Assignment 4

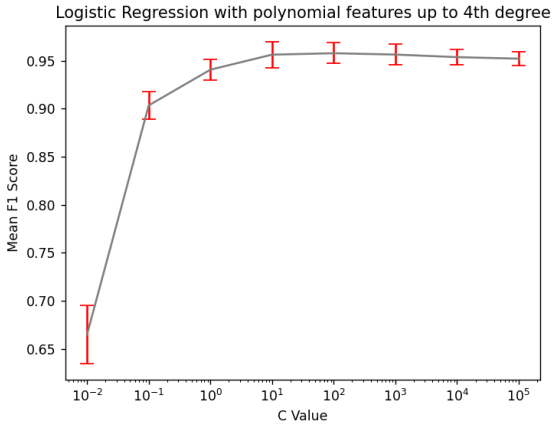
Dataset 1: # id6--6-6-0 Dataset 2: # id6-12-6-0

Question i(a) - the first dataset

C Value	Poly=1	Poly=2	Poly=3	Poly=4	Poly=5	Poly=6
10^{-3}	0	0	0	0	0	0
10^{-2}	0.538987	0.606452	0.661494	0.665008	0.666211	0.668685
10^{-1}	0.803371	0.89204	0.87618	0.903565	0.902025	0.904562
1	0.810664	0.938022	0.936625	0.940739	0.940739	0.93942
10	0.809586	0.957871	0.956453	0.956419	0.956474	0.957778
10^{2}	0.809586	0.956548	0.956697	0.957872	0.957872	0.956435
10^{3}	0.809586	0.955278	0.956601	0.956522	0.955163	0.955163
10^{4}	0.809586	0.955278	0.956601	0.953823	0.95514	0.955087
Max Score	0.810664	0.957871	0.956697	0.957872	0.957872	0.957778

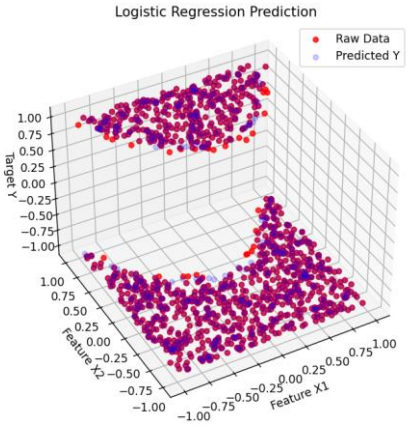
As the penalty term $\alpha = {}^{1}/_{C}$, a smaller C results in a stronger penalty impact on coefficients.

When C=0.001, all coefficients turn to zero, making it unnecessary to test smaller C values. Polynomial features are powered up to 6 and select those features that equal to 4 via cross-validation as it yields the highest F1 score. Although the F1 score is also 0.957872 when features are powered up to 5, it is not considered due to the potential risk of overfitting.

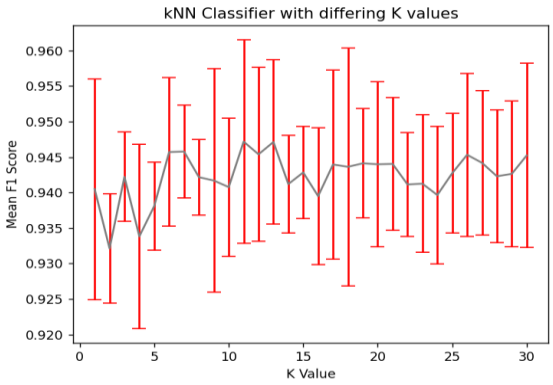


Utilize the 4th polynomial degree to compute cross-validation scores for a range of C values. The below plot illustrates that when C=100, the logistic regression model achieves a notably high mean F1 score, as proved by the above table indicating the highest F1 score among a series of C is 0.957872.

The underneath 3D plot displays training data and prediction made by a logistic regression model using the 4th order of polynomial features when C=100. The red dots represent actual target values, while the blue dots correspond to predicted target values. And the visualization shows that a large portion of predicted values overlapped with the real target values. However, there are a few blue points not correctly predicted.



