Assignment Week 4

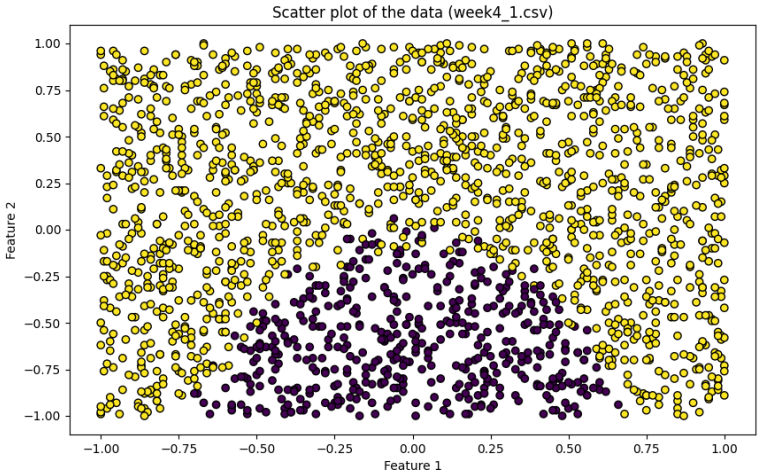
Student Name: Patrick Farmer Student Number: 20501828

Date: 18-10-2024 Dataset1: 12-24-12-0 Dataset2: 12-12-12-0

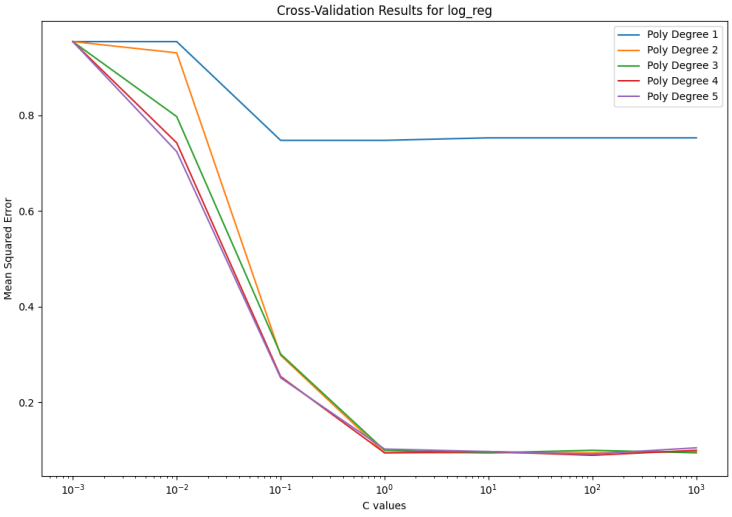
Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

I(a)

First the data read and plotted onto a scatter plot as can be seen below in figure 1. We can see clearly from the data that the data is non linear. We can also see that it is likely that the data set would work best with a polynomial of order 2. We can confirm this with the use of cross validation.

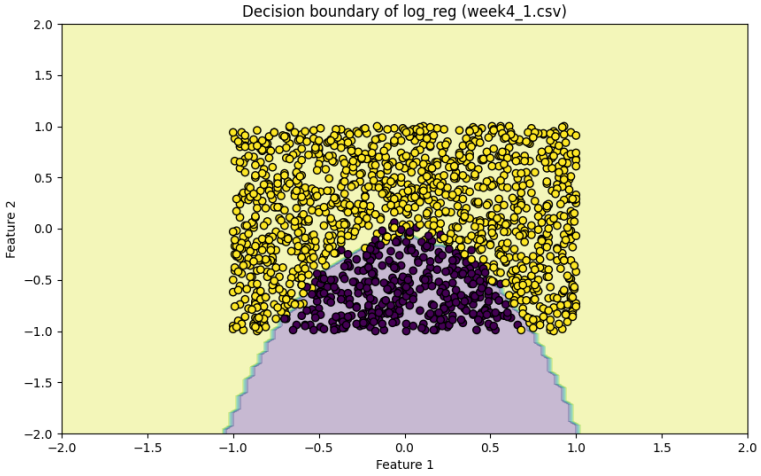


From the cross validation results we can see that the difference between the mean squared error for degree 1 and degree 2 is very large. This is due to the data being non linear. We can also see that the difference between the degrees 2, 3, 4 and 5 are negligible. This is due to the underlying structure of the data being simple enough to be accurately modelled by a polynomial of degree 2.
This graph also shows us that the value of C beyond 1 has very little effect on the mean squared error, so the code will select a value of approximately 1.



