Machine Learning Assignment 4

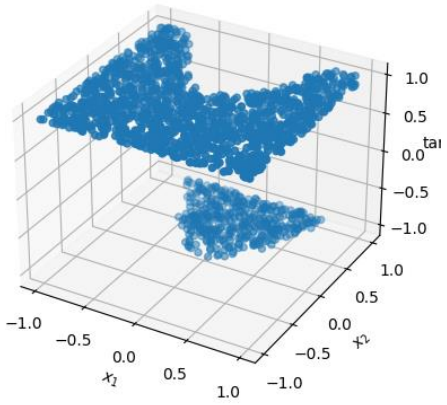
Dataset 1 – # id:13-26--13-0

Dataset 2 - # id:13-13--13-0

(i)

(a)

Firstly, the data was plotted on a 3D plot to gain a better understanding of the dataset.



The data is split by a curve into two sections on a different dimension. A model that can capture non-linearity is likely to have more success over others.

The goal was to train a logistic regression model on the data with an L2 penalty and polynomial features. Cross validation was used with 5-folds to select the values for C in the penalty parameter and the degree of polynomial features.

Nested for loops, iterating over the range of polynomial features and values for C, were used to achieve this:

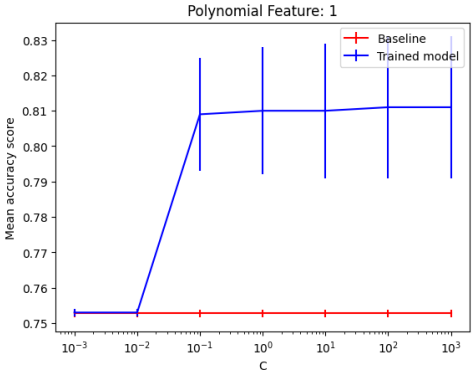
```
for i in poly_range:
     mean_error = []
     std_error = []
     X_poly = PolynomialFeatures(i).fit_transform(X)
     for j in c_range:
      model = LogisticRegression(C=j, penalty='l2')
      model.fit(X_poly, y)
      scores = cross_val_score(model, X_poly, y, cv=5, scoring='accuracy')
```

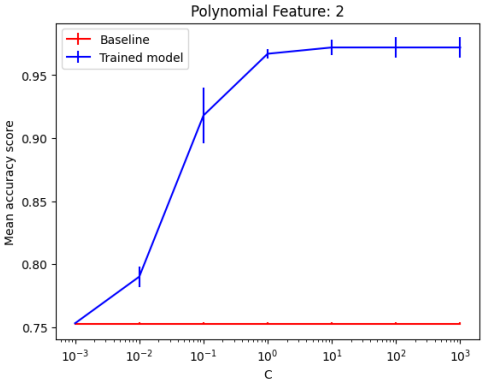
This generates a list of mean errors for each value of C tested on a level of polynomial features. These were then graphed against the values of C to see which values were most effective. When selecting the range of C values, a wide range of [0.001,0.01,0.1,1,10,100,1000] was chosen to help narrow down on the optimal value. The range of [1,2,3,4,5,6] was less wide for the polynomial features based on the initial graph of data. Other higher values (10,20) were tested but resulted in less accurate models. The metric used was accuracy, which is the total number of correct predictions over the total number of predictions.

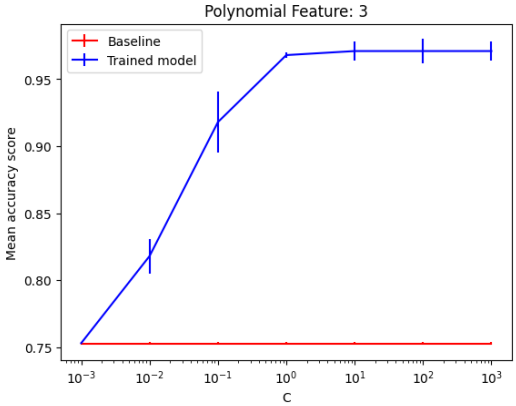
A baseline predictor that chose the most frequent class was created to compare against the performance of the of the logistic regression model. This was chosen as the initial graph of the data showed one class was reasonably larger than the other and therefore this baseline predictor will be reasonably strong.

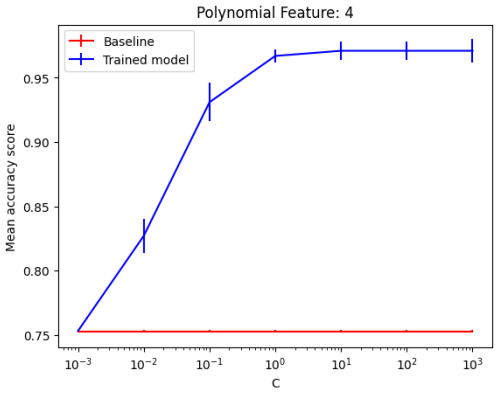
The average accuracy and standard error were plotted against the range of values chosen for C. Here are the mean accuracy values for each value of C and polynomial degree.

