

## Assignment Week 4

Student Name: Patrick Farmer Student Number: 20331828

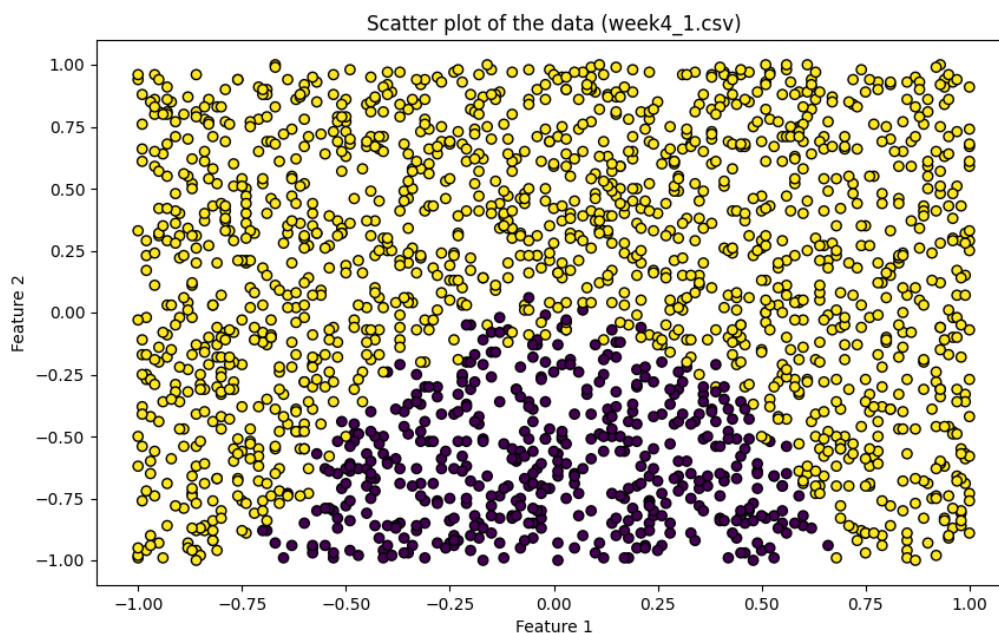
Date: 2024-07-10 Dataset1: 12-24-12-0 Dataset2: 12-12-12-0



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

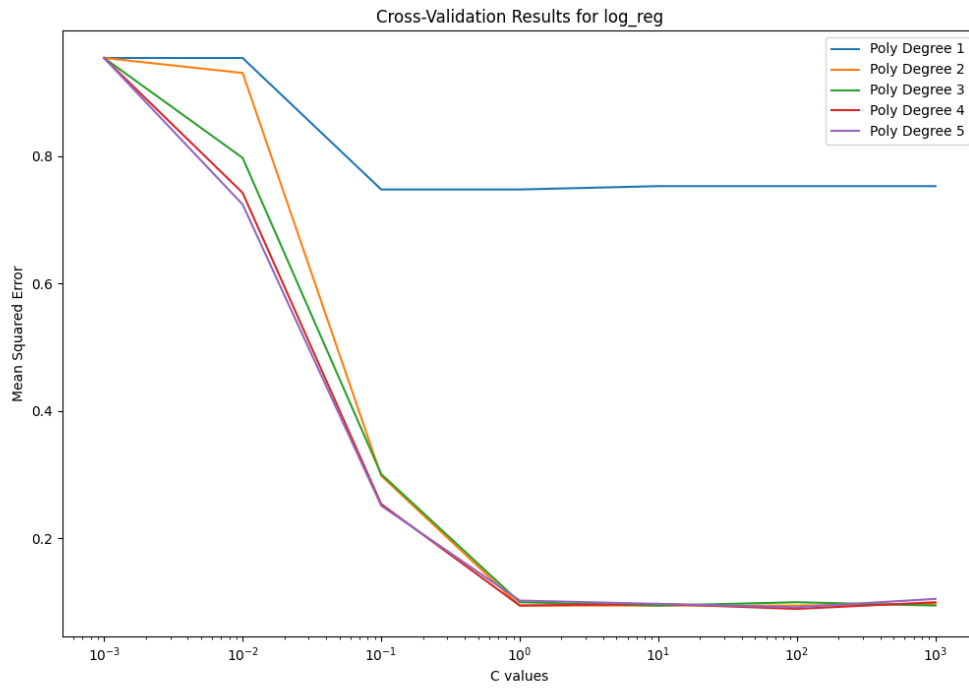
## I(a)

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 we can choose a value of 1 for  $C$ .



