Machine Learning Assignment 4

Thomas Kelly
16323455

First Line of Data Set: # id:10-20--10-1

Part (i) | A

After attempting to fit an accurate model to the first dataset, it quickly became clear to me that this would not be possible. The dataset is simply too noisy to successfully model. The program *'week4_part1.py'* firstly uses cross-validation to select the optimal value for 'k' k-fold split value. This is then used to test the values of hyperparameter C and Polynomial degree Q, choosing the value in each case that results in the least amount of error.

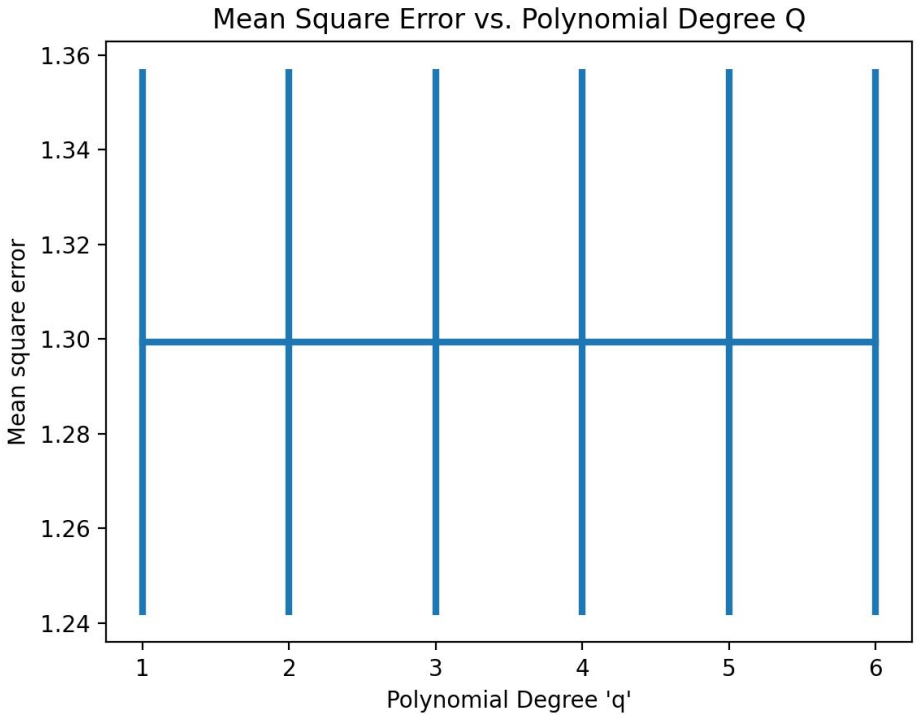

Figure 1 - MSE vs. Q (Polynomial Degree)

The poor quality of the data starts to become evident here, when a constant mean square error is observed for every value of polynomial degree (C is constant = 1). This would indicate that no matter how many features are used, there is no effect on the accuracy of the model - pointing towards noisy data.

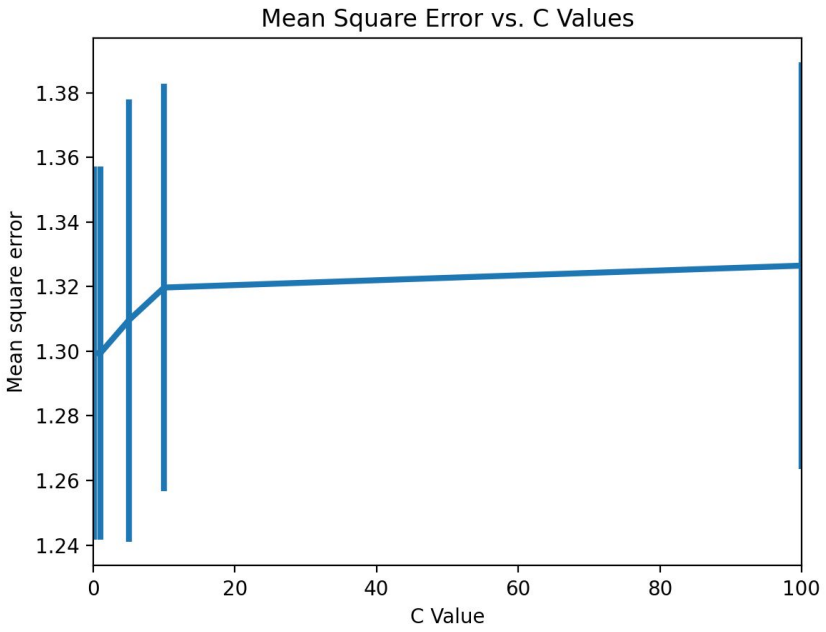

Figure 2 - MSE vs. C values

A low value for C results in the lowest error. This low C value will result in a large penalty for model parameters, meaning many of them will be forced close to zero (L2 regularisation). Using the initially reported optimal values; model parameters q = 1 and C = 0.001 are set, resulting in the following training data & prediction visualisation.

