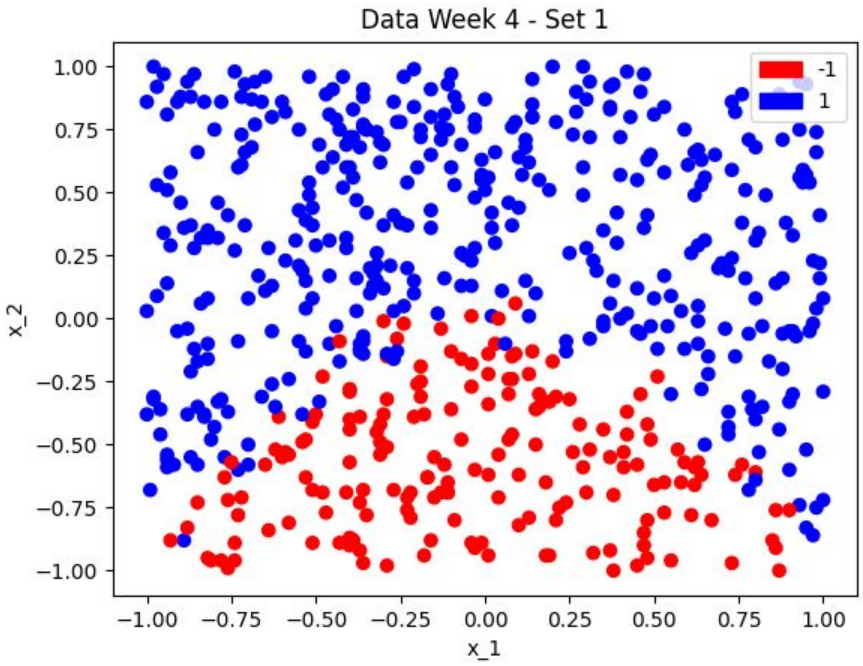
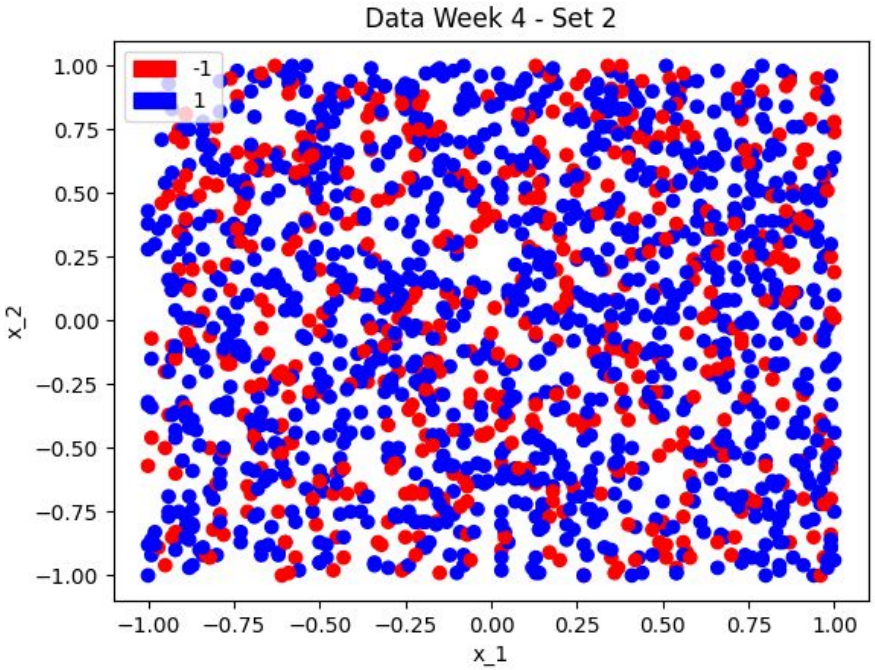
Machine Learning Assignment 4

Saul O'Driscoll (17333932)