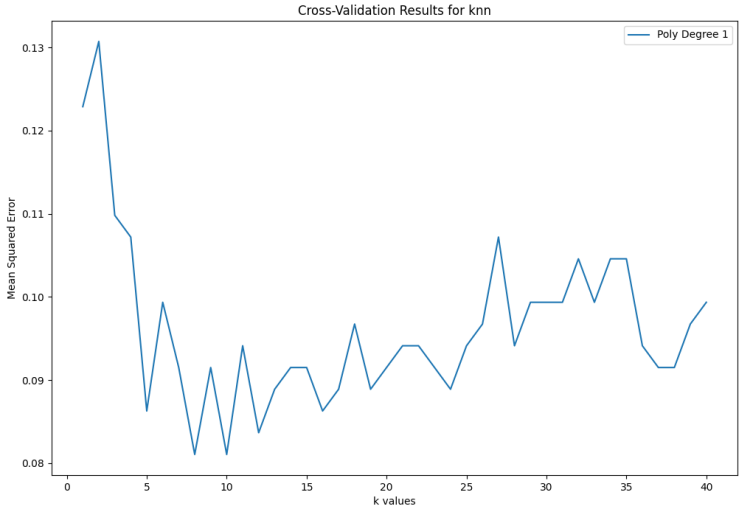
Now knowing that the order of the model is degree 2 we can train a logistic regression model on this

and plot the decision boundary as can be seen in the graph below. As the cross-validation results showed the logistic regression model captures the underlying structure of the data well.



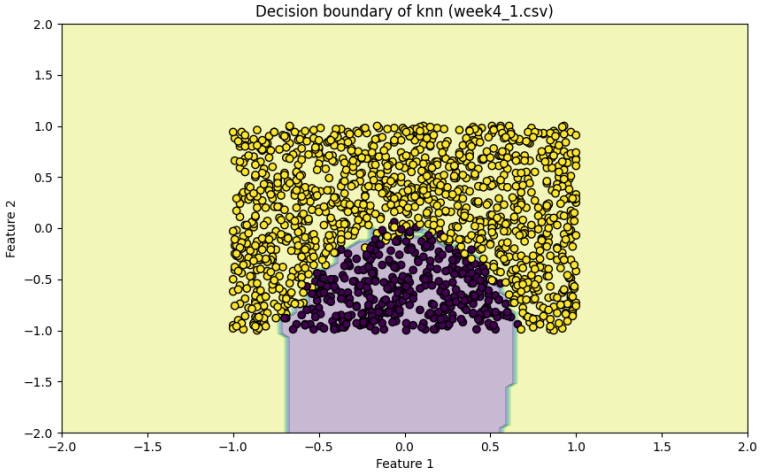
I(b)

The same process was repeated for the KNN model. Due to the simplicity of the data set we are likely to see good results for all values of K and a rather quick drop off in the mean square error as K increases.
We do get these predicted results and we see a minimum of MSE at K=8, K=10 and K=12. All of which would produce similar results. The code will select K=8 as the best parameter due to the negligible difference in MSE between the three values.



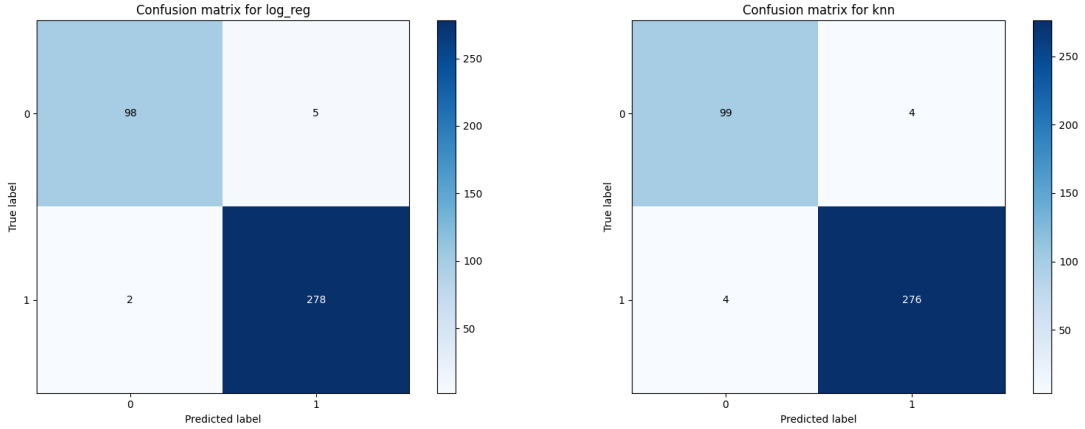
Next we can plot the decision boundary for the KNN model. We can see that the KNN model is

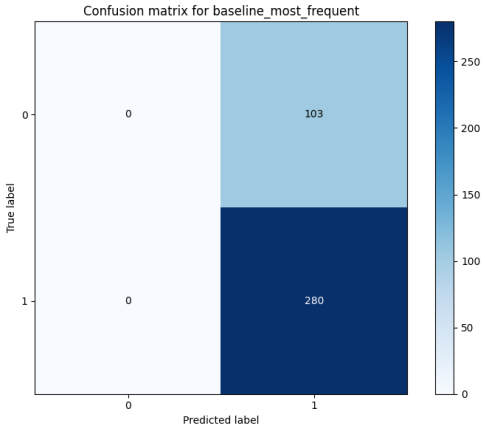
accurate for this data. it does not actually capture any structure of the data though like logistic regression does. This does not matter if the test data is in the same range as the training data but if for some reason it were to go beyond [-1, 1] the accuracy would fall significantly. For the given data though the model is accurate.