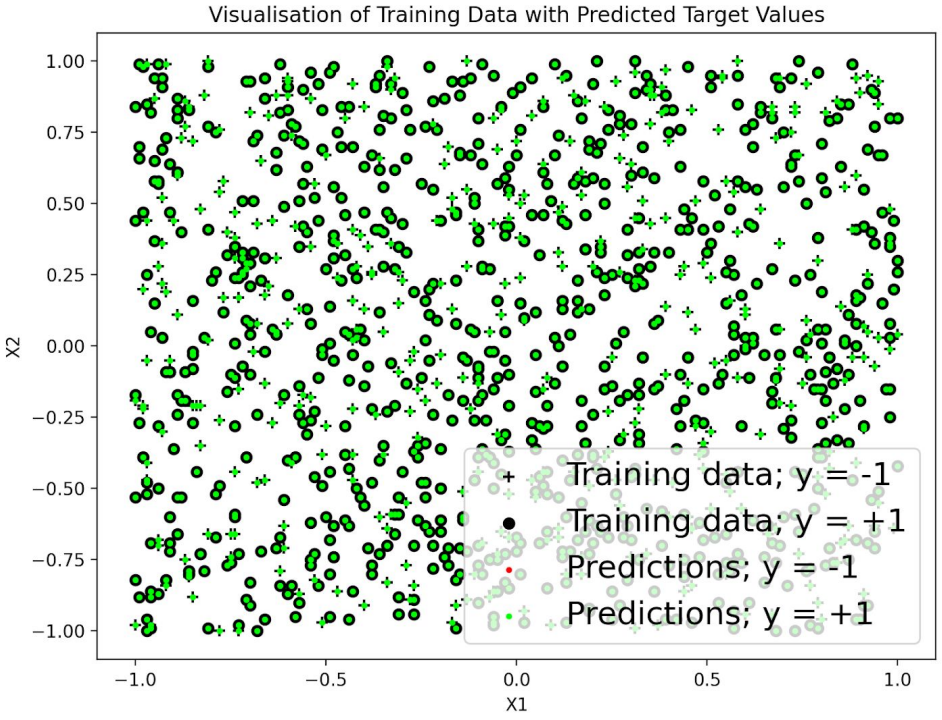

Figure 3 - Training data and predictions visualisation, q = 1, C = 0.001

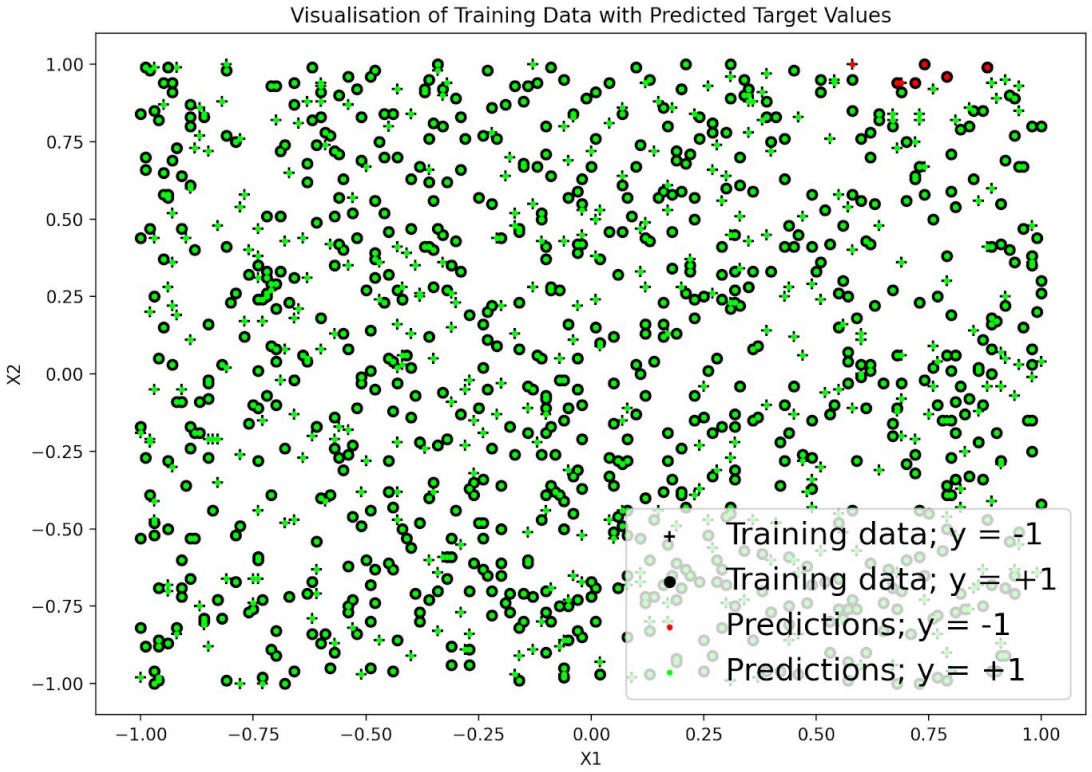

Figure 4 - Training data and predictions; q = 5, C = 5

Examining Figure 3, it is evident that the model is clearly inaccurate and useless, as it does not even predict any values for y = -1. If the values for q and C are increased experimentally to q = 5 and C = 5, we finally get some y = -1 predictions, however the model is still completely useless and fails to capture the behaviour of the data. We can conclude that even by increasing C (more nonzero parameters) and increasing q (more features), the data is simply too noisy to model accurately, evident from *Figure 4*.

Part (i) | B | Select a 'k' and justify for KNN

Based on the reported value of optimal k-fold splits from the *week1_part1.py* file, a value of k = 5 is chosen to keep computational cost low while also maintaining a low error.
A plot is generated for mean square error of the knn model vs. the number of neighbours used;

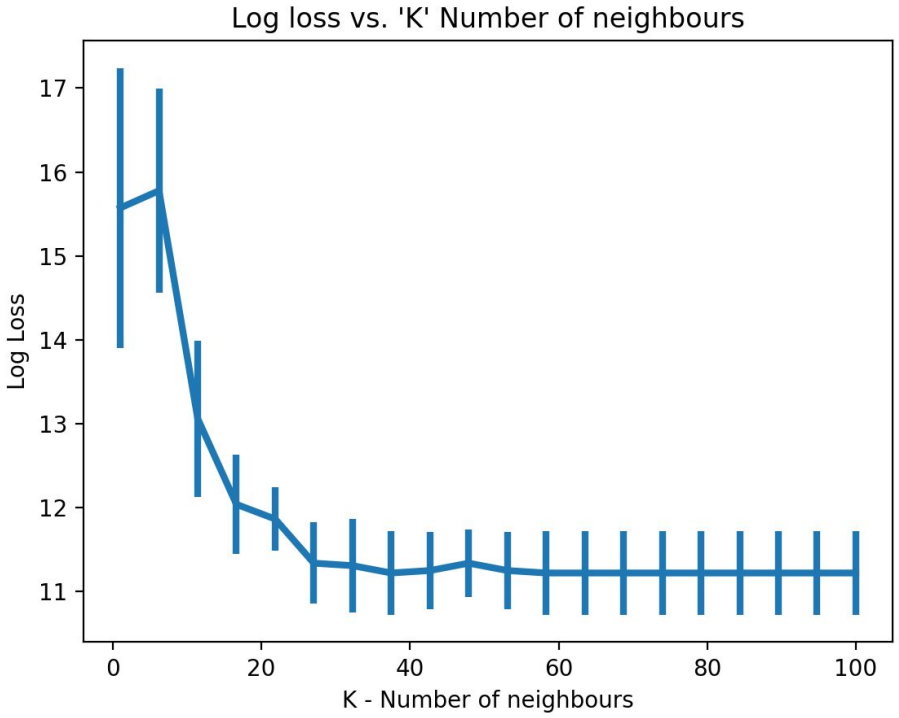

Figure 6 - Log loss vs. K- neighbours

A range of k = [1...100] is used because it captures the most interesting behaviour. Judging from the errorbar plot and its reported values, the optimal k to choose would be approx. k = 35, even though error is still quite high. The predictions are examined;

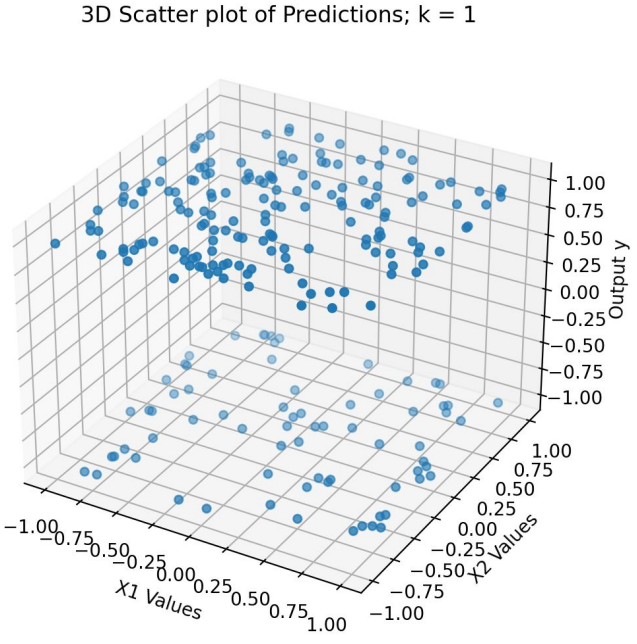

Figure 7 - Test data and classifications; k = 20

Evidently here, as is indicated by the high error, the model is useless. With knowledge that knn models also start to underfit as we increase k, this is reflected in the predictions scatter plots;

