CS7CS4 Machine Learning
Week 4 Assignment

Michael Millard
Student ID: 24364218
Dataset 1 ID: # id:20-40-20-0
Dataset 2 ID: # id:11-22-11-0

October 19, 2024

Question (i)

Question (i)(a)

The first dataset for this assignment consisted of two input features, X_1 and X_2, and target values y, which were labels of either -1 or +1. This dataset had 1887 records, 1285 of which were +1 labels and 602 of which were -1 labels. As such, the ratio of +1 labels to -1 labels was 2.13455; suggesting that this dataset is moderately imbalanced with more than twice as many +1 labels as there are -1 labels. As a result of this, in conjunction with the fact that no context is provided regarding desirable model performance (e.g. minimal false positvies and false negatives), Receiver Operating Characteristic Area Under the Curve (ROC AUC) was chosen over F_1 score as the appropriate metric when evaluating model performance as it is more robust when working with imbalanced datasets. Figure 1 below depicts the entire dataset with input feature X_1 plotted against X_2 in a scatter plot where the shape and colour of the marker is determined by the corresponding label value (green cross for +1, turquoise dot for -1).

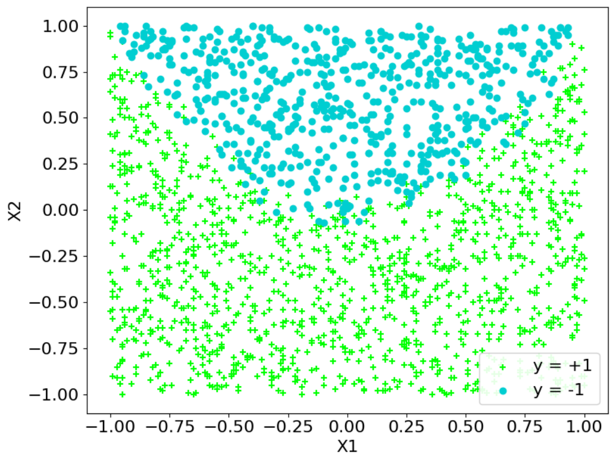

Figure 1: Scatter plot of the entire first dataset

The dataset was split 80:20 into training and testing subsets using sklearn's *train_test_split* function. All model training was done on the training dataset, X_{train} and y_{train}, and all model performance evaluations were done on the testing dataset, X_{test} and y_{test}.

The first classifier trained for this dataset was a logistic regression model with an L_2 penalty added to the cost function. In order to select the best logistic regression model for this dataset, 5-fold cross validation was performed to find the optimal values of the polynomial features power, q, and the penalty weight parameter, C. Initially, cross validation was used to find these parameter values individually (q first, then C) and it was found that the optimal values were q = 4 and C = 8. However, this approach does not consider all possible combinations of q and C that might yield greater model performance than finding them sequentially does. As such, cross validation was performed in a nested loop (looping through q values in the outer loop, C values in the inner loop). In each outer loop iteration, the mean (μ_{AUC}) an standard deviation (σ_{AUC}) of the ROC AUC, as well as an additional "worst case" metric (the difference of μ_{AUC} and σ_{AUC}) were calculated. For each q value, the C value that achieved the best "worst case" result was recorded. The best model was chosen as the one with the best "worst case" value as this metric takes the deviation of the model performance into consideration, rather than simply selecting the model with the highest ROC AUC mean. Table 1 below shows the cross validation results for each of these aforementioned parameters and metrics.

q	Best C	μ_{AUC}	σ_{AUC}	$\mu_{AUC} - \sigma_{AUC}$
1	6.0	0.9461	0.0089	0.9371
2	6.0	0.9969	0.0006	0.9964
3	4.0	0.9970	0.0005	0.9964
4	8.0	0.9969	0.0006	0.9963
5	6.0	0.9968	0.0006	0.9962
6	10.0	0.9967	0.0007	0.9960
7	10.0	0.9967	0.0007	0.9961
8	10.0	0.9967	0.0007	0.9960

Table 1: Cross validation results for various q values and their best performing C values

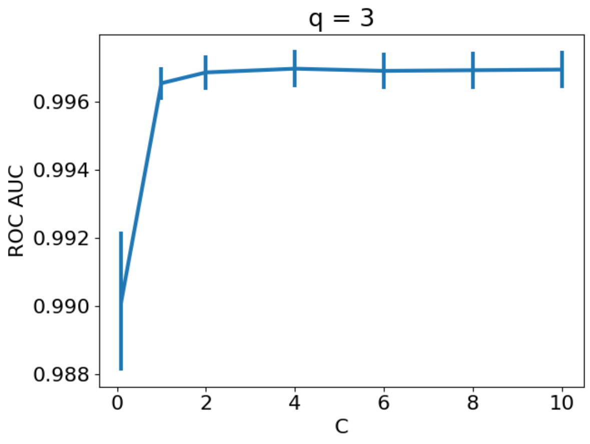

Figure 2: ROC AUC error bar plot for various values of C when q = 3

It can be seen in Table 1 above that all models trained on a polynomial features dataset with q > 1 had extremely high means and very low standard deviations. The best performing model was found to be a logistic regression model trained on a polynomial features dataset of maximum order polynomial q = 3 and a L_2 penalty weight parameter C = 2. This row is highlighted in Table 1 and the error bar plot corresponding to where this combination occurs is shown in Figure 2 above. The plot shows the mean and standard deviation of the ROC AUC when q = 3 as a function of penalty weight parameter C.

Having found the optimal values of q and C, the model performance was evaluated by seeing how well it generalized to the test dataset, X_{test}. Figure 3 below shows the predictions made by the model superimposed on the true test dataset labels with the decision boundary plotted as well. It is evident that the model performs extremely well. This was confirmed using sklearn's *classification_report* function, which reported that the model had an accuracy of 96% on the test dataset.

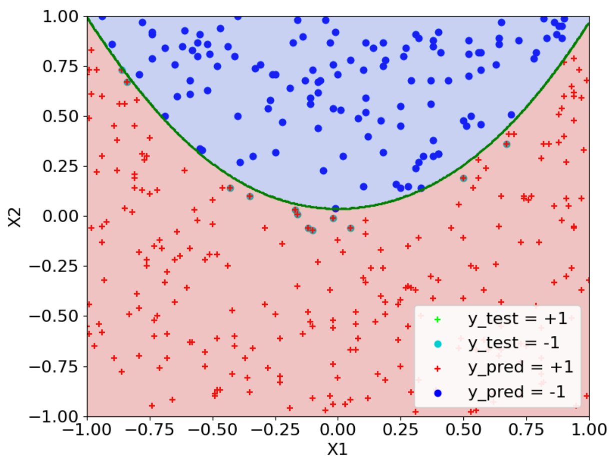

Figure 3: Scatter plot of best logistic regression model predictions on test dataset with decision boundary

Question (i)(b)

Two k-Nearest Neighbours (kNN) models were then trained. The first had uniform weights applied to the k-nearest neighbours and the second had distance weights applied to the k-nearest points. Cross validation was employed to find the optimal k value for each of these kNN models. Again, ROC AUC was chosen as the metric for best model selected with the mean (μ_{AUC}) and standard deviation (σ_{AUC})

of the ROC AUC, along with the "worst case" ((μ_{AUC}) - (σ_{AUC}) value being computed for each value of k. The results for the first kNN model with uniform weights are tabulated below in Table 2 alongside Figure 4, which depicts the error bar plot corresponding to these values.