Question i(b) - the first dataset



To select a proper K value, a range of K values from 1 to 30 were tested using cross-validation. The result showed that when K is set to 11, it yields the highest mean F1 score among the K range tested, with a relatively good standard deviation of F1 score.

Question i(c) - the first dataset

Logistic Regression model(C=100, 4th polynomial degree, 80% training data, 20% testing data)
The confusion matrix:

[[142 2]
[4 84]]

The classification report:

	precision	recall	f1-score	support
-1	0.97	0.99	0.98	144
1	0.98	0.95	0.97	88
accuracy			0.97	232
macro avg	0.97	0.97	0.97	232
weighted avg	0.97	0.97	0.97	232

From the above report, the overall accuracy of the trained logistic regression model is 0.97, which is the same as the macro average and weighted average. This indicates that the trained model

performs very well.

kNN classifier(K=11, 80% training data, 20% testing data)

The confusion matrix:

[[162 2]
[2 66]]

The classification report:

	precision	recall	f1-score	support
-1	0.99	0.99	0.99	164
1	0.97	0.97	0.97	68
accuracy			0.98	232
macro avg	0.98	0.98	0.98	232
weighted avg	0.98	0.98	0.98	232

Notably, for the labelled class -1, its precision, recall and f1-score have achieved 99% and for the other labelled class 1, all of the prediction rates are also good with a value of 97%. Its overall accuracy is 0.98, which is the same as the macro average and weighted average. Thus, the kNN model performs really excellent.

Most frequent classifier

The confusion matrix:

[[793 0]
[367 0]]

The classification report:

	precision	recall	f1-score	support
-1	0.68	1.00	0.81	793
1	0.00	0.00	0.00	367
accuracy			0.68	1160
macro avg	0.34	0.50	0.41	1160
weighted avg	0.47	0.68	0.56	1160

The most frequent classifier only predicts the labelled class -1 and consequently its prediction accuracy is quite low with a value of 68%. The macro average and the weighted average are worse with values of 0.41 and 0.56 respectively.

Random confusion matrix

The confusion matrix:

[[366 427]
[198 169]]

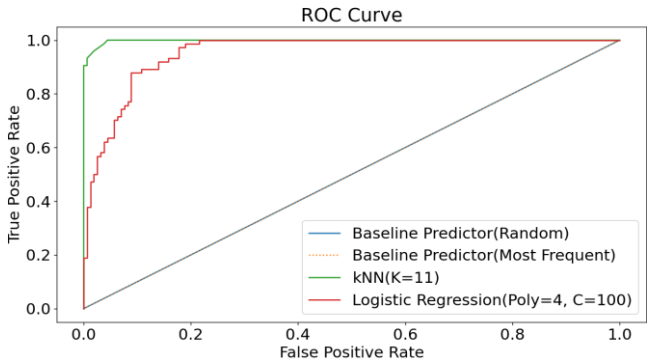
The classification report:

	precision	recall	f1-score	support
-1	0.65	0.46	0.54	793
1	0.28	0.46	0.35	367
accuracy			0.46	1160
macro avg	0.47	0.46	0.45	1160
weighted avg	0.53	0.46	0.48	1160

With randomly generating class labels, it poorly predicts correct target values, whose accuracy of prediction is only 46% and similar to the macro average (45%) and weighted average (48%)

Question i(d) - the first dataset

In the ROC plot below, the green line represents the kNN model(K=11), the red line is the logistic regression model(Poly features=4, C=100), the blue solid line indicates the baseline predictor (Random) and the orange dotted line represents the baseline predictor (most frequent).



Question i(e) - the first dataset

In the ROC curve of the different classifiers, both the kNN and logistic regression model outperform the baseline predictors that are the most frequent classifier and random classifier. This is evident from the fact that both the kNN and the logistic regression model are closer to the upper-left corner, indicating superior model performance. Whereas, the diagonal line that indicates baseline predictors poorly performs and essentially makes random class assignments. This can be further supported by the table of (c), which reveals that the kNN model and regression both have higher accuracy scores (98% and 97% respectively) than the baseline predictors' (46% for the random classifier and 68% for the most frequent classifier).