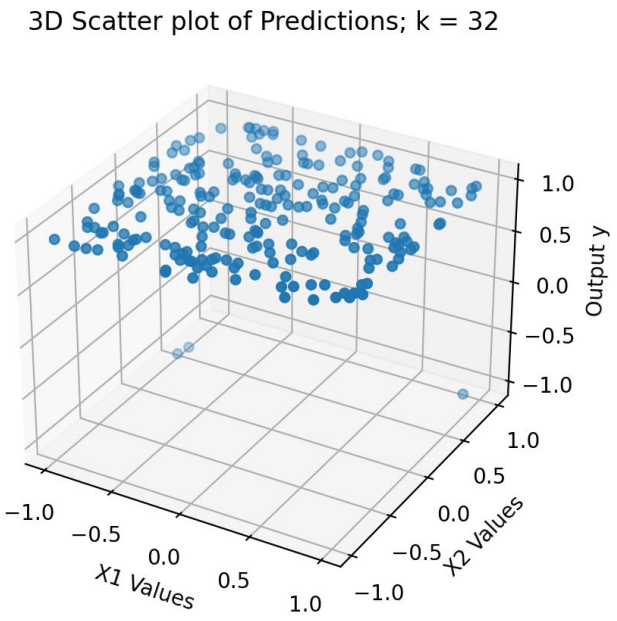

Figure 8 - Predictions

As we increase k, the model starts to predict the most common class for every test data point. It is evident here that choosing any k will result in an awful prediction - the data is simply too noisy.

Is it worth introducing Polynomial features?

Introducing new features to the model will have the effect of generating more parameters. However, setting the polynomial degree to q = 5 and taking a look at the mean square error graph compared with the number of neighbours ‘k’ we don’t see much different in terms of the trend & values.

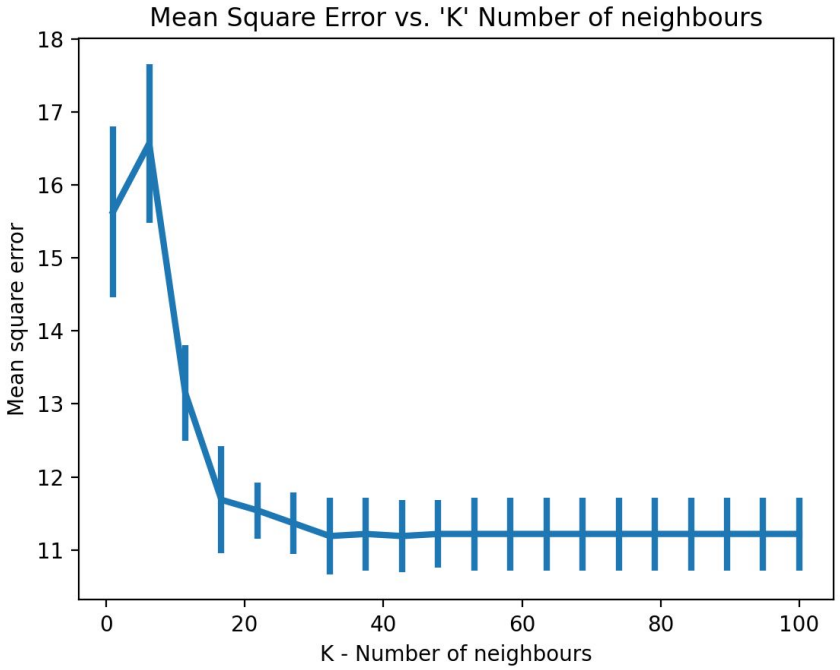

Figure 10 - Polynomial degree = 5; error vs knn

I would argue that introducing new features only serves to add even more noise to the data and doesn't enhance the model much at all.

Part (i) | C | Confusion Matrices for Logistic Regression, KNN & a baseline classifier

I note that no polynomial features are introduced to any of these classifiers, as they do not seem to increase the accuracy of the models as discussed above. The optimal values hyperparameter C = 0.001 (Logistic Regression) and KNN = 5 are used in the respective classifiers.

K Nearest Neighbours

TN = 8	FP = 69
FN = 25	TP = 135

Logistic Regression

TN = 0	FP = 77
FN = 0	TP = 160

Baseline Classifier - Predicts most frequent class

TN = 0	FP = 77
FN = 0	TP = 160

Analysis of the matrices discussed in (e)

Part (i) D | Plot the ROC curves for KNN, Logistic and Baseline Classifiers

For each classifier in question, values for 'y_score' (retrieved by either the *model.predict_proba()* method or *model.decision_function()* method) are passed along with y_test values to the *plot_roc_curve* function;

```
def plot_roc_curve(y_test, y_score, model):
   fpr, tpr, _ = roc_curve(y_test, y_score)
   plt.plot(fpr, tpr)
   plt.title(model + " ROC Curve")
   plt.xlabel('False positive rate')
   plt.ylabel('True positive rate')
   plt.show()
```

The roc curves are displayed below for each classifier;