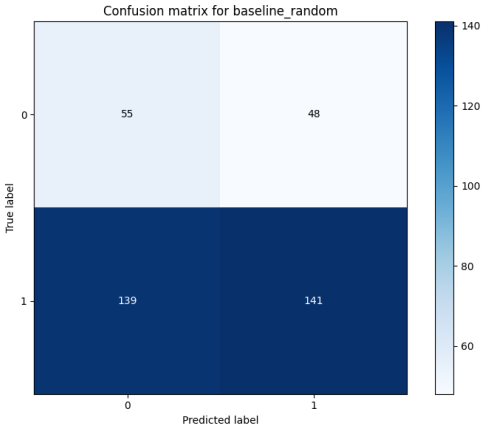


I(c)

Next the confusion matrices were plotted for both models and for a dummy classifier alongside them. The test to training data split in this case was 20/80. We can see from this that the logistic regression model is marginally better than the KNN model (Only 1 more correct prediction). We can also see that the logistic regression model tends to predict false positives more often than false negatives as apposed to the KNN model which has an equal distribution of both. This is likely due to the logistic regression model capturing the underlying structure of the data and handling the cluster of data points at the top of the curve with positives and negatives overlapping differently to the KNN model. The most frequent dummy classifier is of course not ass accurate. It predicts true on all ocassions. This model being substantially worse shows that our models are not just selecting the most likely in all scenarios but are actually capturing the complexities of the data. We also see our random dummy classifier being worse again than the most frequent.

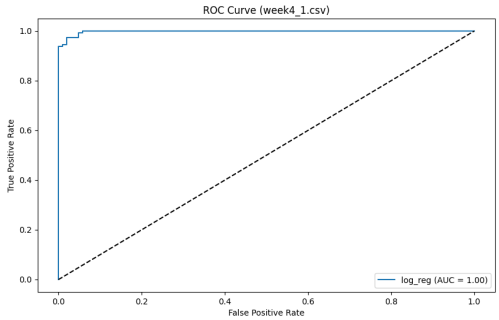


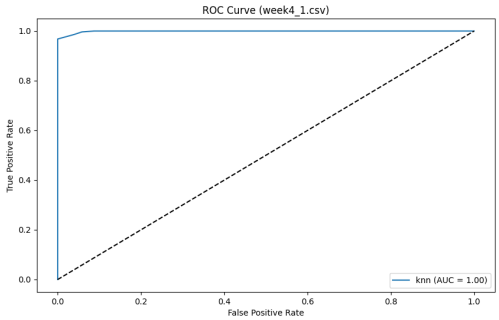


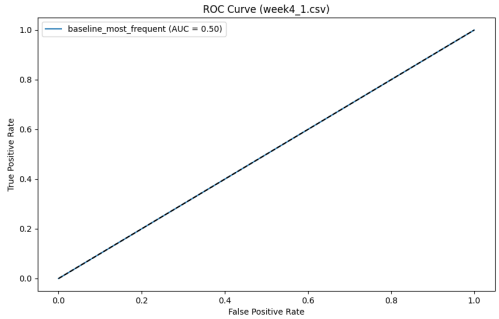


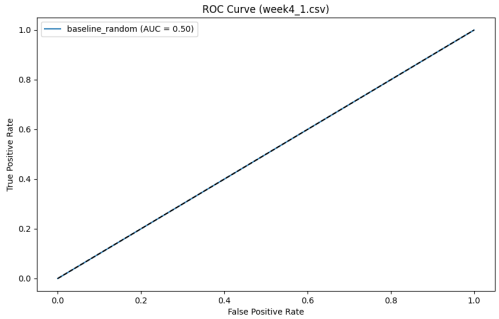
I(d)

Next the ROC curves were plotted. The KNN and logistic regression curves are very similar except for the ROC curve of the KNN converging faster originally but then logistic regression catches up and just about overtakes it. We can then also see that both the dummy classifiers are very innacurate.









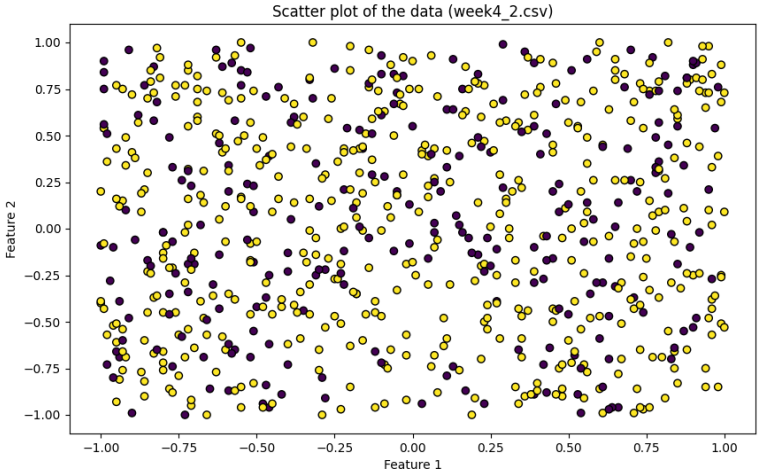
I(e)