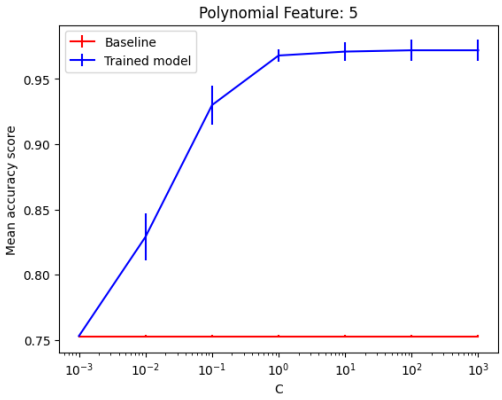
Degree	C=0.001	C=0.01	C=0.1	C=1	C=10	C=100	C=1000
1	.753	.753	.809	.81	.81	.811	.811
2	.753	.79	.918	.967	.972	.972	.972
3	.753	.827	.931	.967	.971	.971	.971
4	.753	.829	.93	.968	.971	.972	.972
5	.753	.829	.93	.968	.971	.972	.972
6	.753	.83	.931	.968	.972	.972	.972

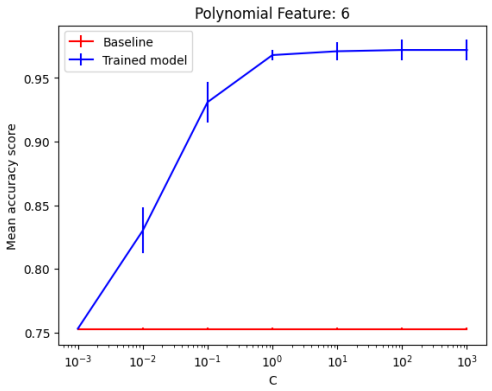












Each model easily eclipsed the accuracy of the baseline model. All values of C <1 clearly underperformed greater values of C. The mean accuracy was the same for values of C greater than 1 and for all polynomial features. The only difference between these iterations of models were differences in the standard error.

Degree	C=1	C=10
1	0.018	0.019
2	0.004	0.006
3	0.002	0.007
4	0.005	0.007
5	0.005	0.005
6	0.004	0.007

The most optimal value appears to be degree three, C=1 model which has a mean accuracy of 0.967 and a standard error of 0.002 or a degree 2 polynomial, C=10 model with a mean accuracy of 0.972 and a standard error of 0.006. Due to the smaller difference in mean accuracy, the model chosen was the degree three, C=1 model as its extremely low standard error means it stable and might best adapt to unseen data.

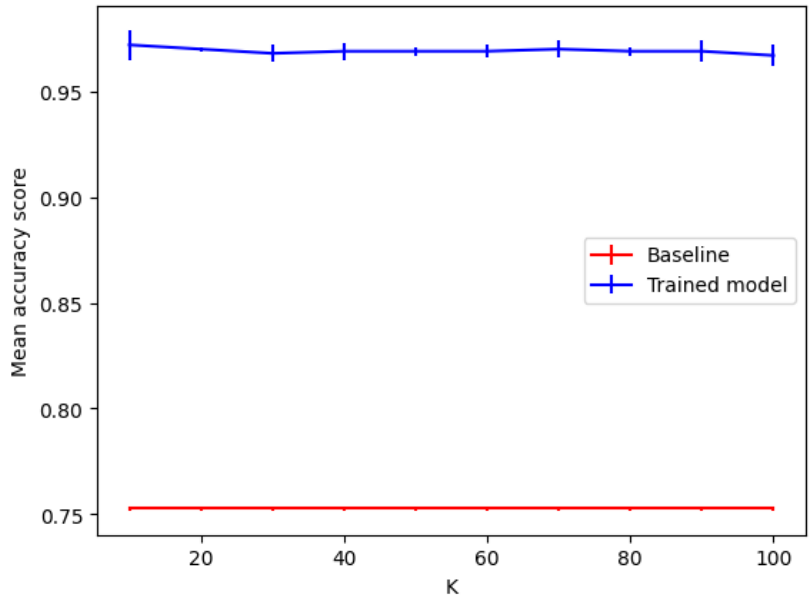
(b)

The goal was to train a K-nearest neighbours model using cross-validation.