DATA USED = # id:22-22-22-0

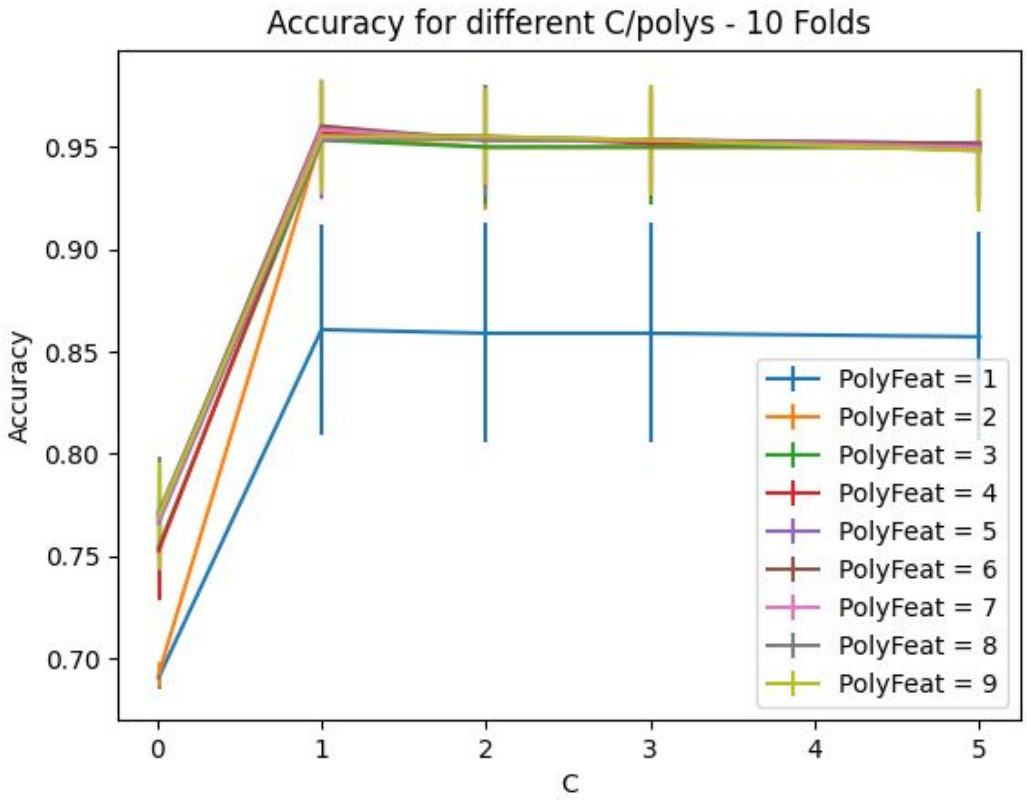




1 a)

Above on the first page I have plotted dataset one and two. Using python I split the data up and saved it in 2 different csv files to make it easier to work with.

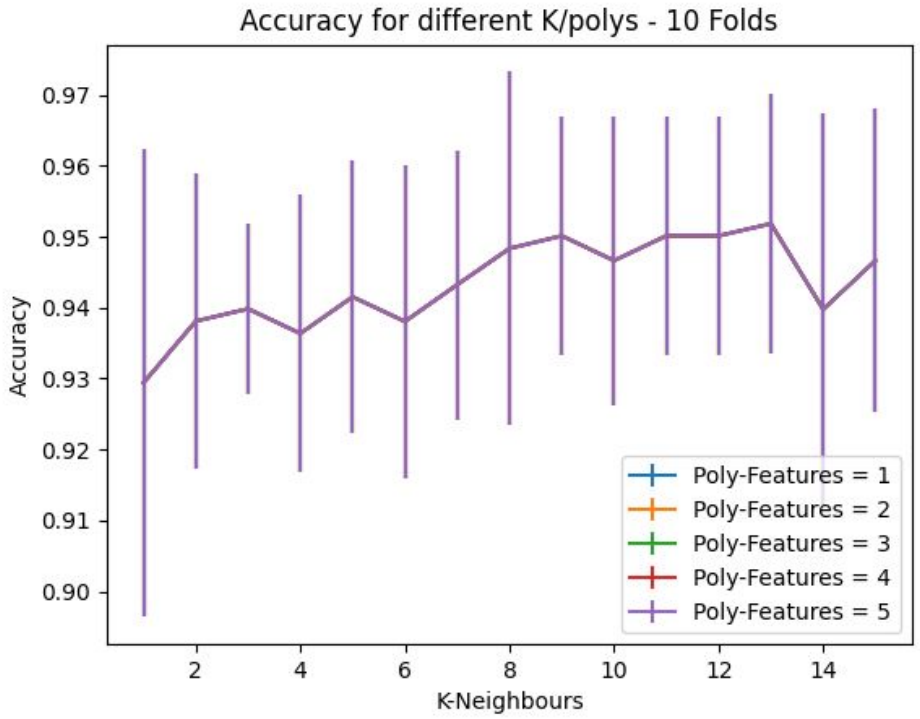
To choose the best C and best order of polynomial I iterated through an array of options for both and tried every combination. I plotted the error bars for both to see which one performed best.



As you can see orders of polynomials higher than 3 performed roughly the same across all Cs. According to the data displayed in text form by my program order 6 and 7 polynomial perform best and C = 1 is the best C for this dataset incl lowest standard dev

```
C: 1 PolyFeatures: 3 -> Accuracy: 0.95 (+/- 0.05)
C: 1 PolyFeatures: 4 -> Accuracy: 0.96 (+/- 0.05)
C: 1 PolyFeatures: 5 -> Accuracy: 0.95 (+/- 0.06)
C: 1 PolyFeatures: 6 -> Accuracy: 0.96 (+/- 0.04)
C: 1 PolyFeatures: 7 -> Accuracy: 0.96 (+/- 0.04)
C: 1 PolyFeatures: 8 -> Accuracy: 0.96 (+/- 0.06)
C: 1 PolyFeatures: 9 -> Accuracy: 0.96 (+/- 0.06)
C: 2 PolyFeatures: 3 -> Accuracy: 0.95 (+/- 0.06)
```

1 B)
To find the best K amount of neighbours and amount of polynomial features for the KN classifier I did the same as in part 1A and iterated through both to compare the accuracy.