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.

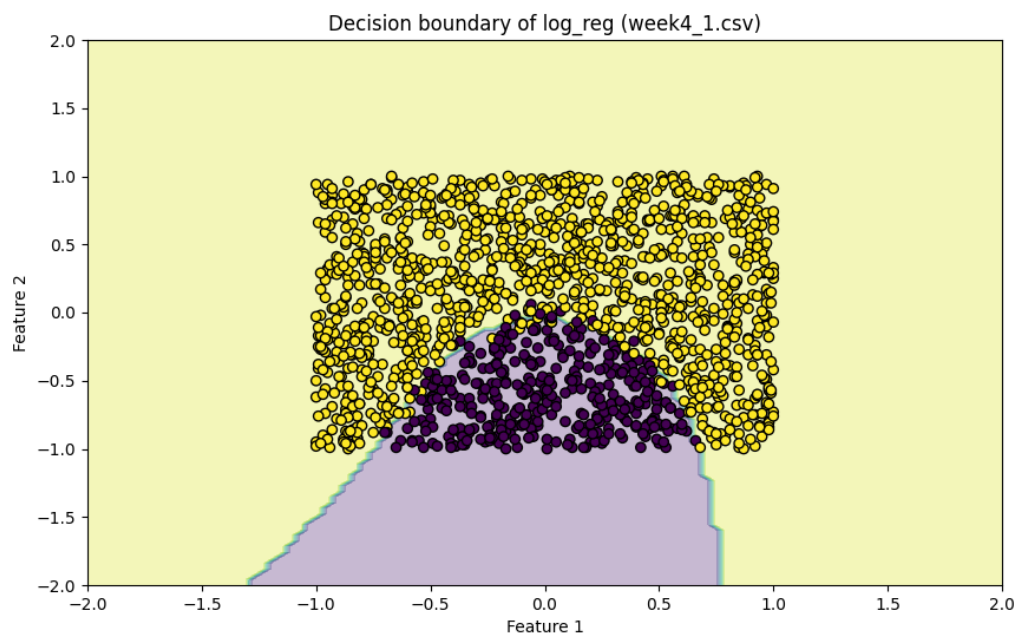
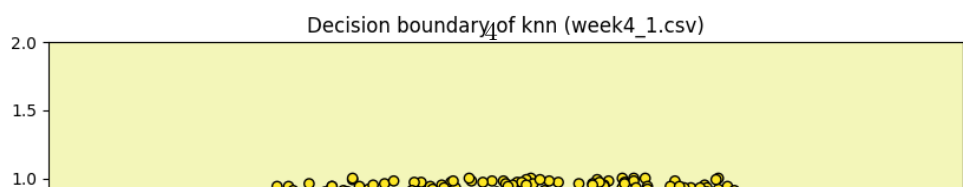
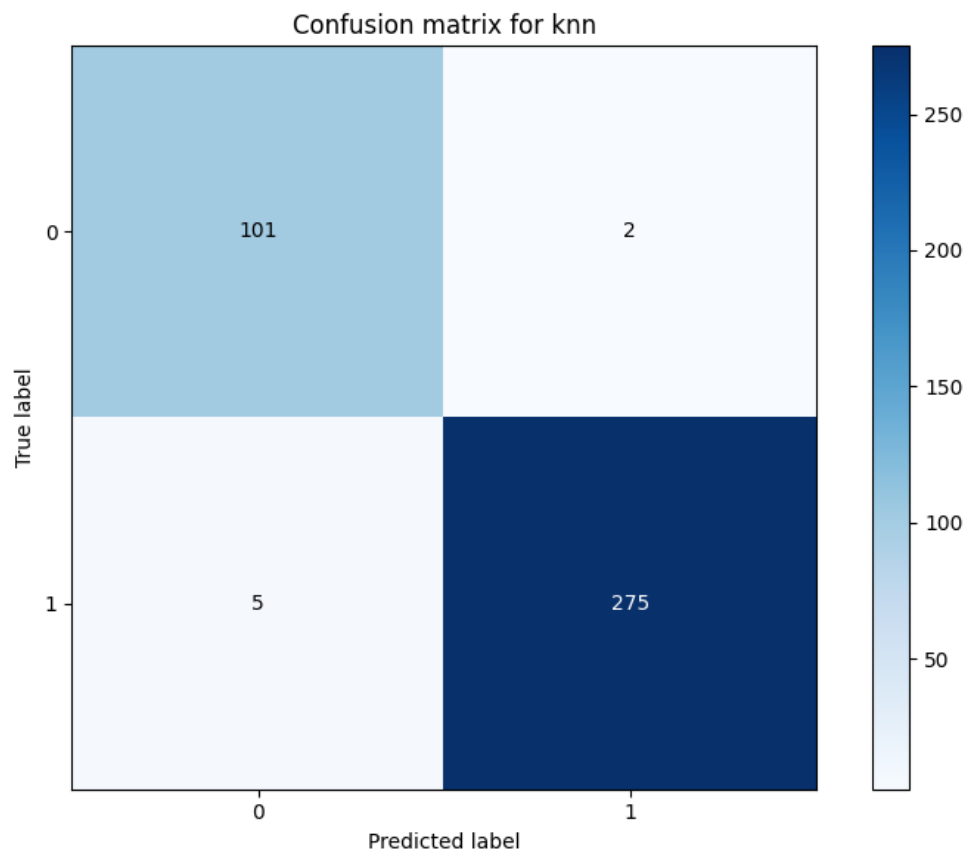