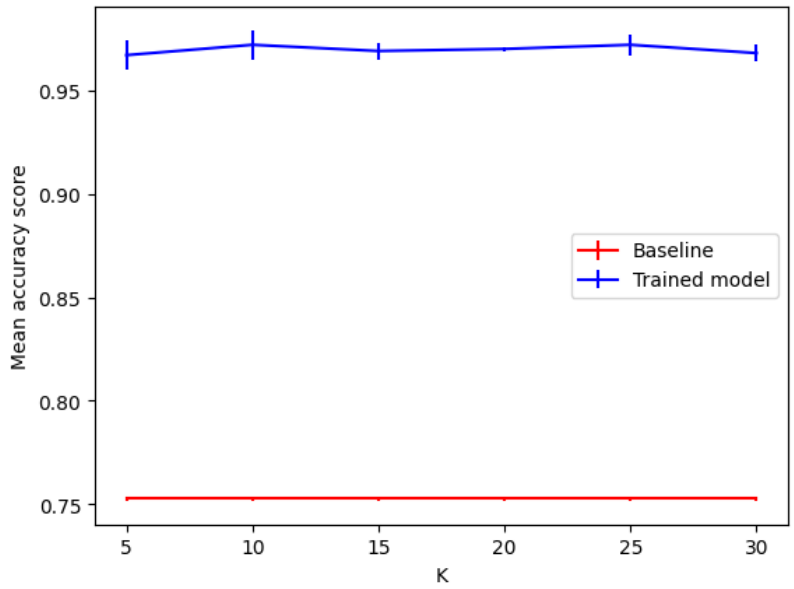
Five-fold cross validation was used to test for optimal values of k. Accuracy was again the chosen metric. The following code was used to generate the mean error and standard error for given ranges of k:

```
for k in k_range:
        model = KNeighborsClassifier(n_neighbors=k)
        model.fit(X,y)
        scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
        mean_error.append(round(scores.mean(),3))
        std_error.append(round(scores.std(),3))
```

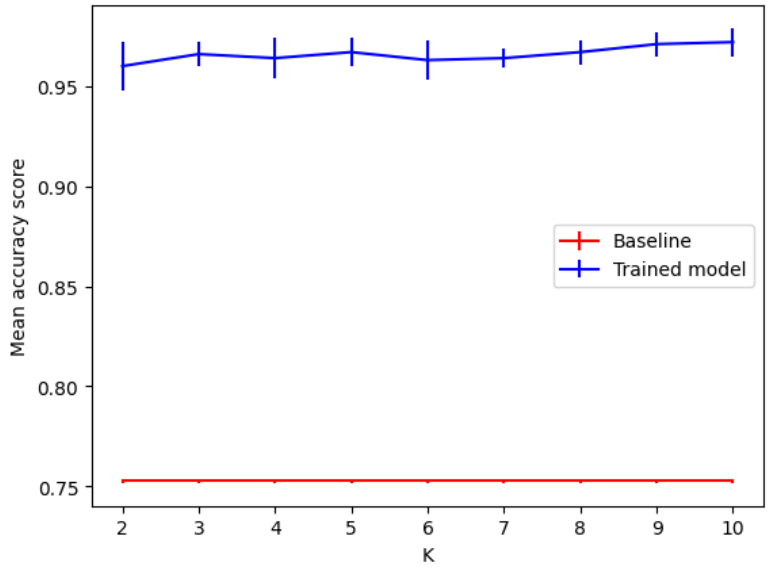
Three different ranges of K were tested to identify narrow down on which values of k were optimal. Firstly, a wider range of [10,20,30,40,50,60,70,80,90,100] was chosen. The mean and standard accuracy were plotted against the chosen values of K. A baseline model which always chose the most frequent class was chosen and graphed for comparison.



Here the trained model immediately outperforms the baseline. It is also more accurate than initial values for the unoptimized logistic regression model. Standard error was also low with all values <0.1. Its also clear that that the optimal value for K, was < 20. Therefore, a smaller range of [5,10,15,20,25,30] was tested to narrow down on the correct value of K.



This graph identified the optimal value of K as <=10. Therefore, the last range tested was [2,3,4,5,6,7,8,9,10].



The optimal value for k was clearly 10, which had a standard error of 0.005 and a mean accuracy of 0.972.

(c)

For the optimal versions of the selected logistic regression and k-nearest neighbours' models, confusion matrixes were generated from sklearn's metrics module. The following code was used to generate the matrix's:

```
X_train, X_test, y_train, y_test = train_test_split(X_poly, y)

  opt_log = LogisticRegression(C=0.1, penalty='l2')
  opt_log.fit(X_train, y_train)
  y_pred_log = opt_log.predict(X_test)
  print("Confusion Matrix for Logistic Regression:")
  print(confusion_matrix(y_test, y_pred_log))
```

Training sets and test sets were employed to maintain validity. The test sets had a chosen size of 0.3. Confusion matrix's were also generated for the baseline predictor.