k	μ_{AUC}	σ_{AUC}	$\mu_{AUC} - \sigma_{AUC}$
1	0.9460	0.0096	0.9364
3	0.9755	0.0055	0.9700
5	0.9894	0.0050	0.9844
7	0.9909	0.0039	0.9871
9	0.9933	0.0030	0.9903
11	0.9954	0.0008	0.9945
13	0.9953	0.0009	0.9944
15	0.9953	0.0006	0.9947
25	0.9959	0.0006	0.9952
35	0.9956	0.0006	0.9950

Table 2: Cross validation results of kNN model with uniform weights for various values of k

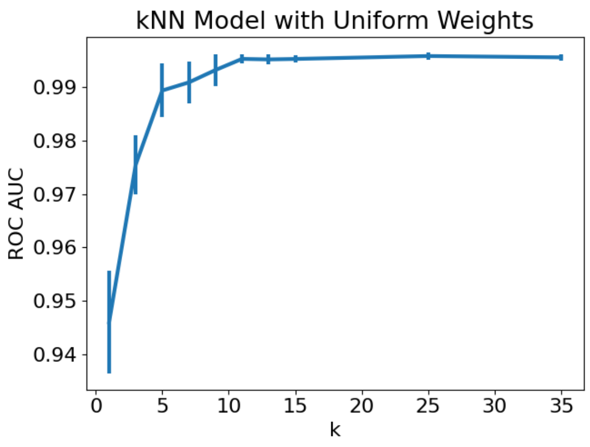

Figure 4: ROC AUC error bar plot of kNN model with uniform weights for various values of k

From Table 2 and Figure 4 above, it was found that k = 25 yielded the greatest model performance with a "worst case" ROC AUC value of 0.9952. Having selected the best kNN model with uniform weights, the same cross validation was then performed with kNN models with distance weights. The results and their corresponding error bar plot are provided below in Table 3 and Figure 5.

k	μ_{AUC}	σ_{AUC}	$\mu_{AUC} - \sigma_{AUC}$
1	0.9460	0.0096	0.9364
3	0.9723	0.0044	0.9679
5	0.9856	0.0028	0.9829
7	0.9873	0.0018	0.9855
9	0.9896	0.0025	0.9872
11	0.9917	0.0033	0.9883
13	0.9917	0.0032	0.9885
15	0.9918	0.0032	0.9886
25	0.9919	0.0032	0.9887
35	0.9918	0.0033	0.9885

Table 3: Cross validation results of kNN model with distance weights for various values of k

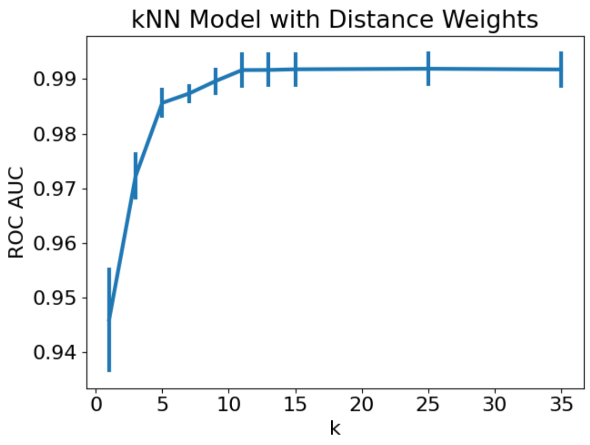

Figure 5: ROC AUC error bar plot of kNN model with distance weights for various values of k

It was found that once again that having k = 25 resulted in the best performing kNN model, this time with distance weights. The best kNN model in this case achieved a "worst case" value of 0.9887, slightly lower than the best performing kNN model with uniform weights. As such, the kNN model used in subsequent questions was the uniformly weighted one with k = 25.

The model performance was then evaluated on the test dataset. The predictions it made are superimposed on the true test dataset labels in the scatter plot in Figure 6 below, with the kNN decision boundary drawn in green. This kNN model performed exceptionally well with an accuracy of 95%, as returned by sklearn's *classification_report*.

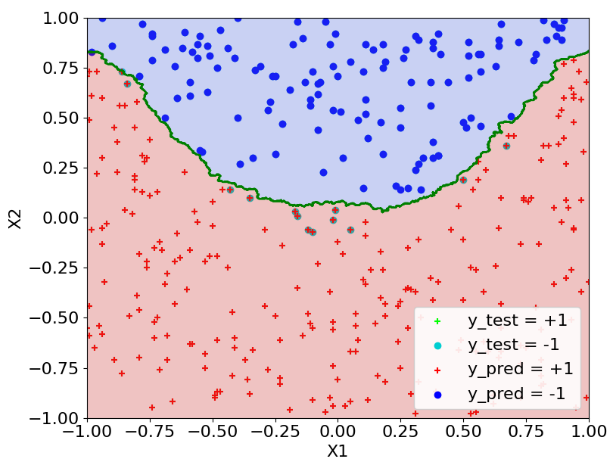

Figure 6: Scatter plot of best kNN model predictions on test dataset with decision boundary

Question (i)(c)

The confusion matrices for the best performing logistic regression and kNN models above were obtained using sklearn's *confusion_matrix* function. For both models, a prediction vector, y_{pred} was created as the predictions made by each model on the test dataset, X_{test}. The confusion matrix was then created by passing the predictions, y_{pred}, and the true test set labels, y_{test}, to the *confusion_matrix* function as arguments. The confusion matrix for the logistic regression model is shown below in Table 4, alongside the confusion matrix for the kNN model in Table 5.

True Positive	123	12
True Negative	4	239
	Predicted Positive	Predicted Negative

Table 4: Confusion matrix for logistic regression model

True Positive	122	13
True Negative	5	238
	Predicted Positive	Predicted Negative

Table 5: Confusion matrix for k-Nearest Neighbours model

It is evident from Tables 4 and 5 above that the both of these models perform extremely similarly with only one extra false positive (bottom left corner of matrix) and one extra false negative (top right corner of matrix) made by the kNN model in comparison to the logistic regression model. Both models have extremely low false positives and false negatives, suggesting that their performance is exceptionally good.

Two baseline classifiers were then introduced to compare the performance of the logistic regression and kNN models with. The first was a most frequent classifier and the second was a random classifier. Both of these baseline models were created using sklearn's *DummyClassifier* function, with the 'strategy' argument set to 'most_frequent' and 'uniform', respectively. The confusion matrices for these baseline classifiers were produced on the same test set, y_{pred}, and are shown below in Tables 6 and 7, respectively.

True Positive	0	135
True Negative	0	243
	Predicted Positive	Predicted Negative

Table 6: Confusion matrix for most frequent dummy classifier

True Positive	70	65
True Negative	111	132
	Predicted Positive	Predicted Negative

Table 7: Confusion matrix for random dummy classifier

As expected, the most frequent classifier only predicted one label (-1 in this case), suggesting that y_test consisted of more -1 labels than +1 labels. As such, it was able to predict the label correctly more than half the time, with a high number of false positives being an obvious outcome. The random classifier made roughly the same number of +1 predictions as -1 predictions, as expected. Since there were more -1 labels than +1 labels in y_test, it naturally made a high number of false positive predictions.

Question (i)(d)

For this question, the ROC curves of both the logistic regression model and the kNN model, along with those of the two baseline classifiers, were plotted on the same set of axes, as shown below in Figure 7. The ROC curve for each of the models was obtained using sklearn's *roc_ curve* function.