As one can see the Knn-classifier does improve slightly with the amount of Ks up to a certain point then falls off. Secondly one can also see that the amount of polynomial features does not matter with this classifier. You can tell because all of the lines overlap and appear at one meaning they are identical across polynomial orders. This means it is not worth augmenting the data by adding polynomial features but it is worth it to change the amount of K-neighbours. In this case I have found that K = 12 and K = 13 perform best. I have decided to go with 12 for validating my classifiers against dummy classifiers in later stages. I have also decided to not use polynomial features at all in later stages.

K: 13 PolyFeatures: 1 -> Accuracy: 0.95 (+/- 0.04)
K: 13 PolyFeatures: 2 -> Accuracy: 0.95 (+/- 0.04)
K: 13 PolyFeatures: 3 -> Accuracy: 0.95 (+/- 0.04)
K: 13 PolyFeatures: 4 -> Accuracy: 0.95 (+/- 0.04)
K: 13 PolyFeatures: 5 -> Accuracy: 0.95 (+/- 0.04)

Accuracy is identical for 2 decimal spaces for the same K across any number of polynomial features.

1C)
I used the sklearn libraries to get the confusion matrix and accuracy of both classifiers along with a baseline dummy classifier which predict the most common class.

KNN Classifier

Logistic KNN Classifier	Predicted Negatives	Predicted Positives
Actual Negatives	62	0
Actual Positives	3	110

Accuracy: 0.9828571428571429

Dummy Classifier	Predicted Negatives	Predicted Positives
Actual Negatives	0	52
Actual Positives	0	123

Accuracy: 0.7028571428571428

Logistic Regression Classifier

Logistic Regression (poly-data)	Predicted Negatives	Predicted Positives
Actual Negatives	47	4
Actual Positives	6	118

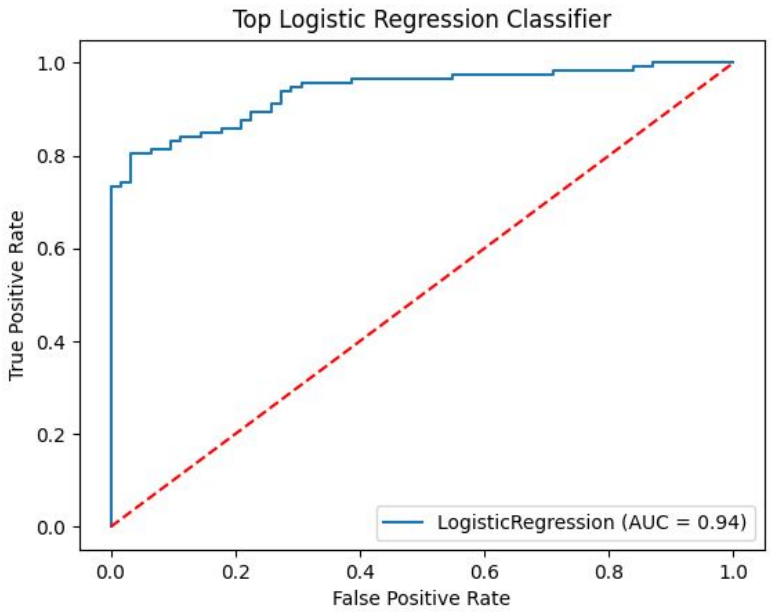
Accuracy: 0.9428571428571428

Dummy Classifier (poly-data)	Predicted Negatives	Predicted Positives
Actual Negatives	0	47
Actual Positives	0	128

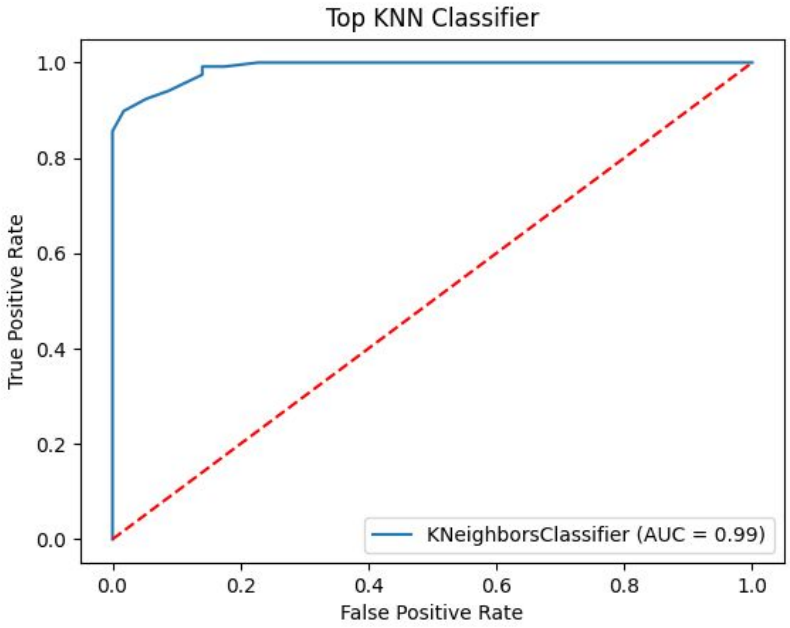
Accuracy: 0.7314285714285714

Above I have listed the confusion matrices and accuracy for 4 different types of classifiers. The data was split 80/20 for training-data and test-data respectively. The first one is the Knn classifier with the best parameter out of the ones that I tested (no polynomial features and K=12). The second one is a dummy classifier that predicts the most common class on the same data. The third one is the best logistic regression classifier with the best parameters from the tests in 1A), C=1 and 6 order polynomial features. The fourth one is a dummy classifier that predicts the most common class on the same data.