Confusion Matrix for Logistic Regression:

TP: 110	FN: 7
FP: 3	TN: 314

Confusion Matrix for k-Nearest Neighbours:

TP: 108	FN: 9
FP: 5	TN: 312

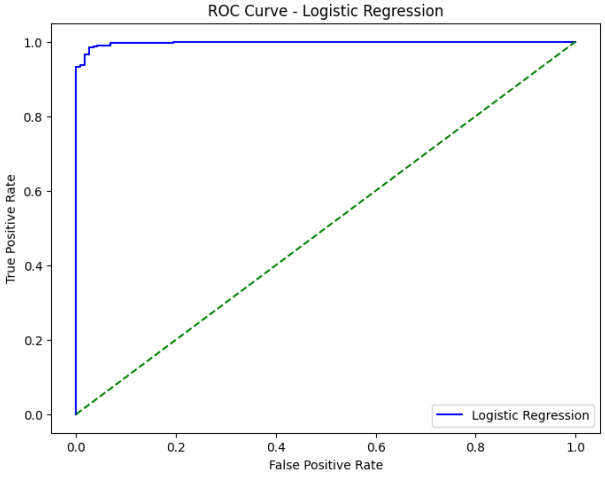
Confusion Matrix for Dummy model:

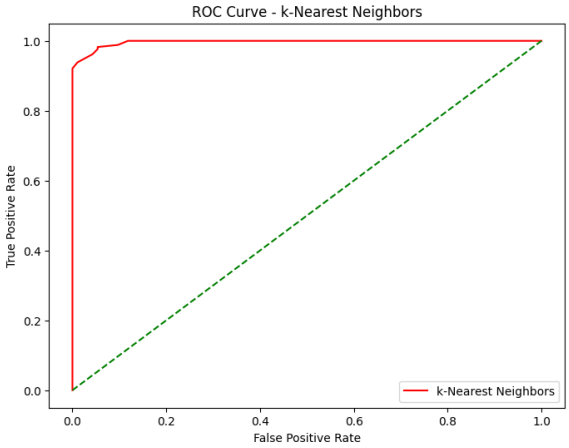
TP: 0	FN: 115
FP: 0	TN: 319

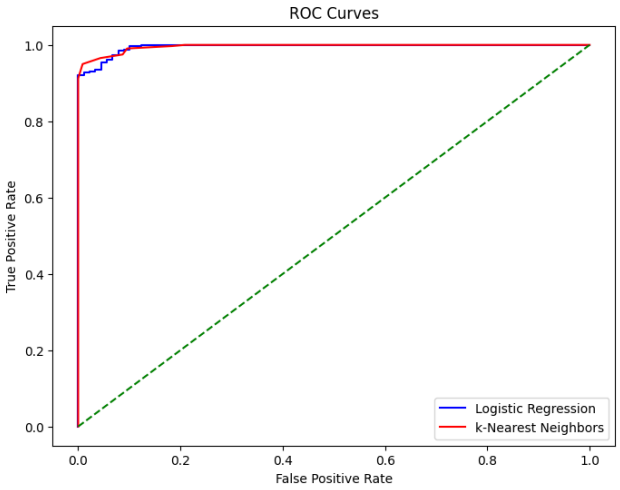
Logistic regression had an accuracy of 97.7% and a true positive rate of 97.3%. The k-nearest neighbours had an accuracy of 96.7% and a true positive rate of 95.6%. These scores are considerably higher than dummy classifier which had an accuracy of 63.94%. The dummy classifier had a true positive rate of 0% as it always predicted negative (the most common class).

(d)

The ROC curve graphing the true positive rate was graphed for each model separately and then put together on the same curve for comparison.







The baseline model draws a line completely across the middle of the graph as it has both a true positive and false positive rate of 0%. This contrasts with the other models which are extremely close to the top left corner, showing a true positive rate of almost 100%.

(e)

Based on the data generated in parts (c) and (d), both models considerably improved over the dummy model and were highly effective. Both models have extremely high accuracies and true positive rates

when compared to the dummy model (+30%). The ROC curve is extremely close to the top left corner indicating an almost perfect true positive rate.

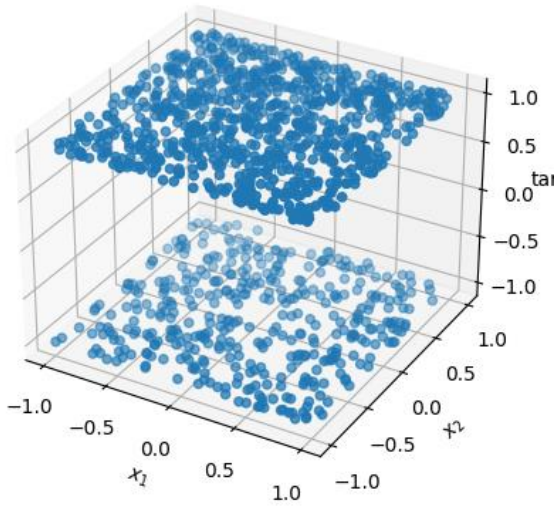
When comparing the two it appears logistic regression is preferable by a small margin. Not only did it have higher accuracy, but it had less false positives and less false negative then K-nearest neighbours meaning it is preferable regardless of what the classes represent. It also had a slightly lower standard error, and which means it could likely handle unseen data better. That being said, both models fit the data extremely well and are both valid choices.