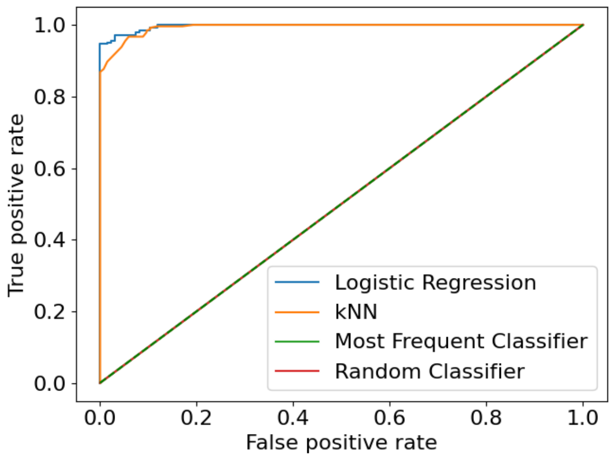

Figure 7: ROC curves for logistic regression model, kNN model, most frequent classifier (baseline 1), and random classifier (baseline 2)

It is evident in Figure 7 that both the logistic regression model (blue ROC curve) and the kNN model (orange ROC curve) perform extremely well in that they get very close to the top left corner of the ROC space (top left corresponds to an ideal model with a 100% true positive rate and 0% false positive rate). On the other hand, the baseline classifiers (green ROC curve for most frequent classifier, red ROC curve for random classifier) are along the diagonal. This is expected in the case of the most frequent classifier in that it only ever predicts one label (the most common one), meaning it will either correctly predict all true positive labels, with every incorrect prediction being a false positive (most common +1 case); or it will correctly predict all true negative labels, with every incorrect prediction being a false negative (most common -1 case). In the case of the random classifier, the ROC curve is expected to be along the diagonal as it is randomly predicting one of two possible labels, meaning over the entire dataset it's +1 label and -1 label predictions will be roughly 50:50.

Question (i)(e)

Some of the analysis needed for this question has already been done above. As mentioned in question (i)(c), both the logistic regression model and the kNN model generalized extremely well with very low false positive and false negative predictions, evident in their confusion matrices shown in Tables 4 and 5, respectively. These findings are consistent with the ROC curves for these models shown in Figure 7 above, where both models get very close to the top left corner of the ROC plot (close to performing like ideal models). The logistic regression model very slightly outperforms the kNN model, with one extra true positive and one extra true negative prediction over the kNN model. This is also consistent with their respective "worst case" metrics in questions (i)(a) and (i)(b), respectively; where the logistic regression model had a "worst case" ROC AUC of 0.9964 (Table 1), compared to the kNN model's 0.9952 (Table 2).

Both of these models significantly outperform both the most frequent classifier and the random classifier (the baseline models used). The most frequent classifier predicted correctly more than half the time as expected since the test dataset was slightly imbalanced (more -1 labels than +1 labels). However, it naturally had a very high number of false negatives. The random classifier made roughly the same number of positive predictions as it did negative ones. as expected. It had a significant number of false positives and false negatives due to the random nature of its predictions and as a result performed considerably worse than the other models.

Even though both the logistic regression model and kNN model had extremely similar performance results that far outperformed the baseline classifiers, the logistic regression model did ultimately outperform the kNN model and as such it would be my model of choice. However, it is worth noting that the kNN model would still perform extremely well on a dataset of this nature. There is no clear winner.

Question (ii)

Question (ii)(a)

The second dataset for this assignment was then read in. Again, it contained two input featurs, X_1 and X_2, and target labels, y. This dataset consisted of 1686 records, 1120 of which had labels of +1 and 566 of which had labels of -1. Therefore, the ratio of +1 labels to -1 labels was 1.9788 for this dataset, suggesting that it was also moderately imbalanced with roughly twice as many +1 labels as -1 labels. The scatter plot below in Figure 8 shows input feature X_1 plotted against X_2 with the marker of each point determined by its label (green cross for +1, blue dot for -1).

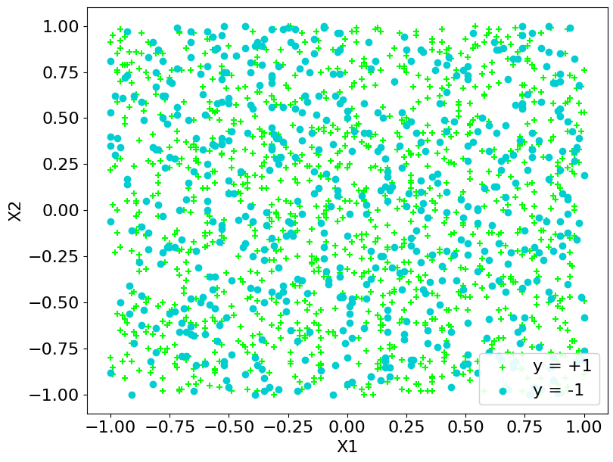

Figure 8: Scatter plot of the entire second dataset

In the same manner to the first dataset, this dataset was split 80:20 into training and test subsets, X_{train} and X_{test}, using sklearn's *train_ test_ split* function.

It was established in the first question that the nested loop approach to finding the optimal values of q and C for a logistic regression model with an L_2 penalty added to the cost function is better than the sequential approach (finding q first, then C). As such, the nested loop 5-fold cross validation approach was adopted for this question and the results are tabulated below in Table 8. Again, the ROC AUC was the chosen model performance metric and the mean (μ_{AUC}), the standard deviation (σ_{AUC}), and the "worst case" metric (μ_{AUC} - σ_{AUC}) were computed for each value of q.

q	Best C	μ_{AUC}	σ_{AUC}	$\mu_{AUC} - \sigma_{AUC}$
1	1.0	0.5104	0.0285	0.4819
2	0.1	0.4991	0.0265	0.4726
3	0.1	0.4818	0.0191	0.4627
4	10.0	0.4712	0.0115	0.4598
5	10.0	0.4861	0.0169	0.4692
6	8.0	0.4797	0.0159	0.4638
7	8.0	0.4847	0.0123	0.4724
8	10.0	0.4843	0.0126	0.4717

Table 8: Cross validation results for various q values and their best performing C values

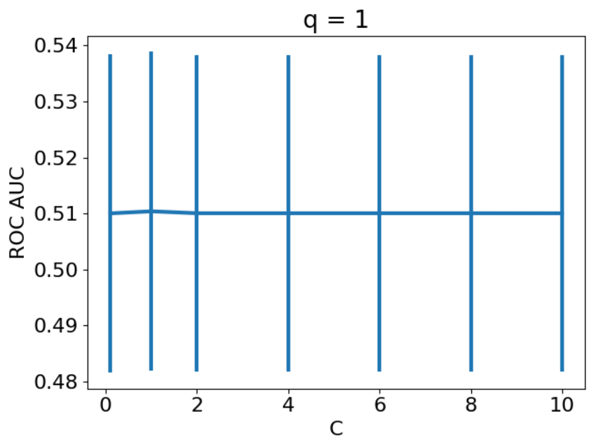

Figure 9: ROC AUC error bar plot for various values of C when q = 1

On this dataset, it was found that the optimal values for the logistic regression model were q = 1 and C = 1, as shown in the highlighted row in Table 8 above, The error bar plot above in Figure 9 shows the ROC AUC mean and standard deviation for various values of C when q = 1. It can be seen that the mean peaks slightly at C = 1 with the standard deviation remaining much the same for all values of C. It is clear that a logistic regression model performs poorly on this dataset in that only one of the resulting models (q = 1, C = 1) achieved a mean ROC AUC of greater than 0.5 (which is on par with the most frequent and random baseline classifiers). Factoring in the standard deviation (σ_{AUC}) for this model to calculate its "worst case", results in a value of 0.4819. This is significantly worse than the logistic regression model performance on the first dataset.