1D)



Above is the ROC curve for the logistic regression classifier along with a dummy classifier in red which predicts a random class. As you can see accuracy is quite high around 94%.



Above is the ROC curve for the KNN classifier along with a dummy classifier in red which predicts a random class. As you can see accuracy is quite high around 94%.

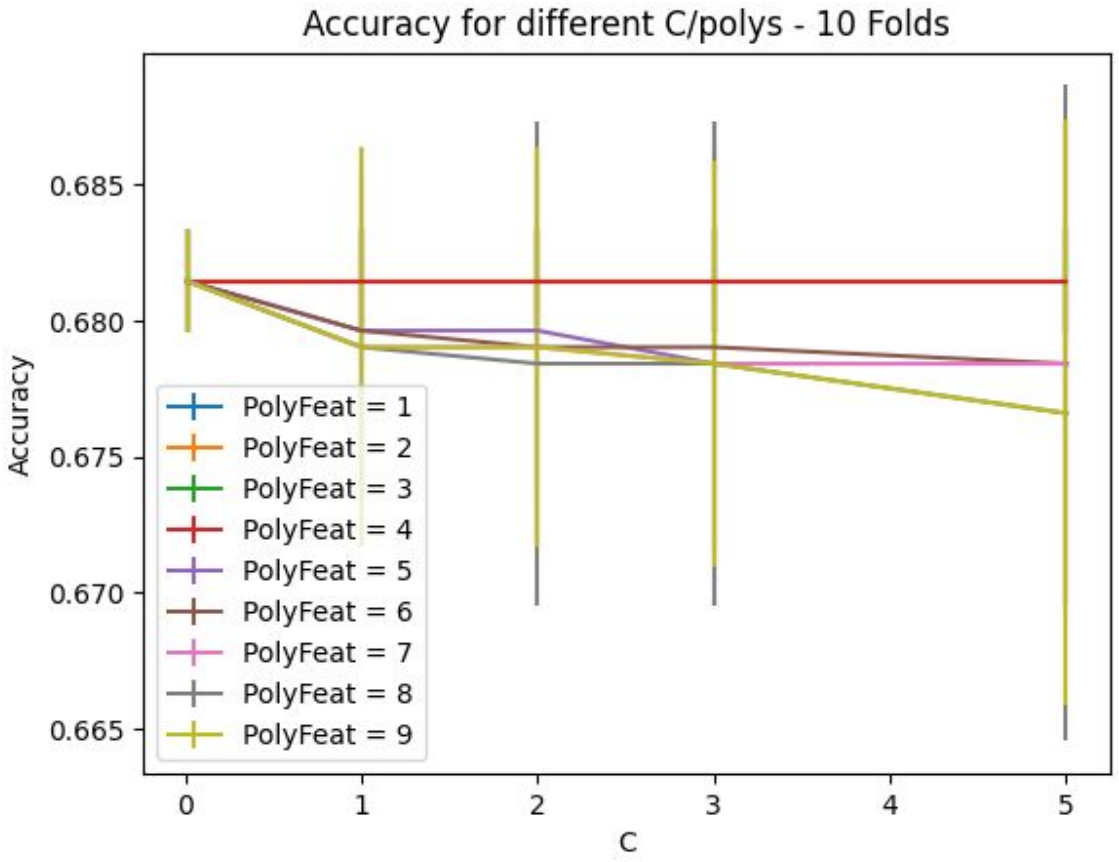
1E)

Both the Logistic regression classifier and the KNN classifier have very high accuracy on the first non-noise dataset. They yield 90%+ results and with tweaking one is able to get an even higher score. They far outperform the baseline classifier. In this case I would still choose the logistic regression classifier because when looking at how the data is spread out one can see that drawing a decision boundary is quite trivial. Logistic regression excels at this. While the KNN ROC curve tends to look better it seems that when we look closely at the numbers the LogReg clf works better and we also have to consider the fact that this depends on how much training data we have as the Logistic regression classifier seems to perform better when we have a lower test to train ratio ie less training examples.

1 a)

Above on the first page I have plotted dataset one and two. Using python I split the data up and saved it in 2 different csv files to make it easier to work with.

