Assignment 4

Student number: 20324412

Module: CS7CS4/CSU44061

Part I - Dataset I

The data includes two features X_1, X_2 and a target variable y of two classes (1,-1). A visualisation of the data is shown in figure 1. The decision boundary between the two target classes (y – 1, y = -1) is a non-linear boundary. This is good to keep in mind for subsequent model fitting.

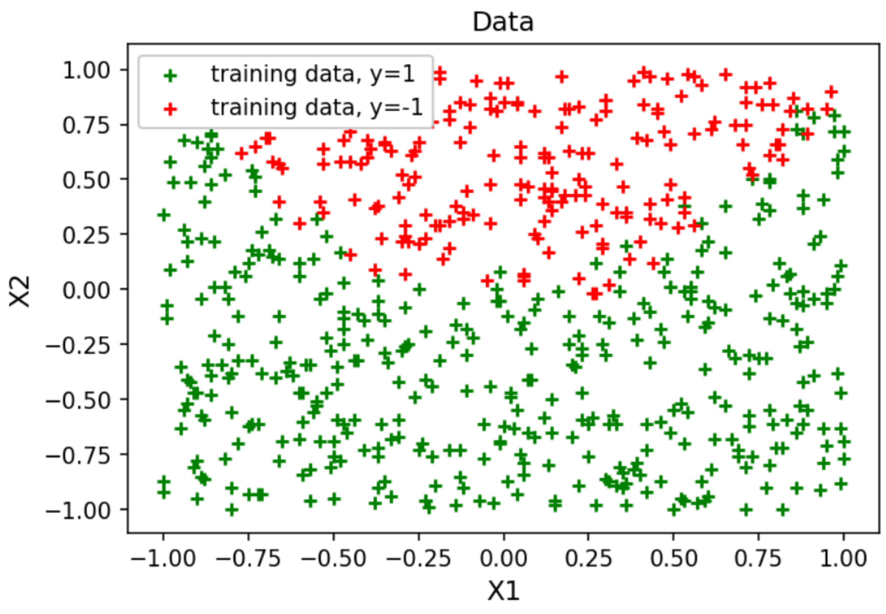


Figure 1. Visualisation of the data

Part a. Logistic Regression

Choice of q

A Logistic regression model with l2 penalty was trained on the data. The optimal value of q was determined in the range; [1, 2, 3, 4, 5] using 5 fold cross validation. This was implemented for three variations of the parameter C [0.01, 1, 1000]. The resultant plot is shown in Figure 2, obtained using the function *regression_model_range_c* as below. The model performs best for C =1, C = 1000 as the MSE is significantly less than that of C = 0.01 for all values of q. Thus the focus will be on the results obtained at these values.

The MSE drops dramatically when the degree of the polynomial is increased from one to two, from 0.54 to 0.22 (Figure 2b). This result is in line with the data visualisation in figure 1 in which the decision boundary splitting the data into it's target classes appears to be quadratic and so a polynomial of degree 2 seems like a good fit. There is subsequently little reduction in the MSE for polynomials of degrees 3, 4 and 5. Given the incentive to choose as simple a model as possible, a polynomial model of degree 2 seems optimal. Note in Figure 2b that the standard deviation of the MSE is greatest for the model of polynomial degree 1, implying higher variability in model performance, while it is lowest for q = 2.

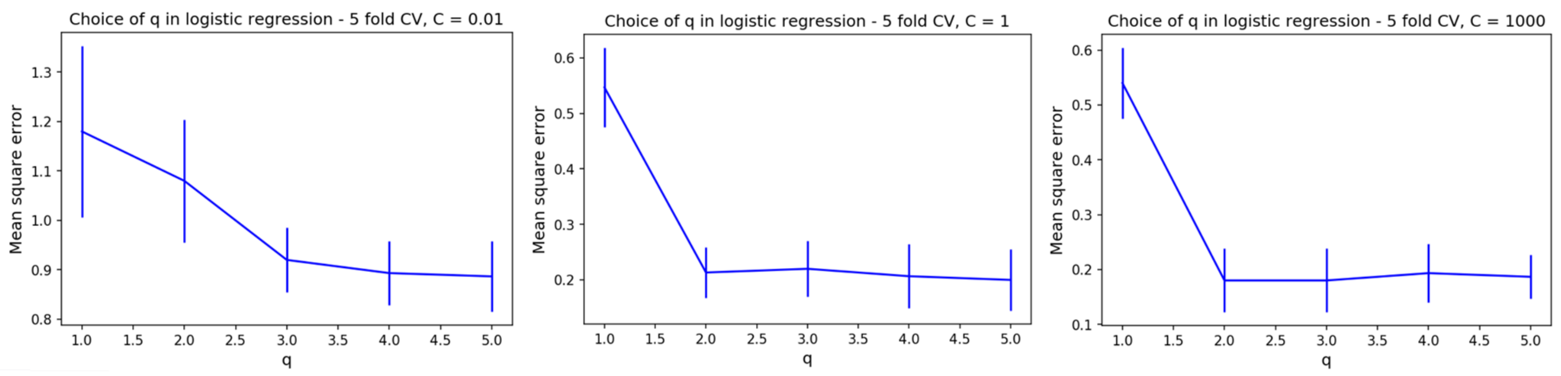


Figure 2. Plot of the degree of the polynomial (q) vs the Mean Square Error obtained via 5 fold cross validation for a logistic regression model fit to the data. For C values of 0.01 (a), 1 (b) and 100 (c)

Function - choose_q_cv

```
def choose_q_cv(X, y, c_param, q_range, plot_color):

    '''Implement 5 fold cross validation for testing

    regression model (lasso or ridge) and plot results''

    #Param setup

    kf = KFold(n_splits = 5)

    mean_error=[]; std_error=[];

    model = LogisticRegression(penalty= 'l2', C = c_param)

    #Loop through each k fold

    for q in q_range:

        mse_temp = []

        Xpoly = PolynomialFeatures(q).fit_transform(X)

        for train, test in kf.split(Xpoly):

            model.fit(Xpoly[train], y[train])

            ypred = model.predict(Xpoly[test])

            mse = mean_squared_error(y[test],ypred)

            mse_temp.append(mse)

        #Get mean & variance

        mean_error.append(np.array(mse_temp).mean())

        std_error.append(np.array(mse_temp).std())
    #Plot

    plt.errorbar(q_range, mean_error, yerr=std_error, color = plot_color)

    plt.xlabel('q')

    plt.ylabel('Mean square error')

    plt.title('Choice of q in logistic regression - 5 fold CV, C =
{}'.format(c_param))

    plt.show()
```

Choice of C

The optimal value of the penalty term C in logistic regression was then determined using 5 fold cross validation. A range of values for C were trialled starting with [0.01, 0.02, 0.05, 1, 5, 10, 50, 100, 500, 1000] as shown in Figure 3a. The plot was obtained using the function *choose_c_cv* as below. The models were fit to the data of polynomial degree 2. It is evident from Figure 3a that the greatest reduction occurs at approximately C = 1.0 and so the plot was zoomed in as in Figure 3b. The MSE drops dramatically as C approaches 1, reaching a minimum at C = 5.0. Given that smaller C values penalise more complex models, the smallest viable value for C was deemed optimal, thus a value of 5 was chosen.

Figure 3 (a-b). Penalty term (*C) vs* MSE of logistic regression model – 5 fold Cross Validation

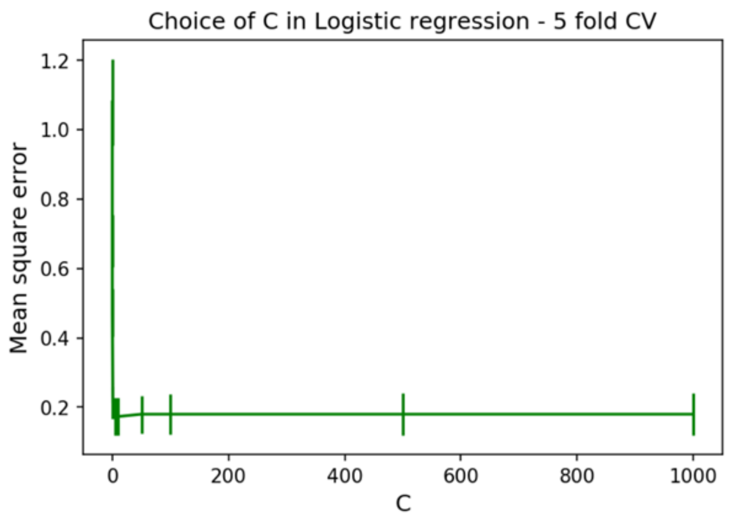


Figure 3a. C = [1, 5, 10, 50, 100, 500].