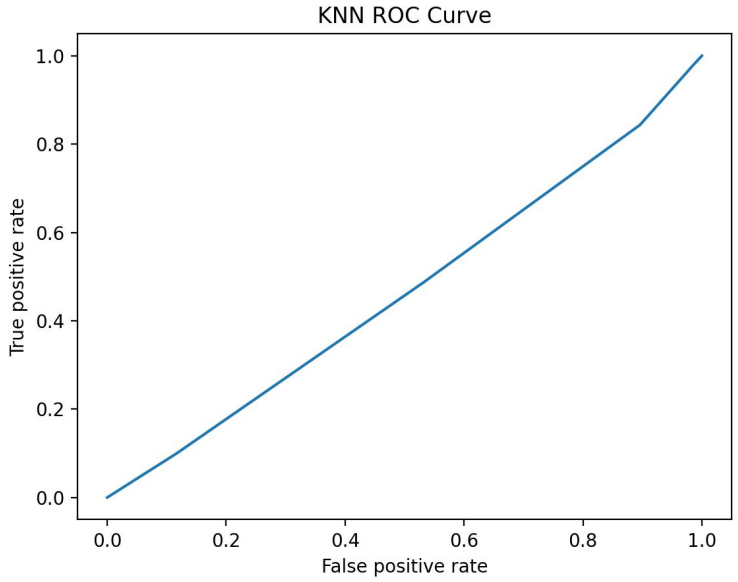

Figure 11 - KNN ROC Curve

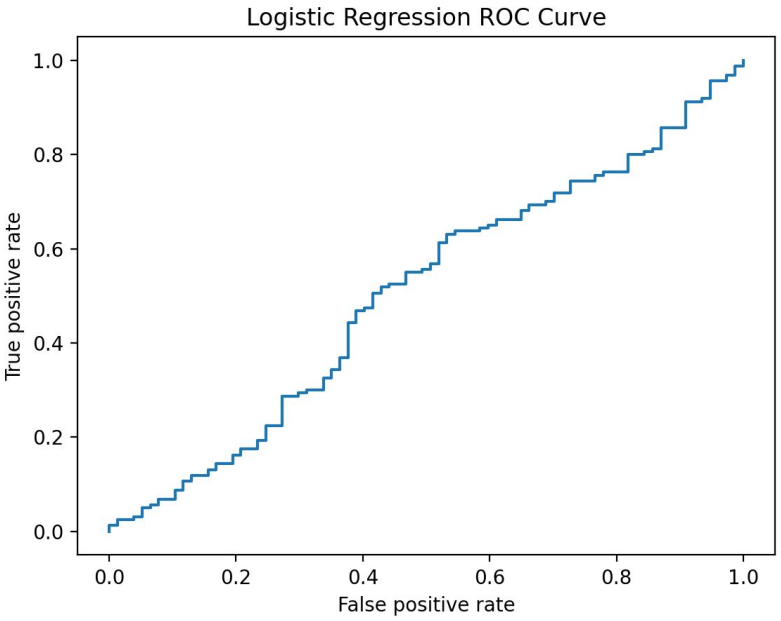

Figure 12 - Logistic Regression ROC Curve

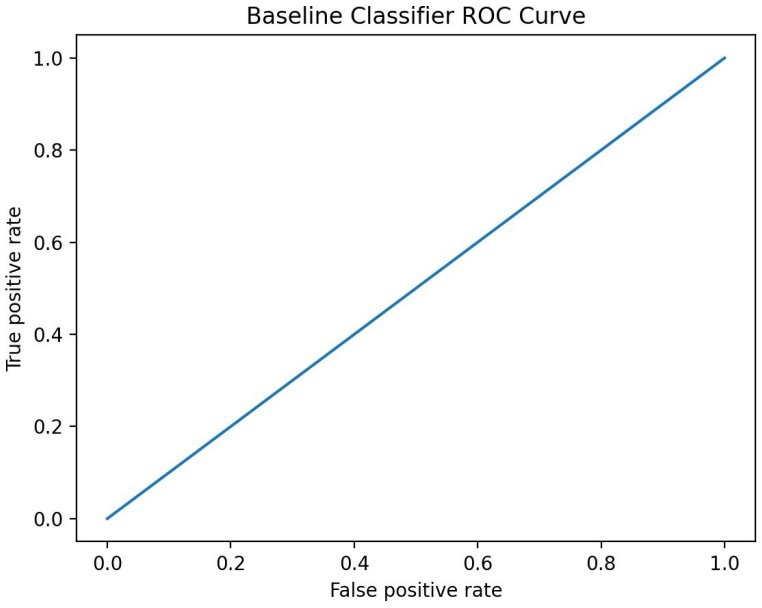

Figure 13 - Baseline Classifier ROC Curve

Part (i) E | Compare the classifiers and discuss

Evident from the confusion matrices and ROC plots, all of the classifiers perform quite poorly. With the behaviour of KNN & Logistic classifiers being extremely close in terms of their ROC plot to the baseline classifier, which simply classifies based on the most frequent class. This can be backed up with reference to the confusion matrices for each classifier, noting a pretty high false positive occurrence in all classifiers. Logistic regression notably seems to behave in the same way as the baseline classifier, simply resorting to classification

based on the most frequent class - caused by the poor quality of the data. KNN is not vulnerable to this flaw, but still performs just as poorly.

I would not recommend use of any of these classifiers as there is a very high chance they will be inaccurate. It is worth noting some metrics like accuracy & precision of each classifier here to put this in perspective.

- Accuracy = $\frac{TP + TN}{TP + TN + FP + FN}$
- FPR = $\frac{FP}{TN+FP}$

KNN Model

- Accuracy = 8 + 135 / (8 + 135 + 25 + 69) = 0.60
- FPR = 69 / 69 + 8 = 0.89

Logistic Regression

- Accuracy = 160 / 77 + 160 = 0.675
- FPR = 77 / 77 = 1

Baseline

- Accuracy = 160 / 160 + 77 = 0.675
- FPR = 77 / 77 = 1

While the reported accuracy isn't too bad, the FPR for all classifiers is pretty awful, again mounting more evident against these classifiers being useful at all for this data set. None of these classifiers would be suitable for this task - the training data is too poor.

Part (ii) | Repeat for data set 2

The same code was used for this part, simply changing the data read to the second data set.

A | Choosing an appropriate q and C for logistic regression model.

Mean squared error is plotted against q and C in each case. An optimal k-fold value of k=5 is chosen.