Taking the information from the ROC curves and confusion matrices we can see that the logistic regression model and KNN model are very similar in performance but the logistic regression model is performing very slightly better as was commented on for both the ROC curve and the confusion matrix. The logistic regression model is also slightly more desireable as it manages to capture the underlying

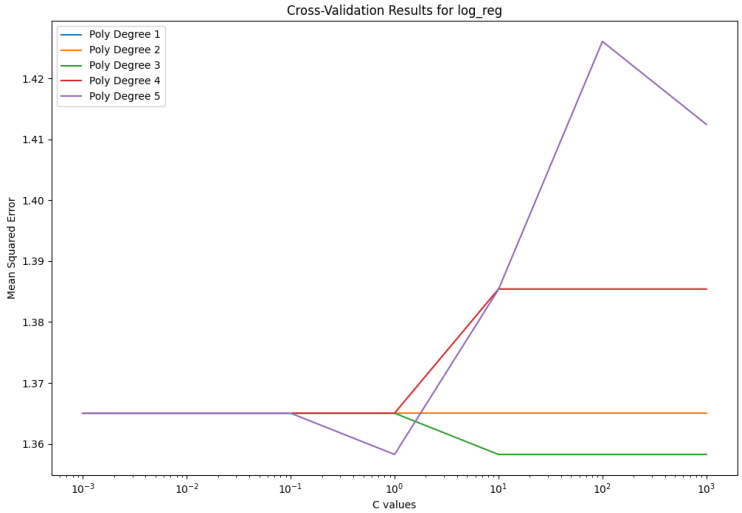
structure of the data. The KNN model is also very accurate but does not capture the structure of the data. Both of the models performed much better than the baseline models. This is shown in both the ROC curve and the confusion matrix and proves that the models are not just performing well due to the data being simple but because they are actually capturing the complexities of the data.

II(a)

No additional code was needed to run the models on the second data set. The loop was simply incremented. We can from the initial scatter plot produced that the code is seemingly just random with no structure. We can also see that the class distribution of the data is very similar to the first data set so we will be likely to see the most frequent baseline predictor perform the same on this data.



The cross validation results from the logistic regression show very little change in the data as the value of C and power of the polynomial changes. This is because there is no underlying structure to capture and every iteration of the model is essential just the same as the dummy random classifier.



The decision boundary for the logistic regression model predicts that the entire data set is one class (The most frequent class). This is the exact same behaviour as the most frequent dummy classifier.

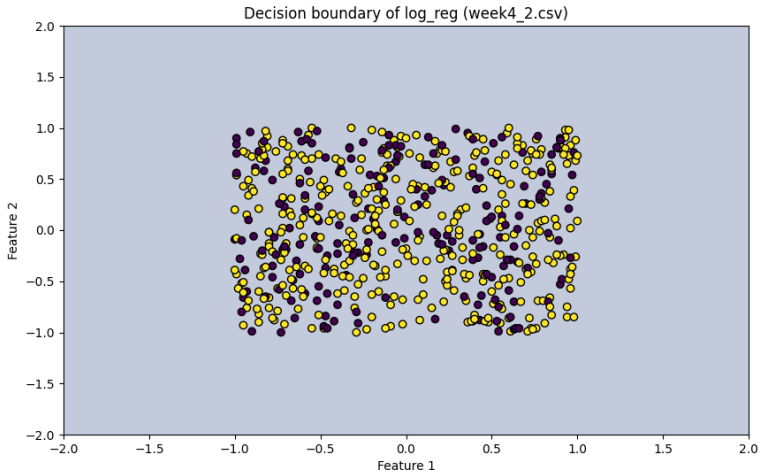
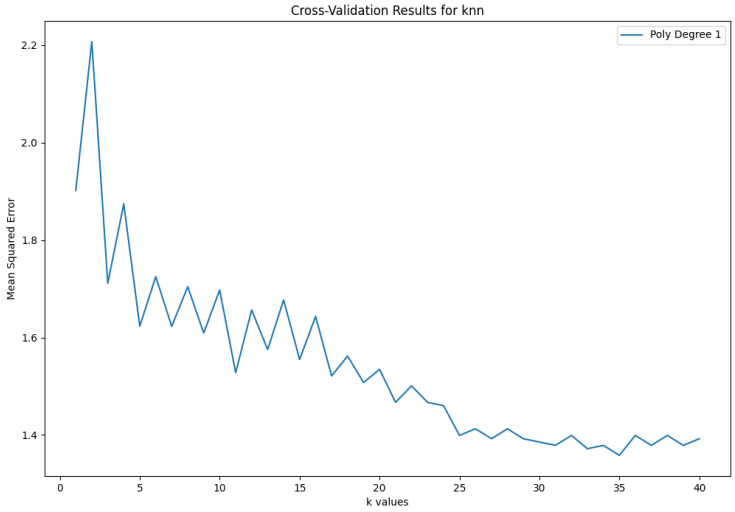
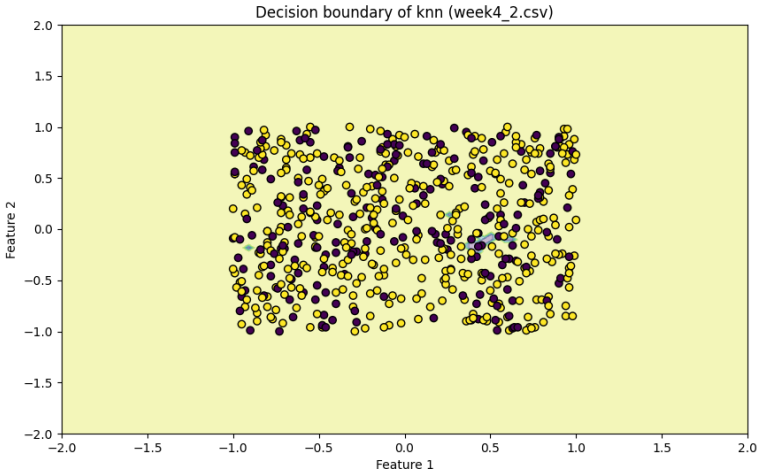