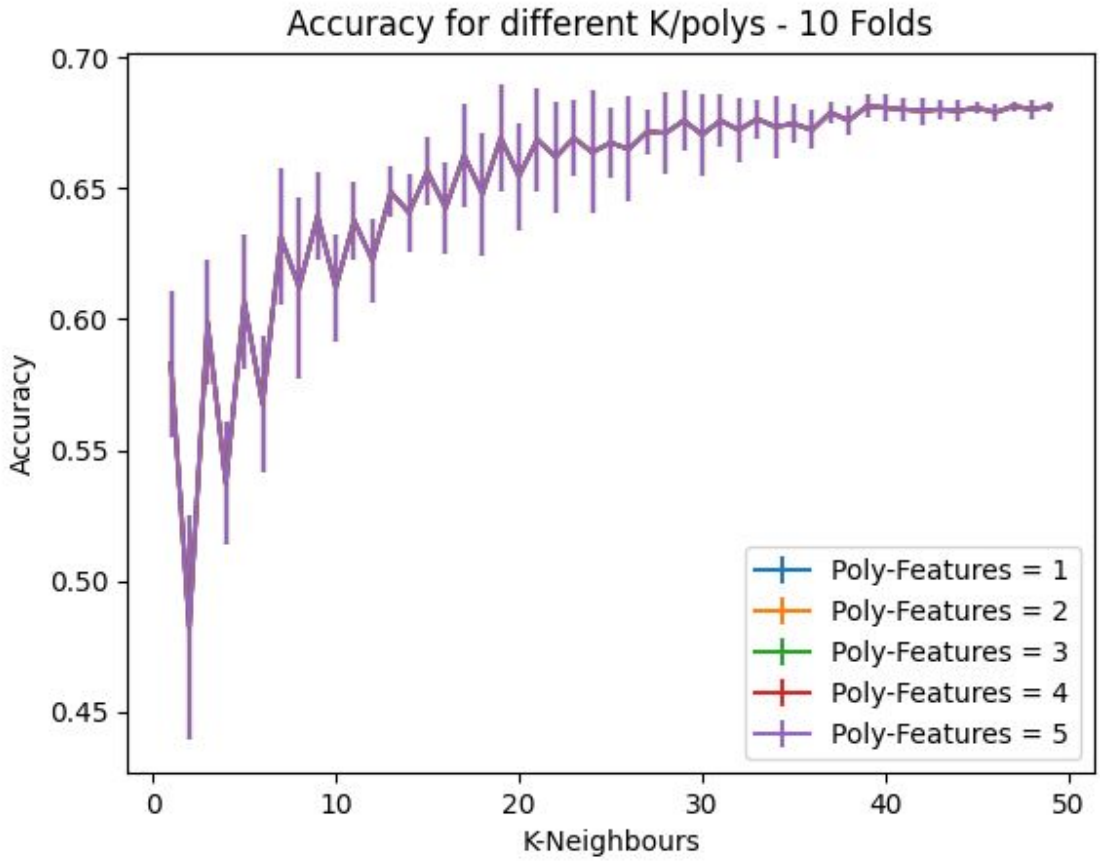
To choose the best C and best order of polynomial I iterated through an array of options for both and tried every combination. I plotted the error bars for both to see which one performed best.



As you can see on this random noisy dataset the Logistic regression algorithm has a hard time. It cannot break bast 70% accuracy and tweaking parameters does not really seem to help when trying to improve the all. According to some data displayed in text form by my program all parameters seem to perform similarly.

```
C: 0.01 PolyFeatures: 4 -> Accuracy: 0.68 (+/- 0.00)
C: 1 PolyFeatures: 4 -> Accuracy: 0.68 (+/- 0.00)
C: 2 PolyFeatures: 4 -> Accuracy: 0.68 (+/- 0.00)
C: 3 PolyFeatures: 4 -> Accuracy: 0.68 (+/- 0.00)
C: 5 PolyFeatures: 4 -> Accuracy: 0.68 (+/- 0.00)
```

1 B)
To find the best K amount of neighbours and amount of polynomial features for the KN classifier I did the same as in part 1A and iterated through both to compare the accuracy.



As one can see the Knn-classifier does improve slightly with the amount of Ks up to a certain point then falls off. Secondly one can also see that the amount of polynomial features does not matter with this classifier. You can tell because all of the lines overlap and appear at one meaning they are identical across polynomial orders. This means it is not worth augmenting the data by adding polynomial features but it is worth it to change the amount of K-neighbours. In this case I have found that K = 40 performs best. Again with this noisy data the classifiers never seem to be able to break past 70% accuracy making them a little better than a random dummy classifier.

K: 35 PolyFeatures: 5 -> Accuracy: 0.67 (+/- 0.02)
K: 40 PolyFeatures: 5 -> Accuracy: 0.68 (+/- 0.01)
K: 41 PolyFeatures: 5 -> Accuracy: 0.68 (+/- 0.01)
K: 42 PolyFeatures: 5 -> Accuracy: 0.68 (+/- 0.01)
K: 43 PolyFeatures: 5 -> Accuracy: 0.68 (+/- 0.01)

Accuracy is identical for 2 decimal spaces for the same K across any number of polynomial features. Polynomial features are not effective in this case

1C)
I used the sklearn libraries to get the confusion matrix and accuracy of both classifiers along with a baseline dummy classifier which predict the most common class.

