Question (i) [Dataset #1]

Dataset #1 identified by # id:13-13–13-0. Dataset #2 identified by # id:13–26-13-0.

Part (a)

I plotted the data to get a feel for the data I was going to be working with.

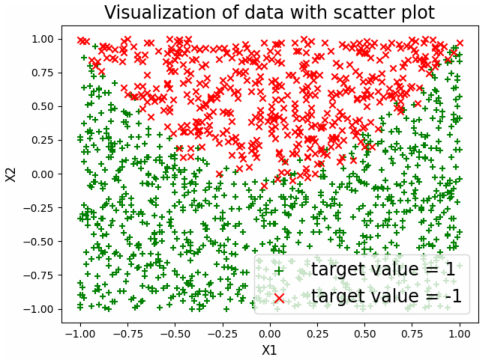


Figure 1: Plot of dataset #1 which has a quadratic shape.

Part (a) (i)

I chose q (the maximum order polynomial to use) using code 1.4 in the appendix. This consisted of plotting the F1 Score (which measures the accuracy of the model) against values of q to see what is the lowest value of q (to avoid overfitting) that will yield the highest F1 Score. From plotting the data above, we can see that the data is quadratic in shape, meaning q=2 makes the most sense. Therefore, we can try values for q such as 1,2 and 3 during 5-fold cross validation. Also, we can try to use a range of values for C such as 0.01, 1, and 1000 to see if there are any abnormalities which we can discuss in (a) (ii).

```
for q in range_of_values_for_q:
    Xpolynomial = PolynomialFeatures(q).fit_transform(X)
    scores = cross_val_score(model, Xpolynomial, y, cv=5,
    scoring="f1")
    std_error.append(np.array(scores).std())
    mean_error.append(np.array(scores).mean())
```

We can see from figure 2 below that our assumption of choosing $q = 2$ would be the best value for q is correct, as it is the lowest value of q that also yields the highest F1 Score. Using $q < 1$ leads to underfitting as seen by the low F1 Score below, and for $q > 2$ the F1 Score decreases slightly due to overfitting.

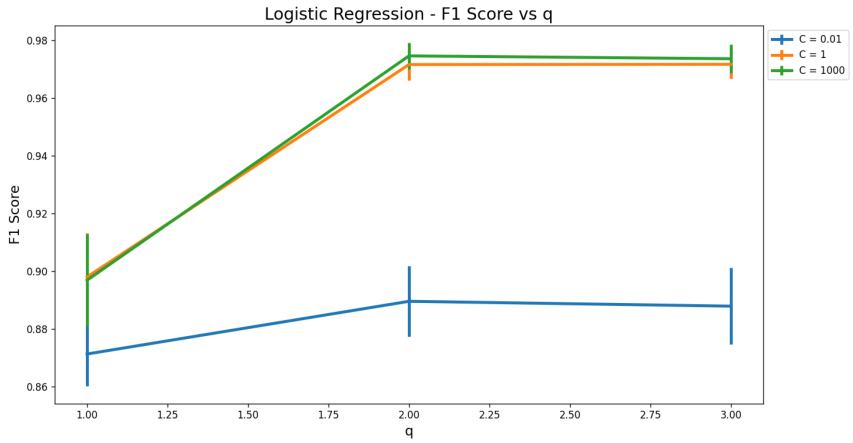
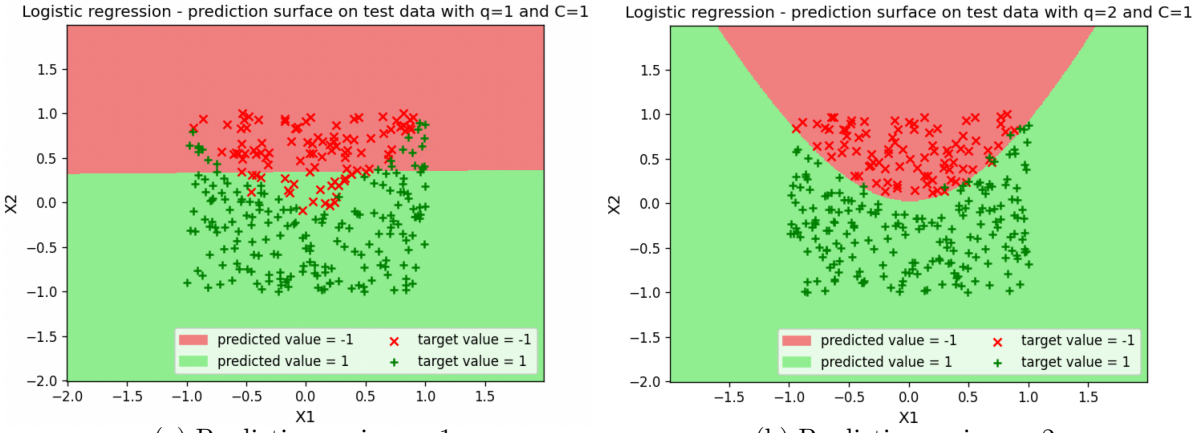


Figure 2: Plot of F1 Scores against q for the logistic regression classifier.

We can confirm our choice of $q = 2$ by plotting the model predictions with their prediction surface. We can do this by creating a meshgrid from the test data.

```
X, Y = np.meshgrid(np.arange(Xtest_X_minimum, Xtest_X_maximum, 0.01),
                   np.arange(Xtest_y_minimum, Xtest_y_maximum, 0.01))
Xtest = PolynomialFeatures(q).fit_transform(np.c_[X.ravel(),
Y.ravel()])
predictions = model.predict(Xtest).reshape(X.shape)
  colors_for_prediction_surface = ListedColormap(["lightcoral",
  "lightgreen"])
plt.pcolormesh(X, Y, predictions, cmap=colors_for_prediction_surface)
```



(a) Predictions using q=1 (b) Predictions using q=2

Figure 3: Plotting predictions of our trained model (predicted value) against the real test data (target value). We can confirm that using $q = 2$ is best.