Figure 1: Figure.16

II(b)

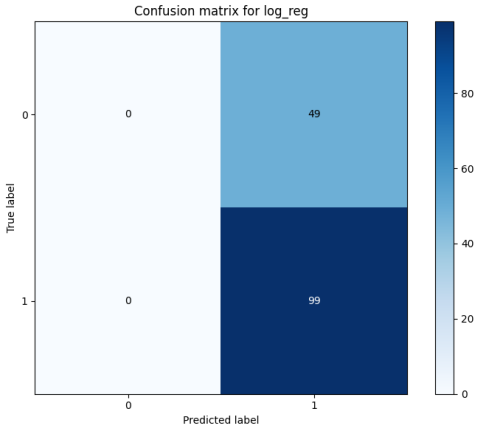
The KNN model also performs poorly on the data and will come to predict the most likely class in every instance like the logistic regression. However, as can be seen from the cross-validation plot it will take longer to converge to this solution as the value of k will have to become large enough to in every scenario have a majority of neighbours in the most frequent class. For smaller values of k the model will be much more random. If the data had even a small amount of clustering the model would perform better at a low value of k but since this is not true in this instance it will find its minimum MSE at a high value of k.

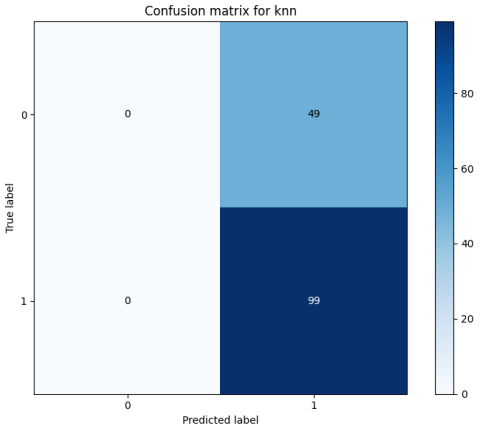


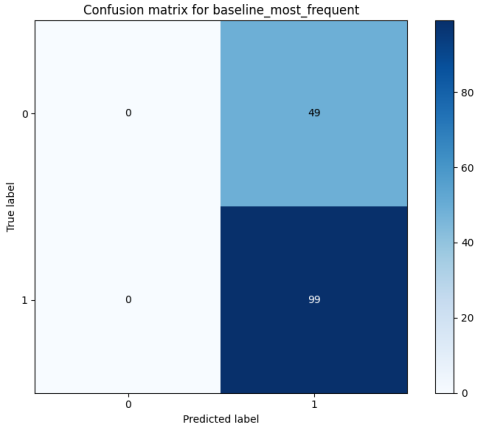


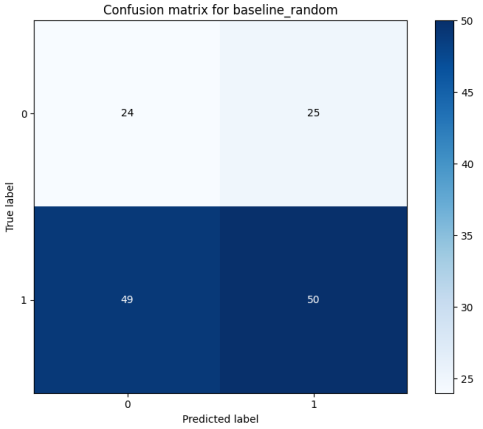
II(c)

From the confusion matrix we can see the proof of the already summised information. The logistic regression model, KNN model and the most frequent dummy classifier all perform the same because they have come to the same conclusion. The only model with a different confusion matrix is the random dummy classifier. Behind the scenes though the KNN model is the worst of the 3 as the amount of neighbours that are being taken into consideration the computation cost of this specific KNN model is very high.

