# 📘 模擬程式題（Machine Learning Midterm – Programming Style）

## Q1. Logistic Regression (Gradient Descent)

(a) Implement logistic regression from scratch using gradient descent (no library implementation).  
(b) Plot the cost function (log-loss) over iterations.  
(c) Use the model to predict 5 test samples and explain results.

🔽 Sample Code:

import numpy as np  
import matplotlib.pyplot as plt  
  
# Sigmoid function  
def sigmoid(z):  
 return 1 / (1 + np.exp(-z))  
  
# Logistic loss  
def compute\_cost(X, y, theta):  
 m = len(y)  
 h = sigmoid(X @ theta)  
 return -(1/m) \* np.sum(y \* np.log(h + 1e-5) + (1 - y) \* np.log(1 - h + 1e-5))  
  
# Gradient descent  
def logistic\_regression(X, y, lr=0.1, epochs=1000):  
 m, n = X.shape  
 theta = np.zeros((n, 1))  
 costs = []  
  
 for \_ in range(epochs):  
 h = sigmoid(X @ theta)  
 gradient = (X.T @ (h - y)) / m  
 theta -= lr \* gradient  
 costs.append(compute\_cost(X, y, theta))  
 return theta, costs  
  
# Generate synthetic data  
X = np.array([[1, 2], [1, 3], [1, 5], [1, 6], [1, 8]])  
y = np.array([[0], [0], [1], [1], [1]])  
  
# Train  
theta, costs = logistic\_regression(X, y)  
  
# Plot cost  
plt.plot(costs)  
plt.title("Log-Loss over Iterations")  
plt.xlabel("Epoch")  
plt.ylabel("Loss")  
plt.show()  
  
# Predict 5 test points  
X\_test = np.array([[1, 2], [1, 4], [1, 5.5], [1, 7], [1, 9]])  
preds = sigmoid(X\_test @ theta)  
print("Predicted Probabilities:", preds.ravel())  
print("Predicted Classes:", (preds > 0.5).astype(int).ravel())

## Q2. K-Fold Cross Validation for Linear Regression

(a) Implement K-Fold Cross Validation (K=5) for linear regression.  
(b) For each fold, compute the MSE and return average MSE.  
(c) Use numpy (no sklearn).

🔽 Sample Code:

import numpy as np  
from sklearn.metrics import mean\_squared\_error  
  
def linear\_regression(X, y):  
 return np.linalg.pinv(X.T @ X) @ X.T @ y  
  
def k\_fold\_cv(X, y, k=5):  
 fold\_size = len(X) // k  
 mse\_list = []  
  
 for i in range(k):  
 start, end = i \* fold\_size, (i + 1) \* fold\_size  
 X\_val, y\_val = X[start:end], y[start:end]  
 X\_train = np.concatenate((X[:start], X[end:]), axis=0)  
 y\_train = np.concatenate((y[:start], y[end:]), axis=0)  
  
 theta = linear\_regression(X\_train, y\_train)  
 y\_pred = X\_val @ theta  
 mse = mean\_squared\_error(y\_val, y\_pred)  
 mse\_list.append(mse)  
   
 return np.mean(mse\_list)  
  
# Example Data  
X = np.hstack((np.ones((20, 1)), np.arange(20).reshape(-1, 1)))  
y = 2 + 3 \* X[:, 1:2] + np.random.randn(20, 1)  
  
avg\_mse = k\_fold\_cv(X, y, k=5)  
print("Average MSE:", avg\_mse)

## Q3. Confusion Matrix + Evaluation Metrics

(a) Write a function to compute TP, TN, FP, FN.  
(b) Calculate accuracy, precision, recall, F1-score.  
(c) Do not use sklearn.metrics.

🔽 Sample Code:

import numpy as np  
  
def compute\_metrics(y\_true, y\_pred):  
 y\_true, y\_pred = np.array(y\_true), np.array(y\_pred)  
 TP = np.sum((y\_true == 1) & (y\_pred == 1))  
 TN = np.sum((y\_true == 0) & (y\_pred == 0))  
 FP = np.sum((y\_true == 0) & (y\_pred == 1))  
 FN = np.sum((y\_true == 1) & (y\_pred == 0))  
  
 accuracy = (TP + TN) / len(y\_true)  
 precision = TP / (TP + FP + 1e-5)  
 recall = TP / (TP + FN + 1e-5)  
 f1 = 2 \* precision \* recall / (precision + recall + 1e-5)  
  
 return TP, TN, FP, FN, accuracy, precision, recall, f1  
  
# Example  
y\_true = [1, 0, 1, 1, 0, 0, 1]  
y\_pred = [1, 0, 0, 1, 0, 1, 1]  
  
results = compute\_metrics(y\_true, y\_pred)  
print("TP, TN, FP, FN:", results[:4])  
print("Accuracy, Precision, Recall, F1:", results[4:])