["no Polyfeatures", "PolyFeatures(2)", "PolyFeatures(3)"], loc='lo
wer right')
    mpl.show()

    #using best value of k
    model = KNeighborsClassifier(n_neighbors=8, weights = 'uniform').fit(firstXTr
ain, firstTargetValTrain)
    firstYPred = model.predict(firstXTest)
    firstTargetValTest = np.asarray(firstTargetValTest)
    firstTargetValTrain = np.asarray(firstTargetValTrain)
    testColor = classifier(firstTargetValTest, 'red', 'cornflowerblue')
    predictColour = classifier(firstYPred, 'black', 'lime')
    for labels in ["Test data y = 1", "Test data y = -1"]:
        mpl.scatter(firstXTest[:,0], firstXTest[:,1], c = testColor, label = labe
ls)
    for labels in ["Predicted Test data y = 1", "Predicted Test data y = -1"]:
        mpl.scatter(firstXTest[:,0], firstXTest[:,1], c = predictColour, marker =
 '+', label = labels)
    mpl.xlabel("feature X1")
    mpl.ylabel("feature X2")
    mpl.title("kNN Classifier k = 8 using Testing Data")
    mpl.gca()
    leg = mpl.legend()
    leg.legendHandles[0].set_color('red')
    leg.legendHandles[1].set_color('cornflowerblue')
    leg.legendHandles[2].set_color('black')
    leg.legendHandles[3].set_color('lime')
    mpl.show()

    #CONFUSTION MATRICES first dataset
    dummyModel = DummyClassifier(strategy="most_frequent").fit(firstXTrain, first
TargetValTrain)
    dummyPred = dummyModel.predict(firstXTest)

    print("Confusion Matrix for Logistic Regression Classifier")
    print(confusion_matrix(firstTargetVal, ypred))

    print("Confusion Matrix for kNN")
    print(confusion_matrix(firstTargetValTest, firstYPred))

    print("Confusion Matrix for Dummy Classifier")
    print(confusion_matrix(firstTargetValTest, dummyPred))
```

```
    #ROC Curves
#Logistic
    Xtrain, Xtest, ytrain, ytest = train_test_split(bestPolyFit, firstTargetVal,
test_size=0.2)
    logisticModel = LogisticRegression(penalty='l2', solver='lbfgs', C = 1)
    logisticModel.fit(Xtrain, ytrain)
    fprLog, tprLog,_ = roc_curve(ytest, logisticModel.decision_function(Xtest))
    mpl.plot(fprLog, tprLog)
#kNN
    y_scores = model.predict_proba(firstXTest)
    fprknn, tprknn, _ = roc_curve(firstTargetValTest, y_scores[:,1])
    mpl.plot(fprknn, tprknn)
#baseline
    mpl.plot([0, 1], [0, 1], color='green',linestyle='--')
    mpl.xlabel("False Positive Rate")
    mpl.ylabel("True Positive Rate")
    mpl.title("ROC Curve with Logistic Regression and knn and baseline classifier
")
    mpl.legend(["Logistic", "kNN", "Baseline"])
    mpl.show()

#USING THE SECOND DATASET NOW!!!

    second_plot(X3, X4, secondTargetVal, "ScatterPlot of Dataset 2")
    secondMeanArrayC, secondStdArrayC = fold10_cVals(secondX, secondTargetVal, cV
als, 'Logistic', 2)
    secondPolyMeanArray = np.array([0]*len(degrees), dtype=float)
    secondPolyStdArray = np.array([0]*len(degrees), dtype=float)
    for n in range(len(degrees)):
        fit = getPoly(degrees[n], secondX, secondTargetVal)
        model = selectClassifier(fit, "Logistic", 1, secondX, secondTargetVal)
        mean, std = fold10_polyFeat(model, fit, secondTargetVal, degrees[n])
        secondPolyMeanArray[n] = mean
        secondPolyStdArray[n] = std

    plot_error(cVals, secondMeanArrayC, secondStdArrayC,"grey","blue", "10-
fold cross validation, mean and  standard deviation of f1 scoring vs C", "Values
of C using Logistic", "F1 scoring mean")
    leg_point0 = mpl.Rectangle((0,0), 1, 1, fc = 'grey')
    leg_point1 = mpl.Rectangle((0,0), 1,1 , fc = 'blue')
    mpl.legend([leg_point0,leg_point1], ["f1 score mean", "Standard Deviation"])
    mpl.show()

    plot_error(degrees, secondPolyMeanArray, secondPolyStdArray,"grey","blue", "1
0-
fold cross validation, f1 scoring vs degrees of Polynomial Features", "Polynomial
Features(n)", "F1 scoring mean")
```

```
    leg_point0 = mpl.Rectangle((0,0), 1, 1, fc = 'grey')
    leg_point1 = mpl.Rectangle((0,0), 1,1 , fc = 'blue')
    mpl.legend([leg_point0,leg_point1], ["Standard Deviation", "f1 scoring mean"]
)
    mpl.show()