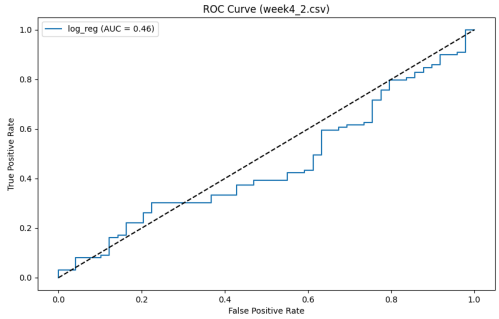

Confusion matrix for log_reg
True label
Predicted label
0
1
0
49
0
99
80
60
40
20
0

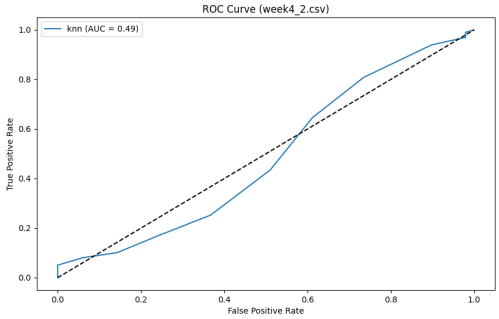

Confusion matrix for knn
True label
Predicted label
0
1
0
49
0
99
80
60
40
20
0

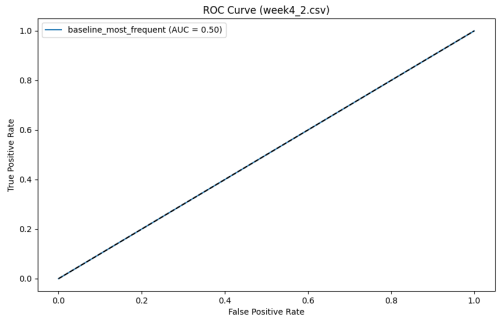

Confusion matrix for baseline_most_frequent
True label
Predicted label
0
1
0
49
0
99
80
60
40
20
0

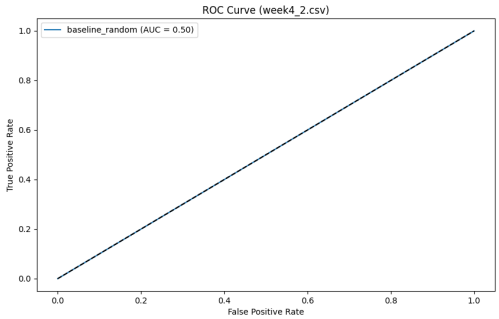

Confusion matrix for baseline_random
True label
Predicted label
0
1
24
25
49
50
50
45
40
35
30
25

II(d)


ROC Curve (week4_2.csv)
log_reg (AUC = 0.46)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


ROC Curve (week4_2.csv)
knn (AUC = 0.49)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


ROC Curve (week4_2.csv)
baseline_most_frequent (AUC = 0.50)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


ROC Curve (week4_2.csv)
baseline_random (AUC = 0.50)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

Appendix

I(a)

```
for index in range(1, 3):
    data = pd.read_csv(f'week4_{index}.csv', skiprows=1)
    X = data.iloc[:, :-1].values
    y = data.iloc[:, -1].values
    plt.figure(figsize=(10, 6))
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o')
    plt.title(f'Scatter plot of the data (week4_{index}.csv)')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    i_string = 'i' * index
    plt.savefig(f'Images/{i_string}(a(1)).png')
    plt.close()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    log_reg_runner = model_runner.ModelRunner('log_reg', X_train, y_train, X_test, y_test, i_string=i
    log_reg_runner.train_model()
    log_reg_runner.predict()
    log_reg_runner.plot_decision_boundary(index)
```

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

class ModelRunner:
    def __init__(self, model_name, X_train, y_train, X_test, y_test, i_string=''):
        self.model_name = model_name
        self.X_train = X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test
        self.i_string = i_string
        self.final_model = None
        self.best_params = None
        self.y_pred = None
        self.y_prob = None

    def train_model(self):
        best_score = float('inf')
        best_model = None
        best_params = None
        k_values = list(range(1, 41))
        C_values = np.logspace(-3, 3, 7)
        poly_degrees = list(range(1, 6))
```