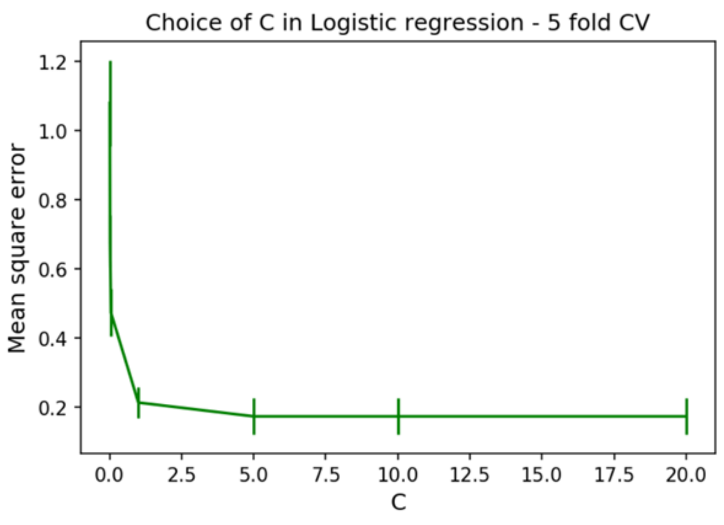


*Figure 3b._C = [*0.01, 0.02, 0.05, *1, 5, 10, 20, 50]*

Function - choose_c_cv

```
def choose_C_cv(X, y, q, c_range, plot_color):

    '''Implement 5 fold cross validation for testing

    regression model (lasso or ridge) and plot results'''
```

```
#Param setup

kf = KFold(n_splits = 5)

Xpoly = PolynomialFeatures(q).fit_transform(X)

mean_error=[]; std_error=[];

#Loop through each k fold

for c_param in c_range:

    mse_temp = []

    model = LogisticRegression(penalty= 'l2', C = c_param)

    for train, test in kf.split(Xpoly):

        model.fit(Xpoly[train], y[train])

        ypred = model.predict(Xpoly[test])

        mse = mean_squared_error(y[test],ypred)

        mse_temp.append(mse)

    #Get mean & variance

    mean_error.append(np.array(mse_temp).mean())

    std_error.append(np.array(mse_temp).std())

#Plot

plt.errorbar(c_range, mean_error, yerr=std_error, color = plot_color)

plt.xlabel('C')

plt.ylabel('Mean square error')

plt.title('Choice of C in Logistic regression - 5 fold CV')

plt.show()
```

Final model

The final logistic regression implemented was a polynomial of degree 2 with a penalty term C equal to 5 as below. A train test split of 67% to 33% was implemented given the relatively small size of the data (size = 600)

```
Xpoly_train, Xpoly_test, ytrain, ytest,indices_train,indices_test =
train_test_split(Xpoly, y,indices, test_size= 0.33, random_state=42)

log_reg_model = LogisticRegression(penalty= 'l2', C = 5.0)

log_reg_model.fit(Xpoly_train, Xpoly_test)
```

The resultant model is as follows whereby θ_0 is the intercept and $\theta_1 \dots \theta_5$ are the slope parameters

$$\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2$$

$$\theta^T x = 0.057 - 0.169\, x_1 - 9.349 x_2 + 9.420 x_1^2 - 0.267 x_1 x_2 - 9.814 x_2^2$$

These can be interpreted as follows;

$$y = +1 \text{ when } 0.057 - 0.169\, x_1 - 9.349 x_2 + 9.420 x_1^2 - 0.267 x_1 x_2 - 9.814 x_2^2 > 0$$

$$y = -1 \text{ when } 0.057 - 0.169\, x_1 - 9.349 x_2 + 9.420 x_1^2 - 0.267 x_1 x_2 - 9.814 x_2^2 < 0$$

Predictions

The trained logistic regression classifier was then used to predict the target values in the training data and the results are shown in figure 4. This was obtained using the code as below. The model provides a good fit to the data and the vast majority of cases are correctly classified. The model has captured the quadratic decision boundary that separates the classes.

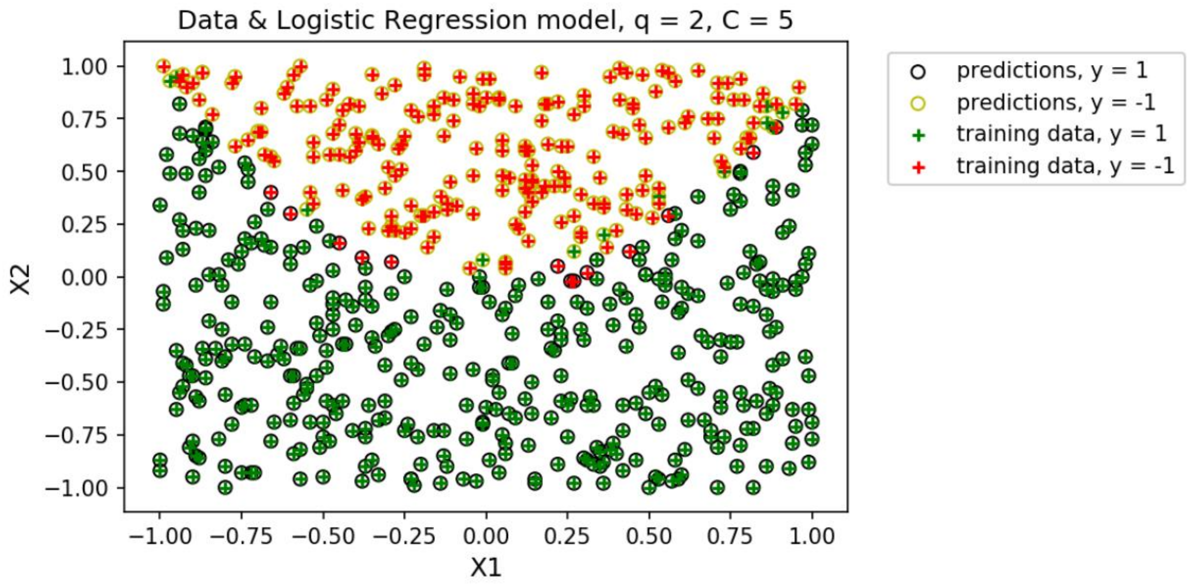


Figure 4. *Training data and predictions obtained via logistic regression of polynomial degree 2.*

```
#Function to plot predictions

def plot_data_preds(df, model_name, preds_col):

    'Plot data, logistic regression predictions'

    #Plot of Predictions

    plt.scatter(df.loc[df[preds_col] == 1, 'X1'], df.loc[df[preds_col] ==
1, 'X2'], marker = 'o', facecolors='none', edgecolors= 'k')
```

```
    plt.scatter(df.loc[df[preds_col] == -1, 'X1'], df.loc[df[preds_col] ==
-1, 'X2'], marker = 'o',facecolors='none', edgecolors= 'y')

    #Plot of Training Data

    plt.scatter(df.loc[df['y'] == 1, 'X1'], df.loc[df['y'] == 1, 'X2'],
marker = '+', c = 'g')

    plt.scatter(df.loc[df['y'] == -1, 'X1'], df.loc[df['y'] == -1, 'X2'],
marker = '+', c = 'r')

    #Labels

    plt.xlabel('X1')

    plt.ylabel('X2')

    plt.title('Data & {}'.format(model_name))

    plt.legend(['predictions, y = 1', 'predictions, y = -1', 'training
data, y = 1','training data, y = -1'], fancybox=True, framealpha=1,
bbox_to_anchor=(1.04,1), loc="upper left") #:D

    plt.show()


#Implement

predictions = log_reg_model.predict(X)

df2['preds'] = predictions

plot_data_preds(df, model_name, preds_col)
```

Part b. k Nearest Neighbour

A k nearest neighbour (knn) model was trained on the data. To determine the optimal k value, i.e the number of neighbours 5 fold cross validation was used for each value of k in the following range; [2, 3, 5, 7, 10, 15, 20, 25, 40, 60, 100]. Theoretically it has been argued that the optimal value of k is equal to $\sqrt{N}$, which would be $\sqrt{600} = 24.49$ and so this value was included in the range. The resultant plot is shown in Figure 5. The plot was obtained using the function *choose_k_knn* as below.