(ii)

For part (ii), the same code was used to generate the models, graphs, and metrics as in part (i). The same baseline model that chose the most frequent class was also used. The only adjustments made were in the ranges of values chosen for testing for the relevant hyperparameters. Thus, I will not include the same code snippets for the sake of brevity.

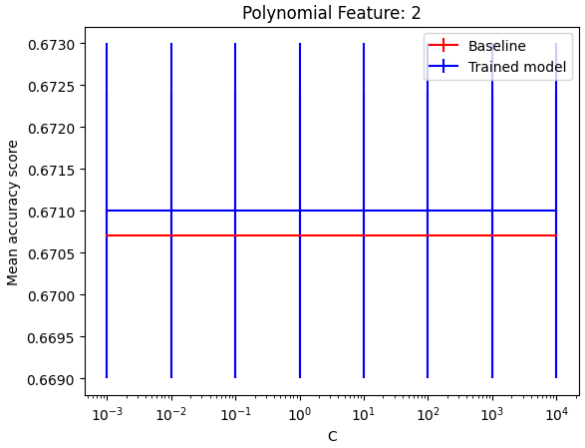
(a)

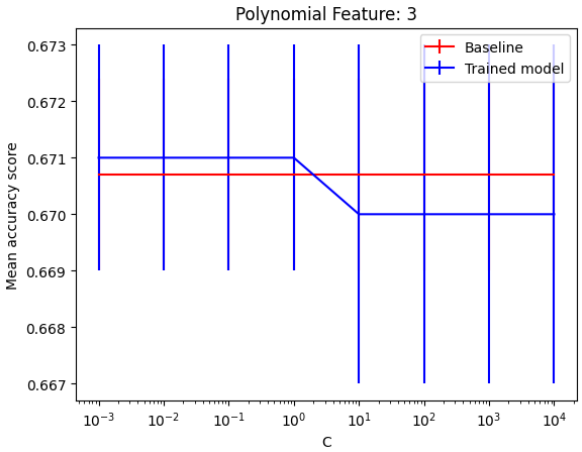
The data from the second dataset has no discernible visual pattern unlike the data from part one. The data is split seemingly randomly between +1 and -1. It is clear from the graph, that +1 had more entries then the other class.

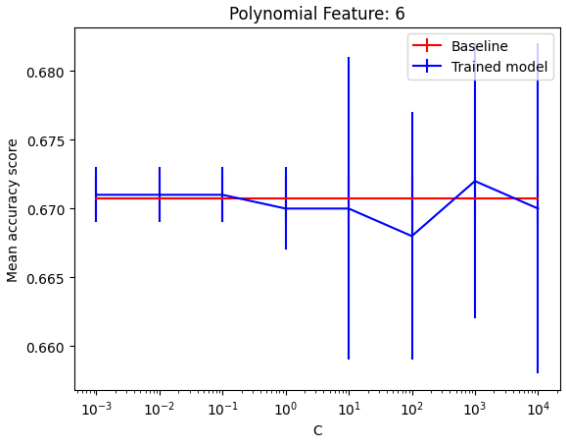


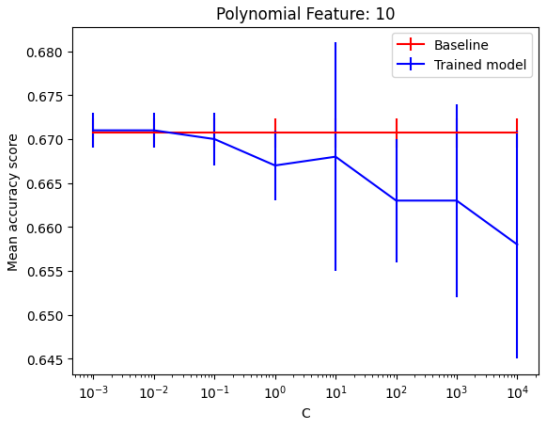
The same initial range for polynomial features of [1,2,3,4,5,6] was chosen for the second dataset. However, as the model was not performing well wider values [10,15,20], were tested but did not result in any notable improvement.

This is also true for the range chosen for the penalty parameter C, [0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]. Increasing C past 10 resulted in drops in accuracy and increased standard error.