KNN Classifier

Logistic KNN Classifier	Predicted Negatives	Predicted Positives
Actual Negatives	1	260
Actual Positives	0	562

Accuracy: 0.68408262454435

Dummy Classifier	Predicted Negatives	Predicted Positives
Actual Negatives	0	271
Actual Positives	0	552

Accuracy: 0.6707168894289186

Logistic Regression Classifier

Logistic Regression (poly-data)	Predicted Negatives	Predicted Positives
Actual Negatives	0	263
Actual Positives	0	560

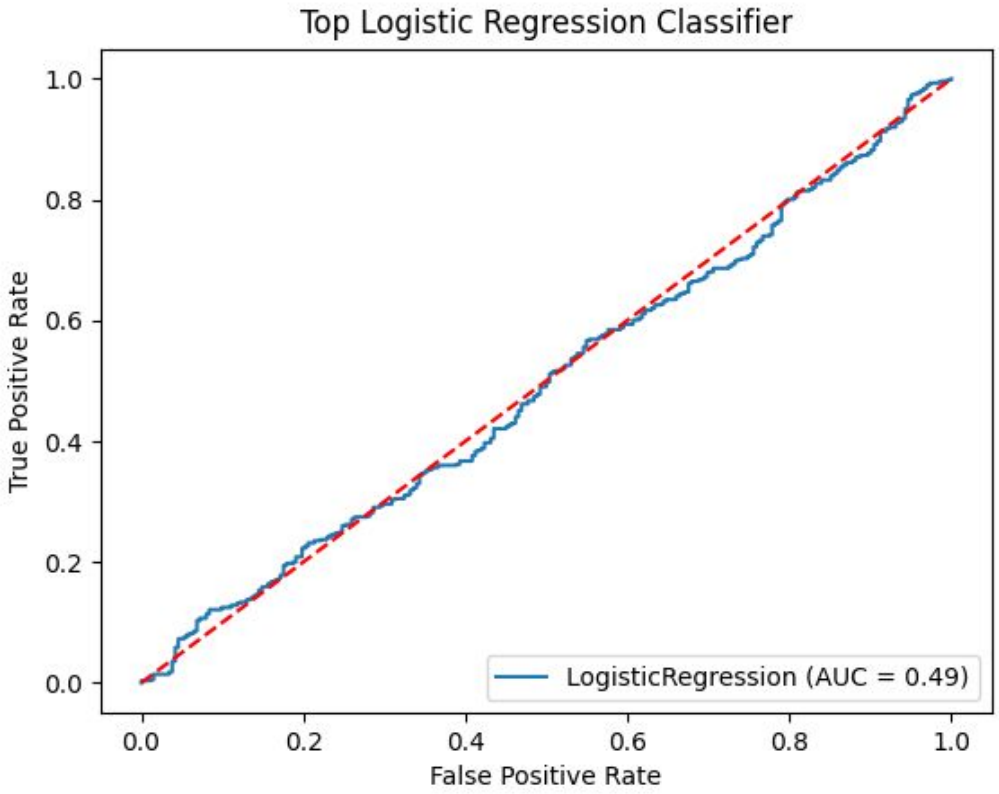
Accuracy: 0.6804374240583232

Dummy Classifier (poly-data)	Predicted Negatives	Predicted Positives
Actual Negatives	0	272
Actual Positives	0	551