```
cv_results = []

improvement_threshold = 0.01

for poly_degree in poly_degrees:
    poly = PolynomialFeatures(degree=poly_degree)
    X_poly = poly.fit_transform(self.X_train)

    if self.model_name == 'log_reg':
        for C_value in C_values:
            model = LogisticRegression(C=C_value)
            scores = cross_val_score(model, X_poly, self.y_train, cv=5, scoring='neg_mean_squ
            mean_score = -scores.mean()
            cv_results.append((poly_degree, C_value, mean_score))

            if best_score - mean_score > improvement_threshold:
                best_score = mean_score
                best_model = model
                best_params = {'poly_degree': poly_degree, 'C': C_value}
    elif self.model_name == 'knn':
        for k_value in k_values:
            model = KNeighborsClassifier(n_neighbors=k_value)
            scores = cross_val_score(model, X_poly, self.y_train, cv=5, scoring='neg_mean_squ
            mean_score = -scores.mean()
            cv_results.append((poly_degree, k_value, mean_score))

            if best_score - mean_score > improvement_threshold:
                best_score = mean_score
                best_model = model
                best_params = {'poly_degree': poly_degree, 'n_neighbors': k_value}
    elif self.model_name == 'baseline_most_frequent':
        model = DummyClassifier(strategy='most_frequent')
        scores = cross_val_score(model, X_poly, self.y_train, cv=5, scoring='neg_mean_squared
        mean_score = -scores.mean()
        cv_results.append((poly_degree, 'most_frequent', mean_score))

        if best_score - mean_score > improvement_threshold:
            best_score = mean_score
            best_model = model
            best_params = {'poly_degree': poly_degree, 'strategy': 'most_frequent'}
    elif self.model_name == 'baseline_random':
        model = DummyClassifier(strategy='uniform')
        scores = cross_val_score(model, X_poly, self.y_train, cv=5, scoring='neg_mean_squared
        mean_score = -scores.mean()
        cv_results.append((poly_degree, 'random', mean_score))

        if best_score - mean_score > improvement_threshold:
            best_score = mean_score
            best_model = model
            best_params = {'poly_degree': poly_degree, 'strategy': 'uniform'}

plt.figure(figsize=(12, 8))
```

```
        for poly_degree in poly_degrees:
            if self.model_name == 'log_reg':
                scores = [result[2] for result in cv_results if result[0] == poly_degree]
                plt.plot(C_values, scores, label=f'Poly Degree {poly_degree}')
            elif self.model_name == 'knn':
                scores = [result[2] for result in cv_results if result[0] == poly_degree]
                plt.plot(k_values, scores, label=f'Poly Degree {poly_degree}')
            elif self.model_name in ['baseline_most_frequent', 'baseline_random']:
                scores = [result[2] for result in cv_results if result[0] == poly_degree]
                plt.plot([0], scores, label=f'Poly Degree {poly_degree}', marker='o')

        plt.xlabel('C values' if self.model_name == 'log_reg' else 'k values' if self.model_name == '
        plt.ylabel('Mean Squared Error')
        plt.title(f'Cross-Validation Results for {self.model_name}')
        plt.legend()
        plt.xscale('log' if self.model_name == 'log_reg' else 'linear')
        if self.model_name == 'log_reg':
            plt.savefig(f'Images/{self.i_string}(a(2)).png')
        elif self.model_name == 'knn':
            plt.savefig(f'Images/{self.i_string}(b(2)).png')
        plt.close()

        self.best_params = best_params
        self.poly = PolynomialFeatures(degree=best_params['poly_degree'])
        X_poly_train = self.poly.fit_transform(self.X_train)
        self.final_model = best_model.fit(X_poly_train, self.y_train)

    def predict(self):
        poly = PolynomialFeatures(degree=self.best_params['poly_degree'])
        X_poly_test = poly.fit_transform(self.X_test)
        self.y_pred = self.final_model.predict(X_poly_test)
        self.y_prob = self.final_model.predict_proba(X_poly_test)[:, 1]

    def plot_decision_boundary(self, index):
        xx, yy = np.meshgrid(np.linspace(self.X_train[:, 0].min() - 1, self.X_train[:, 0].max() + 1,
                             np.linspace(self.X_train[:, 1].min() - 1, self.X_train[:, 1].max() + 1,
        X_poly_mesh = self.poly.transform(np.c_[xx.ravel(), yy.ravel()])
        Z = self.final_model.predict(X_poly_mesh)
        Z = Z.reshape(xx.shape)

        plt.figure(figsize=(10, 6))
        plt.contourf(xx, yy, Z, alpha=0.3)
        plt.scatter(self.X_train[:, 0], self.X_train[:, 1], c=self.y_train, edgecolors='k', marker='o
        plt.title(f'Decision boundary of {self.model_name} (week4_{index}.csv)')
        plt.xlabel('Feature 1')
        plt.ylabel('Feature 2')
        if self.model_name == 'log_reg':
            plt.savefig(f'Images/{self.i_string}(a(3)).png')
        else:
            plt.savefig(f'Images/{self.i_string}(b(3)).png')
        plt.close()
```

I(b)

```
for index in range(1, 3):
    knn_runner = model_runner.ModelRunner('knn', X_train, y_train, X_test, y_test, i_string=i_string)
    knn_runner.train_model()
    knn_runner.predict()
    knn_runner.plot_decision_boundary(index)
```

I(c)