Figure 1: Figure.3

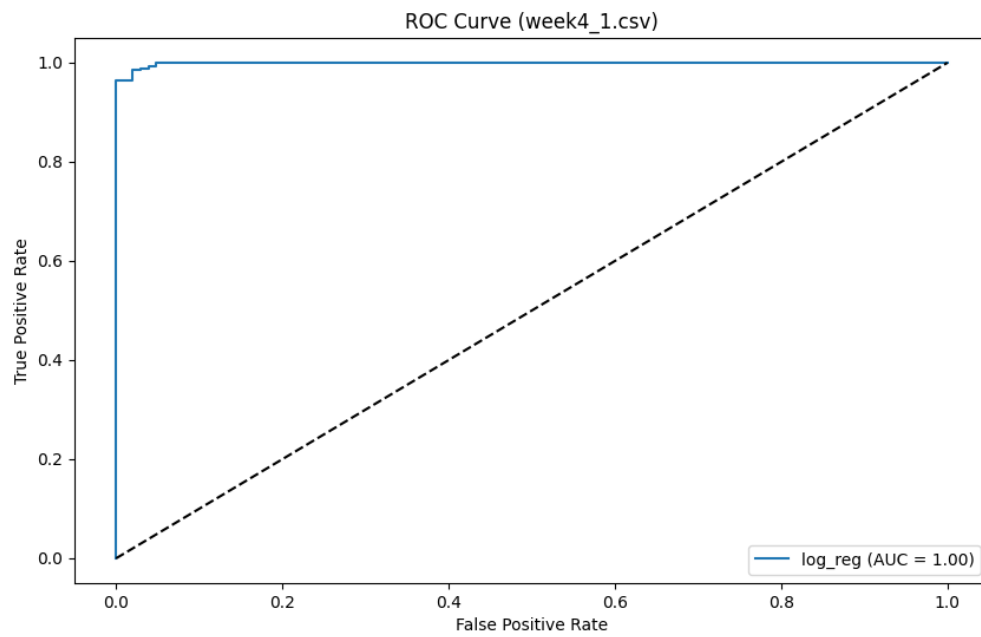
I(b)



I(c)