Part (a) (ii)

To choose C I plotted the F1 Score against values of C to find the lowest value of C that yields the highest F1 Score using code from 1.5 in the appendix. We already know that C=1 works well for our model from the plots in (a)(i). Therefore we can try out a wide range of values for C such as 1, 5, 10, 50, 100, 150, 250, 450, 650, 850 and 1000 to find the best value for C using 5-fold cross validation.

```
for C in range_of_values_of_C:
        model = LogisticRegression(C=C, penalty="l2", max_iter=1000)
        scores = cross_val_score(model, Xpolynomial, y, cv=5, scoring="f1")
        std_error.append(np.array(scores).std())
        mean_error.append(np.array(scores).mean())
```

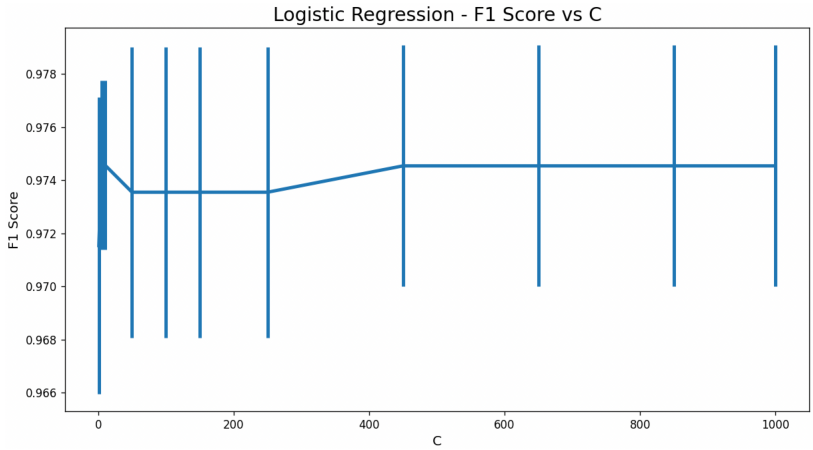
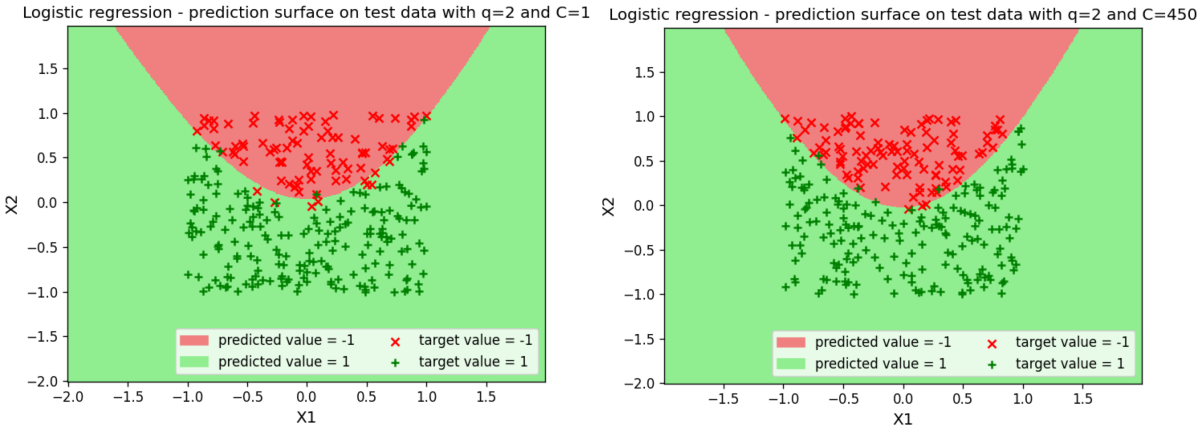


Figure 4: Plot of F1 Scores against C for the logistic regression classifier.

We can see from figure 4 that using $C = 450$ is the best value for C since that is when the F1 score is the highest. We can't use $C > 450$, since we want the smallest value for C to avoid overfitting (we want the model as simple as possible). We can see below that there is very little difference in the predictions with using $C = 1$ vs $C = 450$. This is because using q=2 and C=1 already fits the data very well from figure 3 (b), meaning that using $C = 450$ did not improve the accuracy by much, although it is the best value for C.



(a) Predictions using q=2 and C=1. (b) Predictions using q=2 and C=450.

Figure 5: Plotting predictions of our trained model (predicted value) against the real test data (target value).

Part (b)

I trained a kNN classifier on the data, and chose a value for k using 5-fold cross validation using code from 1.6 in the appendix. I did this by plotting the F1 Score against a range of values of k to see which value of k was best. I used a range of values for k such as 1,5,15,25,35,45 and 55 to see how the F1 score changed as k is increased.

```
for k in range_of_values_of_k:
    model = KNeighborsClassifier(n_neighbors=k, weights="uniform")
    scores = cross_val_score(model, Xpoly, y, cv=5, scoring="f1")
    std_error.append(np.array(scores).std())
    mean_error.append(np.array(scores).mean())
```

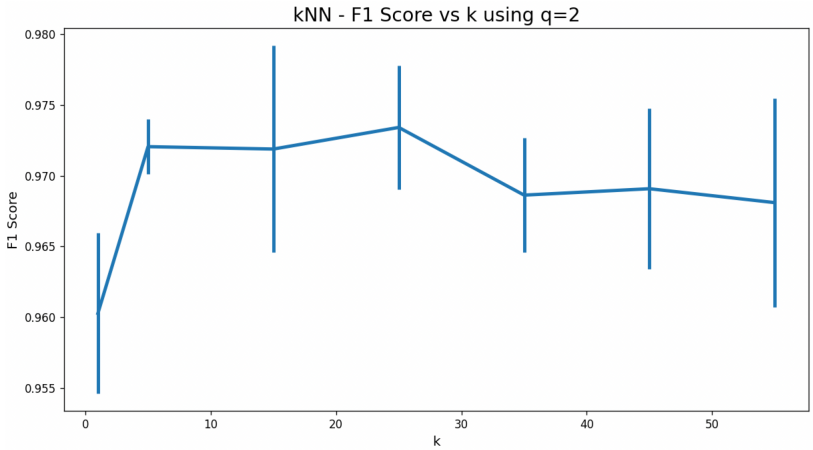


Figure 6: Plot of F1 Scores against k for the kNN classifier.

Referring to the plot above, we can see that the F1 Score increases until k=25, then tends to decrease. Therefore the best value for k is 25, since it yields the highest F1 Score. We can confirm our observations by plotting the prediction surface for the kNN classifier.