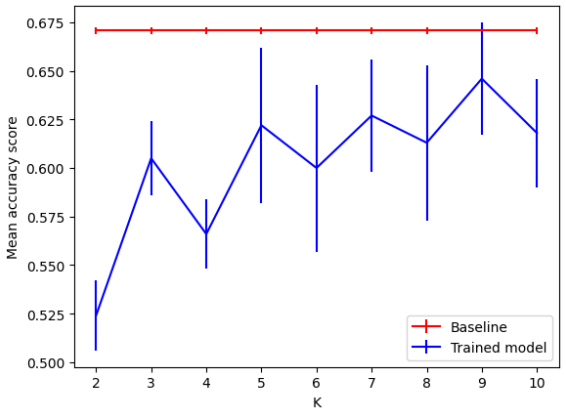


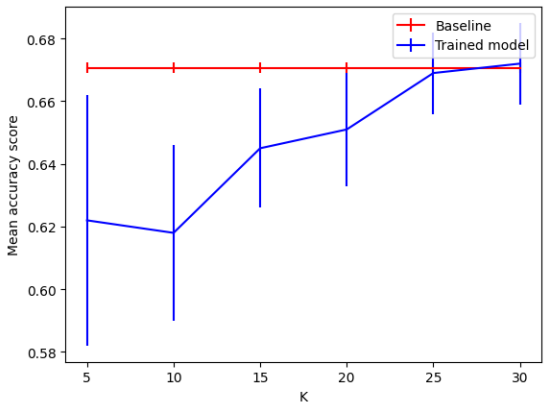
None of the iterations of the model ever outperforms the dummy model. It has slightly higher accuracy at points, but the standard error is so high it can be dismissed. We can see there is little difference when the hyperparameters change. The only meaningful difference is as C increases when higher polynomial features are used there is a drastic drop off in accuracy.

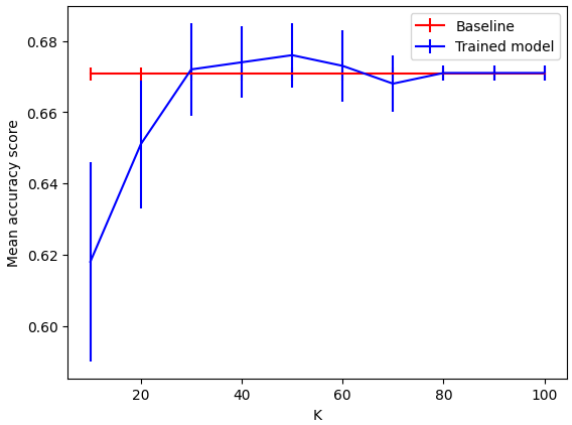
The optimal model was polynomial 7 and C=0.1, which performed at the same accuracy as the baseline although the standard error was higher.

(b)

The K-nearest neighbours' model did not perform better in any significant way then the logistic regression model. The same initial ranges for k were tested as part one ,[10,20,30,40,50,60,70,80,90,100], however different narrower ranges tested due to a different optimal value for K. The narrower range tested were between 40 and 80.







For lower values of K (< 30), the model performed worse than the dummy classifier. As K increased the optimal range was between 60 and 80, however it only brought it on par with the dummy classifier.

The optimal value for K was 50, with a mean accuracy of 0.687.

(c)

The following confusion matrix's were generated for the selected models:

Confusion Matrix for Logistic Regression:

TP: 0	FN: 106
FP: 0	TN: 204

Confusion Matrix for k-Nearest Neighbours:

TP: 19	FN: 71
FP: 52	TN: 168

Confusion Matrix for Dummy model:

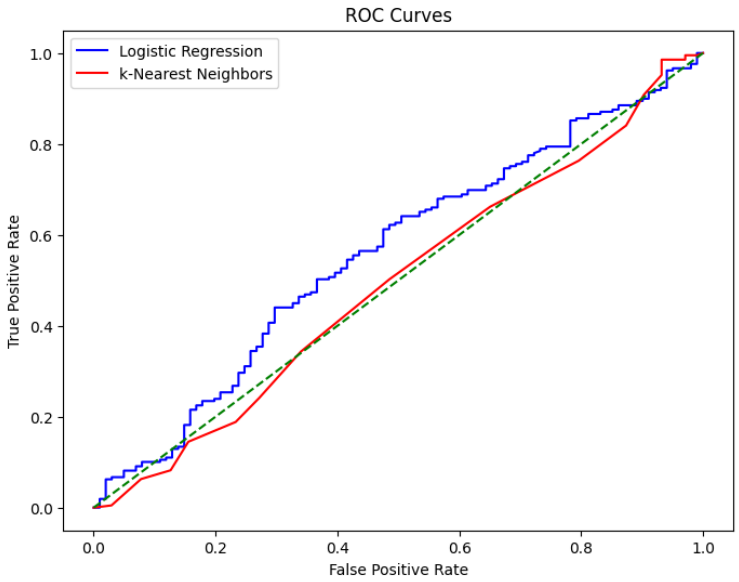
TP: 0	FN: 103
FP: 0	TN: 207

The accuracy for the logistic regression model was 65.8%. The accuracy for the k-nearest neighbours' model was 60.03%. The true positive rate was 26.8%.

Neither had a higher accuracy then the dummy model which had 66.8%.

(d)

The ROC curve for both models are shown below.