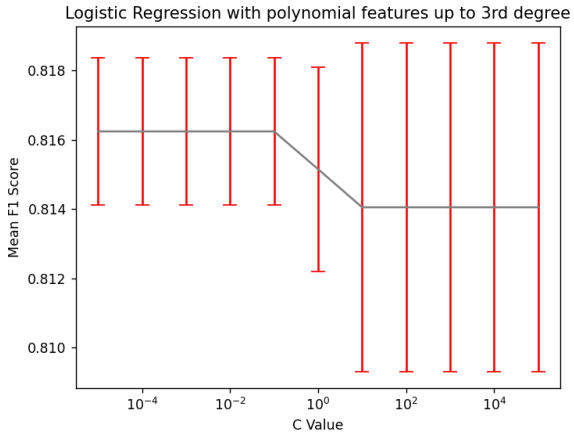
In this case, a recommended classifier is the kNN model as it has a slightly higher accuracy score than the logistic regression model, and the ROC curve of the former is much closer to the upper-left corner, showing its better prediction. Furthermore, the logistic regression performs poorly at correctly identifying the class at the beginning of the curve, despite having a high accuracy score. This phenomenon could be attributed to the imbalanced classes, where the model tends to favour the majority class to achieve high accuracy.

Question i(a) – the second dataset

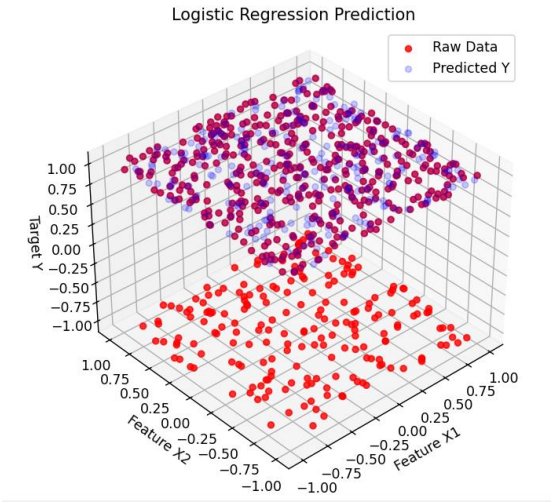
C Value	Poly=1	Poly=2	Poly=3	Poly=4	Poly=5	Poly=6
10^{-5}	0.81625	0.81625	0.81625	0.81625	0.81625	0.81625
10^{-4}	0.81625	0.81625	0.81625	0.81625	0.81625	0.81625
10^{-3}	0.81625	0.81625	0.81625	0.81625	0.81625	0.81625
10^{-2}	0.81625	0.81625	0.81625	0.81625	0.81625	0.81625
10^{-1}	0.81625	0.81625	0.81625	0.81625	0.81625	0.81625
1	0.81625	0.81625	0.81516	0.81625	0.81516	0.81405
10	0.81625	0.81625	0.81406	0.81516	0.81295	0.81295
10^{2}	0.81625	0.81625	0.81406	0.81516	0.81074	0.8052
10^{3}	0.81625	0.81625	0.81406	0.81516	0.8104	0.79803
10^{4}	0.81625	0.81625	0.81406	0.81516	0.8104	0.79954
10^{5}	0.81625	0.81625	0.81406	0.81516	0.8093	0.80067
Max Score	0.81625	0.81625	0.81625	0.81625	0.81625	0.81625

Using the same polynomial range but extending C range values for the second dataset. The coefficients cannot be completely reduced to zero, even with a series of low C values being set to the logistic regression model. The overall F1 scores are not outstanding, with only around 81.6%. In this case, raising polynomial features to the power of 3 provides relatively acceptable F1 scores across all C values. This choice allows the model to balance the trade-off between complexity and simplicity, which could avoid over-complication and over-simplification.

Utilize the 3rd polynomial degree to compute cross-validation scores for a range of C values. The plot below shows that when C=10, the logistic regression model achieves a relatively high mean F1 score.