The performance of the best logistic regression model was assessed using test dataset, X_{test}, to see how well it generalizes. The resulting scatter plot is shown below in Figure 10. It can be seen that this model only predicted +1 labels, and as such will have identical performance to a most frequent classifier (provided +1 is the most frequent label in y_{test}). Using sklearn's *classification_report* function, it was found that this logistic regression model has an accuracy of 67%.

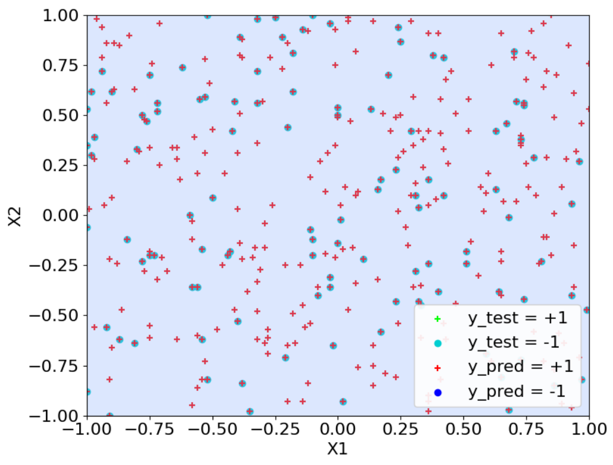

Figure 10: Scatter plot of best logistic regression model predictions on test dataset with decision boundary

Question (ii)(b)

Two kNN models were then trained on this dataset. Again, the first had uniform weights and the second had weights influenced by distance, The range of k values was changed slightly for this dataset (up to k = 55) to see if considering a greater number of neighbours yielded greater model performance. The ROC AUC was chosen as the appropriate model performance metric and the mean (μ_{AUC}), the standard deviation (σ_{AUC}), and the "worst case" metric (μ_{AUC} - σ_{AUC}) were computed for each value of k. The results of 5-fold cross validation to find the optimal value of k for a kNN model with uniform weights are provided below in Table 9, alongside the error bar plot depicting these same results in Figure 11.

k	μ_{AUC}	σ_{AUC}	$\mu_{AUC} - \sigma_{AUC}$
1	0.5124	0.0238	0.4885
3	0.4960	0.0231	0.4730
5	0.4876	0.0187	0.4689
7	0.4881	0.0222	0.4659
9	0.4893	0.0314	0.4579
15	0.4836	0.0246	0.4590
25	0.4947	0.0128	0.4819
35	0.5115	0.0191	0.4924
45	0.4990	0.0148	0.4842
55	0.4949	0.0168	0.4781

Table 9: Cross validation results of kNN model with uniform weights for various values of k

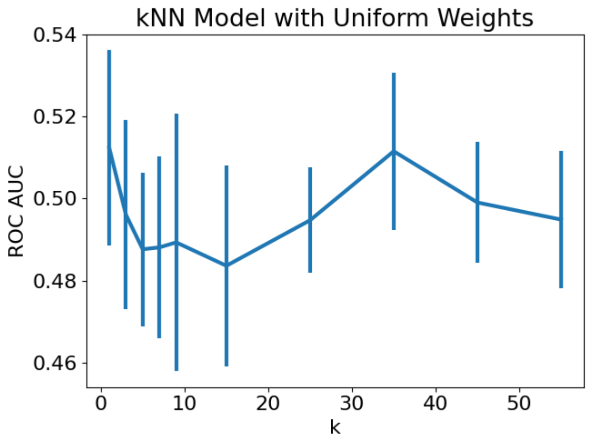

Figure 11: ROC AUC error bar plot of kNN model with uniform weights for various values of k

It is clear from the ROC AUC spikes at k = 1 and k = 35 in Figure 11 above that the kNN model performs best when considering these number of neighbours. This is a seemingly coincidental result considering the random nature of the dataset (see Figure 8). The "worst case" value is slightly greater when k = 35 (0.4924) than when k = 1 (0.4885) and as such, this was taken as the optimal value of k for a uniformly weighted kNN model. However, both of these results are still poor and indicate that a kNN model is not well suited to this dataset. Both of these results are roughly on par with the most frequent and random baseline classifiers.

The same 5-fold cross validation was then performed on a kNN model with distance weights to find the optimal value of k. The results are shown below in Table 10 and Figure 12.

k	μ_{AUC}	σ_{AUC}	$\mu_{AUC} - \sigma_{AUC}$
1	0.5124	0.0238	0.4885
3	0.5047	0.0251	0.4796
5	0.5016	0.0239	0.4777
7	0.5001	0.0225	0.4777
9	0.5004	0.0278	0.4726
15	0.5004	0.0289	0.4715
25	0.5085	0.0239	0.4846
35	0.5170	0.0223	0.4947
45	0.5101	0.0199	0.4902
55	0.5097	0.0173	0.4924

Table 10: Cross validation results of kNN model with distance weights for various values of k

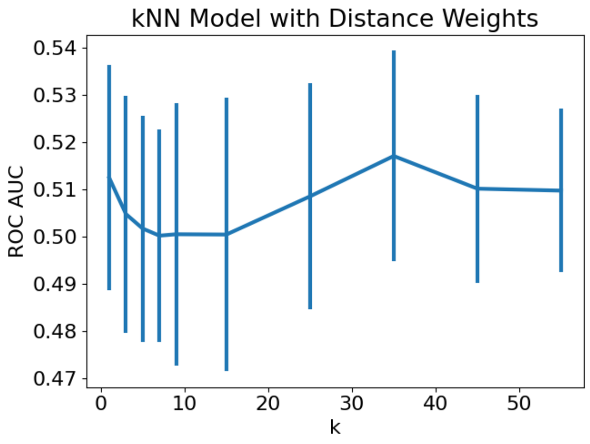

Figure 12: ROC AUC error bar plot of kNN model with distance weights for various values of k

In a similar manner to the uniformly weighted kNN model above, the ROC AUC error bar plot shown in Figure 12 peaks at k = 1 and k = 35. In general, the kNN model with distance weights had slightly higher mean (μ_{AUC}) and standard deviation (σ_{AUC}) values than the uniformly weighted kNN model. However, the overall performance of these kNN model on this dataset is still poor, again being roughly on par with the most frequent and random baseline classifiers. Since the "worst case" metric was highest for k = 35, this was taken as the optimal k value.

The best kNN model with distance weights very slightly outperformed the best performing uniformly weighted kNN model on this dataset and as such was the kNN model used in subsequent questions. The performance of this kNN model was evaluated using X_{test} and the resulting scatter plot showing its predictions superimposed on the true labels is provided below in Figure 13. Also included are the decision boundaries of the model, plotted as green contours. The model generalized averagely, as expected considering its mediocre performance on the training data. This is confirmed by sklearn's *classification_report* which returned an accuracy value of 64%.

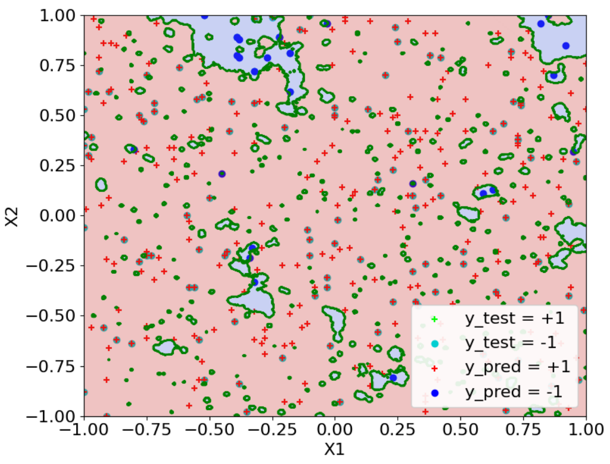

Figure 13: Scatter plot of best kNN model predictions on test dataset with decision boundary

Question (ii)(c)

The confusion matrices for the best performing logistic regression and kNN models above were once again obtained using sklearn's *confusion_matrix* function. For each model, prediction on the test dataset were computed and stored in a prediction vector, y_{pred}, which, along with the true test set labels, y_{test}, were passed to the *confusion_ matrix* function as arguments. The confusion matrix for the logistic regression model is shown in Table 11 below, and the confusion matrix for the kNN model is next to it in Table 12.