The MSE decreases as the number of neighbours k is increased towards 15. It then increases significantly for greater values of k (40, 60, 100). The MSE is lowest when the number of neighbours k is 15 and so this was chosen as the optimal value.

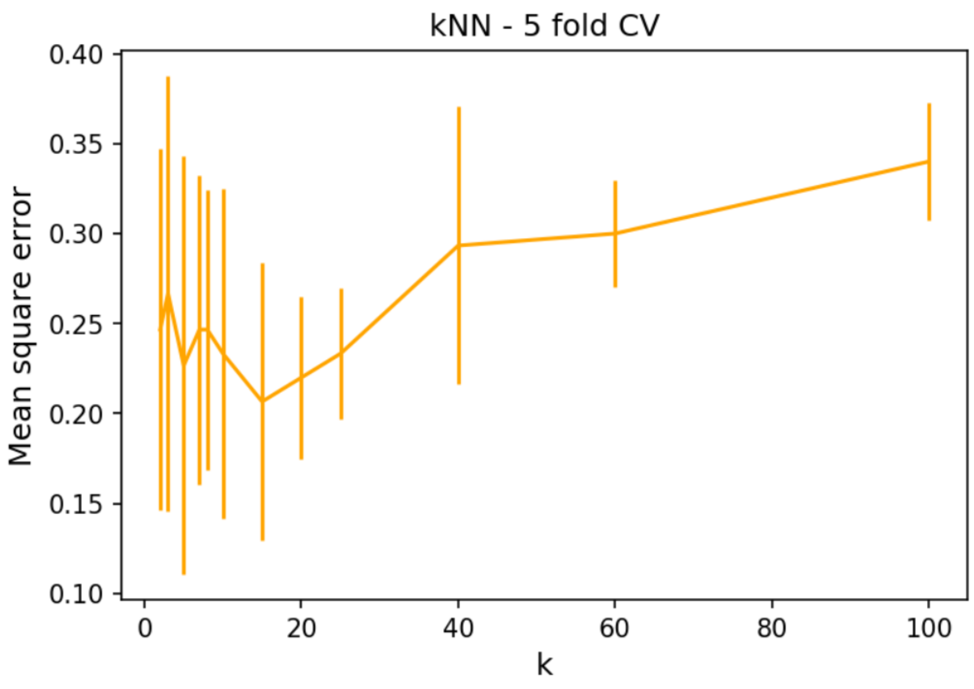


Figure 6. k nearest neighbour - p*lot of the number of neighbours k vs the Mean Square Error of the model obtained via 5 fold cross validation*

Function – choose_k_knn

```
def choose_k_knn(X, y, k_range, plot_color):
    '''knn - Implement 5 fold cross validation for determinine optimal k'''
    #Param setup
    kf = KFold(n_splits = 5)
    mean_error=[]; std_error=[];
    #Loop through each k value
    for k in k_range:
        mse_temp = []
        model = KNeighborsClassifier(n_neighbors = k, weights= 'uniform')
        for train, test in kf.split(X):
            model.fit(X[train], y[train])
            ypred = model.predict(X[test])
            mse = mean_squared_error(y[test],ypred)
            mse_temp.append(mse)
        #Get mean & variance
        mean_error.append(np.array(mse_temp).mean())
        std_error.append(np.array(mse_temp).std())
    #Plot
    plt.errorbar(k_range, mean_error, yerr=std_error, color = plot_color)
    plt.xlabel('k')
    plt.ylabel('Mean square error')
    plt.title('kNN - 5 fold CV')
    plt.show()
```

Addition of polynomial features

It is not worth adding polynomial features to the knn model. Using a knn model, an observation is classified by a majority vote of its neighbours and is assigned to the class most common among its *k* nearest neighbours. The model does not learn the overall global relationship between X and y. Thus it does not need to identify patterns such as nonlinear ones between x and y and so the addition of polynomial features is inappropriate. To confirm that no benefit was obtained by including polynomial features, 5 fold cross validation was implemented to test if there was any reduction in the MSE. As shown in Figure 7, no reduction was obtained and the MSE is lowest for a polynomial of degree 1 at a value of 0.215, thus no polynomial features were included in the final knn model.

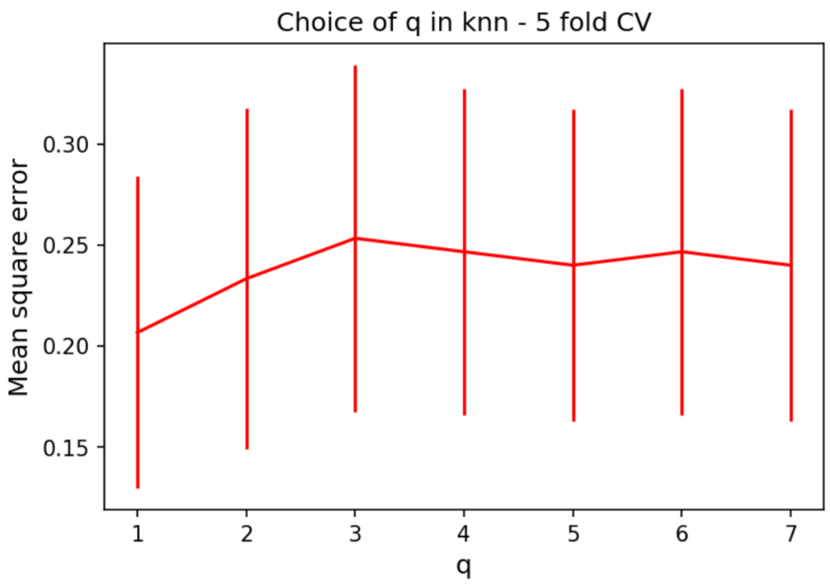


Figure 8. Plot of the degree of the polynomial (q) vs the Mean Square Error obtained via 5 fold cross validation for a knn model fit to the data

Final knn Model

```
k = 15

knn_model = KNeighborsClassifier(n_neighbors = k, weights= 'uniform')

knn_model.fit(X[indices_train], ytrain)
```

Predictions

The trained knn classifier was then used to predict the target values in the training data as shown in figure 8. The model provides a good fit to the data and the vast majority of cases are correctly classified.

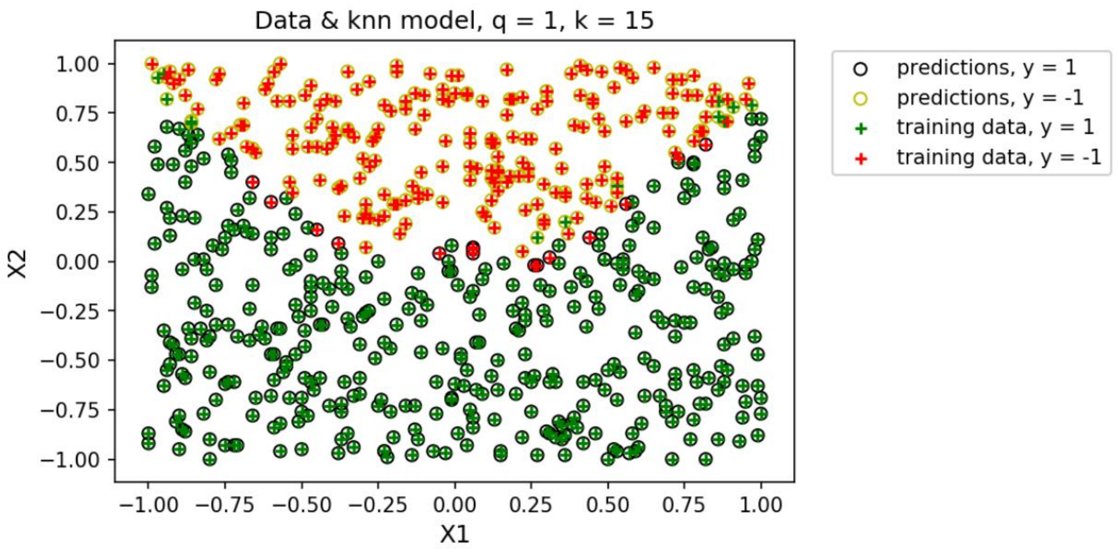


Figure 8. Data and model predictions from knn model

Part c – Confusion matrix

Confusion matrices for the logistic regression model, knn model and baseline model were generated as below.