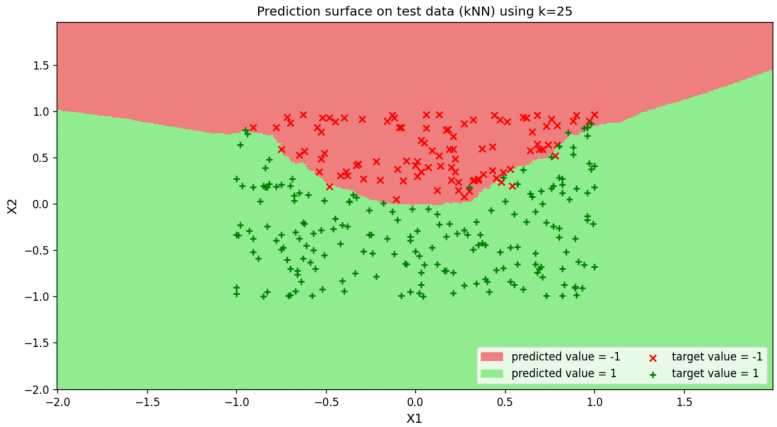


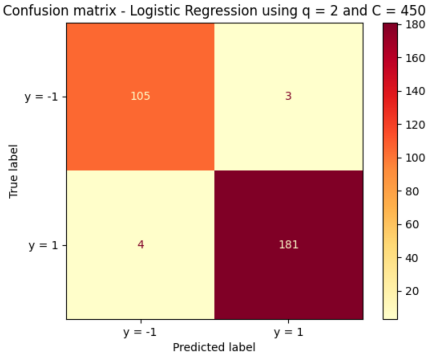
Figure 7: Plotting predictions of the kNN classifier using k=25.

It is evident from figure 7 that the kNN classifier is very accurate on the test data, however it doesn't follow the quadratic shape that the logistic regression classifier did. This can be seen from the outer top left and right edges of the training data, where the kNN classifier does not follow a quadratic shape at all for potentially new data.

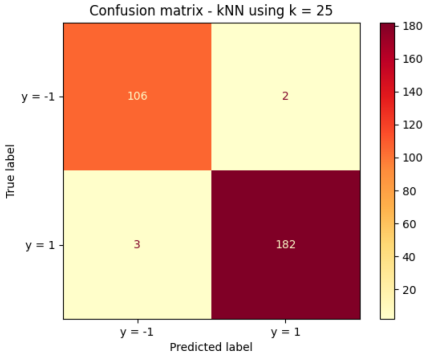
Part (c)

I calculated the confusion matrices for the trained logistic regression classifier, the kNN classifier, and two baseline classifiers (one which predicts the most frequent class, and one which predicts randomly) using code from 1.7 in the appendix. I split the training data and used the same test data (20%) collected for all 4 classifiers.

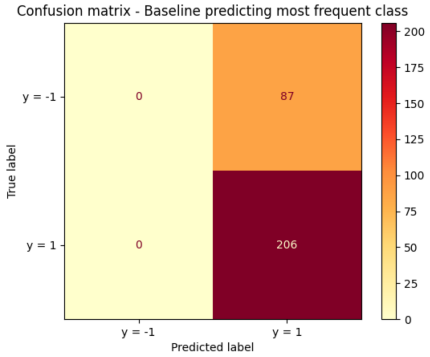
```
model.fit(Xtrain, ytrain)
ConfusionMatrixDisplay.from_estimator(model, Xtest, ytest,
display_labels=["y = -1", "y = 1"], cmap=plt.cm.YlOrRd)
```



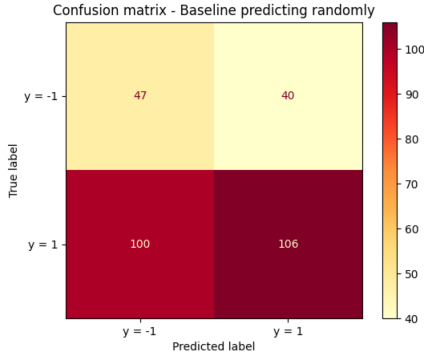
(a) Logistic regression classifier (q=2 & C=450).



(b) kNN classifier (k=25).



(c) Baseline classifier (most frequent).



(d) Baseline classifier (random).

Figure 8: Confusion matrices for the logistic regression classifier, the kNN classifier, and two baseline classifiers.

Part (d)

I plotted the ROC curves for the logistic regression and kNN classifiers, as well as the two baseline classifiers mentioned in (a)(c) using code from 1.8 in the appendix.

```
model = LogisticRegression(penalty="l2", C=C).fit(Xtrain, ytrain)
fpr, tpr, _ = roc_curve(ytest, model.decision_function(Xtest))
plt.plot(fpr, tpr, label="Logistic Regression (q = " + str(q) + " and
C = " + str(C) + ")", color="red")
model = KNeighborsClassifier(n_neighbors=k,
weights="uniform").fit(Xtrain, ytrain)
fpr, tpr, _ = roc_curve(ytest, model.predict_proba(Xtest)[:, 1])
plt.plot(fpr, tpr, label="kNN (k = " + str(k) + ")", color="green")
```

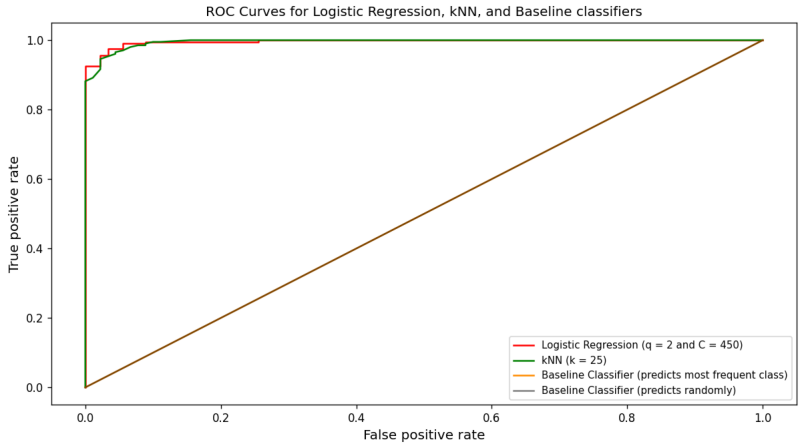


Figure 9: ROC curves for logistic regression, kNN, and 2 baseline classifiers.

Part (e)

I can evaluate and compare the performance of the logistic regression, kNN, and two baseline classifiers by calculating each of their accuracy, true positive rate, false positive rate and precision as discussed in lectures about evaluating a model.