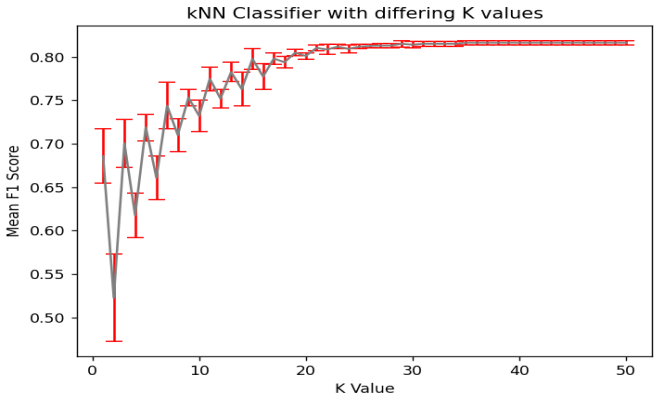


The 3D plot below shows training data and prediction made by a logistic regression model using the 3rd order of polynomial features when C=10. The red dots represent actual target values, while the blue dots correspond to predicted target values. However, the visualization displays that a large portion of predicted values are not matched with actual target values, showing as the blue dots on the top plane with the target value 1. Moreover, the predicted value should overlap as many as the red dots on the bottom plane with a target value of -1.



Question i(b) – the second dataset

Using the extended range of K values from 1 to 50 to test cross-validation for the second dataset. The result showed that when K is set to 29, it yields the highest mean F1 score among the K range tested, with a relatively good standard deviation of F1 score.



Question i(c) - the second dataset

Logistic Regression model(C=10, 3rd polynomial degree, 80% training data, 20% testing data)

The confusion matrix:

[[0 39]
[0 90]]

The classification report:

	precision	recall	f1-score	support
-1	0.00	0.00	0.00	39
1	0.70	1.00	0.82	90
accuracy			0.70	129
macro avg	0.35	0.50	0.41	129
weighted avg	0.49	0.70	0.57	129

From the above classification report, the overall accuracy of the trained logistic regression model is 0.76. This indicates that the trained model does not perform well with all labelled value -1 not correctly predicted.

kNN classifier(K=29, 80% training data, 20% testing data)

The confusion matrix:

[[1 34]
[1 93]]

The classification report:

	precision	recall	f1-score	support
-1	0.50	0.03	0.05	35
1	0.73	0.99	0.84	94
accuracy			0.73	129
macro avg	0.62	0.51	0.45	129
weighted avg	0.67	0.73	0.63	129

From the above classification report, the overall accuracy of the trained kNN classifier is 0.76. This indicates that the trained model does not perform well and all labelled values -1 are not correctly predicted.

Most frequent classifier

The confusion matrix:

[[0 199]
[0 442]]

The classification report:

	precision	recall	f1-score	support
-1	0.00	0.00	0.00	199
1	0.69	1.00	0.82	442
accuracy			0.69	641
macro avg	0.34	0.50	0.41	641
weighted avg	0.48	0.69	0.56	641

The most frequent classifier only predicts the labelled class 1 and consequently its prediction accuracy is quite low with a value of 69%. The macro average and the weighted average are worse with values of 0.41 and 0.56 respectively.

Random confusion matrix

The confusion matrix:

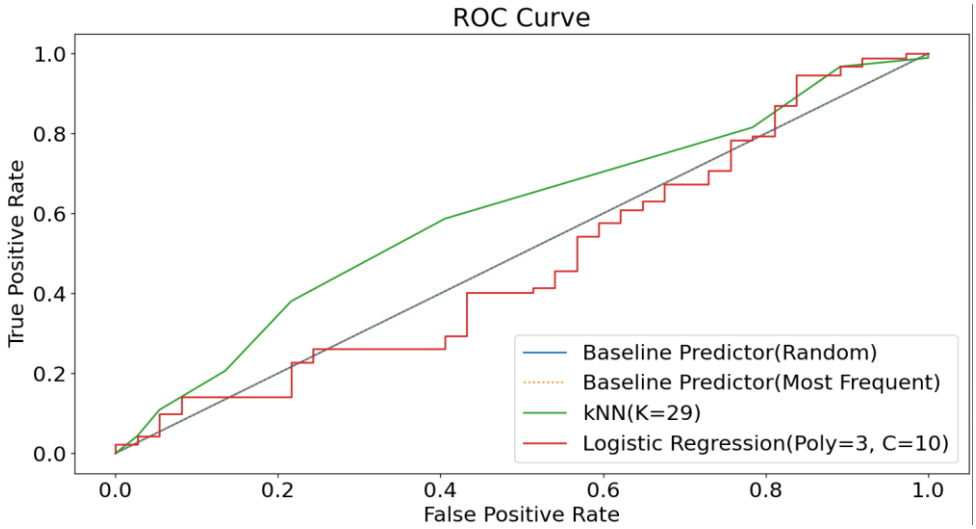
[[105 94]
[214 228]]

The classification report:

	precision	recall	f1-score	support
-1	0.33	0.53	0.41	199
1	0.71	0.52	0.60	442
accuracy			0.52	641
macro avg	0.52	0.52	0.50	641
weighted avg	0.59	0.52	0.54	641

With randomly generating class labels, it poorly predicts correct target values, whose accuracy of prediction is only 52% and similar to the macro average (50%) and weighted average (54%).

Question i(d) - the second dataset



Both the logistic regression model and the kNN model are poorly performed with the second dataset and even are under baseline predictors

Question i(e) - the second dataset

In the ROC curve of the different classifiers, both the kNN and logistic regression model underperform the baseline predictors that are the most frequent classifier and random classifiers. This is shown by the fact that both the kNN and the logistic regression model are close to the baseline predictors and mostly under the diagonal line. The diagonal line indicates that baseline predictors poorly perform and essentially make random class assignments. This can be further supported by the table of (c), which shows that all models have quite low accuracy with kNN(73%), Logistic Regression(70%), the most frequent baseline predictor(69%), and the random baseline predictor(52%)

In this case, a recommended classifier is the kNN model as it has the highest accuracy score, and its ROC curve is above the baseline predictors and the logistic regression, showing its relatively better prediction.

Appendix

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

warnings.filterwarnings("ignore")


def update_dataset(dataset):
    global df, x1, x2, X, Y, pf, poly_X, x_train, x_test, y_train,
y_test, x_train_poly, x_test_poly, y_train_poly, y_test_poly,
model_knn, model_log

    ds1 = '1# id6--6-6-0.csv'  # dataset 1
    ds2 = '2# id6-12-6-0.csv'  # dataset 2

    # the first dataset
    if dataset == 1:
        df = pd.read_csv(ds1)
        pf = PolynomialFeatures(degree=4, include_bias=False)  #
choose polynomial degree=4 for dataset 1
        model_knn = KNeighborsClassifier(n_neighbors=11,
weights='uniform')  # choose K=11 for dataset 1
        model_log = LogisticRegression(penalty='l2', C=100)  # choose
C=100 for dataset 1

    # the second dataset
    elif dataset == 2:
        df = pd.read_csv(ds2)
        pf = PolynomialFeatures(degree=3, include_bias=False)  #
choose polynomial degree=3 for dataset 2
        model_knn = KNeighborsClassifier(n_neighbors=29,
weights='uniform')  # choose K=21 for dataset 2
```

```
        model_log = LogisticRegression(penalty='l2', C=10)  # choose
C=10 for dataset 2

    # input data
    x1 = df.iloc[:, 0]
    x2 = df.iloc[:, 1]
    X = np.column_stack((x1, x2))
    Y = df.iloc[:, 2]
    poly_X = pf.fit_transform(X)
    x_train, x_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2)
    x_train_poly, x_test_poly, y_train_poly, y_test_poly =
train_test_split(poly_X, Y, test_size=0.2)