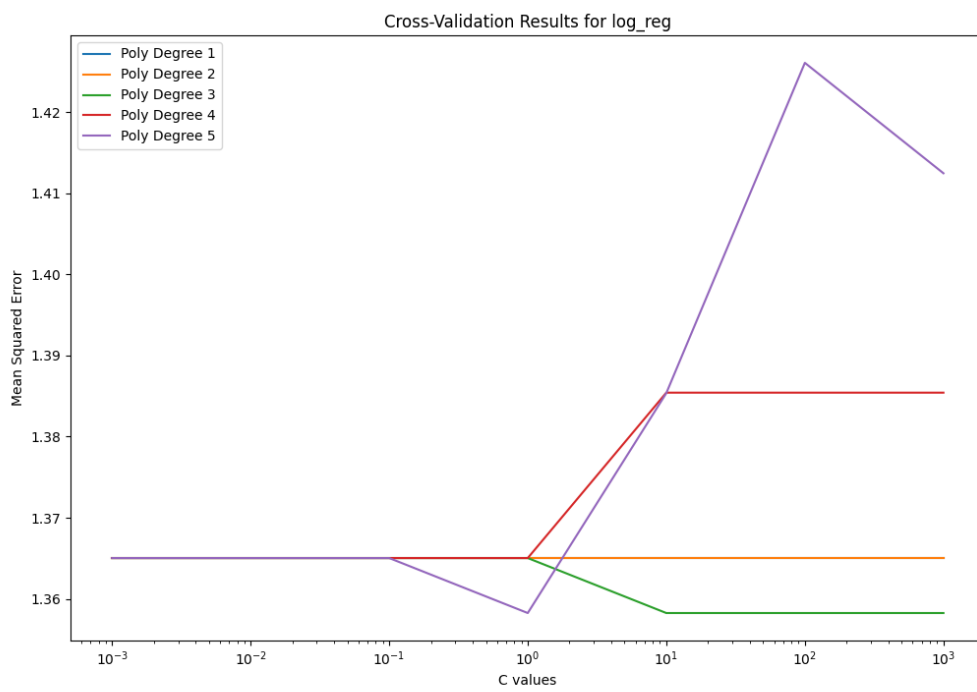
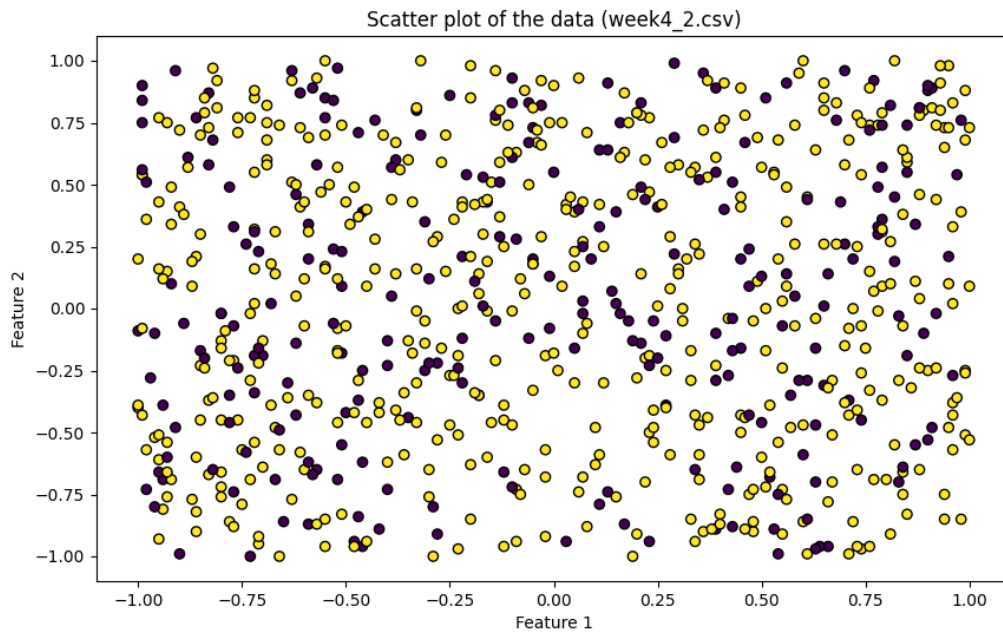