1. **Logistic regression classifier (q=2, C=450)**
   - Accuracy $= \frac{TN+TP}{TN+TP+FN+FP} = \frac{105+181}{105+181+4+3} = 0.976$
   - True positive rate $= \frac{TP}{TP+FN} = \frac{181}{181+4} = 0.978$
   - False positive rate $= \frac{FP}{TN+FP} = \frac{3}{105+3} = 0.028$
   - Precision $= \frac{TP}{TP+FP} = \frac{181}{181+3} = 0.984$
2. **kNN classifier (k=25)**
   - Accuracy $= \frac{TN+TP}{TN+TP+FN+FP} = \frac{106+182}{106+182+3+2} = 0.983$
   - True positive rate $= \frac{TP}{TP+FN} = \frac{182}{182+3} = 0.984$
   - False positive rate $= \frac{FP}{TN+FP} = \frac{2}{106+2} = 0.019$
   - Precision $= \frac{TP}{TP+FP} = \frac{182}{182+2} = 0.989$
3. **Baseline classifier (predicting most frequent class)**
   - Accuracy $= \frac{TN+TP}{TN+TP+FN+FP} = \frac{0+106}{0+206+0+87} = 0.362$
   - True positive rate $= \frac{TP}{TP+FN} = \frac{206}{206+0} = 1$
   - False positive rate $= \frac{FP}{TN+FP} = \frac{87}{0+87} = 1$
   - Precision $= \frac{TP}{TP+FP} = \frac{206}{206+87} = 0.794$

4. **Baseline classifier (predicting randomly)**
   - Accuracy $= \frac{TN+TP}{TN+TP+FN+FP} = \frac{47+106}{47+106+100+40} = 0.522$
   - True positive rate $= \frac{TP}{TP+FN} = \frac{106}{106+100} = 0.515$
   - False positive rate $= \frac{FP}{TN+FP} = \frac{40}{47+40} = 0.460$
   - Precision $= \frac{TP}{TP+FP} = \frac{106}{106+40} = 0.726$

We can see from the above calculations that the logistic regression classifier and kNN classifier does better than both baseline classifiers in respect to having a true positive rate as close to 1 and a false positive rate as close to 0 as possible. This is because the [**true positive rate, false positive rate**] of [0.978, 0.028] and [0.984, 0.019] for the logistic regression and kNN classifier respectively is much better than [1,1] and [0.515, 0.460] for the baseline classifiers. In terms of [**accuracy, precision**], the logistic regression and kNN classifiers are far better than the baseline classifiers too since [0.976, 0.984] and [0.983, 0.989] are much higher than [0.362, 0.794] and [0.522, 0.726]. **Since we now have concluded that the logistic regression and kNN classifiers are better than the baseline classifiers**, we can compare the logistic regression and kNN classifiers to see which is better. We can see from the calculations above that the kNN classifier is slightly better in accuracy ($0.983 > 0.976$), true positive rate ($0.984 > 0.978$), false positive rate ($0.019 < 0.028$) and precision ($0.989 > 0.984$), although this difference is negligible. To further analyze their performance, we can look at the ROC curves for both classifiers. The ideal classifier gives 100% true positives and 0% false positives which is at the top-left corner of Figure 9. We can see that the lines for both classifiers overlap when the false positive rate = 0, however the red line representing the logistic regression classifier is slightly higher in the top left corner than the kNN classifier by a marginal amount. **Compared to a random classifier**, we can see that the baseline classifiers' ROC curves overlap meaning they are somewhat equally accurate, but much worse than the logistic regression and kNN classifiers. **The classifier I would recommend would be the logistic regression classifier.** This is because although from the calculations the kNN classifier is slightly better, it struggles with new data that isn't test data as seen from figure 7. The logistic regression classifier suits the quadratic shape of the data better as seen in figure 5.

Question (i) [Dataset #2, using same code as #1]

Part (a)

I plotted the data to get a feel for the data I was going to be working with.

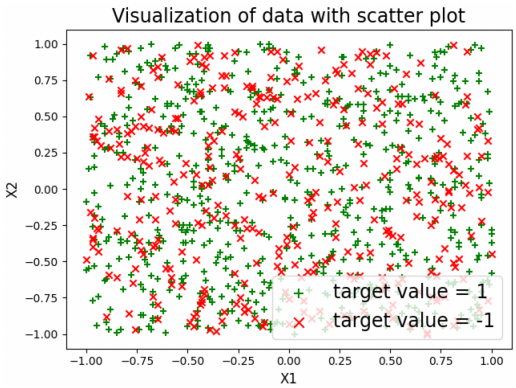


Figure 10: Plot of dataset #2 which is noisy.

Part (a) (i)

Once again, I chose q using code 1.4 in the appendix by plotting the F1 Score against values of q to see what is the lowest value of q (to avoid overfitting) that will yield the highest F1 Score. From plotting the data above, we can see that the data is very noisy meaning $q = 1$ makes the most sense. Therefore, we can try a wider range of values for q such as 1,2,3,4 and 5 during 5-fold cross validation. Also, we can try to use the same range of values for C such as 0.01, 1, and 1000 from dataset #1. We can see from figure 11 below that our assumption that $q = 1$ would be the best value for q is correct, as it is the lowest value of q that also yields the highest F1 Score. Using $q > 3$ leads to overfitting as seen by the F1 Score decreasing.