```
for index in range(1, 3):
    log_reg_runner.plot_confusion_matrix()
    knn_runner.plot_confusion_matrix()

    baseline_runner = model_runner.ModelRunner('baseline_most_frequent', X_train, y_train, X_test, y_
    baseline_runner.train_model()
    baseline_runner.predict()
    baseline_runner.plot_confusion_matrix()
    baseline_runner.plot_decision_boundary(index)
    baseline_runner.plot_roc_curve(index)

    baseline_runner = model_runner.ModelRunner('baseline_random', X_train, y_train, X_test, y_test, i
    baseline_runner.train_model()
    baseline_runner.predict()
    baseline_runner.plot_confusion_matrix()
    baseline_runner.plot_decision_boundary(index)
    baseline_runner.plot_roc_curve(index)
```

```
def plot_confusion_matrix(self):
    cm = confusion_matrix(self.y_test, self.y_pred)

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title(f'Confusion matrix for {self.model_name}')
    plt.colorbar()
    tick_marks = np.arange(len(np.unique(self.y_test)))
    plt.xticks(tick_marks, tick_marks)
    plt.yticks(tick_marks, tick_marks)

    fmt = 'd'
    thresh = cm.max() / 2.
    for i, j in np.ndindex(cm.shape):
        plt.text(j, i, format(cm[i, j], fmt),
                 ha="center", va="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    if self.model_name == 'log_reg':
        plt.savefig(f'Images/{self.i_string}(c(1)).png')
    elif self.model_name == 'knn':
        plt.savefig(f'Images/{self.i_string}(c(2)).png')
```

```
    elif self.model_name == 'baseline_most_frequent':
        plt.savefig(f'Images/{self.i_string}(c(3)).png')
    elif self.model_name == 'baseline_random':
        plt.savefig(f'Images/{self.i_string}(c(4)).png')
    plt.close()

    return cm
```

I(d)

```
for index in range(1, 3):
    log_reg_runner.plot_roc_curve(index)
    knn_runner.plot_roc_curve(index)
```

```
def plot_roc_curve(self, index):
    fpr, tpr, _ = roc_curve(self.y_test, self.y_prob)
    auc_score = auc(fpr, tpr)

    plt.figure(figsize=(10, 6))
    plt.plot(fpr, tpr, label=f'{self.model_name} (AUC = {auc_score:.2f})')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve (week4_{index}.csv)')
    plt.legend(loc='best')
    if self.model_name == 'log_reg':
        plt.savefig(f'Images/{self.i_string}(d(1)).png')
    elif self.model_name == 'knn':
        plt.savefig(f'Images/{self.i_string}(d(2)).png')
    elif self.model_name == 'baseline_most_frequent':
        plt.savefig(f'Images/{self.i_string}(d(3)).png')
    elif self.model_name == 'baseline_random':
        plt.savefig(f'Images/{self.i_string}(d(4)).png')
    plt.close()
```

II(a)

The code is generic and runs for other dataset automatically

```
for index in range(1, 3):
```

Other (main)

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, GridSearchCV, train_test_split
from sklearn.metrics import confusion_matrix, roc_curve, auc
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
import os
from collections import Counter
from sklearn.pipeline import Pipeline
```

```
import model_runner

os.makedirs('Images', exist_ok=True)
script_dir = os.path.dirname(os.path.abspath(__file__))
os.chdir(script_dir)

for index in range(1, 3):
    data = pd.read_csv(f'week4_{index}.csv', skiprows=1)
    X = data.iloc[:, :-1].values
    y = data.iloc[:, -1].values
    plt.figure(figsize=(10, 6))
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o')
    plt.title(f'Scatter plot of the data (week4_{index}.csv)')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    i_string = 'i' * index
    plt.savefig(f'Images/{i_string}(a(1)).png')
    plt.close()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    log_reg_runner = model_runner.ModelRunner('log_reg', X_train, y_train, X_test, y_test, i_string=i
    log_reg_runner.train_model()
    log_reg_runner.predict()
    log_reg_runner.plot_confusion_matrix()
    log_reg_runner.plot_decision_boundary(index)
    log_reg_runner.plot_roc_curve(index)

    knn_runner = model_runner.ModelRunner('knn', X_train, y_train, X_test, y_test, i_string=i_string)
    knn_runner.train_model()
    knn_runner.predict()
    knn_runner.plot_confusion_matrix()
    knn_runner.plot_decision_boundary(index)
    knn_runner.plot_roc_curve(index)

    baseline_runner = model_runner.ModelRunner('baseline_most_frequent', X_train, y_train, X_test, y_
    baseline_runner.train_model()
    baseline_runner.predict()
    baseline_runner.plot_confusion_matrix()
    baseline_runner.plot_decision_boundary(index)
    baseline_runner.plot_roc_curve(index)

    baseline_runner = model_runner.ModelRunner('baseline_random', X_train, y_train, X_test, y_test, i
    baseline_runner.train_model()
    baseline_runner.predict()
    baseline_runner.plot_confusion_matrix()
    baseline_runner.plot_decision_boundary(index)
    baseline_runner.plot_roc_curve(index)
```