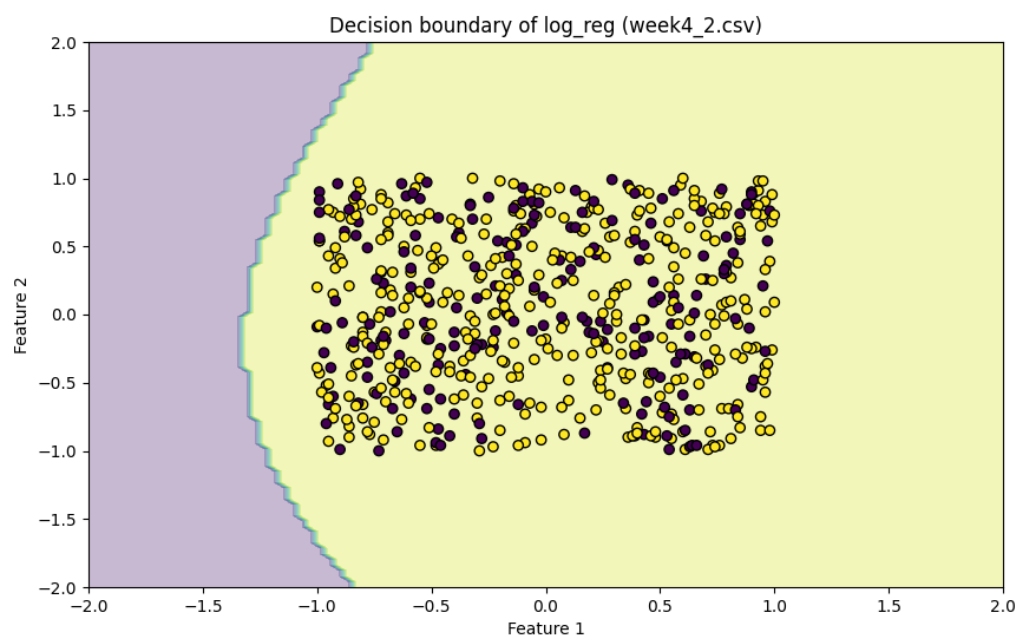
I(d)



I(e)