The K-nearest Neighbours model either mirrored the baseline (which has a true positive rate of 0%) or performed worse. The logistic regression model does slightly better but still is extremely close to the middle representing almost 0 correct positive predictions.

(e)

Both models had a lower mean accuracy than the dummy model (which always predicted the most common class). The models also had a high standard error as seen in the training graphs, which means the results are inconsistent as well as inaccurate. It seems there was no distinguishing features that acted as predictors for the models, even when polynomial features were considered. It is difficult to know whether another model could be effective at capturing the data without knowing more about the dataset.

I would not recommend the logistic regression or k-nearest neighbours' model as neither captured the data well. Both models were able to handle non-linearity and still could not beat the baseline predictors. I would be hesitant to recommend any other machine learning models as they are unlikely to be able to predict the data based on the visualisation created and would be increasingly computationally expensive. Depending on what the data is based on, you could possibly improve on the baseline slightly, maybe by always picking the same class unless a certain condition is reached or if the data is a series of recordings using time-forecasting to find a pattern.

Appendix

Code was written in a Jupyter notebook, paragraphs represent the different code blocks

```
# - import datasets
import pandas as pd
import numpy as np

data1 = pd.read_csv("dataset1.csv"
```

```
# - Using sklearn augment the two features in the dataset with polynomial
features
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import KFold
from sklearn.dummy import DummyClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve
import numpy as np
import warnings
# Suppress ConvergenceWarnings
warnings.filterwarnings("ignore", category=UserWarning, module="sklearn")


# Takes in a dataframe and breaks it up into variables for the models
def data(df1):
  X1 = df1.iloc[:,0]
  X2 = df1.iloc[:,1]
  Y = df1.iloc[:,2]
  X = np.column_stack((X1,X2))
  return X,Y

def plot_features(df):
   #Didn't use data function here becuase didn't want X1 X2 in one variable
    X1 = df.iloc[:,0]
    X2 = df.iloc[:,1]
    Y = df.iloc[:,2]
    fig = plt.figure()
    # Creates a 3D scatter plot of the data
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(X1, X2,Y )
    ax.set_xlabel('$x_1$')
```

```
    ax.set_ylabel('$x_2$')
    ax.zaxis.set_rotate_label(False)
    ax.set_zlabel('target', rotation=0)
    plt.show()

#  # Using the dummyc classifier model to generate mean accuracys and standard
error for the data
def baselines(X,y,length):
    idiot = DummyClassifier(strategy='most_frequent')
    idiot_scores = cross_val_score(idiot, X, y, cv=5, scoring='accuracy')
    idiot_mean, idiot_std = idiot_scores.mean(), idiot_scores.std()
    idiot_mean = [idiot_mean] * length
    idiot_std = [idiot_std] * length
    return idiot_mean, idiot_std

def test_logistic(poly_range, c_range, df1):
    # Store data from dataframe
    X, y = data(df1)
    #generate baselines
    idiot_mean, idiot_std = baselines(X,y,len(c_range))

    # outerloop controlled by range of polynomial features
    for i in poly_range:
     mean_error = []
     std_error = []
     X_poly = PolynomialFeatures(i).fit_transform(X)
     # inner loop controlled by range of C values
     for j in c_range:
      model = LogisticRegression(C=j, penalty='l2')
      model.fit(X_poly, y)
      # mean accuracy calculated in the inner loop, and reset in the outer
loop
      scores = cross_val_score(model, X_poly, y, cv=5, scoring='accuracy')
      mean_error.append(round(scores.mean(),3))
      std_error.append(round(scores.std(),3))

     # Plot the mean accuracies versus C for each polynomial degree
     plt.errorbar(c_range,idiot_mean,yerr=idiot_std,fmt='r')
     plt.errorbar(c_range, mean_error, yerr=std_error, fmt='b')
     plt.xlabel('C')
     plt.ylabel('Mean accuracy score')
     plt.xscale('log')
     plt.legend(['Baseline','Trained model'])
     plt.title(f'Polynomial Feature: {i}')
     plt.show()
     print(f'{i}:  {std_error}')

    return mean_error, std_error
```

```
plot_features(data1)
```

```
p_range = [1,2,3,4,5,6]
c_range = [1,10,100,1000]
mean = []
std = []
mean,std,  = test_logistic(p_range,c_range,data1)
```

```
# Tests k, same logic used as before just one loop
def test_knn(k_range,df):