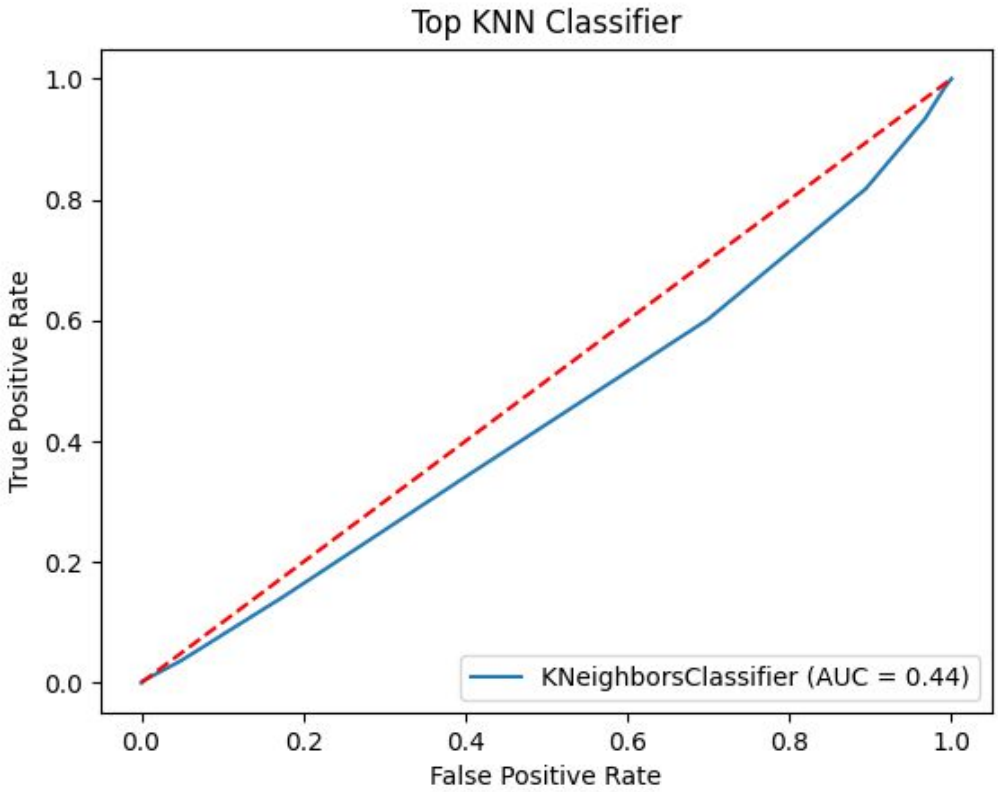
Accuracy: 0.669501822600243

Above I have listed the confusion matrices and accuracy for 4 different types of classifiers. The data was split 80/20 for training-data and test-data respectively. The first one is the Knn classifier with the best parameter out of the ones that I tested (no polynomial features and K=40). The second one is a dummy classifier that predicts the most common class on the same data. The third one is the best logistic regression classifier with the best parameters from the tests in 1A), C=1 and 6 order polynomial features. The fourth one is a dummy classifier that predicts the most common class on the same data.

1D)



Above is the ROC curve for the logistic regression classifier along with a RANDOM dummy classifier in red which predicts a random class. As you can see the AUC is quite low around 0.44.



Above is the ROC curve for the KNN classifier along with a RANDOM dummy classifier in red which predicts a random class. As you can see the AUC is quite low around 0.44.

1E)
Both the Logistic regression classifier and the KNN classifier have accuracy close to the dummy classifier which predicts the most common class. Tweaking them does not help much at all. They do not outperform the baseline classifier by much. In this case I would still choose the KNN classifier because it performs marginally better. Logistic regression excels at this. The ROC curves above are comparing against a random dummy classifier while the numbers before are comparing a against a classifier which predicts the most common class. The models don't change much based on how much training vs test data you show them.

