Lab 3

24/3/25

Linear Regression

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

def estimate_coef(x, y):
    n = np.size(x)
    m_x = np.mean(x)
    m_y = np.mean(y)
    SS_xy = np.sum((x - m_x) * (y - m_y))
    SS_xx = np.sum((x - m_x)**2)
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1 * m_x
    return (b_0, b_1)

def plot_regression_line(x, y, b):
    plt.scatter(x, y, color="m", marker="o", s=30)
    y_pred = b[0] + b[1] * x
    plt.plot(x, y_pred, color="g")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title("Linear Regression")
    plt.show()

file_path = input("/content/marketing.csv")
df = pd.read_csv(file_path)

x = df.iloc[:, 0].values
y = df.iloc[:, 1].values

b = estimate_coef(x, y)
print(f"estimated coefficients:\nb_0 = {b[0]}
      \nb_1 = {b[1]}")
plot_regression_line(x, y, b)
```

Output
b_0 = 7.03
b_1 = 0.04

Multiple Linear Regression

```
data = {
    "Feature 1": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    "Feature 2": [2, 3, 5, 7, 11, 13, 17, 19, 23, 25],
    "Feature 3": [3, 6, 9, 12, 15, 18, 21, 24, 27, 30],
    "Target": [5, 9, 15, 22, 31, 41, 53, 66, 80, 96]
}

df = pd.DataFrame(data)

X = df.drop(columns = ["Target"]).values
y = df["Target"].values.reshape(-1, 1)

X = np.hstack((np.ones((X.shape[0], 1)), X))

beta = np.linalg.solve(X.T @ X + 0.01 * np.identity(
        X.shape[1]), X.T @ y)

y_pred = X @ beta

mse = np.mean((y - y_pred)**2)
total_variance = np.sum((y - np.mean(y))**2)
explained_variance = np.sum((y_pred -
    np.mean(y))**2)
r2 = explained_variance / total_variance

print("Model coefficients:", beta[1:].flatten())
print("Intercept:", beta[0][0])
```

print("Mean Squared Error:", mse)
print("R-squared Score:", r2)

Output

Model coefficients: [0.04 3.31 0.12]
Intercept: -3.15
Mean Squared Error: 5.36
R-squared Error: 0.99.

Logistic Regression.

```
def sigmoid(z):
    return 1/(1 + np.exp(-z))

def compute_cost(X, y, theta):
    m = len(y)
    h = sigmoid(X @ theta)
    cost = (-1/m) * np.sum(y * np.log(h) +
                (1 - y) * np.log(1 - h))
    return cost

def gradient_descent(X, y, theta, alpha,
    iterations):
    m = len(y)
    cost_history = []

    for _ in range(iterations):
        gradient = (1/m) * X.T @ (sigmoid
                (X @ theta) - y)
        theta -= alpha * gradient
        costs_history.append(compute_cost(X, y, theta)

    return theta, cost_history
```

```
def predict(X, theta):
    return sigmoid(X @ theta) >= 0.5
    .astype(int)

np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = (X > 5).astype(int).ravel()

X_b = np.c_[np.ones((X.shape[0], 1)), X]

theta = np.zeros(X_b.shape[1])
alpha = 0.1
iterations = 1000

theta, cost_history = gradient_descent(X_b,
                      y, theta, alpha, iterations)
y_pred = predict(X_b, theta)
```

Output

Accuracy: 0.97

[0, 0, 1, 0, 0, 1, 1]

Lab 8B
24/3/25