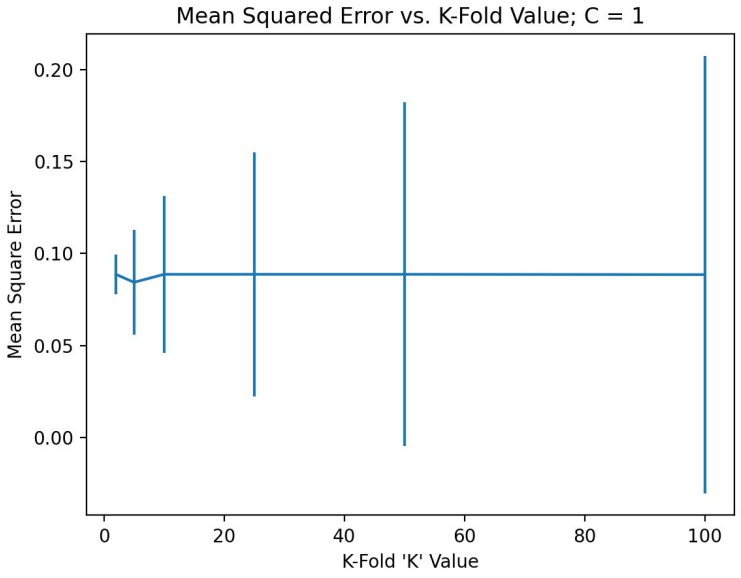

Figure 14 - MSE vs K-fold

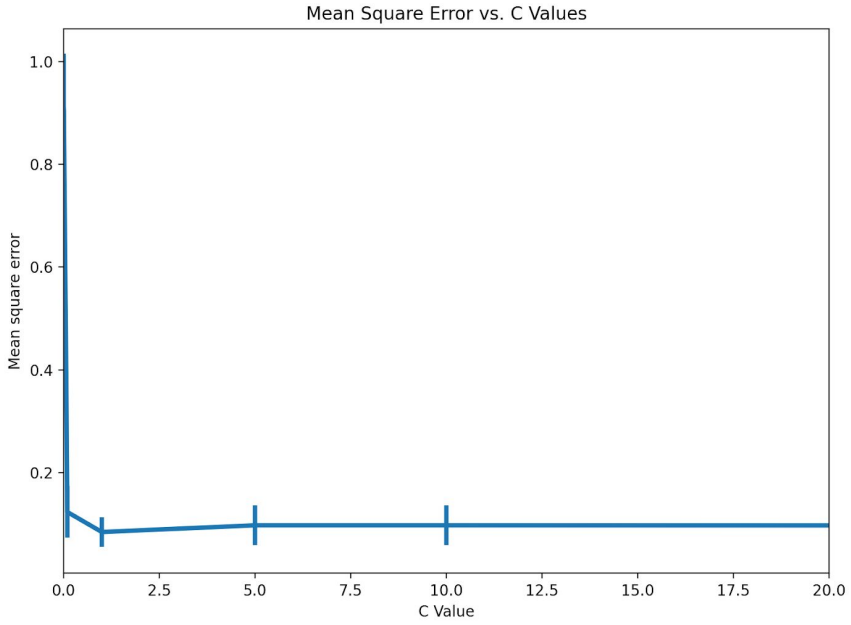

Figure 15 - MSE vs C Values

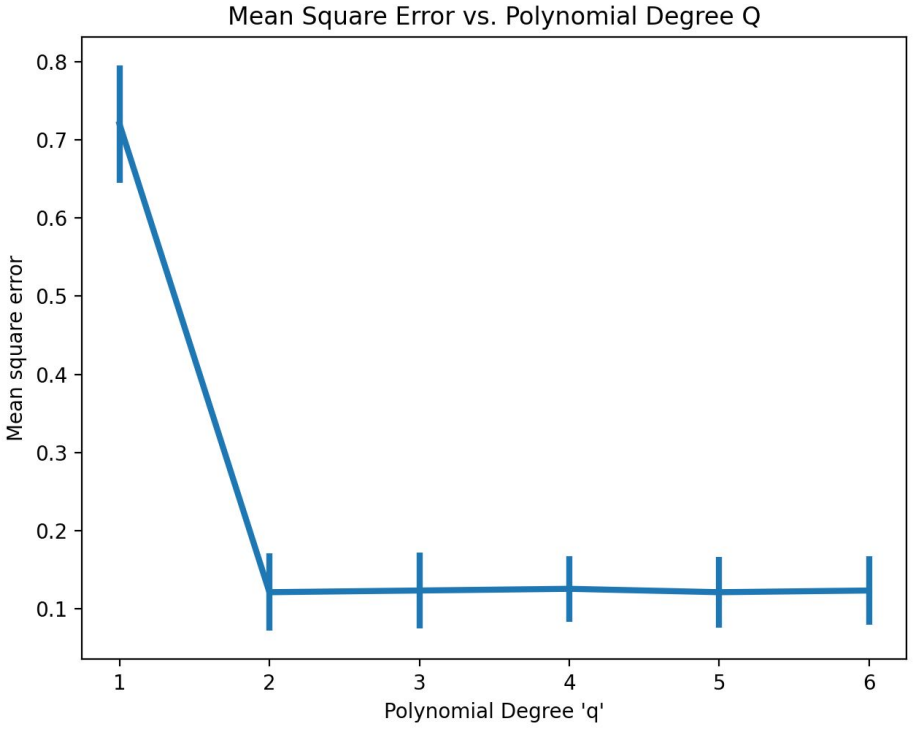

Figure 16 - MSE vs Polynomial order Q

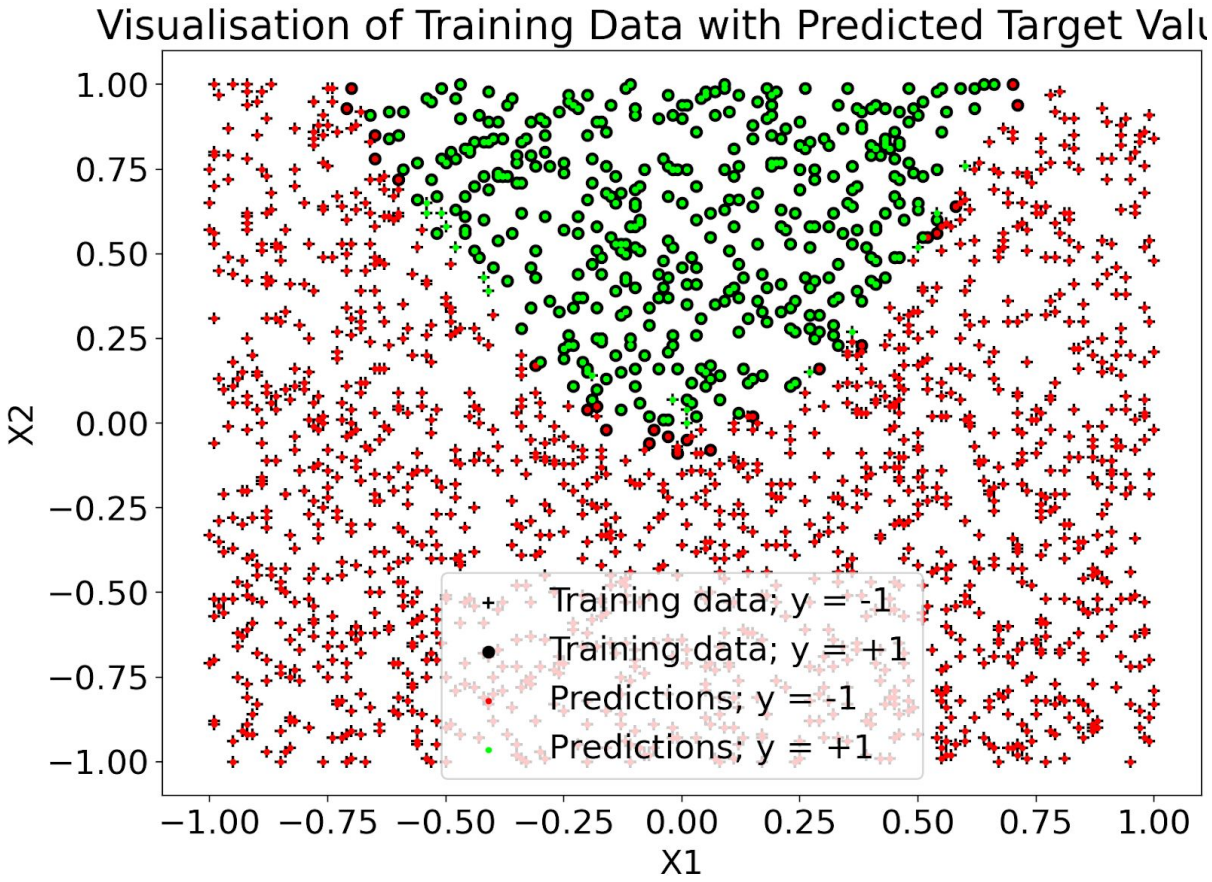

Figure 17 - Training data with predictions for q = 2, C = 1

The optimal values q = 2 and C = 1 are chosen because they have low error and avoid under and overfitting the model. Examining the prediction scatter plot they look pretty accurate. This data is of a much higher quality than that of the first set, and can be modelled pretty accurately with logistic regression.

B | Use KNN to model the data, choosing optimal 'k'

Generally, for KNN, a reasonable value for k will lie around $\sqrt{num\ data\ entries}$. This returns a value of approx k = 35. Examining the graph for Log loss vs KNN, an optimal value of about k = 43 can be selected.