Logistic Regression

```
predictions = log_reg_model.predict(Xpoly_test)

print(confusion_matrix(ytest, predictions))
```

	Predicted Positive	Predicted Negative
True Positive	121	6
True Negative	4	67

Table 2a. Confusion Matrix - Logistic Regression

K Nearest neighbour

```
predictions_knn = knn_model.predict(X[indices_test])

print(confusion_matrix(ytest, predictions_knn))
```

	Predicted Positive	Predicted Negative
True Positive	121	6
True Negative	6	65

Table 2b. Confusion Matrix – knn model

Baseline Model

A baseline model was implemented which predicts the majority class for all cases. In this instance the majority class is y = 1 (388 observations), vs y = -1 (212 observations)

```
dummy_clf = DummyClassifier(strategy="most_frequent")

dummy_clf.fit(X[indices_train], y[indices_train])

predictions_dummy = dummy_clf.predict(X[indices_test])

print(confusion_matrix(ytest, predictions_dummy))
```

	Predicted Positive	Predicted Negative
True Positive	127	0
True Negative	71	0

Table 2c. Confusion Matrix – Baseline Model

Part d – ROC Curve

The Receiver Operator Curves for the Logistic Regression model, knn model and random classifier are shown in Figure 8. The following function df was used to generate the plot.

```
def plot_roc_models(X, ytest, indices_test, log_reg_model, knn_model,
preds_baseline):

    'Plot ROC Curve of implemented models'

    #Data

    Xpoly = PolynomialFeatures(2).fit_transform(X)

    #Logistic Regression model

    scores = log_reg_model.decision_function(Xpoly[indices_test])

    fpr, tpr, _= roc_curve(ytest, scores)

    plt.plot(fpr,tpr, label = 'Logistic Regression')

    #knn model

    scores = knn_model.predict_proba(X[indices_test])

    fpr, tpr, _= roc_curve(ytest, scores[:, 1])

    plt.plot(fpr,tpr, color = 'r', label = 'knn')

    #Baseline Model

    scores_bl = dummy_clf.predict_proba(X[indices_test])

    fpr, tpr, _= roc_curve(ytest, scores_bl[:, 1])

    fpr, tpr, _= roc_curve(ytest, scores_bl[:, 1])

    plt.plot(fpr,tpr, color = 'orange', label = 'baseline model')

    #Random Choice

    plt.plot([0, 1], [0, 1],'g--')

    plt.legend(['Logistic Regression', 'knn', 'Baseline ','Random
Classifier'])

    #Labels

    plt.xlabel('False positive rate')

    plt.ylabel('True positive rate')

    plt.title('ROC Curve') #  - Logistic Regression')

    plt.show()
```

```
#Implement function

plot_roc_models(X, ytest, indices_test, log_reg_model, knn_model,
dummy_clf)
```

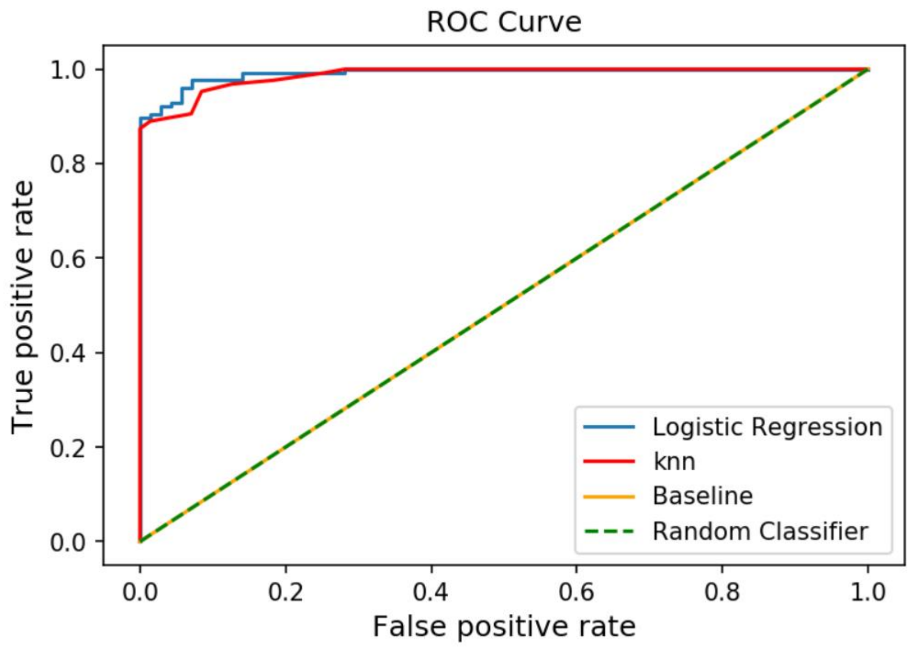


Figure 9. ROC Curve for various models; Logistic Regression, KNN, Baseline model and Random Classifier

Part e – Performance Evaluation

Both the logistic regression classifier and the knn classifier are performing well when their ROC curves are considered (Figure 7). A perfect classifier (100% true positives, 0% false positives) would produce a point in the top-left corner of the ROC plot and the ROC curves of both models are close to this point. The Logistic Regression model is marginally closer to this corner point, i.e at the optimal threshold it's True Positive Rate is higher and it's False Positive rate is lower then that of the knn model. Both models perform significantly better than that of the random and baseline classifiers. These models could be considered 'no-skill' classifies in that they cannot discriminate between classes and would predict a random class in the case of the former and a constant class (the majority class) for all observations in the case of the latter. Their ROC Curves thus lie on the 45 degree line, which can be interpreted as the 'flip a coin' line, i.e if the ROC curve isn't far from this 45 degree line, the model is no better than flipping a coin to categorize a binary response.

These results also correspond with the confusion matrices in table 2a-2c. It can be quickly observed that the logistic regression model and the knn model are performing well, given that the diagonal entries are both high, i.e the models' ability to detect the true positives and true negatives. Conversely the off-diagonals are low which shows that there is a low number of false positives and false negatives. Quantifying these results, the *true positive rate* (recall) of both the logistic regression and the knn models is high and identical in both cases i.e;

True Positive Rate $= \frac{TP}{TP+FN} = \frac{121}{121+6} = 95.28\%$ (Logistic Regression & knn)

Considering the false positive rate (specificity) of both models, the logistic regression model performs better in that the rate is lower, i.e there is less overall false positives;

False Positive Rate $= \frac{FP}{FP+TN} = \frac{4}{4+67} = 5.6\%$ (Logistic Regression) FPR $= \frac{4}{4+67} = 8.4\%$ (knn)

The logistic regression model is also more precise than the knn model. Of the observations it predicts from the positive class, more of them are actually positive, i.e;

Precision $= \frac{TP}{TP+FP} = \frac{121}{121+4} = 96.8\%$ (Logistic Regression) FPR $= \frac{121}{121+6} = 95.3\%$ (knn)

Interestingly, for the baseline model, the *true positive rate* (recall) is 100%, given that there are no false negatives, as all cases are predicted to come from the positive class. However as a result there is a large amount of false positives and no true negatives. Therefore the model has a very poor false positive rate and precision, at 100% and 61% respectively.

Based on the ROC curve and the performance metrics obtained from the confusion matrix above the logistic regression model performs marginally better the knn model and significantly better than the baseline model and so would be the classifier of choice for the first dataset.

Part 2: Dataset II

The same analyses was again implemented on the second dataset. The same functions and code was used as above and for brevity, will not be repeated.

The data includes two features X_1, X_2 and a target variable y of two classes (1,-1). A visualisation of the data is shown in figure I. There is no obvious decision boundary between the two target classes (y – 1, y = -1) and the data is not linearly separable based on the features X_1 and X_2.