#Logistic Regression Classifier using best polyfeature and C Value.
    bestPolyFit = getPoly(2, secondX, secondTargetVal)
    model = LogisticRegression(penalty='l2', solver='lbfgs', C = 0.01)
    model.fit(bestPolyFit, secondTargetVal)
    ypred = model.predict(bestPolyFit)
    print(ypred)
    ax = mpl.subplot()
    trainColor = classifier(secondTargetVal, 'red', 'cornflowerblue')
    predictColour = classifier(ypred, 'black', 'lime')
    for labels in ["Trained data y = 1", "Trained data y = -1"]:
        ax.scatter(X3, X4, c = trainColor, label = labels)
    for labels in ["Predicted data y = 1", "Predicted data y = -1"]:
        ax.scatter(X3, X4, c = predictColour, marker = '+', label = labels, s= 5*
*2)
    mpl.xlabel("feature X1")
    mpl.ylabel("feature X2")
    mpl.title("Logistic Regression Classifier with polyfeature of degree 2 and C
= 0.01")
    mpl.legend(loc = 'lower center')
    ax = mpl.gca()
    leg = ax.get_legend()
    leg.legendHandles[1].set_color("cornflowerblue")
    leg.legendHandles[2].set_color('black')
    leg.legendHandles[3].set_color('lime')
    mpl.show()

#nearest neighbors
    secondXTrain, secondXTest, secondTargetValTrain, secondTargetValTest = train_
test_split(secondX, secondTargetVal, test_size=0.2)
    kVals = np.array([2, 4, 6, 8, 10, 12, 14, 16, 18, 20])
    meanArrayK, stdArrayK = getBestK(kVals, secondXTrain, secondTargetValTrain)
    print(stdArrayK)
    meanPolyK, stdPolyK = getBestKAugmentPoly(kVals, secondXTrain, secondTargetVa
lTrain, 2)
    meanPolyK1, stdPolyK1 = getBestKAugmentPoly(kVals, secondXTrain, secondTarget
ValTrain, 3)

    plot_error(kVals, meanArrayK, stdArrayK, "lime", 'red', "10-
fold cross validation of kNN to determine best k", "k", "f1 scoring mean" )
    leg_point0 = mpl.Rectangle((0,0), 1, 1, fc = 'lime')
    leg_point1 = mpl.Rectangle((0,0), 1,1 , fc = 'red')
```

```
    mpl.legend([leg_point0,leg_point1], ["Standard Deviation", "f1 scoring mean"]
)
    mpl.show()
    plot_error(kVals, meanArrayK, stdArrayK, "lime", 'red', None, None, None )
    plot_error(kVals, meanPolyK1, stdPolyK1, "pink", 'green', None, None, None )
    plot_error(kVals, meanPolyK, stdPolyK, "grey", 'blue', None, None, None )
    mpl.title("kNN using no augmented and augmented polynomial features")
    mpl.xlabel("k = number of neighbors")
    mpl.ylabel("f1 scoring mean")
    mpl.legend(["no Polyfeatures", "PolyFeatures(2)", "PolyFeatures(3)"], loc='lo
wer right')
    mpl.show()

    #using best value of k
    model = KNeighborsClassifier(n_neighbors=8, weights = 'uniform').fit(secondXT
rain, secondTargetValTrain)
    secondYPred = model.predict(secondXTest)
    secondTargetValTest = np.asarray(secondTargetValTest)
    secondTargetValTrain = np.asarray(secondTargetValTrain)
    testColor = classifier(secondTargetValTest, 'red', 'cornflowerblue')
    predictColour = classifier(secondYPred, 'black', 'lime')
    for labels in ["Test data y = 1", "Test data y = -1"]:
        mpl.scatter(secondXTest[:,0], secondXTest[:,1], c = testColor, label = la
bels)
    for labels in ["Predicted Test data y = 1", "Predicted Test data y = -1"]:
        mpl.scatter(secondXTest[:,0], secondXTest[:,1], c = predictColour, marker
 = '+', label = labels)
    mpl.xlabel("feature X1")
    mpl.ylabel("feature X2")
    mpl.title("kNN Classifier k = 10 using Testing Data")
    mpl.gca()
    leg = mpl.legend()
    leg.legendHandles[0].set_color('red')
    leg.legendHandles[1].set_color('cornflowerblue')
    leg.legendHandles[2].set_color('black')
    leg.legendHandles[3].set_color('lime')
    mpl.show()

    #CONFUSTION MATRICES second dataset
    dummyModel = DummyClassifier(strategy="most_frequent").fit(secondXTrain, seco
ndTargetValTrain)
    dummyPred = dummyModel.predict(secondXTest)

    print("Confusion Matrix for Logistic Regression Classifier")
    print(confusion_matrix(secondTargetVal, ypred))

    print("Confusion Matrix for kNN")
    print(confusion_matrix(secondTargetValTest, secondYPred))
```

```
    print("Confusion Matrix for Dummy Classifier")
    print(confusion_matrix(secondTargetValTest, dummyPred))

    #ROC Curves
#Logistic
    Xtrain, Xtest, ytrain, ytest = train_test_split(bestPolyFit, secondTargetVal,
 test_size=0.2)
    logisticModel = LogisticRegression(penalty='l2', solver='lbfgs', C = 1)
    logisticModel.fit(Xtrain, ytrain)
    fprLog, tprLog,_ = roc_curve(ytest, logisticModel.decision_function(Xtest))
    mpl.plot(fprLog, tprLog)
#kNN
    y_scores = model.predict_proba(secondXTest)
    fprknn, tprknn, _ = roc_curve(secondTargetValTest, y_scores[:,1])
    mpl.plot(fprknn, tprknn)
#baseline
    mpl.plot([0, 1], [0, 1], color='green',linestyle='--')
    mpl.xlabel("False Positive Rate")
    mpl.ylabel("True Positive Rate")
    mpl.title("ROC Curve with Logistic Regression and knn and baseline classifier
")
    mpl.legend(["Logistic", "kNN", "Baseline"])
    mpl.show()


if __name__ == "__main__":
    main()
```