    one = True
    X, y = data(df)
    idiot_mean, idiot_std = baselines(X,y,len(k_range))
    mean_error = []
    std_error = []
    for k in k_range:
        model = KNeighborsClassifier(n_neighbors=k)
        model.fit(X,y)
        scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
        mean_error.append(round(scores.mean(),3))
        std_error.append(round(scores.std(),3))

    print(mean_error)
    print(std_error)
    print
    plt.errorbar(k_range,idiot_mean,yerr=idiot_std,fmt='r')
    plt.errorbar(k_range, mean_error, yerr=std_error, fmt='b')
    plt.xlabel('K')
    plt.ylabel('Mean accuracy score')

    plt.legend(['Baseline','Trained model'])
    plt.show()
```

```
k_range = [10,20,30,40,50,60,70,80,90,100]
test_knn(k_range,data1)
k_range = [5,10,15,20,25,30]
test_knn(k_range,data1)
k_range = [2,3,4,5,6,7,8,9,10]
test_knn(k_range,data1)
```

```
# add the optimal values for each model then calculate the tpr and fpr and
grah
def test_optimal_models2(df1):
```

```
    one = True
    X, y = data(df1)

    X_poly = PolynomialFeatures(3).fit_transform(X)
    X_train, X_test, y_train, y_test = train_test_split(X_poly, y)

    opt_log = LogisticRegression(C=1, penalty='l2')
    opt_log.fit(X_train, y_train)
    y_pred_log = opt_log.predict(X_test)
    print("Confusion Matrix for Logistic Regression:")
    print(confusion_matrix(y_test, y_pred_log))

    fpr_log, tpr_log, _ = roc_curve(y_test, opt_log.decision_function(X_test))

    plt.figure(figsize=(8, 6))
    plt.plot(fpr_log, tpr_log, color='blue', label='Logistic Regression')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.plot([0, 1], [0, 1], color='green', linestyle='dashed')
    plt.title('ROC Curve - Logistic Regression')
    plt.legend()
    plt.show()

    opt_knn = KNeighborsClassifier(n_neighbors=10)
    X_train, X_test, y_train, y_test = train_test_split(X, y)
    opt_knn.fit(X_train, y_train)
    y_pred_knn = opt_knn.predict(X_test)
    print("Confusion Matrix for k-Nearest Neighbors:")
    print(confusion_matrix(y_test, y_pred_knn))

    fpr_knn, tpr_knn, _ = roc_curve(y_test, opt_knn.predict_proba(X_test)[:,
1])

    plt.figure(figsize=(8, 6))
    plt.plot(fpr_knn, tpr_knn, color='red', label='k-Nearest Neighbors')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.plot([0, 1], [0, 1], color='green', linestyle='dashed')
    plt.title('ROC Curve - k-Nearest Neighbors')
    plt.legend()
    plt.show()

# Same as other function but prints each model on one graph
def test_optimal_models3(df1):
    one = True
    X, y = data(df1)

    X_poly = PolynomialFeatures(7).fit_transform(X)
```

```
    X_train, X_test, y_train, y_test = train_test_split(X_poly, y)

    opt_log = LogisticRegression(C=0.1, penalty='l2')
    opt_log.fit(X_train, y_train)
    y_pred_log = opt_log.predict(X_test)
    print("Confusion Matrix for Logistic Regression:")
    print(confusion_matrix(y_test, y_pred_log))

    fpr_log, tpr_log, _ = roc_curve(y_test, opt_log.decision_function(X_test))

    opt_knn = KNeighborsClassifier(n_neighbors=50)
    X_train, X_test, y_train, y_test = train_test_split(X, y)
    opt_knn.fit(X_train, y_train)
    y_pred_knn = opt_knn.predict(X_test)
    print("Confusion Matrix for k-Nearest Neighbors:")
    print(confusion_matrix(y_test, y_pred_knn))

    idiot = DummyClassifier(strategy='most_frequent')
    idiot.fit(X_train,y_train)
    i_pred = idiot.predict(X_test)
    print("Confusion Matrix for Dummy model: ")
    print(confusion_matrix(y_test, i_pred))

    fpr_knn, tpr_knn, _ = roc_curve(y_test, opt_knn.predict_proba(X_test)[:,
1])

    plt.figure(figsize=(8, 6))
    plt.plot(fpr_log, tpr_log, color='blue', label='Logistic Regression')
    plt.plot(fpr_knn, tpr_knn, color='red', label='k-Nearest Neighbors')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.plot([0, 1], [0, 1], color='green', linestyle='dashed')
    plt.title('ROC Curves')
    plt.legend()
    plt.show()
```

```
test_optimal_models2(data1)
test_optimal_models3(data1)
```

```
# Call all the other functions for the other dataset
data2 = pd.read_csv("dataset2.csv")
plot_features(data2)
p_range = [1,2,3,4,5,6,7,8,9,10,20,30,40,50]
c_range = [0.001,0.01,0.1,1,10,100,1000,10000]
mean = []
```

```
std = []
mean,std,  = test_logistic(p_range,c_range,data2)
k_range = [10,20,30,40,50,60,70,80,90,100]
test_knn(k_range,data2)
k_range = [5,10,15,20,25,30]
test_knn(k_range,data2)
k_range = [2,3,4,5,6,7,8,9,10]
test_knn(k_range,data2)
test_optimal_models2(data2)
test_optimal_models3(data2)
```