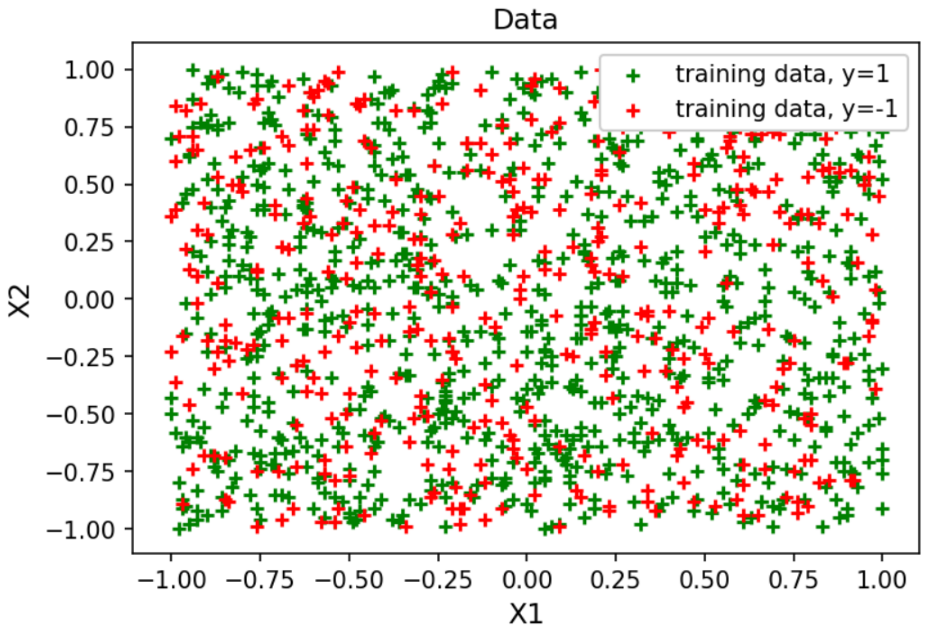


Figure I. Visualisation of the data

Part a. Logistic Regression

Choice of q

A Logistic reqression model with l2 penalty was trained on the data. The optimal value of q was determined in the range; [1, 2, 3, 4, 5] and implemented for three variations of the parameter C [0.01, 1, 1000] to get an idea if the results were consisted across values of C. The resultant plot is shown in Figure II obtained again using the function *regression_model_range_c*.

All models seem to be performing very poorly across all values of q and C. This is unsurprising given the nature of the data in figure 1, in that the classes are not separable based on the features X_1 and X_2 and the logistic regression model is unable to determine a decision boundary to separate the classes. The MSE is high for all values of q. In Figure IIc, the MSE is seen to increase as the polynomial is increased to orders of 5, 6, 7. Given that no reduction is seen in the MSE with the inclusion of polynomial features, it was deemed unnecessary to include them in the model and so the degree of the polynomial of X was left at one.

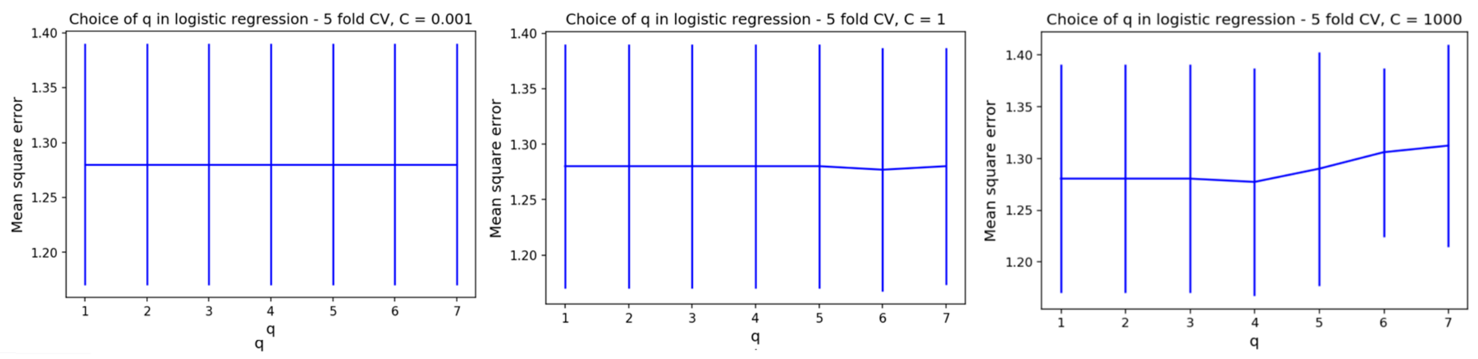
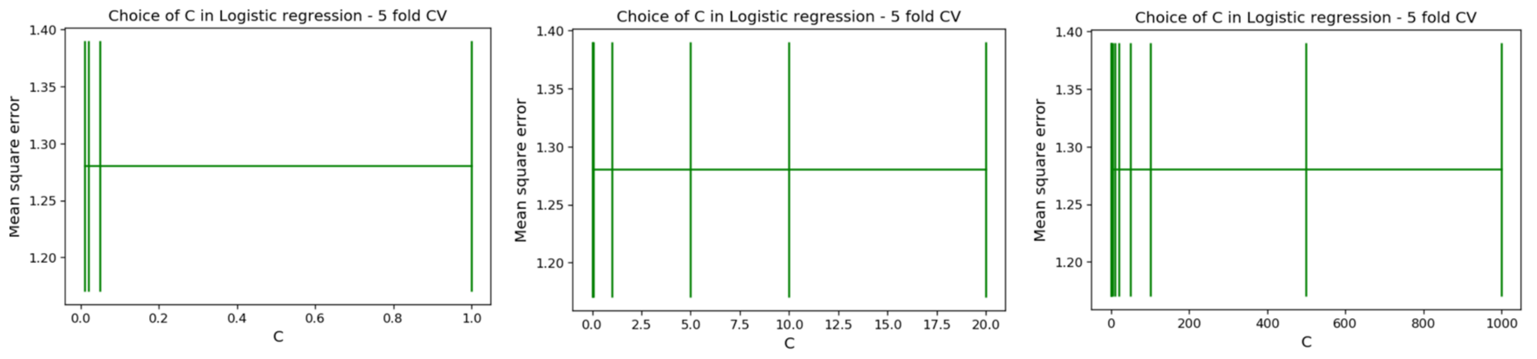


Figure II. *Plot of the degree of the polynomial (q) vs the Mean Square Error obtained via 5 fold cross validation for a logistic regression model fit to the data. For C values of 0.01 (a), 1 (b) and 100 (c)*

Choice of C

The optimal value of the penalty term C in logistic regression was then determined in the range [0.01, 0.02, 0.05, 1, 5, 10, 50, 100, 500, 1000] as shown in Figure III. The plot was again obtained using the function *choose_c_cv*. The same poor model performance results are again seen. No gain in performance is achieved for varying the penalty term C. Again the logistic regression model is unable to determine a decision boundary to separate the classes. Thus for the final model value of the penalty term was left at the default value of 1.

Figure III (a-c). Penalty term (*C) vs* MSE of logistic regression model – 5 fold Cross Validation



C =[0.01, 0.02, 0.05, *1].* *C = [*0.01, 0.02, 0.05, *1, 5, 10, 20]* *C = [*0.01, 0.02, 0.05, *1, 5, 10, 20, 20 20, 50, 100,1000]*

Final model

The final logistic regression implemented was a polynomial of degree 1 with a penalty term C equal to 1.0

The resultant model is as follows whereby θ_0 is the intercept and θ_1, θ_2 are the slope parameters

$$\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0.772 + 0.027\, x_1 - 0.214 x_2$$

These can be interpreted as follows;

$y = +1$ when $0.772 + 0.027\, x_1 - 0.214 x_2 > 0$

$y = -1$ when $0.772 + 0.027\, x_1 - 0.214 x_2 < 0$

Predictions

The trained logistic regression classifier was then used to predict the target values in the training data and the results are shown in figure IV. The plot was again obtained using the function *plot_data_preds.* All model predictions appear to come from the positive class (y = 1). Upon inspection, the dataset contains a significant class in-balance. The positive class (y =1) has significantly more observations (854 observations, 68%) than that of the negative class (y = -1) (402 observations, 32%). The model is over-fitting to the dominant class.