# select the maximum order of polynomial to use
def question_a_i(dataset=1):
    update_dataset(dataset)  # choose dataset

    if dataset == 1:
        ci_range = [1e-3, 1e-2, 0.1, 1, 10, 1e2, 1e3, 1e4, 'Max']
    elif dataset == 2:
        ci_range = [1e-5, 1e-4, 1e-3, 1e-2, 0.1, 1, 10, 1e2, 1e3, 1e4,
1e5, 'Max']

    ret = dict()  # store result
    for i in range(1, 7):  # add polynomial features power to 6
        tmp_pf = PolynomialFeatures(degree=i, include_bias=False)  #
temp polynomial features
        temp_poly_X = tmp_pf.fit_transform(X)

        tmp_scores = list()
        for ci in ci_range[:-1]:
            model = LogisticRegression(C=ci, penalty='l2',
solver='lbfgs')
            tmp_f1 = cross_val_score(model, temp_poly_X, Y, cv=5,
scoring='f1')
            tmp_scores.append(tmp_f1.mean())
        tmp_scores += [max(tmp_scores)]
        ret[i] = tmp_scores

    with pd.option_context('display.expand_frame_repr', False):
        df_ret = pd.DataFrame(ret, index=ci_range)
        print(df_ret)
```

```
        from Common import table_writer
        table_writer.write(df_ret, index=True, index_label=ci_range +
['Max'])


# select the weight C given to penalty in the cost function
def question_a_ii(dataset=1):
    update_dataset(dataset)  # choose dataset

    if dataset == 1:
        ci_range = [1e-2, 0.1, 1, 10, 100, 1e3, 1e4, 1e5]
    elif dataset == 2:
        ci_range = [1e-5, 1e-4, 1e-3, 1e-2, 0.1, 1, 10, 1e2, 1e3, 1e4,
1e5]

    f1_mean = list()
    f1_std = list()

    for ci in ci_range:
        model = LogisticRegression(penalty='l2', C=ci)
        tmp_scores = cross_val_score(model, poly_X, Y, cv=5,
scoring='f1')
        f1_mean.append(tmp_scores.mean())
        f1_std.append(tmp_scores.std())

    plt.xscale('log')
    plt.errorbar(ci_range, f1_mean, yerr=f1_std, color='gray',
ecolor='r', capsize=5)
    plt.xlabel('C Value')
    plt.ylabel('Mean F1 Score')
    if dataset == 1:
        plt.title('Logistic Regression with polynomial features up to
4th degree')
    elif dataset == 2:
        plt.title('Logistic Regression with polynomial features up to
3rd degree')
    plt.show()


# plot data
def question_a_iii(dataset=1):
    update_dataset(dataset)  # choose dataset

    # train logistic regression model using training dataset
```

```
    model_log.fit(x_train_poly, y_train_poly)

    # use trained model to predict Y
    Y_pred = model_log.predict(poly_X)

    # set plot attributes
    ax = plt.subplot(111, projection='3d')
    ax.scatter(x1, x2, Y, c='red', label='Raw Data', alpha=0.8)
    ax.scatter(x1, x2, Y_pred, c='blue', label='Predicted Y',
alpha=0.2)
    plt.xlabel('Feature X1')
    plt.ylabel('Feature X2')
    ax.set_zlabel('Target Y')
    plt.title("Logistic Regression Prediction")
    plt.legend()
    plt.tight_layout()
    plt.show()


# train KNN classifier and choose K=11 based on result
def question_b(dataset=1):
    update_dataset(dataset)  # choose dataset

    if dataset == 1:
        k_range = range(1, 31)
    elif dataset == 2:
        k_range = range(1, 51)

    f1_mean = list()
    f1_std = list()

    for k in k_range:
        model = KNeighborsClassifier(n_neighbors=k, weights='uniform')
        tmp_f1 = cross_val_score(model, X, Y, cv=5, scoring='f1')
        f1_mean.append(tmp_f1.mean())
        f1_std.append(tmp_f1.std())

    plt.errorbar(k_range, f1_mean, yerr=f1_std, color='gray',
ecolor='r', capsize=5)
    plt.xlabel('K Value')
    plt.ylabel('Mean F1 Score')
    plt.title('kNN Classifier with differing K values')
    plt.show()
```

```
# calculate confusion matrices
def question_c(dataset=1):
    update_dataset(dataset)  # choose dataset