True Positive	0	113
True Negative	0	225
	Predicted Positive	Predicted Negative

Table 11: Confusion matrix for logistic regression model

True Positive	9	104
True Negative	17	208
	Predicted Positive	Predicted Negative

Table 12: Confusion matrix for k-Nearest Neighbours model

Once again, the chosen baseline classifiers to compare with were the most frequent classifier and a random classifier. The confusion matrices corresponding to these baseline classifiers are shown in Table 13 for the most frequent classifier, and Table 14 for the random classifier.

True Positive	0	113
True Negative	0	225
	Predicted Positive	Predicted Negative

Table 13: Confusion matrix for most frequent dummy classifier

True Positive	61	52
True Negative	117	108
	Predicted Positive	Predicted Negative

Table 14: Confusion matrix for random dummy classifier

It is immediately evident that the logistic regression model (Table 12) performs identically to the most frequent classifier (Table 13), making only predictions of -1. It will therefore correctly predict all true negatives but have a high number of false negatives. The kNN model (Table 12) made significantly more negative predictions than positive ones (only 26 of its 338 predictions were positive) and as such also had a high number of false negatives. The random classifier (Table 14) made roughly the same number of positive predictions as it did negative ones, as expected. Since the test dataset was imbalanced with more -1 labels than +1 labels, the random classifier made a high number of false positive predictions, unlike the other three models.

Question (ii)(d)

The ROC curves of the logistic regression model, the kNN model, the most frequent classifier, and the random classifier were plotted on the same set of axes, as shown below in Figure 14. Again, these ROC curves were obtained using sklearn's *roc_ curve* function for each model.

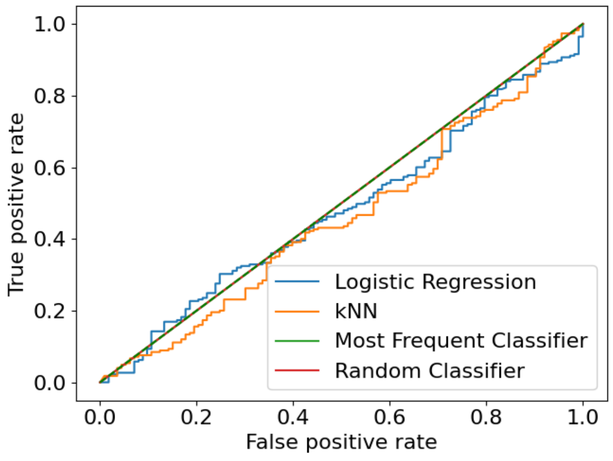

Figure 14: ROC curves for logistic regression model, kNN model, most frequent classifier (baseline 1), and random classifier (baseline 2)

It is immediately evident that none of the models perform well on the dataset. Again, the ROC curves of the baseline classifiers are along the diagonal, as expected; with the ROC curves of the logistic regression model (blue) and the kNN model (orange) roughly along the diagonal as well. As such, both of these models have a ROC AUC of approximately 50, which is consistent with their results observed during model selection (Table 8 for logistic regression and Table 10 for kNN). Neither of these models approach anywhere near the top left corner (ideal model) as they did in the case of the first dataset. Therefore, neither of these models outperform the baseline classifiers and are poor models for this dataset.

Question (ii)(e)

Since neither the best logistic regression model nor the best kNN model were able to outperform the baseline classifiers (most frequent and random classifiers), neither of them should be recommended as appropriate models for data of this nature. The best performing linear regression model performed identically to the most frequent classifier, only predicting negative labels. The kNN model performed similarly by making significantly more negative label predictions than positive ones, as shown in Table 12). As such all of these models made a very high number of false negative predictions. This was consistent with the resulting ROC curves for each of these models shown in Figure 14 above, where their curves were all roughly along the diagonal.

As such, I would recommend none of these models to be used to make predictions data of the same nature as that contained in this second dataset.

References

All Python code used in this assignment was based on snippets taken from lectures provided for this course. This includes the use of the sklearn toolkit, the numpy and pandas libraries, as well as plotting using matplotlib.