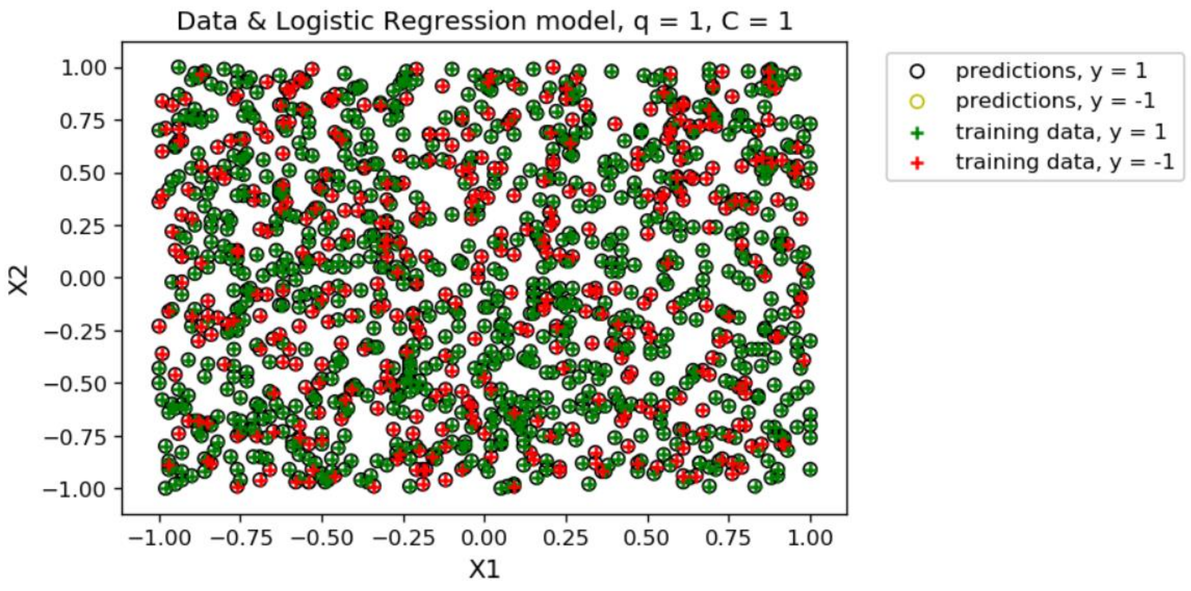


Figure 4. *Training data and predictions obtained via logistic regression of polynomial degree 2.*

Part b. k Nearest Neighbour

A k nearest neighbour (knn) model was trained on the data. The optimal k value, i.e the number of neighbours was determined from the following range; [2, 3, 5, 7, 10, 15, 25, 35, 50, 80, 100, 150, 200]. In this case $\sqrt{N} = \sqrt{1256} = 35.44$ and so this value was included in the range. The resultant plot is shown in Figure V which was again obtained using the function *choose_k_knn*. The MSE drops significantly as k is increased and appears to reach it's minimum at k = 100. Thus this was chosen as the optimal k value. This seems an unusually high value for the parameter k, but was chosen based on the results of the MSE.

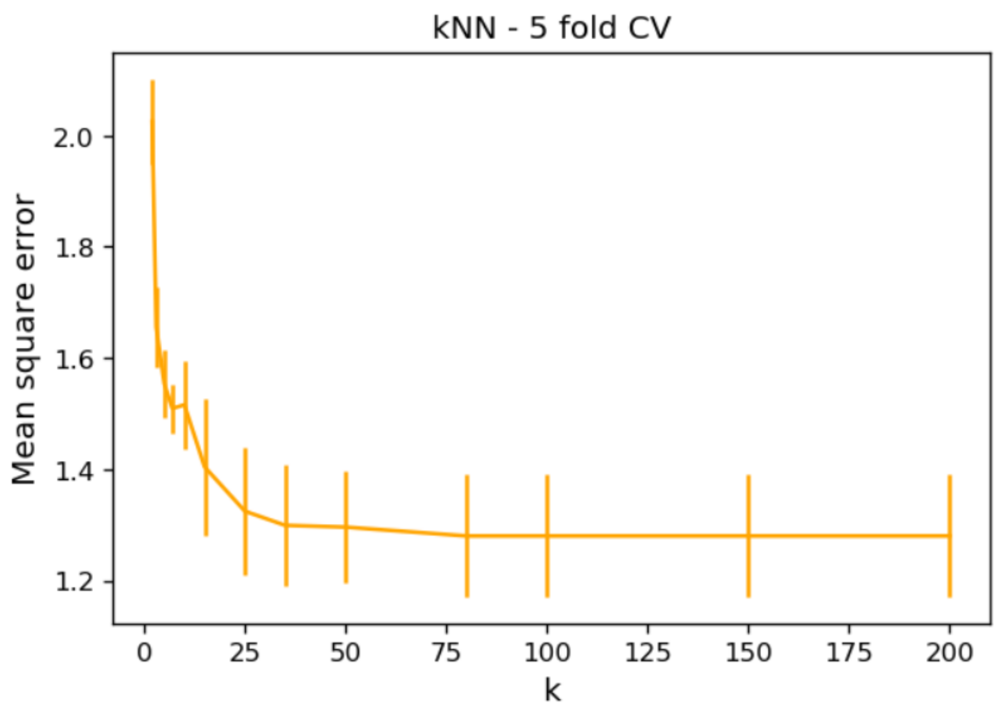


Figure VI. k nearest neighbour - p*lot of the number of neighbours k vs the Mean Square Error of the model obtained via 5 fold cross validation*

Addition of polynomial features

As described for dataset 1, it would not be expected that the addition of polynomial features would provide an improvement to the MSE. This is confirmed by plotting the degree of the polynomial vs the MSE as in Figure VII. No improvement is obtained and so no polynomial features were included in the knn model.

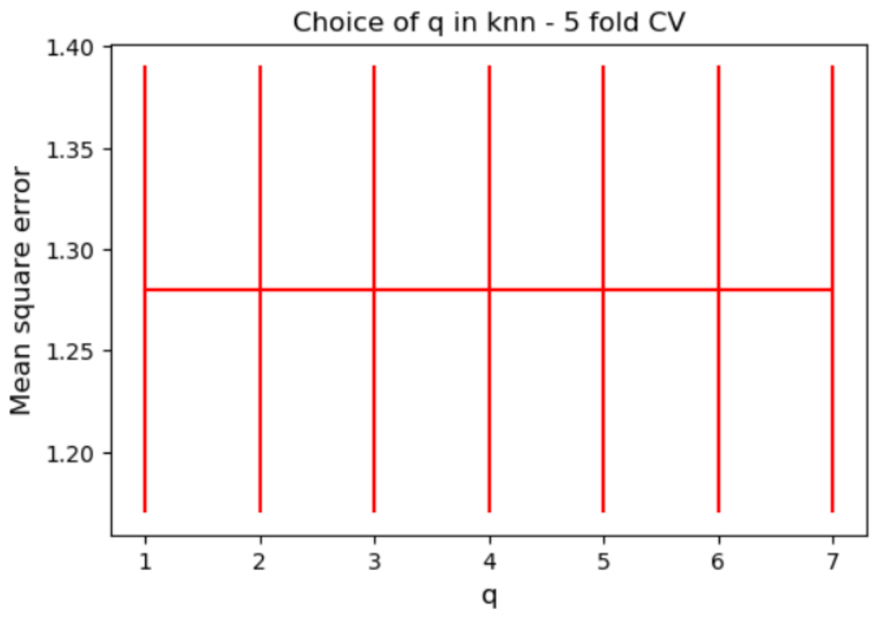


Figure 8. Plot of the degree of the polynomial (q) vs the Mean Square Error obtained via 5 fold cross validation for a knn model fit to the data

Final knn Model

```
k = 100

knn_model = KNeighborsClassifier(n_neighbors = k, weights= 'uniform')

knn_model.fit(Xtrain, ytrain)
```

Predictions

The plot of the predictions obtained from the knn model is plotted with the data in figure XIII. Again all model predictions appear to come from the positive class (y = 1), the model is over-fitting to the dominant class.

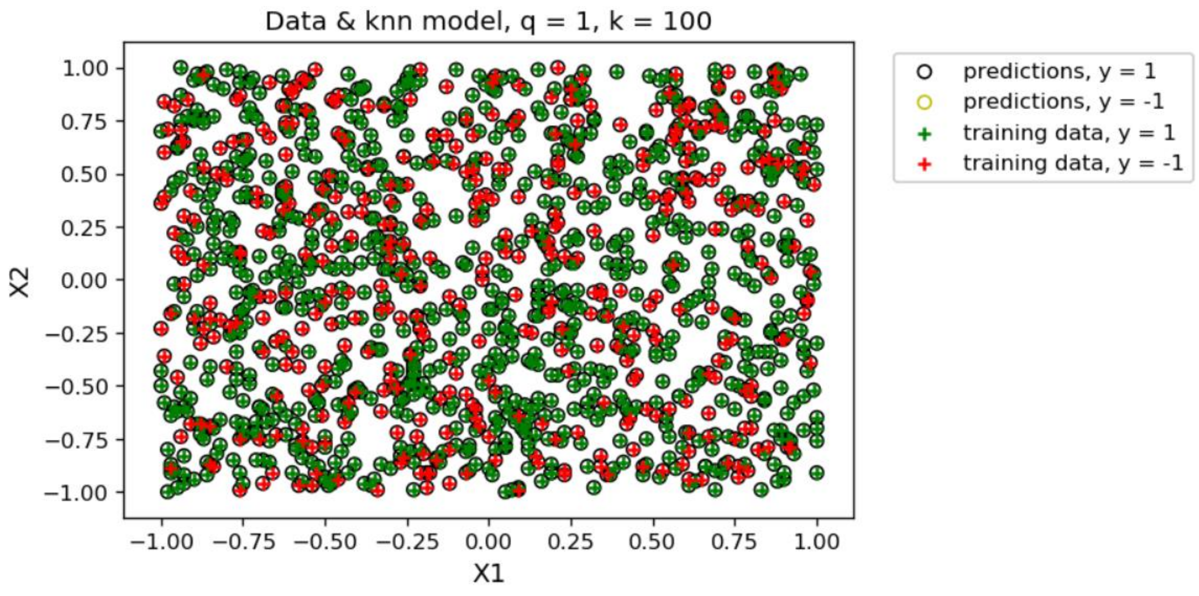


Figure XIII. Model predictions using final knn model

Part c – Confusion matrix

Confusion matrices for the logistic regression model, knn model and baseline model were generated as below.