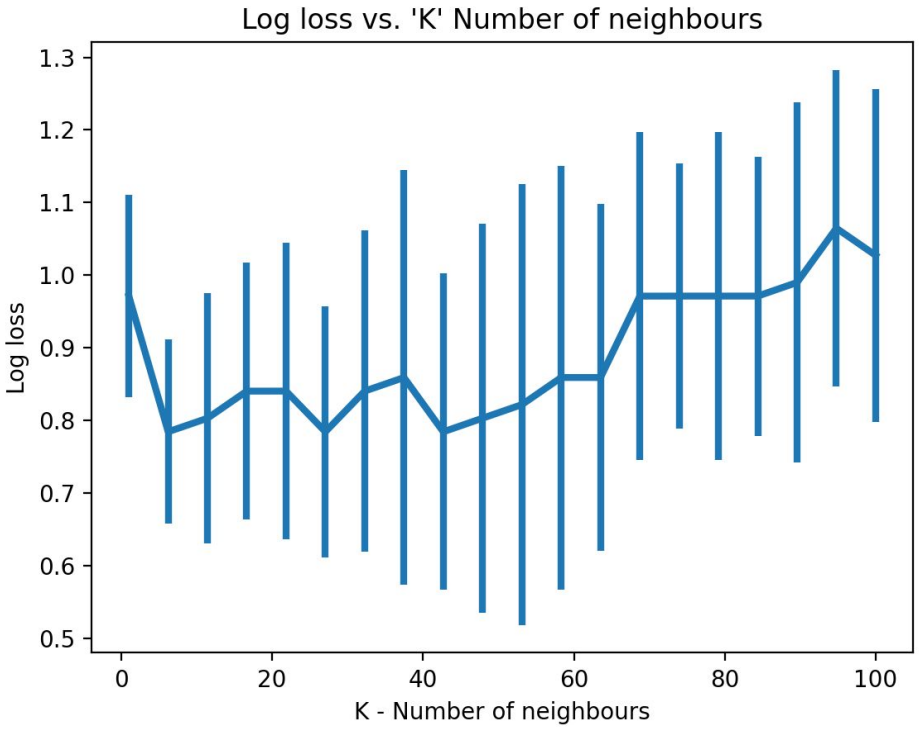

Figure 18 - Log loss vs. KNN

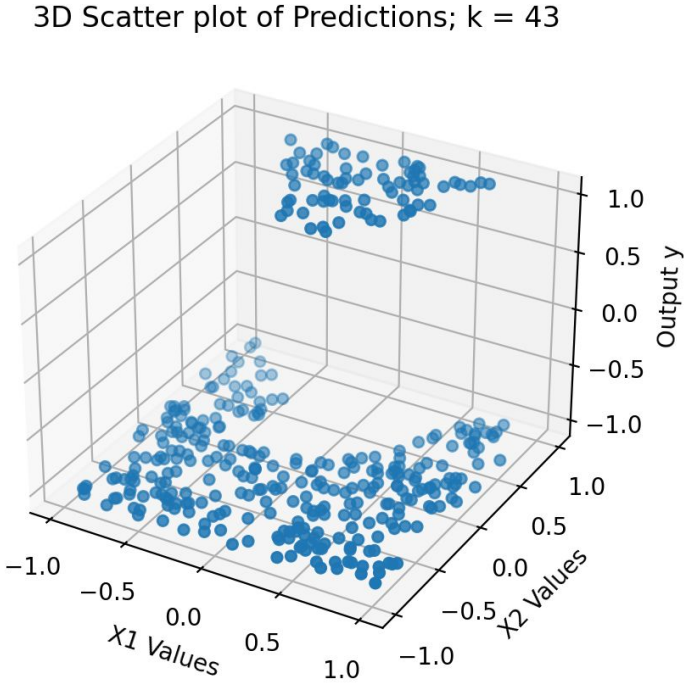

Figure 19 - Predictions for knn = 43

Regarding polynomial features, once again in this case adding them will only serve to add noise to the data and is unnecessary. The plot for log loss vs knn in figure 18 above is completely unchanged by adding more features. Best to keep this model simple and utilize the existing features as they do a pretty good job of classifying new data.