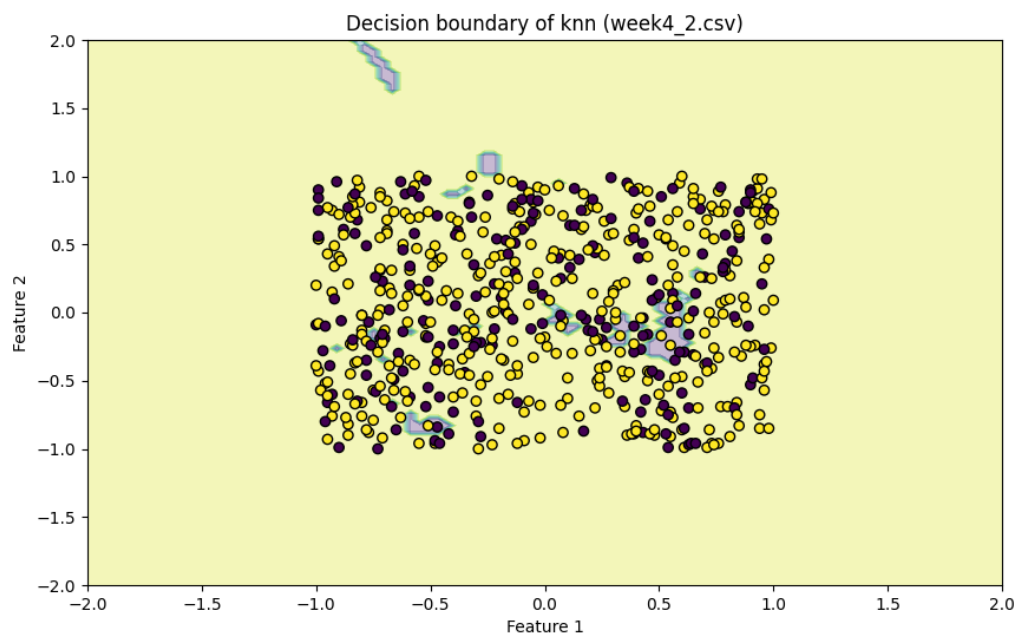
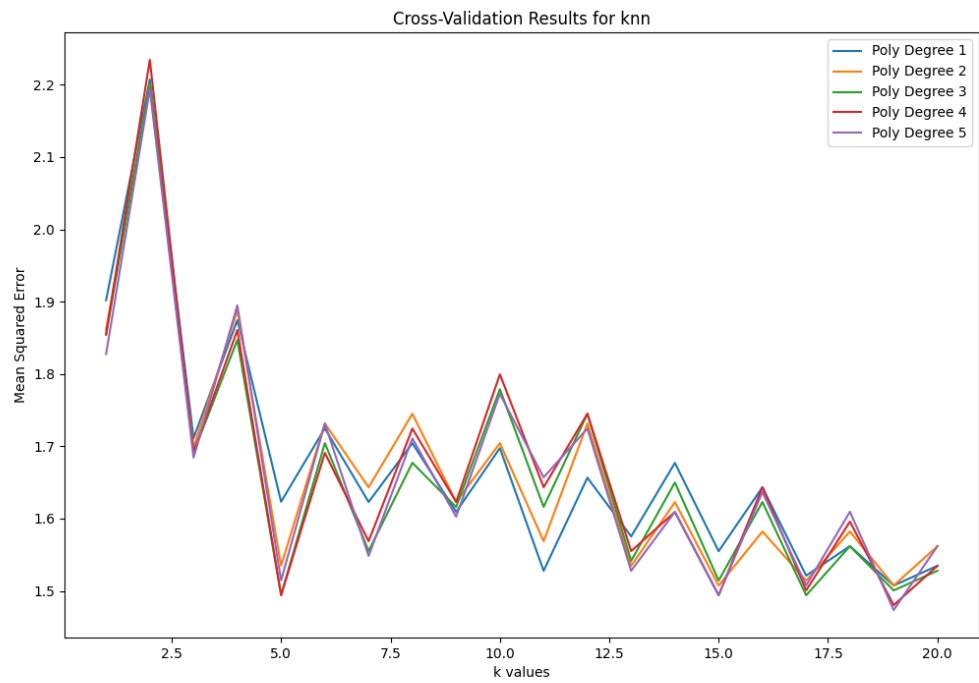
II(a)