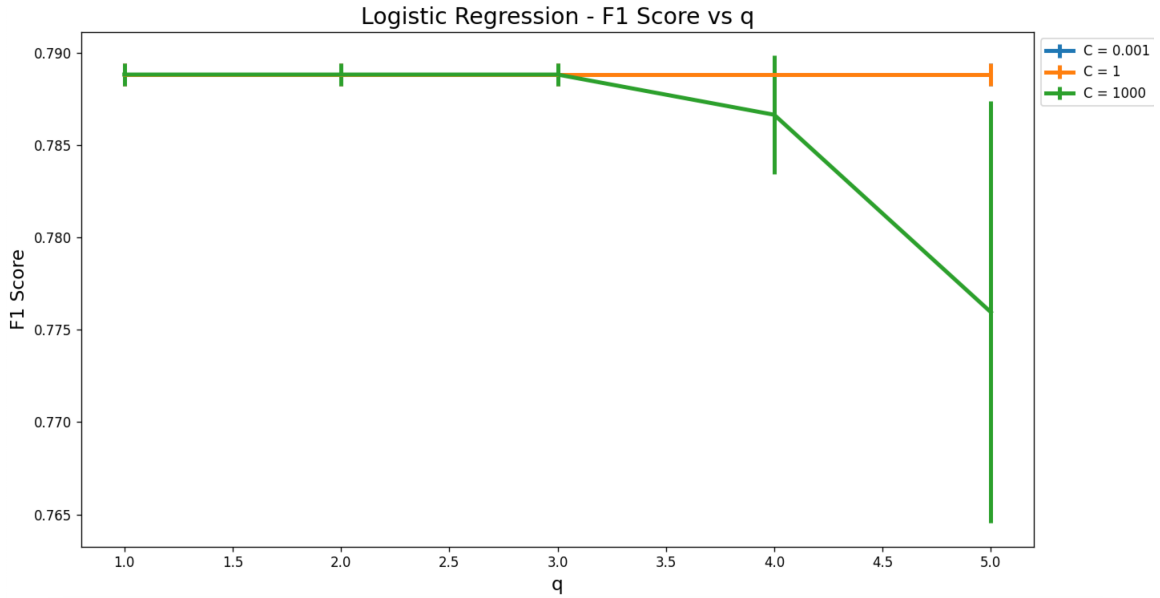
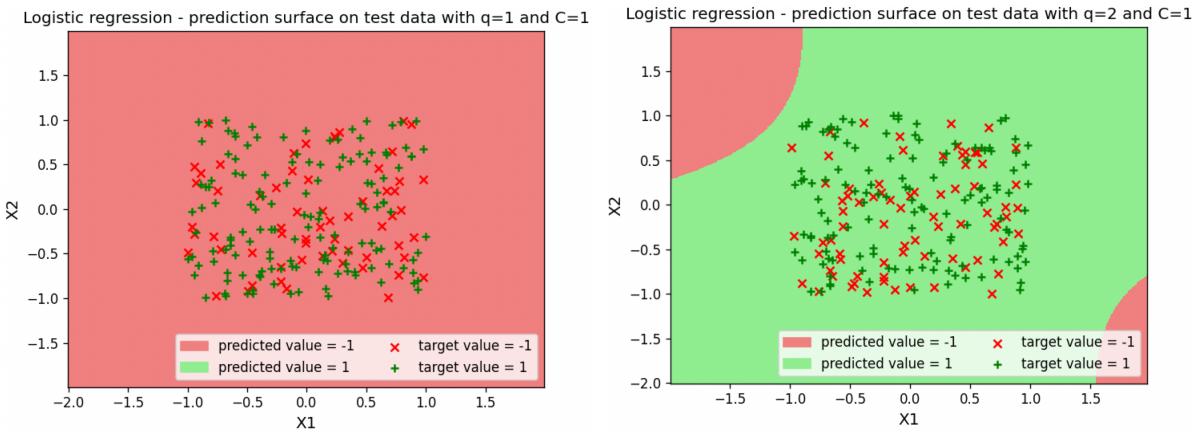


Figure 11: Plot of F1 Scores against q for the logistic regression classifier.

Since the data is very noisy, the predictions will likely not be as accurate but we can see what our choice of $q = 1$ looks like with the model predictions.



(a) Predictions using q=1 (b) Predictions using q=2

Figure 12: Plotting predictions of our trained model (predicted value) against the real test data (target value).

We can confirm from figure 12 that since the data is very noisy, the classifier predicts only one class for the training data and is very inaccurate. We can keep our choice of q=1 as it seems to be the best from figure 11.

Part (a) (ii)

Once again, to choose C I plotted the F1 Score against values of C to find the lowest value of C that yields the highest F1 Score using code from 1.5 in the appendix. We already know that C=1 works well for our model from the plots in (a)(i). Therefore we can try out a wide range of values for C such as 1, 5, 10, 50, 100, 150, 250, 450, 650, 850, 1000 to find the best value for C using 5-fold cross validation.

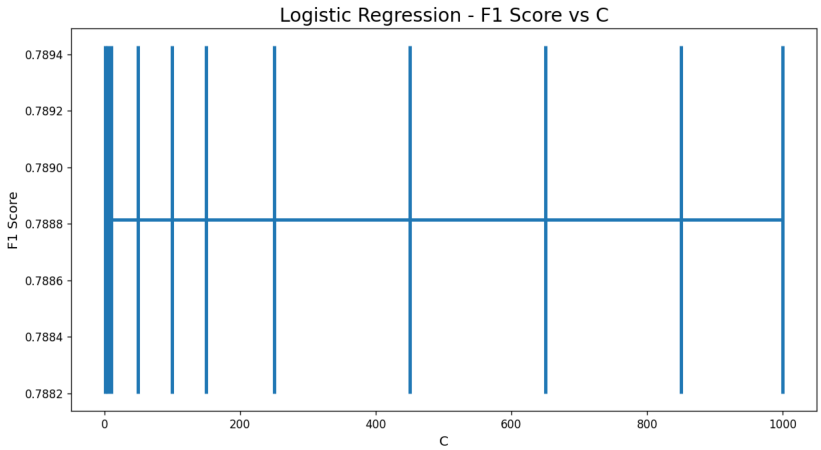
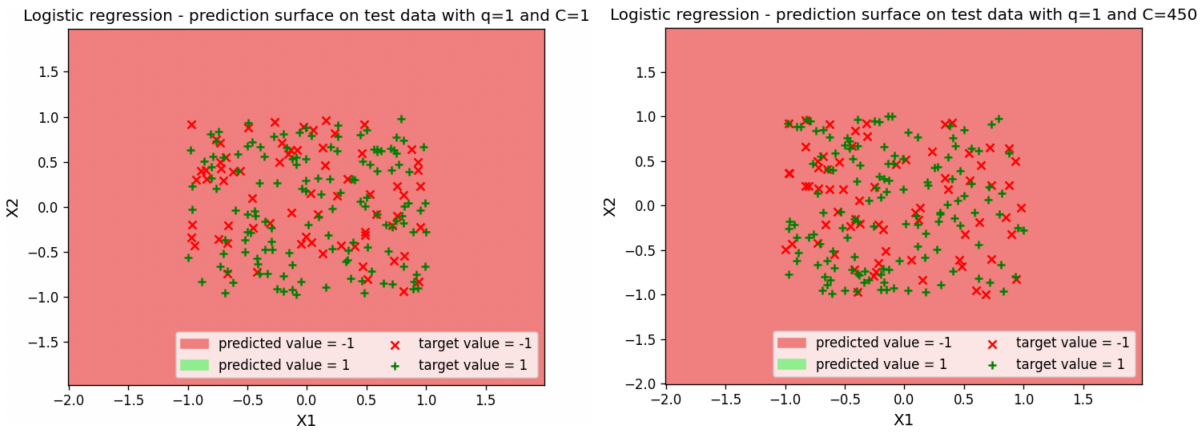


Figure 13: Plot of F1 Scores against C for the logistic regression classifier.