C | Confusion Matrices…

KNN Model

TN = 284	FP = 4
FN = 6	TP = 76

Logistic Regression

TN = 283	FP = 5
FN = 11	TP = 71

Baseline Model - Predicts most frequent class

TN = 288	FP = 0
FN = 82	TN = 0

D | Plot ROC Curves

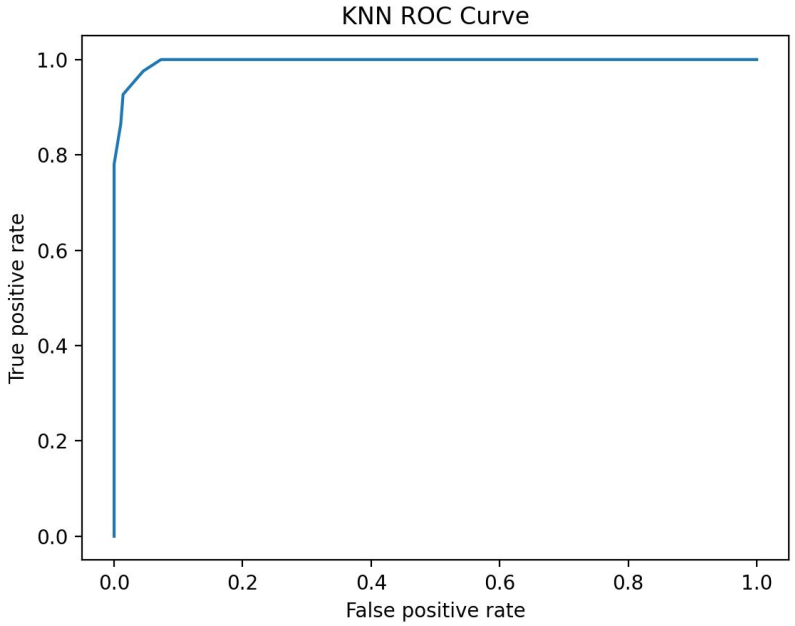

Figure 20 - ROC curve for KNN

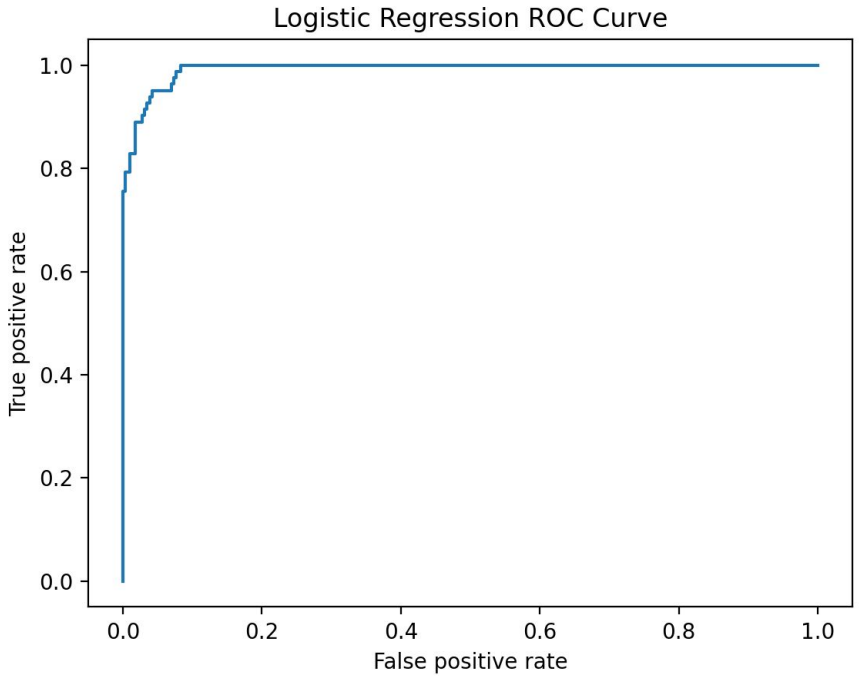

Figure 21 - ROC Logistic Regression

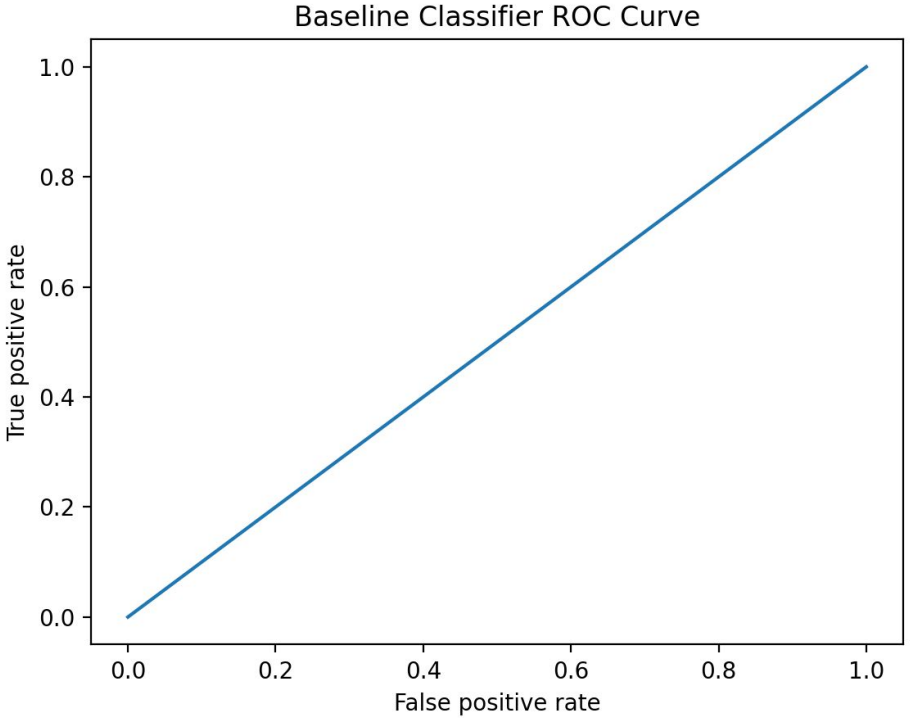

Figure 22 - ROC Baseline Classifier

E | Compare and contrast

False positive rate and accuracy of each classifier is computed to give more context to the performance of the models.

- Accuracy = $\frac{TP + TN}{TP + TN + FP + FN}$
- FPR = $\frac{FP}{TN+FP}$

KNN Model

- Accuracy = 0.972
- FPR = 0.0138

Logistic Regression

- Accuracy = 0.9567
- FPR = 0.0173661

Baseline

- Accuracy = 0.7783
- FPR = 0 however FNR = 1

KNN seems to be the best classifier for this job, however only slightly. Examining the ROC curves and taking other metrics of accuracy and false positive rate, KNN wins out only slightly over logistic regression. It also does not require addition of more features, simplifying the training process greatly. For this reason I believe the knn model is the best choice for this task.

Both models outperform the baseline model by a good margin, with a desirable ROC curve exhibiting high true positive rate vs. false positives.

Code

Logistic Regression, choosing q and C - Week4_part1.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
import sklearn.preprocessing as prep
from sklearn.model_selection import KFold

df = pd.read_csv("week4_set1.csv")

X1 = df.iloc[:, 0]
X2 = df.iloc[:, 1]
```

```
X = np.column_stack((X1, X2))
y = df.iloc[:, 2]



dataframe = pd.DataFrame({'column1': X1, 'column2': X2, 'column3': y})
minusDF = []
postivieDF = []

for grouped_y, grouping_x in dataframe.groupby(["column3"]):
   if grouped_y == -1:
       minusDF = grouping_x  # X rows that have y = -1
   else:
       positiveDF = grouping_x  #  X rows that have y = +1

polynomial_features = prep.PolynomialFeatures(degree=3)  #  use initial q
new_features = polynomial_features.fit_transform(X)

initial_C = 1
split_values = [2, 5, 10, 25, 50, 100]
mean_array = []*len(split_values)
stan_dev_array = []*len(split_values)
for i, split in enumerate(split_values):

   print("--- SPLIT = ", split, " ---")
   kf = KFold(n_splits=split)
   loopCount = 0
   sum_errors = 0
   error_array = []*split
   for train, test in kf.split(new_features):

       loopCount = loopCount + 1
       logRegressionModel = LogisticRegression(
           C=initial_C, penalty='l1', solver="saga", max_iter=7000,
random_state=0)
       logRegressionModel.fit(new_features[train], y[train])

       predictions = logRegressionModel.predict(new_features[test])
       # print("Intercept: ",
       #       lassoModel.intercept_, "\nCOEF: ", lassoModel.coef_, "ERROR:
", mean_squared_error(y[test], predictions))
       mse = mean_squared_error(y[test], predictions)
```

```
        sum_errors = sum_errors + mse
        error_array.append(mse)
    mean = sum_errors / split
    variance = np.var(error_array)
    print("Mean: ", mean, "- Variance: ", variance)
    mean_array.append(mean)
    stan_dev_array.append(np.array(error_array).std())

plt.figure(1)
plt.errorbar(split_values, mean_array, yerr=stan_dev_array)
plt.xlabel("K-Fold 'K' Value")
plt.ylabel("Mean Square Error")
plt.title("Mean Squared Error vs. K-Fold Value; C = 1")
plt.show()

index_of_min_k = mean_array.index(min(mean_array))
print("MIN INDEX: ", index_of_min_k)
optimal_k = split_values[index_of_min_k]
print("optimal K", optimal_k)


def determine_optimal_q(c):
    kf = KFold(n_splits=optimal_k)
    mean_error = []
    std_error = []
    q_range = [1, 2, 3, 4, 5, 6]
    for i, q in enumerate(q_range):
        print("--- Trying Q Value: ", q)
        polynomial_features = prep.PolynomialFeatures(degree=q)
        new_features = polynomial_features.fit_transform(X)
        logRegressionModel = LogisticRegression(
            C=1, penalty='l2', solver="saga", max_iter=7000, random_state=0)
        temp = []

        for train, test in kf.split(new_features):
            logRegressionModel.fit(new_features[train], y[train])
            predictions = logRegressionModel.predict(new_features[test])
            temp.append(mean_squared_error(y[test], predictions))

        mean_error.append(np.array(temp).mean())
```

```
        std_error.append(np.array(temp).std())
    plt.figure(2)
    plt.errorbar(q_range, mean_error, yerr=std_error, linewidth=3)
    plt.xlabel("Polynomial Degree 'q'")
    plt.ylabel("Mean square error")
    plt.title("Mean Square Error vs. Polynomial Degree Q")
    # plt.show()
    curr_min = min(mean_error)
    indexOfMinimum = [i for i, j in enumerate(mean_error) if j == curr_min]
    return indexOfMinimum[0] + 1  #  take first element for simplest model


c_values = [0.001, 0.01, 0.1, 1, 5, 10, 100, 1000]


def determine_optimal_C():
    kf = KFold(n_splits=optimal_k)
    mean_error = []
    std_error = []
    for i, C in enumerate(c_values):
        print("--- Trying C Value: ", C)
        polynomial_features = prep.PolynomialFeatures(degree=5)
        new_features = polynomial_features.fit_transform(X)
        logRegressionModel = LogisticRegression(
            C=C, penalty='l1', solver="saga", max_iter=7000, random_state=0)

        temp = []
        for train, test in kf.split(new_features):
            logRegressionModel.fit(new_features[train], y[train])
            predictions = logRegressionModel.predict(new_features[test])
            temp.append(mean_squared_error(y[test], predictions))

        mean_error.append(np.array(temp).mean())
        std_error.append(np.array(temp).std())
    plt.figure(3)
    plt.errorbar(c_values, mean_error, yerr=std_error, linewidth=3)
    plt.xlabel("C Value")
    plt.ylabel("Mean square error")
    plt.title("Mean Square Error vs. C Values")
    plt.xlim([0, 20])
    # plt.show()
```