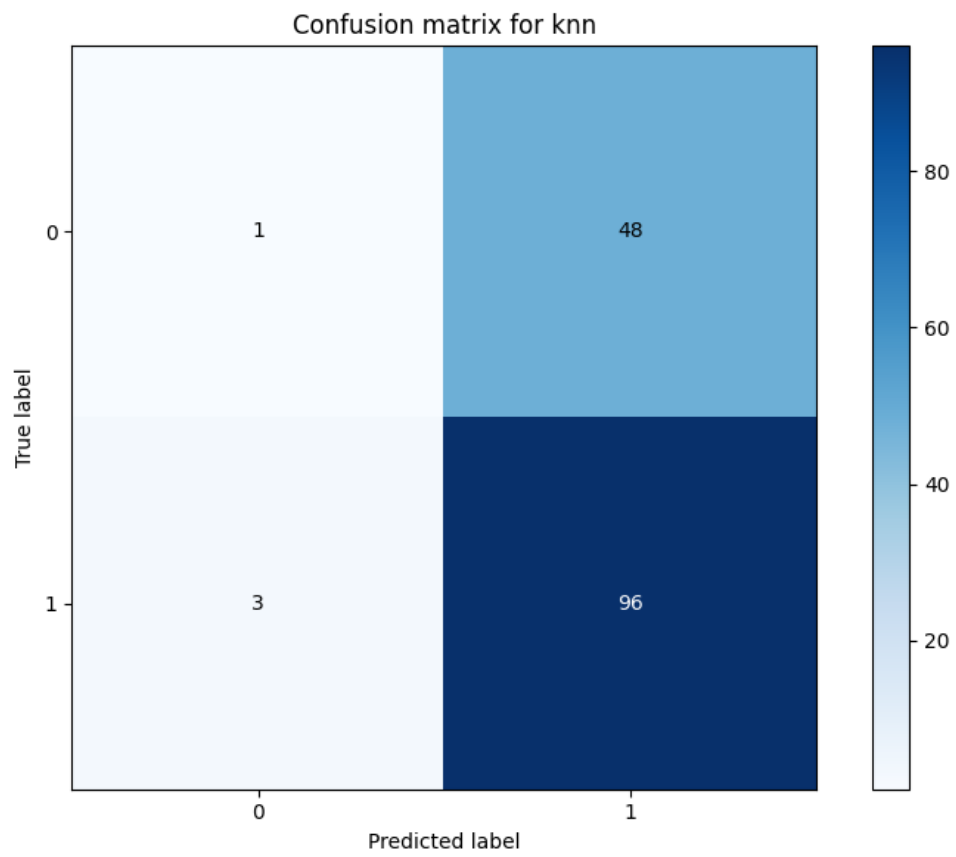




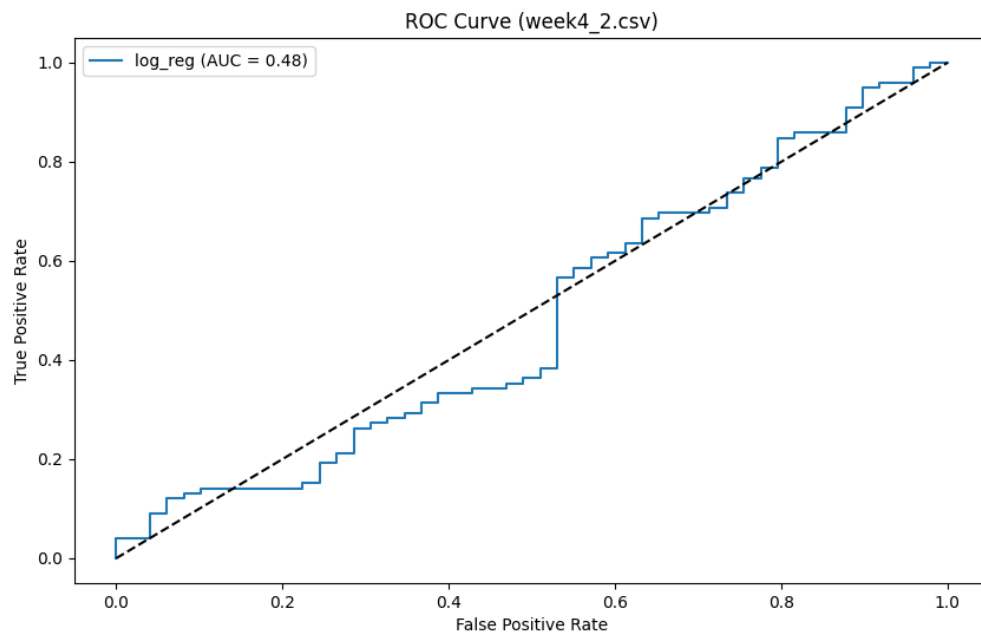
II(b)



II(c)



II(d)



II(e)

## 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_string)
    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, 21))
        C_values = np.logspace(-3, 3, 7)
        poly_degrees = list(range(1, 6))
```

```

cv_results = []

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_squared_error')
            mean_score = -scores.mean()
            cv_results.append((poly_degree, C_value, mean_score))

            if mean_score < best_score:
                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_squared_error')
            mean_score = -scores.mean()
            cv_results.append((poly_degree, k_value, mean_score))

            if mean_score < best_score:
                best_score = mean_score
                best_model = model
                best_params = {'poly_degree': poly_degree, 'n_neighbors': k_value}

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}')

plt.xlabel('C values' if self.model_name == 'log_reg' else 'k values')
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')
else:
    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)
        if self.model_name == 'log_reg':
            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()

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()
plt.savefig(f'Images/{self.i_string}(c).png')
plt.close()

print(f"Confusion matrix for {self.model_name}:\n{cm}")
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):
    if self.y_prob is not None:
        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')
        plt.savefig(f'Images/{self.i_string}(d).png')
        plt.close()

```

II(a)

The code is generic and runs for other dataset automatically

```

for index in range(1, 3):

```