We can see from figure 13 that changing C does not have much of an impact on the classifier. This is because the hyperparameter C is used to apply a penalty to the parameters in the model, however since the data is very noisy, there is a massive uncertainty as to where the model can apply this penalty to, meaning all values of C yield fairly similar results in the plot above. We can confirm this by plotting the prediction surface as seen below. Henceforth, we can use C=1 as it seemed to work well from (a)(i).



(a) Predictions using q=1 and C=1. (b) Predictions using q=1 and C=450.

Figure 14: Plotting predictions of our trained model (predicted value) against the real test data (target value).

Part (b)

Once again, I chose a value for k using 5-fold cross validation using code from 1.6 in the appendix. I did this by plotting the F1 Score against a range of values of k such as 1, 10, 40, 70, 100, 130, 160, 190 to see how the F1 score changed as k is increased.

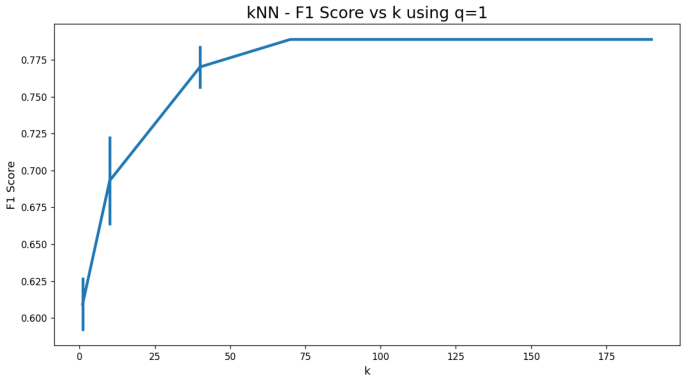


Figure 15: Plot of F1 Scores against k for the kNN classifier.

We can see that the F1 Score increases until k=70, then decreases slightly. The best value for k is 70, since it yields the highest F1 Score. The prediction surface for the kNN classifier predicts one class. We can keep k=70 as it was the best from figure 15.

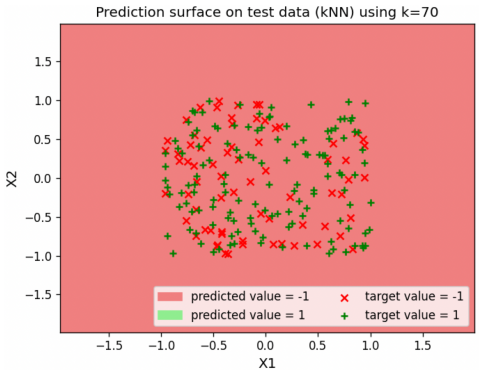
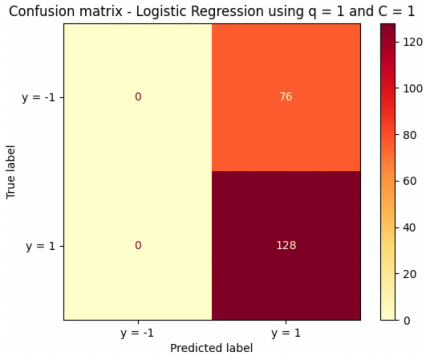
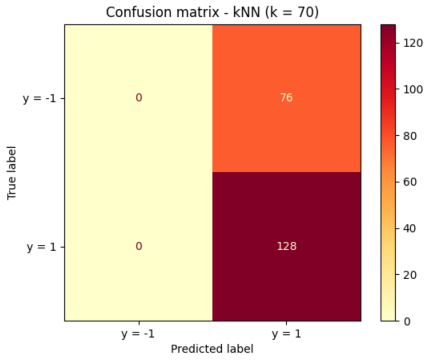


Figure 16: Plotting predictions of the kNN classifier using k=70.

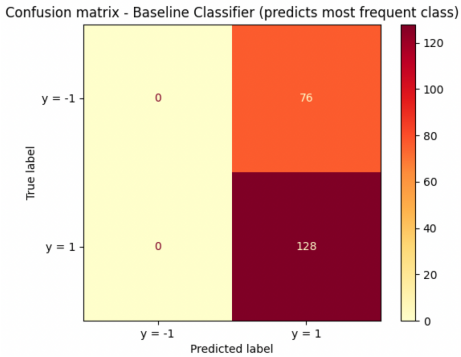
Part (c)



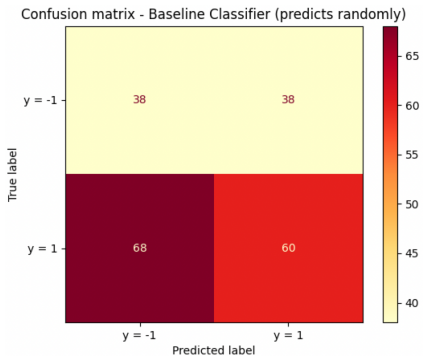
(a) Logistic regression classifier (q=1 & C=1).



(b) kNN classifier (k=70).



(c) Baseline classifier (most frequent).



(d) Baseline classifier (random).

Figure 17: Confusion matrices for 4 classifiers.

Part (d)

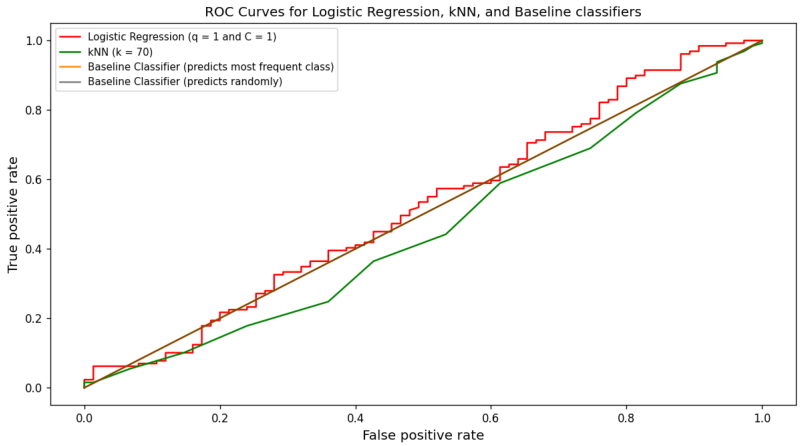


Figure 18: ROC curves for 4 classifiers.