```
    curr_min = min(mean_error)
    indexOfMinimum = [i for i, j in enumerate(mean_error) if j == curr_min]
    print("OPTIMAL C must be: ", c_values[indexOfMinimum[0]])
    #  take first element for simplest model
    return indexOfMinimum[0]


def run(q, c):
    polynomial_features = prep.PolynomialFeatures(degree=q)
    new_features = polynomial_features.fit_transform(X)
    logRegressionModel = LogisticRegression(
        C=c_values[c], penalty='l1', solver="saga", max_iter=7000,
random_state=0)
    logRegressionModel.fit(new_features, y)
    predictions = logRegressionModel.predict(new_features)

    prediction_dataframe = pd.DataFrame(
        {'column1': X1, 'column2': X2, 'column3': predictions})
    minus_pred_DF = []
    positive_pred_DF = []

    for grouped_y, grouping_x in prediction_dataframe.groupby(["column3"]):
        if grouped_y == -1:
            minus_pred_DF = grouping_x  # X rows that have y = -1
        else:
            positive_pred_DF = grouping_x  #  X rows that have y = +1
    # PLOT
    plt.figure(1)
    plt.title(
        "Visualisation of Training Data with Predicted Target Values")
    plt.rc('font', size=18)
    plt.rcParams['figure.constrained_layout.use'] = True

    plt.scatter(minusDF.iloc[:, 0], minusDF.iloc[:, 1],
                marker="+", c="black", s=40)
    plt.scatter(positiveDF.iloc[:, 0],
                positiveDF.iloc[:, 1], marker="o", c="black")

    plt.scatter(minus_pred_DF.iloc[:, 0],
                minus_pred_DF.iloc[:, 1], marker="o", c="red", s=6)
    plt.scatter(positive_pred_DF.iloc[:, 0],
```

```
                positive_pred_DF.iloc[:, 1], marker="o", c="lime", s=6)

    # plt.xlim(-1, 1)
    plt.legend(["Training data; y = -1", "Training data; y = +1",
                "Predictions; y = -1", "Predictions; y = +1"])

    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.show()


index_of_optimal_c = determine_optimal_C()
optimal_q = determine_optimal_q(index_of_optimal_c)

while(True):
    print("Using Q: ", optimal_q, "\nUsing C: ", c_values[index_of_optimal_c])
    run(optimal_q, index_of_optimal_c)
    increase_q = input(
        "Based on visual analysis, do you want to increase q? y/n")
    if (increase_q == "y"):
        optimal_q = optimal_q + 1
    else:
        break
```

KNN Model, choosing K value - week4_knn_1.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import log_loss
from sklearn.metrics import accuracy_score
import sklearn.preprocessing as prep
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsClassifier


df = pd.read_csv("week4_set2.csv")
```

```
X1 = df.iloc[:, 0]
X2 = df.iloc[:, 1]
X = np.column_stack((X1, X2))
y = df.iloc[:, 2]

# polynomial_features = prep.PolynomialFeatures(degree=5)
# X = polynomial_features.fit_transform(X)

k_range = np.linspace(1, 100, 20)
print("KRANGE: ", k_range)

mean_error = []
std_error = []
kf = KFold(n_splits=5)

for i, k in enumerate(k_range):
    knn_model = KNeighborsClassifier(n_neighbors=round(k))
    print("Trying k: ", round(k))
    temp = []

    for train, test in kf.split(X):
        knn_model.fit(X[train], y[train])
        predictions = knn_model.predict(X[test])
        temp.append(log_loss(y[test], predictions))
        print("--> ", log_loss(y[test], predictions))

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

    fig = plt.figure(k)

    ax = fig.add_subplot(111, projection='3d')

    sc = ax.scatter(X1[test], X2[test], predictions, label="Training Data")


    ax.set_title(
        "3D Scatter plot of Predictions; k = " + str(round(k)))
    ax.set_xlabel("X1 Values")
    ax.set_ylabel("X2 Values")
    ax.set_zlabel("Output y")
```

```
    plt.show()



plt.figure(2)
plt.errorbar(k_range, mean_error, yerr=std_error, linewidth=3)
plt.xlabel("K - Number of neighbours")
plt.ylabel("Log loss")
plt.title("Log loss vs. 'K' Number of neighbours")
plt.show()

k_index_min_error = mean_error.index(min(mean_error))
print("MEAN ERROR: ", mean_error)
optimal_k = k_range[k_index_min_error]
print("OPTIMAL K: ", optimal_k)
```

Confusion Matrices and ROC Curves - week4_confusion_matrix.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
import sklearn.preprocessing as prep
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.dummy import DummyClassifier
from sklearn.metrics import roc_curve



def calculate_confusion_matrix(truth, pred, model):
    # cf = confusion_matrix(truth, pred)
    tn, fp, fn, tp = confusion_matrix(truth, predictions).ravel()
    print("--- Model ", model, " ---")
    print("Confusion Matrix (tn, fp, fn, tp): ", tn, fp, fn, tp)



def plot_roc_curve(y_test, y_score, model):
    fpr, tpr, _ = roc_curve(y_test, y_score)
    plt.plot(fpr, tpr)
```

```
    plt.title(model + " ROC Curve")
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.show()


df = pd.read_csv("week4_set2.csv")

X1 = df.iloc[:, 0]
X2 = df.iloc[:, 1]
X = np.column_stack((X1, X2))
y = df.iloc[:, 2]

# --- KNN MODEL ----
knn_model = KNeighborsClassifier(
    n_neighbors=5, weights="uniform")
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0)

knn_model.fit(X_train, y_train)
predictions = knn_model.predict(X_test)

calculate_confusion_matrix(y_test, predictions, "KNN")
y_score = knn_model.predict_proba(X_test)
plot_roc_curve(y_test, y_score[:, 1], "KNN")

# --- LOGISTIC REGRESSION ---
# not using polynomial features as they dont make much of a difference
polynomial_features = prep.PolynomialFeatures(degree=2)  #  use initial q
X = polynomial_features.fit_transform(X)
log_model = LogisticRegression(C=1, penalty="l2", random_state=0)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0)
log_model.fit(X_train, y_train)
predictions = log_model.predict(X_test)

calculate_confusion_matrix(
    y_test, predictions, "Logistic regression")
y_score = log_model.decision_function(X_test)
plot_roc_curve(y_test, y_score, "Logistic Regression")
```

```
# --- BASELINE ---
dummy_model = DummyClassifier(strategy="most_frequent", random_state=0)
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.2, random_state=0)
dummy_model.fit(X_train, y_train)
predictions = dummy_model.predict(X_test)

calculate_confusion_matrix(
   y_test, predictions, "Dummy Classifier")
y_score = dummy_model.predict_proba(X_test)
plot_roc_curve(y_test, y_score[:, 1], "Baseline Classifier")
```