CODE FOR WEEK 4

```
## id:22-22-22-0
# Saul O'Driscoll 17333932

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics._plot.roc_curve import plot_roc_curve
from sklearn.dummy import DummyClassifier

dummyClf = DummyClassifier(strategy = "most_frequent")

#Indicator of which dataset to use
dataset_switcher = 1

#Creating new files for the split datasets
dataset = open("data4.csv", "r")
dataset_1 = open("data4_1.csv", "w+")
dataset_2 = open("data4_2.csv", "w+")

#splitting the dataset
content = dataset.read()
content_list = content.split("# ")
dataset.close()

content_list[0] = content_list[1]
content_list[1] = content_list[2]

content_list[0] = "# " + content_list[0]
content_list[1] = "# " + content_list[1]

#writing content to the new dataframes
```

```
dataset_1.write(content_list[0])
dataset_2.write(content_list[1])
dataset_1.close()
dataset_2.close()

#reading dataset 1 from new file
df1 = pd.read_csv("data4_1.csv",header=None, comment="#",
names=["X1","X2","X3"])
dataset_1_X_all = df1.drop(columns=["X3"])
dataset_1_X_labels = df1["X3"]

#reading dataset 2 from new file
df2 = pd.read_csv("data4_2.csv",header=None, comment="#",
names=["X1","X2","X3"])
dataset_2_X_all = df2.drop(columns=["X3"])
dataset_2_X_labels = df2["X3"]

#assigning X and y to dataset that was selected earlier
if (dataset_switcher == 1):
   X = dataset_1_X_all
   y = dataset_1_X_labels
else:
   X = dataset_2_X_all
   y = dataset_2_X_labels

def displayDataSet():
   colors=['red' if l == -1 else 'blue' for l in y.values]
   plt.scatter(X["X1"], X["X2"], color=colors)
   plt.title("Data Week 4 - Set {0}".format(dataset_switcher))
   plt.xlabel("x_1")
   plt.ylabel("x_2")
   red_patch = mpatches.Patch(color='red', label='-1')
   blue_patch = mpatches.Patch(color='blue', label='1')
   plt.legend(handles=[red_patch, blue_patch])

   plt.show()

# displayDataSet()
```

```
#Poly Feature Finder for seleting the best C and best Polynomial feature
for logistic regression
def polyFeatureFinder():
   cArray = [0.01, 1, 2, 3, 5]
   polyArray = []

   for i in range(1,10,1):
       polyArray.append(i)

   for p in polyArray:
       polyAccuracy_mean_array = []
       polyAccuracy_std_array = []
       for c in cArray:
           poly = PolynomialFeatures(p)
           polydata = poly.fit_transform(X)

           clf = LogisticRegression(C=c)
           scores = cross_val_score(clf, polydata, y, cv=8)

           polyAccuracy_mean_array.append(scores.mean())
           polyAccuracy_std_array.append(scores.std())

           print("C: " + str(c) + " PolyFeatures: " + str(p) +  " ->
Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

           # print(len(cArray), len(polyAccuracy_mean_array))
       plt.errorbar(cArray, polyAccuracy_mean_array,
yerr=polyAccuracy_std_array, label="PolyFeat = {0}".format(p))

   plt.xlabel("C")
   plt.ylabel("Accuracy")
   plt.title("Accuracy for different C/polys - 10 Folds")
   plt.legend()
   plt.show()

   # Best score
   """
   C: 1 PolyFeatures: 3 -> Accuracy: 0.95 (+/- 0.05)
   C: 1 PolyFeatures: 4 -> Accuracy: 0.96 (+/- 0.05)
   C: 1 PolyFeatures: 5 -> Accuracy: 0.95 (+/- 0.06)
```

```
    C: 1 PolyFeatures: 6 -> Accuracy: 0.96 (+/- 0.04)
    C: 1 PolyFeatures: 7 -> Accuracy: 0.96 (+/- 0.04)
    C: 1 PolyFeatures: 8 -> Accuracy: 0.96 (+/- 0.06)
    C: 1 PolyFeatures: 9 -> Accuracy: 0.96 (+/- 0.06)
    C: 2 PolyFeatures: 3 -> Accuracy: 0.95 (+/- 0.06)
    """

def knnKFinder():
    kArray = []
    polyArray = []

    for i in range(1,50):
        kArray.append(i)

    poly = PolynomialFeatures(6)
    polydata = poly.fit_transform(X)
    for i in range(1,6,1):
        polyArray.append(i)

    for p in polyArray:
        knnAccuracy_mean_array = []
        knnAccuracy_std_array = []
        for k in kArray:
            clf = KNeighborsClassifier(k)

            scores = cross_val_score(clf, polydata, y, cv=8)
            knnAccuracy_mean_array.append(scores.mean())
            knnAccuracy_std_array.append(scores.std())

            print("K: " + str(k) + " PolyFeatures: " + str(p) + " ->
Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
            # print(len(cArray), len(polyAccuracy_mean_array))
        plt.errorbar(kArray, knnAccuracy_mean_array,
yerr=knnAccuracy_std_array, label="Poly-Features = {0}".format(p))

    # plt.errorbar(cArray, polyAccuracy_mean_array,
yerr=polyAccuracy_std_array, label="Accuracy + std deviation")
    plt.xlabel("K-Neighbours")
    plt.ylabel("Accuracy")
    plt.title("Accuracy for different K/polys - 10 Folds")
```

```
    plt.legend(loc=4)
    plt.show()

    #Best Scores
    """
    K: 12 -> Accuracy: 0.950 (+/- 0.03)
    K: 13 -> Accuracy: 0.952 (+/- 0.04)
    """



#Evaluate classifier with confusion matrix and accuracy score
def evaluateClf(clf, X, y):
    X_Train, X_Test, y_Train, y_Test = train_test_split(X, y,
test_size=0.5)
    clf.fit(X_Train, y_Train)

    yPred = clf.predict(X_Test)
    conf_matrix = confusion_matrix(y_Test, yPred)
    accuracy = accuracy_score(y_Test, yPred)
    return conf_matrix, accuracy

#plot ROC curve (does split data into train and test)
def ROCCurve(title,clf,X,y):
    X_Train, X_Test, y_Train, y_Test = train_test_split(X, y,
test_size=0.5)

    clf.fit(X_Train, y_Train)
    yPred = clf.predict(X_Test)

    plot_roc_curve(clf, X_Test, y_Test)
    plt.title(title)
    plt.plot([0,1],[0,1],color="red",linestyle="--")
    plt.show()

def bestKnnDisplay():
    k = 12 # change to 40 for dataset 2
    clf = KNeighborsClassifier(k)
    matrix, accuracy = evaluateClf(clf, X, y)
    print("Best KNN Classifier ")
```

```
    print("Confusion Matrix:\n" + str(matrix) + " -> Accuracy: " +
str(accuracy))
    matrix_dummy, dummy_accuracy = evaluateClf(dummyClf, X, y)
    print("Dummy Classifier ")
    print("Confusion Matrix:\n" + str(matrix_dummy) + " -> Accuracy: " +
str(dummy_accuracy))
    ROCCurve("Top KNN Classifier", clf, X, y)

def bestLogRegDisplay():
    features = 6
    poly = PolynomialFeatures(features)
    polydata = poly.fit_transform(X)

    clf = LogisticRegression(C=1)
    matrix, accuracy = evaluateClf(clf, polydata, y)
    print("Best Logistic Regression Classifier ")
    print("Confusion Matrix:\n" + str(matrix) + " -> Accuracy: " +
str(accuracy))
    matrix_dummy, dummy_accuracy = evaluateClf(dummyClf, polydata, y)
    print("Dummy Classifier ")
    print("Confusion Matrix:\n" + str(matrix_dummy) + " -> Accuracy: " +
str(dummy_accuracy))
    ROCCurve("Top Logistic Regression Classifier", clf, X, y)

polyFeatureFinder()
knnKFinder()
bestKnnDisplay()
bestLogRegDisplay()
```