Part (e)

1. **Logistic regression classifier (q=1, C=1)**
   - Accuracy $= \frac{TN+TP}{TN+TP+FN+FP} = \frac{0+128}{0+128+0+76} = 0.627$
   - True positive rate $= \frac{TP}{TP+FN} = \frac{128}{128+0} = 1$
   - False positive rate $= \frac{FP}{TN+FP} = \frac{76}{0+76} = 1$
   - Precision $= \frac{TP}{TP+FP} = \frac{128}{128+76} = 0.627$
2. **kNN classifier (k=70)**
   - Accuracy $= \frac{TN+TP}{TN+TP+FN+FP} = \frac{0+128}{0+128+0+76} = 0.627$
   - True positive rate $= \frac{TP}{TP+FN} = \frac{128}{128+0} = 1$
   - False positive rate $= \frac{FP}{TN+FP} = \frac{76}{0+76} = 1$
   - Precision $= \frac{TP}{TP+FP} = \frac{128}{128+76} = 0.627$
3. **Baseline classifier (predicting most frequent class)**
   - Accuracy $= \frac{TN+TP}{TN+TP+FN+FP} = \frac{0+128}{0+128+0+76} = 0.627$
   - True positive rate $= \frac{TP}{TP+FN} = \frac{128}{128+0} = 1$
   - False positive rate $= \frac{FP}{TN+FP} = \frac{76}{0+76} = 1$
   - Precision $= \frac{TP}{TP+FP} = \frac{128}{128+76} = 0.627$
4. **Baseline classifier (predicting randomly)**
   - Accuracy $= \frac{TN+TP}{TN+TP+FN+FP} = \frac{38+60}{38+60+68+38} = 0.480$
   - True positive rate $= \frac{TP}{TP+FN} = \frac{60}{60+68} = 0.469$
   - False positive rate $= \frac{FP}{TN+FP} = \frac{38}{38+38} = 0.500$
   - Precision $= \frac{TP}{TP+FP} = \frac{60}{60+38} = 0.612$

We can see from the above calculations that the logistic regression classifier, kNN classifier and baseline classifier predicting the most frequent class all have the same accuracy (0.627), true positive rate (1), false positive rate (1), and precision (0.627). This shows that the logistic regression classifier and kNN classifier are not good at making predictions, as they are equivalent to a dummy classifier that simply chooses the most frequent meanwhile the baseline classifier that predicts randomly is worse than all of them. To further analyze their performance, we can see in figure 18 that the ROC curves for all classifiers are not near the top left of the plot. It's important to note that **compared to a random classifier** the logistic regression and kNN classifiers follow both baseline classifiers' ROC curves closely. **There is no classifier I would recommend for this dataset.** This is because the classifiers are the same as dummy classifiers predicting the most frequent class. Furthermore, the ROC curves clearly show the classifiers following both the baseline classifiers. This leads me to believe that they are both very similar to each other, and is not anymore effective than a dummy classifier. Before using a particular classifier, I would recommend trying to add more features or getting data with less noise such as dataset #1.

1 Appendix

1.1 Helper function to read data from CSV

```
import numpy as np
import pandas as pd

def read_data(Filename):
    # Read data from CSV.
    df = pd.read_csv(Filename)
    X1 = df.iloc[:, 0]
    X2 = df.iloc[:, 1]
    X = np.column_stack((X1, X2))
    y = df.iloc[:, 2]
    return X, y
```

1.2 Plot the dataset

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use("TkAgg")

def plot_dataset(X, y):
    plt.rcParams["figure.constrained_layout.use"] = True
    X_positive = X[np.where(y == 1)]
    X_negative = X[np.where(y == -1)]
    plt.scatter(X_positive[:, 0], X_positive[:, 1], c="green", marker="+",
    label="target value = 1")
    plt.scatter(X_negative[:, 0], X_negative[:, 1], c="red", marker="x",
    label="target value = -1")
    plt.ylabel("X2", fontsize=12)
    plt.xlabel("X1", fontsize=12)
    plt.title("Visualization of data with scatter plot", fontsize=17)
    plt.legend(loc="lower right", fontsize=17, markerscale=1.5)
    plt.show()
    plt.rcParams["figure.constrained_layout.use"] = False
```

1.3 Plot the prediction surface

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from matplotlib.colors import ListedColormap
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.patches as mpatches
matplotlib.use("TkAgg")

def plot_prediction_surface(X, y, q, model):
    plt.figure(dpi=120)
```

```
    Xpolynomial = PolynomialFeatures(q).fit_transform(X)
    Xtrain, Xtest, ytrain, ytest = train_test_split(Xpolynomial, y,
    test_size=0.2)
    Xtest_negative = Xtest[np.where(ytest == -1)]
    Xtest_positive = Xtest[np.where(ytest == 1)]
    model.fit(Xtrain, ytrain)
    # Get the minimum and maximum of X and y to create a meshgrid with
    colors representing the decision boundary.
    Xtest_y_minimum = Xtest[:, 2].min() - 1
    Xtest_y_maximum = Xtest[:, 2].max() + 1
    Xtest_X_minimum = Xtest[:, 1].min() - 1
    Xtest_X_maximum = Xtest[:, 1].max() + 1
    X, Y = np.meshgrid(np.arange(Xtest_X_minimum, Xtest_X_maximum, 0.01),
                       np.arange(Xtest_y_minimum, Xtest_y_maximum, 0.01))
    Xtest = PolynomialFeatures(q).fit_transform(np.c_[X.ravel(),
    Y.ravel()])
    predictions = model.predict(Xtest).reshape(X.shape)
    if model.get_params().__contains__("C"):
        plt.title("Logistic regression - prediction surface on test data
        with q=" + str(q) + " and C=" + str(
            model.get_params()["C"]))
    elif model.get_params().__contains__("n_neighbors"):
        plt.title("Prediction surface on test data (kNN) using k=" +
        str(model.get_params()["n_neighbors"]))
    colors_for_prediction_surface = ListedColormap(["lightcoral",
    "lightgreen"])
    plt.pcolormesh(X, Y, predictions, cmap=colors_for_prediction_surface)
    plt.scatter(Xtest_negative[:, 1], Xtest_negative[:, 2], c="red",
    marker="x", label="target value = -1")
    plt.scatter(Xtest_positive[:, 1], Xtest_positive[:, 2], c="green",
    marker="+", label="target value = 1")
    plt.ylabel("X2", fontsize=12)
    plt.xlabel("X1", fontsize=12)
    handles, labels = plt.gca().get_legend_handles_labels()
    handles.insert(0, mpatches.Patch(color="lightgreen", label="predicted
    value = 1"))
    handles.insert(0, mpatches.Patch(color="lightcoral", label="predicted
    value = -1"))
    plt.legend(loc="lower right", ncol=2, fontsize=10, handles=handles)
    plt.show()
```