    # apply dummy classifier to predict target values
    dummy_type = ['most_frequent', 'uniform']
    for d_type in dummy_type:
        d_c = DummyClassifier(strategy=d_type)
        d_c.fit(X, Y)
        Y_pred_dummy = d_c.predict(X)
        print(f'{d_type} confusion matrix\n{confusion_matrix(Y,
Y_pred_dummy)}')
        print(f'{d_type} classification
report\n{classification_report(Y, Y_pred_dummy)}')

    # apply knn model to predict target values
    model_knn.fit(x_train, y_train)
    Y_pred_knn = model_knn.predict(x_test)
    print(f'KNN confusion matrix\n{confusion_matrix(y_test,
Y_pred_knn)}')
    print(f'KNN classification report\n{classification_report(y_test,
Y_pred_knn)}')

    # apply logistic model to predict target values
    model_log.fit(x_train_poly, y_train_poly)
    Y_pred_log = model_log.predict(x_test_poly)
    print(f'Logistic Regression confusion
matrix\n{confusion_matrix(y_test_poly, Y_pred_log)}')
    print(f'Logistic Regression classification
report\n{classification_report(y_test_poly, Y_pred_log)}')


# plot ROC curves for Logistic Regression and kNN classifiers
def question_d(dataset=1):
    update_dataset(dataset)  # choose dataset

    plt.rc('font', size=18)
    plt.rcParams['figure.constrained_layout.use'] = True

    # create baseline predictor(random)
    dummy = DummyClassifier(strategy="uniform").fit(x_train, y_train)
    y_pred_dummy = dummy.predict_proba(x_test)[:, 1]
    fal_pos_rate, true_pos_rate, _ = roc_curve(y_test, y_pred_dummy)
```

```
    plt.plot(fal_pos_rate, true_pos_rate, label='Baseline
Predictor(Random)')

    # create baseline predictor(most frequent)
    dummy = DummyClassifier(strategy="most_frequent").fit(x_train,
y_train)
    y_pred_dummy = dummy.predict_proba(x_test)[:, 1]
    fal_pos_rate, true_pos_rate, _ = roc_curve(y_test, y_pred_dummy)
    plt.plot(fal_pos_rate, true_pos_rate, linestyle='dotted',
label='Baseline Predictor(Most Frequent)')

    # plot knn prediction
    model_knn.fit(x_train, y_train)
    y_pred_knn = model_knn.predict_proba(x_test)[:, 1]
    fal_pos_rate, true_pos_rate, _ = roc_curve(y_test, y_pred_knn)
    if dataset == 1:
        plt.plot(fal_pos_rate, true_pos_rate, label='kNN(K=11)')
    elif dataset == 2:
        plt.plot(fal_pos_rate, true_pos_rate, label='kNN(K=29)')

    # plot logistic regression prediction
    model_log.fit(x_train, y_train)
    y_pred_log = model_log.predict_proba(x_test)[:, -1]
    fal_pos_rate, true_pos_rate, _ = roc_curve(y_test, y_pred_log)
    if dataset == 1:
        plt.plot(fal_pos_rate, true_pos_rate, label='Logistic
Regression(Poly=4, C=100)')
    elif dataset == 2:
        plt.plot(fal_pos_rate, true_pos_rate, label='Logistic
Regression(Poly=3, C=10)')

    # set plot attribute
    plt.title('ROC Curve')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.show()


# use the second dataset for question i(a).1
def question_a_i_2():
    question_a_i(dataset=2)
```

```
# use the second dataset for question i(a).2
def question_a_ii_2():
    question_a_ii(dataset=2)


# use the second dataset for question i(a).3
def question_a_iii_2():
    question_a_iii(dataset=2)


# use the second dataset for question i(b)
def question_b_2():
    question_b(dataset=2)


# use the second dataset for question i(c)
def question_c_2():
    question_c(dataset=2)


# use the second dataset for question i(d)
def question_d_2():
    question_d(dataset=2)


def main():
    # call function with question number
    question_d()


if __name__ == '__main__':
    main()
```