A Assignment Code

This appendix contains the code used to answer both questions (i) and (ii) of the assignment. The only change between them was changing the "dataset1" boolean below the imports to decide which dataset to read in.

```
# -------------------------------------------------- #
# Imports
# -------------------------------------------------- #

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

# -------------------------------------------------- #
# Read in data and set labels
# -------------------------------------------------- #

# Set which database (true = 1, false = 2)
dataset1 = False

# NB: dataset csv file must be in same directory as this solution
labels = ["X1", "X2", "y"]
if (dataset1):
    df = pd.read_csv("millardm_W4_dataset_1.csv", names=labels)
else:
    df = pd.read_csv("millardm_W4_dataset_2.csv", names=labels)
print("Dataframe␣head:")
print(df.head())

# Split data frame up into X (input features) and y (target values)
X1 = df["X1"].to_numpy()
X2 = df["X2"].to_numpy()
X = np.column_stack((X1, X2))
y = df["y"].to_numpy()
print("Number␣of␣elements:␣", len(y))
print("Number␣of␣+1␣labels:␣", len(y[y == 1]))
print("Number␣of␣-1␣labels:␣", len(y[y == -1]))
print("Labels␣ratio␣(+1/-1):␣", len(y[y == 1]) / len(y[y == -1]))

# Split dataset up into training dataset and test dataset (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
X1_train, X1_test, X2_train, X2_test = X_train[:, 0], X_test[:, 0], X_train[:, 1], X_tes

# -------------------------------------------------- #
# Question (i)(a)
# ------------------------------------------------------------ #
# Visualize dataset
# ------------------------------- #

# Configure plotting params
plt.rc('font', size=16)
plt.rcParams['figure.constrained_layout.use'] = True
plt.rcParams['legend.framealpha'] = 0.9
y_plus_color, y_minus_color, y_pred_plus_color, y_pred_minus_color = 'lime', 'darkturqu

# Create scatter plotof entire dataset ('+' for +1 labels, 'o' for -1 labels)
plt.figure(figsize=(8, 6))
plt.scatter(X1[y == 1], X2[y == 1], marker='+', color=y_plus_color, label='y = +1')
plt.scatter(X1[y == -1], X2[y == -1], marker='o', color=y_minus_color, label='y = -1')

# Label axes and add legend
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend(loc=4) # BR corner

# Save and show the plot
if (dataset1):
    plt.savefig("dataset_scatter_i_a.png")
else:
    plt.savefig("dataset_2_scatter_i_a.png")
plt.show()

# ------------------------------- #
# Finding optimal values of q and C for dataset individually (q first, then C)
# ------------------------------- #
# q cross validation
# ------------------------------- #

# Arrays for mean and std dev of f1 and ROC AUC scores for each q value
mean_f1_score = []
std_f1_score = []
mean_auc_score = []
std_auc_score = []

# Range of q values to sweep through
q_range = [1, 2, 3, 4, 5, 6, 7, 8]

# Sweep through q values (same logistic regression model each time)
log_model = LogisticRegression(penalty='l2', max_iter=1000)
for q in q_range:
    # Create polynomial features dataset
    X_poly = PolynomialFeatures(q).fit_transform(X)
    # Give cross_val_score entire dataset
    f1_cv_scores = cross_val_score(log_model, X_poly, y, cv=5, scoring='f1')
    auc_cv_scores = cross_val_score(log_model, X_poly, y, cv=5, scoring='roc_auc')
    mean_f1_score.append(np.array(f1_cv_scores).mean())
    std_f1_score.append(np.array(f1_cv_scores).std())
    mean_auc_score.append(np.array(auc_cv_scores).mean())
    std_auc_score.append(np.array(auc_cv_scores).std())

# f1 Score error bar
plt.errorbar(q_range, mean_f1_score, yerr=std_f1_score, linewidth=3)
plt.xlabel('q')
plt.ylabel('f1 Score')
if (dataset1):
    plt.savefig("ind_q_f1_errorbar_i_a.png")
else:
    plt.savefig("ind_q_f1_errorbar_i_a_2.png")
plt.show()

# ROC AUC error bar
plt.errorbar(q_range, mean_auc_score, yerr=std_auc_score, linewidth=3)
plt.xlabel('q')
plt.ylabel('ROC␣AUC')
if (dataset1):
    plt.savefig("ind_q_auc_errorbar_i_a.png")
else:
    plt.savefig("ind_q_auc_errorbar_i_a_2.png")
plt.show()

# Take the best q value as the one with the highest worst case score score (f1 - std de
worst_cases = np.array(mean_f1_score) - np.array(std_f1_score)
q_best = q_range[np.argmax(worst_cases)]
X_poly = PolynomialFeatures(q_best).fit_transform(X)
X_train_poly = PolynomialFeatures(q_best).fit_transform(X_train)
X_test_poly = PolynomialFeatures(q_best).fit_transform(X_test)

# Print out results for each q value
print("\nq␣cross-validaion:")
for i in range(len(q_range)):
    print("q,␣mean␣f1,␣std␣dev␣f1,␣mean␣auc,␣std␣dev␣auc,␣worst␣case␣=␣%i␣&␣%.4f␣&␣%.4f␣
print("Best␣q␣value␣is:␣", q_best)

# ------------------------------ #
# C cross validation
# ------------------------------ #

# Empty mean and std dev arrays to reuse for C value sweep
mean_f1_score = []
std_f1_score = []
mean_auc_score = []
std_auc_score = []

# Range of C values to sweep through
C_range = [0.1, 1, 2, 4, 6, 8, 10]

# Sweep through C values
for C_ in C_range:
    # Create new logistic regression model with L2 penalty and new C value
    log_model = LogisticRegression(penalty='l2', C=C_, max_iter=2500)
    # Give cross_val_score entire dataset
    f1_cv_scores = cross_val_score(log_model, X_poly, y, cv=5, scoring='f1')
    auc_cv_scores = cross_val_score(log_model, X_poly, y, cv=5, scoring='roc_auc')
    mean_f1_score.append(np.array(f1_cv_scores).mean())
    std_f1_score.append(np.array(f1_cv_scores).std())
    mean_auc_score.append(np.array(auc_cv_scores).mean())
    std_auc_score.append(np.array(auc_cv_scores).std())

# f1 Score errorbar
plt.errorbar(C_range, mean_f1_score, yerr=std_f1_score, linewidth=3)
plt.xlabel('C')
plt.ylabel('f1␣Score')
if (dataset1):
    plt.savefig("ind_C_f1_errorbar_i_a.png")
else:
    plt.savefig("ind_C_f1_errorbar_i_a_2.png")
plt.show()

# ROC AUC errorbar
plt.errorbar(C_range, mean_auc_score, yerr=std_auc_score, linewidth=3)
plt.xlabel('C')
plt.ylabel('ROC␣AUC')
if (dataset1):
    plt.savefig("ind_C_auc_errorbar_i_a.png")
else:
    plt.savefig("ind_C_auc_errorbar_i_a_2.png")
plt.show()

# Take the best C value as the one with the highest worst case score score (f1 - std de
worst_cases = np.array(mean_auc_score) - np.array(std_auc_score)
C_best = C_range[np.argmax(worst_cases)]
best_log_model = LogisticRegression(penalty='l2', C=C_best, max_iter=2500).fit(X_train_

# Print out results for each C value
print("\nC␣cross-validaion:")
for i in range(len(C_range)):
    print("C,␣mean␣f1,␣std␣dev␣f1,␣mean␣auc,␣std␣dev␣auc,␣worst␣case␣=␣%.1f␣&␣%.4f␣&␣%.4
print("Best␣C␣value␣is:␣", C_best)

# ———————————————————————— #
# Combining q and C cross validations into one nested loop search
# ———————————————————————— #

best_C_vals = []
best_f1_means = []
best_f1_std_devs = []
best_auc_means = []
best_auc_std_devs = []
best_worst_cases = []

# Sweep through q values
for q in q_range:
    # Construct new polynomial features dataset based on q value
    X_poly = PolynomialFeatures(q).fit_transform(X)

    # Mean and std dev arrays
    mean_f1_score = []
    std_f1_score = []
    mean_auc_score = []
    std_auc_score = []

    # Sweep through C values
    for C_ in C_range:
        # Create new logistic regression model with L2 penalty and new C value
        log_model = LogisticRegression(penalty='l2', C=C_, max_iter=2500)
        # Give cross_val_score entire dataset (poly)
        f1_cv_scores = cross_val_score(log_model, X_poly, y, cv=5, scoring='f1')