1.4 Question (a) (i)

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
matplotlib.use("TkAgg")

def part_a_i(X, y, range_of_values_for_q, range_of_values_of_C):
    plt.rcParams["figure.constrained_layout.use"] = True
```

```
    plt.figure(dpi=120)
    for C in range_of_values_of_C:
        model = LogisticRegression(C=C, penalty="l2", max_iter=1000)
        std_error = []
        mean_error = []
        for q in range_of_values_for_q:
            Xpolynomial = PolynomialFeatures(q).fit_transform(X)
            scores = cross_val_score(model, Xpolynomial, y, cv=5,
            scoring="f1")
            std_error.append(np.array(scores).std())
            mean_error.append(np.array(scores).mean())
        plt.errorbar(range_of_values_for_q, mean_error, yerr=std_error,
        label=("C = " + str(C)), linewidth=3)
    plt.title("Logistic Regression - F1 Score vs q", fontsize=17)
    plt.ylabel("F1 Score", fontsize=14)
    plt.xlabel("q", fontsize=14)
    plt.legend(bbox_to_anchor=(1, 1), loc="upper left")
    plt.show()
    plt.rcParams["figure.constrained_layout.use"] = False
```

1.5 Question (a) (ii)

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
matplotlib.use("TkAgg")

def part_a_ii(X, y, q, range_of_values_of_C):
    plt.figure(dpi=120)
    Xpolynomial = PolynomialFeatures(q).fit_transform(X)
    std_error = []
    mean_error = []
    # Use 5-fold cross validation to choose C.
    for C in range_of_values_of_C:
        model = LogisticRegression(C=C, penalty="l2", max_iter=1000)
        scores = cross_val_score(model, Xpolynomial, y, cv=5, scoring="f1")
        std_error.append(np.array(scores).std())
        mean_error.append(np.array(scores).mean())
        # Print parameters from model
        model.fit(Xpolynomial, y)
        print("C =" + str(C))
        print("θ =" + str(np.insert(model.coef_, 0, model.intercept_)))
    plt.errorbar(range_of_values_of_C, mean_error, yerr=std_error,
    linewidth=3)
    plt.title("Logistic Regression - F1 Score vs C", fontsize=17)
    plt.ylabel("F1 Score", fontsize=12)
    plt.xlabel("C", fontsize=12)
    plt.show()
```

1.6 Question (i) (b)

```
import numpy as np
import matplotlib.pyplot as plt
```

```
import matplotlib
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
matplotlib.use("TkAgg")

def part_b(X, y, q, range_of_values_of_k):
    plt.figure(dpi=120)
    Xpoly = PolynomialFeatures(q).fit_transform(X)
    std_error = []
    mean_error = []
    for k in range_of_values_of_k:
        model = KNeighborsClassifier(n_neighbors=k, weights="uniform")
        scores = cross_val_score(model, Xpoly, y, cv=5, scoring="f1")
        std_error.append(np.array(scores).std())
        mean_error.append(np.array(scores).mean())
    plt.errorbar(range_of_values_of_k, mean_error, yerr=std_error,
    linewidth=3)
    plt.title("kNN - F1 Score vs k using q=" + str(q), fontsize=17)
    plt.ylabel("F1 Score", fontsize=12)
    plt.xlabel("k", fontsize=12)
    plt.show()
```

1.7 Question (i) (c)

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from sklearn.metrics import ConfusionMatrixDisplay, roc_curve
from sklearn.preprocessing import PolynomialFeatures
matplotlib.use("TkAgg")

def part_c_helper(X, y, q):
    Xpolynomial = PolynomialFeatures(q).fit_transform(X)
    Xtrain, Xtest, ytrain, ytest = train_test_split(Xpolynomial, y,
    test_size=0.2)
    return Xtrain, Xtest, ytrain, ytest

def part_c(Xtrain, Xtest, ytrain, ytest, model, title):
    model.fit(Xtrain, ytrain)
    ConfusionMatrixDisplay.from_estimator(model, Xtest, ytest,
    display_labels=["y = -1", "y = 1"], cmap=plt.cm.YlOrRd)
    plt.title("Confusion matrix - " + title)
    plt.show()
```

1.8 Question (i) (d)

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from sklearn.dummy import DummyClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
matplotlib.use("TkAgg")
```

```
def part_d(X, y, q, C, k):
    plt.figure(dpi=120)
    Xpolynomial = PolynomialFeatures(q).fit_transform(X)
    Xtrain, Xtest, ytrain, ytest = train_test_split(Xpolynomial, y,
    test_size=0.2)
    # Plot ROC curve for logistic regression
    model = LogisticRegression(penalty="l2", C=C).fit(Xtrain, ytrain)
    fpr, tpr, _ = roc_curve(ytest, model.decision_function(Xtest))
    plt.plot(fpr, tpr, label="Logistic Regression (q = " + str(q) + " and
    C = " + str(C) + ")", color="red")
    # Plot ROC curve for kNN
    model = KNeighborsClassifier(n_neighbors=k,
    weights="uniform").fit(Xtrain, ytrain)
    fpr, tpr, _ = roc_curve(ytest, model.predict_proba(Xtest)[:, 1])
    plt.plot(fpr, tpr, label="kNN (k = " + str(k) + ")", color="green")
    # Plot ROC curve for dummy classifier (predicts most frequent class)
    model = DummyClassifier(strategy="most_frequent").fit(Xtrain, ytrain)
    fpr, tpr, _ = roc_curve(ytest, model.predict_proba(Xtest)[:, 1])
    plt.plot(fpr, tpr, label="Baseline Classifier (predicts most frequent
    class)", color="darkorange")
    # Plot ROC curve for dummy classifier (predicts randomly)
    model = DummyClassifier(strategy="uniform").fit(Xtrain, ytrain)
    fpr, tpr, _ = roc_curve(ytest, model.predict_proba(Xtest)[:, 1])
    plt.plot(fpr, tpr, label="Baseline Classifier (predicts randomly)",
    color="black", alpha=0.5)
    plt.title("ROC Curves for Logistic Regression, kNN, and Baseline
    classifiers")
    plt.ylabel("True positive rate", fontsize=12)
    plt.xlabel("False positive rate", fontsize=12)
    plt.legend(fontsize=9)
    plt.show()
```