Logistic Regression

	Predicted Positive	Predicted Negative
True Positive	279	0
True Negative	136	0

Table IIa. Confusion Matrix - Logistic Regression

K Nearest neighbour

	Predicted Positive	Predicted Negative

True Positive	279	0
True Negative	136	0

Table IIb. Confusion Matrix – knn model

Baseline Model

A baseline model was implemented which predicts the majority class for all cases. In this instance the majority class is y = 1 (854 observations), vs y = -1 (402 observations)

	Predicted Positive	Predicted Negative
True Positive	279	0
True Negative	136	0

Table IIc. Confusion Matrix – Baseline Model

Part d – ROC Curve

The Receiver Operator Curves for the Logistic Regression model, knn model, random classifier and baseline model are shown in Figure 8.

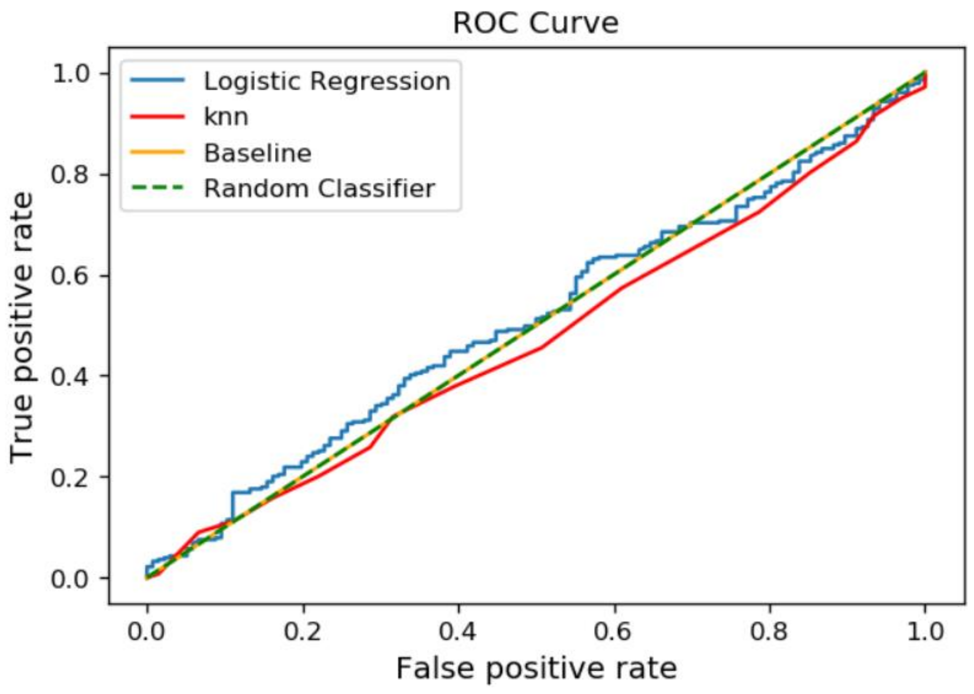


Figure IX. *ROC Curve for various models; Logistic Regression, KNN, Baseline model and Random Classifier*

Part e – Performance Evaluation

Both the logistic regression model and knn model are performing poorly overall. The fundamental problem is there inability to predict the minority class (y = -1). All observations are predicted to come from the majority class (y = 1). Thus they perform no differently to the baseline model, which predicts the majority class (y = 1) or the random model, which picks a class at random. The ROC curves of all models roughly lie on the 45 degree line as shown in Figure VIII, i.e the models are no better than flipping a coin to categorize the binary response.

The same conclusion is drawn when the confusion matrices are considered (Tables IIa – Ic). Given that all models predict all observations to come from the positive class, the number of true positives is very high (279 observations), however the number of false positives is also then very high (136 observations). Given that no observations are predicted to come from the negative class, the number of true negatives predicted and false negatives is zero. Quantifying this, across all models the *true positive rate* (recall) is 100% - the optimal value-, the false positive rate is 100% worst possible value- and the precision is 67.23%.

Interestingly, for the baseline model, the *true positive rate* (recall) is 100%, given that there are no false negatives, as all cases are predicted to come from the positive class. However as a result there is a large amount of false positives and no true negatives. Therefore the model has a very poor false positive rate and precision, at 100% and 67.23% respectively.

Given that the logistic regression model and knn model offer no predictive advantage over the baseline model, due to their inability to predict the minority class, the baseline model would be recommended for this dataset. The model would have to be significantly better than the baseline model to justify their implementation, which is not the case. The baseline model is the simplest model to implement, there is no need to perform hyper-parameter tuning and there is very little parameter tuning involved. The model still has a precision of 67.23%, i.e it correctly classifiers 67% of the data, so it is not the worst predictive model, performing significantly better than chance (50%).

Appendix – Code

```
#Week 4 Assignment - Logistic Regression + knn models

#Imports

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

from sklearn.metrics import roc_curve

from sklearn.neighbors import KNeighborsClassifier

from sklearn.dummy import DummyClassifier

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import PolynomialFeatures

from sklearn.model_selection import KFold

from sklearn.metrics import mean_squared_error


#Plot params

plt.rcParams['figure.dpi'] = 120

matplotlib.rc('xtick', labelsize=10)

matplotlib.rc('ytick', labelsize=10)

pd.set_option('display.max_colwidth', -1)

MEDIUM_SIZE = 12

plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels


#*********************** Part 1 - Dataset 1  ********************


#**************************

#Part i Data + Visualisation

#Data
```

```
data = pd.read_csv('week4_d1.txt',)

data.head()

data.reset_index(inplace=True)

data.columns = ['X1', 'X2', 'y']


#Extract Features

X1 = df.iloc[:,0]

X2 = df.iloc[:,1]

X = np.column_stack((X1,X2))

y= df.iloc[:,2]


#Plot data and color code the observations according to their value of the
target variable y

plt.scatter(data.loc[data['y'] == 1, 'X1'], data.loc[data['y'] == 1, 'X2'],
marker = '+', c = 'g')

plt.scatter(data.loc[data['y'] == -1, 'X1'], data.loc[data['y'] == -1,
'X2'], marker = '+', c = 'r')

plt.xlabel('X1')

plt.ylabel('X2')

plt.title('Data')

plt.legend(['training data, y=1','training data, y=-1'], fancybox=True,
framealpha=1) #:D

plt.show()
```

#***************** Part a Logistic Regression Model ***************

#i. Choose q

```
def choose_q_cv(X, y, q_range, c_param, plot_color):

    '''Implement 5 fold cross validation for testing

    regression model (lasso or ridge) and plot results'''


    #Param setup

    kf = KFold(n_splits = 5)
    mean_error=[]; std_error=[];
    model = LogisticRegression(penalty= 'l2', C = c_param)

    #Loop through each k fold
    for q in q_range:
        mse_temp = []
        Xpoly = PolynomialFeatures(q).fit_transform(X)

        for train, test in kf.split(Xpoly):
            model.fit(Xpoly[train], y[train])
            ypred = model.predict(Xpoly[test])
            mse = mean_squared_error(y[test],ypred)
            mse_temp.append(mse)

        #Get mean & variance
        mean_error.append(np.array(mse_temp).mean())
        std_error.append(np.array(mse_temp).std())

    #Plot
    plt.errorbar(q_range, mean_error, yerr=std_error, color = plot_color)
    plt.xlabel('q')
    plt.ylabel('Mean square error')
    plt.title('Choice of q in logistic regression - 5 fold CV, C =
{}'.format(c_param))
    plt.savefig("log_reg_C.png", bbox_inches='tight')
    plt.show()

#Implement
q_range = [1,2,3,4,5,6,7]
plot_color = 'b'
c_param = 1 #0.001, 1000
```

```
choose_q_cv(X, y, q_range, c_param, plot_color)


#*****************************************
```