        auc_cv_scores = cross_val_score(log_model, X_poly, y, cv=5, scoring='roc_auc')
        mean_f1_score.append(np.array(f1_cv_scores).mean())
        std_f1_score.append(np.array(f1_cv_scores).std())
        mean_auc_score.append(np.array(auc_cv_scores).mean())
        std_auc_score.append(np.array(auc_cv_scores).std())

    ## f1 Score errorbar
    #plt.errorbar(C_range, mean_f1_score, yerr=std_f1_score, linewidth=3)
    #plt.title("q = {}".format(q))
    #plt.xlabel('C')
```

```
        #plt.ylabel('f1 Score')
        #if (dataset1):
        #    plt.savefig("q={i}_C_f1_errorbar_i_a.png".format(i=q))
        #else:
        #    plt.savefig("q={i}_C_f1_errorbar_i_a_2.png".format(i=q))
        #plt.show()
        #
        ## ROC AUC errorbar
        #plt.errorbar(C_range, mean_auc_score, yerr=std_auc_score, linewidth=3)
        #plt.title("q = {}".format(q))
        #plt.xlabel('C')
        #plt.ylabel('ROC AUC')
        #if (dataset1):
        #    plt.savefig("q={i}_C_auc_errorbar_i_a.png".format(i=q))
        #else:
        #    plt.savefig("q={i}_C_auc_errorbar_i_a_2.png".format(i=q))
        #plt.show()

        # Take the best C value as the one with the highest worst case score score (f1 - std
        worst_cases = np.array(mean_auc_score) - np.array(std_auc_score)
        best_C_vals.append(C_range[np.argmax(worst_cases)])
        best_f1_means.append(mean_f1_score[np.argmax(worst_cases)])
        best_f1_std_devs.append(std_f1_score[np.argmax(worst_cases)])
        best_auc_means.append(mean_auc_score[np.argmax(worst_cases)])
        best_auc_std_devs.append(std_auc_score[np.argmax(worst_cases)])
        best_worst_cases.append(np.max(worst_cases))

# Print combined results
print("\nq and C combined cross-validaion:")
q_best = q_range[np.argmax(best_worst_cases)]
C_best = best_C_vals[np.argmax(best_worst_cases)]
for i in range(len(q_range)):
    print("q, best C, best f1 mean, best f1 std dev, best auc mean, best auc std dev, wo
print("Best combination was: q, C = %i, %.1f"%(q_best, C_best))

# Set best polynomial features dataset (best q value) and train best model (best C value
X_poly = PolynomialFeatures(q_best).fit_transform(X)
X_train_poly = PolynomialFeatures(q_best).fit_transform(X_train)
X_test_poly = PolynomialFeatures(q_best).fit_transform(X_test)
best_log_model = LogisticRegression(penalty='l2', C=C_best, max_iter=2500).fit(X_train_p

# ---------------------------------------------- #
# Plotting best model based on optimal q and C values
# ---------------------------------------------- #

plt.figure(figsize=(8, 6))

# Scatter of test data
plt.scatter(X1_test[y_test == 1], X2_test[y_test == 1], marker='+', color=y_plus_color,
plt.scatter(X1_test[y_test == -1], X2_test[y_test == -1], marker='o', color=y_minus_colo

# Scatter of best logistic regression model predictions on training data
y_pred = best_log_model.predict(X_test_poly)
plt.scatter(X1_test[y_pred == 1], X2_test[y_pred == 1], marker='+', color=y_pred_plus_co
plt.scatter(X1_test[y_pred == -1], X2_test[y_pred == -1], marker='o', color=y_pred_minus

# Plot decision boundary
num_pts = 500
X1_grid = np.linspace(np.min(X1), np.max(X1), num_pts).reshape(-1, 1)
X2_grid = np.linspace(np.min(X2), np.max(X2), num_pts).reshape(-1, 1)
```

```
# Make grid of input features
X_grid = []
for i in X1_grid:
    for j in X2_grid:
        X_grid.append([i, j])
X_grid = np.array(X_grid).reshape(-1, 2)
X1_grid, X2_grid = X_grid[:, 0], X_grid[:, 1]

# Create polynomial features for X_grid
X_grid_poly = PolynomialFeatures(q_best).fit_transform(X_grid)
y_grid = best_log_model.predict(X_grid_poly)
# Decision boundary line
plt.contour(X1_grid.reshape(num_pts, num_pts), X2_grid.reshape(num_pts, num_pts), y_grid
# Areas
plt.contourf(X1_grid.reshape(num_pts, num_pts), X2_grid.reshape(num_pts, num_pts), y_gri

# Label axes and add legend
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend(loc=4) # BR corner

# Save and show the plot
if (dataset1):
    plt.savefig("best_log_reg_model_i_a.png")
else:
    plt.savefig("best_log_reg_model_i_a_2.png")
plt.show()

# Print classification report
print(classification_report(y_test, y_pred))

# ---------------------------------------------- #
# Question (i)(b)
# ---------------------------------------------- #
# Uniform weights
# ------------------------ #

# Range of k values to sweep through
k_range = [1, 3, 5, 7, 9, 15, 25, 35, 45, 55]

# Mean and std dev arrays
mean_f1_score = []
std_f1_score = []
mean_auc_score = []
std_auc_score = []

# Sweep through k values
for k in k_range:
    # Uniform weights case - train kNN model for each k on original dataset
    knn_model = KNeighborsClassifier(n_neighbors=k, weights='uniform')
    # Give cross_val_score entire dataset
    f1_cv_scores = cross_val_score(knn_model, X, y, cv=5, scoring='f1')
    auc_cv_scores = cross_val_score(knn_model, X, y, cv=5, scoring='roc_auc')
    mean_f1_score.append(np.array(f1_cv_scores).mean())
    std_f1_score.append(np.array(f1_cv_scores).std())
    mean_auc_score.append(np.array(auc_cv_scores).mean())
    std_auc_score.append(np.array(auc_cv_scores).std())

# f1 Score error bar
plt.errorbar(k_range, mean_f1_score, yerr=std_f1_score, linewidth=3)
plt.xlabel('k')
plt.ylabel('f1␣Score')
plt.title('kNN␣Model␣with␣Uniform␣Weights')
if (dataset1):
    plt.savefig("k_f1_errorbar_uniform_i_b.png")
else:
    plt.savefig("k_f1_errorbar_uniform_i_b_2.png")
plt.show()

# ROC AUC errorbar
plt.errorbar(k_range, mean_auc_score, yerr=std_auc_score, linewidth=3)
plt.xlabel('k')
plt.ylabel('ROC␣AUC')
plt.title('kNN␣Model␣with␣Uniform␣Weights')
if (dataset1):
    plt.savefig("k_auc_errorbar_uniform_i_b.png")
else:
    plt.savefig("k_auc_errorbar_uniform_i_b_2.png")
plt.show()

# Take the best k value as the one with the highest worst case score score (f1 - std dev
worst_cases = np.array(mean_auc_score) - np.array(std_auc_score)
best_uni_worst_case = np.max(worst_cases) # Keep best result to compare to distance kNN
k_best = k_range[np.argmax(worst_cases)]
best_knn_model = KNeighborsClassifier(n_neighbors=k_best, weights='uniform').fit(X_train

# Print out results for k cross-validation
print("\nk␣cross-validaion␣(uniform␣weights):")
for i in range(len(k_range)):
    print("k,␣mean␣f1,␣std␣dev␣f1,␣mean␣auc,␣std␣dev␣auc,␣worst␣case␣=␣%i␣&␣%.4f␣&␣%.4f␣
print("Best␣k␣value␣is:␣", k_best)

# ------------------------------------ #
# Plotting best model based on optimal k value
# ------------------------------------ #

plt.figure(figsize=(8, 6))

# Scatter of test data
plt.scatter(X1_test[y_test == 1], X2_test[y_test == 1], marker='+', color=y_plus_color,
plt.scatter(X1_test[y_test == -1], X2_test[y_test == -1], marker='o', color=y_minus_colo

# Scatter of best kNN model predictions
y_pred = best_knn_model.predict(X_test)
plt.scatter(X1_test[y_pred == 1], X2_test[y_pred == 1], marker='+', color=y_pred_plus_co
plt.scatter(X1_test[y_pred == -1], X2_test[y_pred == -1], marker='o', color=y_pred_minus

# Plot decision boundary
y_grid = best_knn_model.predict(X_grid)
# Decision boundary line
plt.contour(X1_grid.reshape(num_pts, num_pts), X2_grid.reshape(num_pts, num_pts), y_grid
# Areas
plt.contourf(X1_grid.reshape(num_pts, num_pts), X2_grid.reshape(num_pts, num_pts), y_gri

# Label axes and add legend
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend(loc=4) # BR corner

# Save and show the plot
if (dataset1):
    plt.savefig("best_knn_model_uniform_i_b.png")
```

```
else:
    plt.savefig("best_knn_model_uniform_i_b_2.png")
plt.show()

# Print classification report
print(classification_report(y_test, y_pred))

# ------------------------------------ #
# Distance weights
# ------------------------------------ #

# Mean and std dev arrays
mean_f1_score = []
std_f1_score = []
mean_auc_score = []
std_auc_score = []

# Sweep through k values
for k in k_range:
    # Distance weights case - train kNN model for each k on original dataset
    knn_model = KNeighborsClassifier(n_neighbors=k, weights='distance')
    # Give cross_val_score entire dataset
    f1_cv_scores = cross_val_score(knn_model, X, y, cv=5, scoring='f1')
    auc_cv_scores = cross_val_score(knn_model, X, y, cv=5, scoring='roc_auc')
    mean_f1_score.append(np.array(f1_cv_scores).mean())
    std_f1_score.append(np.array(f1_cv_scores).std())
    mean_auc_score.append(np.array(auc_cv_scores).mean())
    std_auc_score.append(np.array(auc_cv_scores).std())

# f1 Score error bar
plt.errorbar(k_range, mean_f1_score, yerr=std_f1_score, linewidth=3)
plt.xlabel('k')
plt.ylabel('f1␣Score')
plt.title('kNN␣Model␣with␣Distance␣Weights')
if (dataset1):
    plt.savefig("k_f1_errorbar_distance_i_b.png")
else:
    plt.savefig("k_f1_errorbar_distance_i_b_2.png")
plt.show()

# ROC AUC errorbar
plt.errorbar(k_range, mean_auc_score, yerr=std_auc_score, linewidth=3)
plt.xlabel('k')
plt.ylabel('ROC␣AUC')
plt.title('kNN␣Model␣with␣Distance␣Weights')
if (dataset1):
    plt.savefig("k_auc_errorbar_distance_i_b.png")
else:
    plt.savefig("k_auc_errorbar_distance_i_b_2.png")
plt.show()

# Take the best k value as the one with the highest worst case score score (f1 - std de
worst_cases = np.array(mean_auc_score) - np.array(std_auc_score)
best_dist_worst_case = np.max(worst_cases) # Keep best result to compare to distance kN
k_dist_best = k_range[np.argmax(worst_cases)]
best_knn_dist_model = KNeighborsClassifier(n_neighbors=k_dist_best, weights='distance').

# Print out results for k cross-validation
print("\nk␣cross-validaion␣(distance␣weights):")
for i in range(len(k_range)):
    print("k,␣mean␣f1,␣std␣dev␣f1,␣mean␣auc,␣std␣dev␣auc,␣worst␣case␣=␣%i␣&␣%.4f␣&␣%.4f␣
print("Best␣k␣value␣is:␣", k_best)

# ------------------------------- #
# Plotting best model based on optimal k value
# ------------------------------- #

plt.figure(figsize=(8, 6))

# Scatter of test data
plt.scatter(X1_test[y_test == 1], X2_test[y_test == 1], marker='+', color=y_plus_color,
plt.scatter(X1_test[y_test == -1], X2_test[y_test == -1], marker='o', color=y_minus_colo

# Scatter of best kNN model predictions
y_pred = best_knn_dist_model.predict(X_test)
plt.scatter(X1_test[y_pred == 1], X2_test[y_pred == 1], marker='+', color=y_pred_plus_co
plt.scatter(X1_test[y_pred == -1], X2_test[y_pred == -1], marker='o', color=y_pred_minu

# Plot decision boundary
y_grid = best_knn_dist_model.predict(X_grid)
# Decision boundary line
plt.contour(X1_grid.reshape(num_pts, num_pts), X2_grid.reshape(num_pts, num_pts), y_grid
# Areas
plt.contourf(X1_grid.reshape(num_pts, num_pts), X2_grid.reshape(num_pts, num_pts), y_gri

# Label axes and add legend
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend(loc=4) # BR corner

# Save and show the plot
if (dataset1):
    plt.savefig("best_knn_model_distance_i_b.png")
else:
    plt.savefig("best_knn_model_distance_i_b_2.png")
plt.show()

# Print classification report
print(classification_report(y_test, y_pred))

# ------------------------------------------------ #
# Question (i)(c)
# ------------------------------------------------ #

# Confusion matrix and classification report for logistic regression model
print("\nConfusion␣matrix␣and␣classification␣report␣for␣logistic␣regression␣model")
y_pred = best_log_model.predict(X_test_poly)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Confusion matrix and classification report for kNN model
print("\nConfusion␣matrix␣and␣classification␣report␣for␣kNN␣model")
# Take whichever kNN performed best
if (best_uni_worst_case > best_dist_worst_case): # Uniform weights better
    y_pred = best_knn_model.predict(X_test)
else: # Distance weights better
    y_pred = best_knn_dist_model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Confusion matrix and classification report for a most frequent dummy classifier
print("\nConfusion␣matrix␣and␣classification␣report␣for␣most␣frequent␣dummy␣classifier")
```

```
dummy_most_model = DummyClassifier(strategy="most_frequent").fit(X_train, y_train)
y_pred = dummy_most_model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Confusion matrix and classification report for a random dummy classifier
print("\nConfusion matrix and classification report for random dummy classifier")
dummy_rand_model = DummyClassifier(strategy="uniform").fit(X_train, y_train)
y_pred = dummy_rand_model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

# ----------------------------------------------------------- #
# Question (i)(d)
# ----------------------------------------------------------- #

# ROC for best logistic regression model
fpr, tpr, _ = roc_curve(y_test, best_log_model.decision_function(X_test_poly))
plt.plot(fpr, tpr, label="Logistic Regression")
#plt.plot([0, 1], [0, 1], color='green', linestyle='--')
#plt.xlabel('False positive rate')
#plt.ylabel('True positive rate')
#plt.title("ROC for Logistic Regression Model")
#if (dataset1):
#    plt.savefig("roc_log_i_d.png")
#else:
#    plt.savefig("roc_log_i_d_2.png")
#plt.show()

# ROC for best kNN model - take whichever kNN performed best
if (best_uni_worst_case > best_dist_worst_case): # Uniform weights better
    fpr, tpr, _ = roc_curve(y_test, best_knn_model.predict_proba(X_test)[:, 1])
else: # Distance weights better
    fpr, tpr, _ = roc_curve(y_test, best_knn_dist_model.predict_proba(X_test)[:, 1])
plt.plot(fpr, tpr, label="kNN")
#plt.xlabel('False positive rate')
#plt.ylabel('True positive rate')
#plt.plot([0, 1], [0, 1], color='green', linestyle='--')
#plt.title("ROC for kNN Model")
#if (dataset1):
#    plt.savefig("roc_knn_i_d.png")
#else:
#    plt.savefig("roc_knn_i_d_2.png")
#plt.show()

# ROC for most frequent dummy classifier
fpr, tpr, _ = roc_curve(y_test, dummy_most_model.predict_proba(X_test)[:, 1])
plt.plot(fpr, tpr, label="Most Frequent Classifier")
#plt.xlabel('False positive rate')
#plt.ylabel('True positive rate')
#plt.plot([0, 1], [0, 1], color='green', linestyle='--')
#plt.title("ROC for Most Frequent Dummy Classifier")
#if (dataset1):
#    plt.savefig("roc_dummy_most_i_d.png")
#else:
#    plt.savefig("roc_dummy_most_i_d_2.png")
#plt.show()

# ROC for most frequent dummy classifier
fpr, tpr, _ = roc_curve(y_test, dummy_rand_model.predict_proba(X_test)[:, 1])
plt.plot(fpr, tpr, label="Random Classifier")
plt.plot([0, 1], [0, 1], color='green', linestyle='--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
#plt.title("ROC for Random Dummy Classifier")
#if (dataset1):
#    plt.savefig("roc_dummy_rand_i_d.png")
#else:
#    plt.savefig("roc_dummy_rand_i_d_2.png")
plt.legend(loc=4)
if (dataset1):
    plt.savefig("rocs_first_dataset.png")
else:
    plt.savefig("rocs_second_dataset.png")
plt.show()

# ------------------------------------------------ #
# Question (i)(e)
# ------------------------------------------------ #

# Written answer

# ------------------------------------------------ #
# END OF ASSIGNMENT
# ------------------------------------------------ #
```

—— END OF ASSIGNMENT ——