#i. Choose C

```
def choose_C_cv(X, y, q, c_range, plot_color):
    '''Implement 5 fold cross validation for testing
    regression model (lasso or ridge) and plot results'''

    #Param setup
    kf = KFold(n_splits = 5)
    Xpoly = PolynomialFeatures(q).fit_transform(X)
    mean_error=[]; std_error=[];

    #Loop through each k fold
    for c_param in c_range:
        mse_temp = []
        model = LogisticRegression(penalty= 'l2', C = c_param)

        for train, test in kf.split(Xpoly):

            model.fit(Xpoly[train], y[train])
            ypred = model.predict(Xpoly[test])
            mse = mean_squared_error(y[test],ypred)
            mse_temp.append(mse)

        #Get mean & variance
        mean_error.append(np.array(mse_temp).mean())
        std_error.append(np.array(mse_temp).std())

    #Plot
    plt.errorbar(c_range, mean_error, yerr=std_error, color = plot_color)

    plt.xlabel('C')

    plt.ylabel('Mean square error')

    plt.title('Choice of C in Logistic regression - 5 fold CV')

    plt.show()


#Implement

q = 2 #Optimal q

c_range = [0.01, 0.02, 0.05, 1, 5, 10, 20, 50, 100, 500, 1000]

q_range = [1,2,3,4,5]

plot_color = 'g'

choose_C_cv(X, y, q, c_range, plot_color)


#********************************************

#Final model

#Data for model training

indices = np.arange(0,600)

Xpoly = PolynomialFeatures(2).fit_transform(X)

Xpoly_train, Xpoly_test, ytrain, ytest,indices_train,indices_test =
train_test_split(Xpoly, y,indices, test_size= 0.33, random_state=42)


#Model

log_reg_model = LogisticRegression(penalty= 'l2', C = 5.0)

log_reg_model.fit(Xpoly_train, ytrain)

log_reg_model.intercept_

log_reg_model.coef_


#***************************************************

#Predictions

predictions_log_reg = log_reg_model.predict(Xpoly_test)

df2['preds_log_reg'] = predictions
```

```
#Plot logistic regression predictions

def plot_data_preds(df, model_name, preds_col):

    'Plot data, logistic regression predictions’

    #Plot of Predictions

    plt.scatter(df.loc[df[preds_col] == 1, 'X1'], df.loc[df[preds_col] ==
1, 'X2'], marker = 'o', facecolors='none', edgecolors= 'k')

    plt.scatter(df.loc[df[preds_col] == -1, 'X1'], df.loc[df[preds_col] ==
-1, 'X2'], marker = 'o',facecolors='none', edgecolors= 'y')

    #Plot of Training Data

    plt.scatter(df.loc[df['y'] == 1, 'X1'], df.loc[df['y'] == 1, 'X2'],
marker = '+', c = 'g')

    plt.scatter(df.loc[df['y'] == -1, 'X1'], df.loc[df['y'] == -1, 'X2'],
marker = '+', c = 'r')

    #Labels

    plt.xlabel('X1')

    plt.ylabel('X2')

    plt.title('Data & {}'.format(model_name))

    plt.legend(['predictions, y = 1', 'predictions, y = -1', 'training
data, y = 1','training data, y = -1'], fancybox=True, framealpha=1,
bbox_to_anchor=(1.04,1), loc="upper left") #:D

    plt.show()

#Implement

preds_col = 'preds'

model_name = 'Logistic Regression model, q = 1, C = 1'

plot_data_preds(df2, model_name, preds_col)




#****************** Part b knn model  ********************************

#i. Choose k

def choose_k_knn(X, y, k_range, plot_color):

    '''knn - Implement 5 fold cross validation for determinine optimal k'''
    #Param setup
    kf = KFold(n_splits = 5)
    mean_error=[]; std_error=[];

    #Loop through each k fold
    for k in k_range:
        mse_temp = []
        model = KNeighborsClassifier(n_neighbors = k, weights= 'uniform')

        for train, test in kf.split(X):

            model.fit(X[train], y[train])
            ypred = model.predict(X[test])
            mse = mean_squared_error(y[test],ypred)
            mse_temp.append(mse)

        #Get mean & variance
        mean_error.append(np.array(mse_temp).mean())
        std_error.append(np.array(mse_temp).std())

    #Plot
    plt.errorbar(k_range, mean_error, yerr=std_error, color = plot_color)
    plt.xlabel('k')
    plt.ylabel('Mean square error')
    plt.title('kNN - 5 fold CV')
    plt.show()

#Implement
k_range = [2,3,5,7,8,10,15,20,25,40, 60, 100]
plot_color = 'orange'
```

```
choose_k_knn(X, y, k_range, plot_color)
```

#ii. Choose q

```
def choose_q_knn_cv(X, y, q_range, plot_color):
    '''Implement 5 fold cross validation for testing
    regression model (lasso or ridge) and plot results'''

    #Param setup
    kf = KFold(n_splits = 5)
    mean_error=[]; std_error=[];
    model = KNeighborsClassifier(n_neighbors = 15, weights= 'uniform')

    #Loop through each k fold
    for q in q_range:
        mse_temp = []
        Xpoly = PolynomialFeatures(q).fit_transform(X)

        for train, test in kf.split(Xpoly):
            model.fit(Xpoly[train], y[train])
            ypred = model.predict(Xpoly[test])
            mse = mean_squared_error(y[test],ypred)
            mse_temp.append(mse)

        #Get mean & variance
        mean_error.append(np.array(mse_temp).mean())
        std_error.append(np.array(mse_temp).std())

    #Plot
    plt.errorbar(q_range, mean_error, yerr=std_error, color = plot_color)
    plt.xlabel('q')
    plt.ylabel('Mean square error')

    plt.title('Choice of q in knn - 5 fold CV')

    plt.show()


#Implement

q_range = [1,2,3,4,5, 6,7]

plot_color = 'r'

choose_q_knn_cv(X, y, q_range, plot_color)


#Final Model

knn_model = KNeighborsClassifier(n_neighbors = 15, weights= 'uniform')

knn_model.fit(X[indices_train], y[indices_train])

predictions_knn = knn_model.predict(X[indices_test])


#Predictions

df2['preds_knn'] = predictions_knn

preds_col = 'preds_knn'

model_name = 'Logistic Regression model, q = 1, C = 1'

plot_data_preds(df2, model_name, preds_col)


#******************* Part c Confusion matrices  ****************

#Logistic regression

print(confusion_matrix(ytest, predictions_log_reg))

#knn

print(confusion_matrix(ytest, predictions_knn))


#Baseline model

dummy_clf = DummyClassifier(strategy="most_frequent")

dummy_clf.fit(X[indices_train], y[indices_train])

predictions_dummy = dummy_clf.predict(X[indices_test])
```

```
#Confusion matrix
print(confusion_matrix(ytest, predictions_dummy))

#****************** Part d - ROC Curve *************
def plot_roc_models(Xtest, ytest, log_reg_model, knn_model, dummy_clf):
    'Plot ROC Curve of implemented models'

    #Logistic Regression model
    scores = log_reg_model.decision_function(Xtest)
    fpr, tpr, _= roc_curve(ytest, scores)
    plt.plot(fpr,tpr, label = 'Logistic Regression')

    #knn model
    scores = knn_model.predict_proba(Xtest)
    fpr, tpr, _= roc_curve(ytest, scores[:, 1])
    plt.plot(fpr,tpr, color = 'r', label = 'knn')

    #Baseline Model
    scores_bl = dummy_clf.predict_proba(Xtest)
    fpr, tpr, _= roc_curve(ytest, scores_bl[:, 1])
    plt.plot(fpr,tpr, color = 'orange', label = 'baseline model')

    #Random Choice
    plt.plot([0, 1], [0, 1],'g--')

    #Labels
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.title('ROC Curve') #  - Logistic Regression')
    plt.legend(['Logistic Regression', 'knn', 'Baseline ','Random
Classifier'])

    plt.show()


#Implement

plot_roc_models(Xtest, ytest, log_reg_model, knn_model, dummy_clf)



#*********************** Part 2 - Dataset 2  ********************

#The same code was ran again for the second dataset, replacing the dataset
week4_d1.txt with week4_d2.txt

data = pd.read_csv('week4_d2.txt',)
```