CEE 554 HW1 — Q1: Perceptron Learning Algorithm (PLA) on Bridge Condition Data

Jack

Problem 1

(a)

$$W_{\text{LR}} = [-2.06799143,\ 0.17494643,\ 0.20105202]$$
$$E_{\text{in}} = 0.0$$

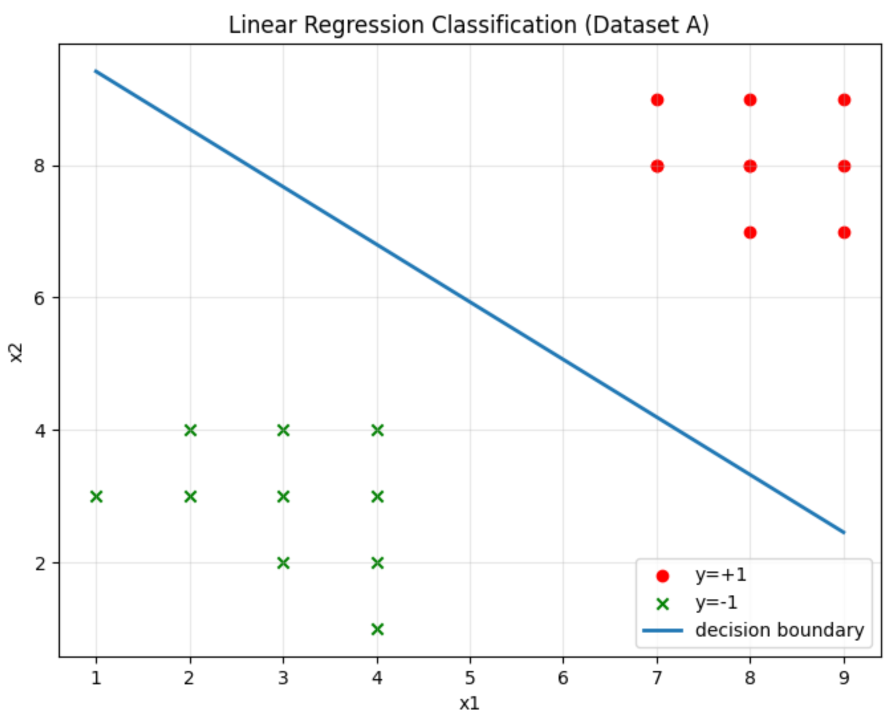

Figure 1: Linear Regression classification result on Dataset A.

(b)

- **(i) Initialize PLA with linear regression weight**

$$\begin{aligned} W_{\text{init}} &= [-2.06799143,\ 0.17494643,\ 0.20105202] \\ \text{PLA updates to converge} &= 0 \\ W_{\text{final}} &= [-2.06799143,\ 0.17494643,\ 0.20105202] \end{aligned}$$

- **(ii) Initialize PLA with random weight (seed = 0)**

$$\begin{aligned} W_{\text{init}} &= [0.12573022,\ -0.13210486,\ 0.64042265] \\ \text{PLA updates to converge} &= 75 \\ W_{\text{final}} &= [-26.87426978,\ 6.86789514,\ -1.35957735] \end{aligned}$$

Problem 2

(a)

$$\text{Average } E_{\text{in}} \text{ over 1000 runs} = 0.503509$$

(b)

$$\begin{aligned} \text{Average } E_{\text{in}} \text{ over 1000 runs} &= 0.118859 \\ \text{Average } W \text{ over 1000 runs} &= \begin{bmatrix} -1.00257846 \\ 1.50050673 \times 10^{-3} \\ -1.86884727 \times 10^{-3} \\ 5.18760347 \times 10^{-4} \\ 1.57664572 \\ 1.57570983 \end{bmatrix} \end{aligned}$$

Closest hypothesis: (i).

Problem 3

(a)

$$\begin{aligned} \mathbb{E}[v_1] &\approx 0.50007 \\ \mathbb{E}[v_{\text{rand}}] &\approx 0.500048 \\ \mathbb{E}[v_{\text{min}}] &\approx 0.037377 \end{aligned}$$

Closest value: (ii) 0.01.

Note. With 1000 coins, it is very likely that at least one coin gets 0 heads in 10 flips, so many runs have $v_{\text{min}} = 0$. When no coin gets 0 heads, the minimum is usually 0.1. Averaging these cases gives about 0.04, not 0.

(b)

c1 is the first coin; crand is a randomly chosen coin; cmin is the coin with the smallest fraction of heads among the 1,000 coins, which is selected based on the data, so it does not satisfy the Hoeffding setting.

```
\section*{appendix(a)}

\subsection*{prompt 1}
import numpy as np
import matplotlib.pyplot as plt

def linear_regression(x, y):
    N = x.shape[0]
    xb = np.c_[np.ones(N), x]
    w = np.linalg.pinv(xb) @ y
    y_pred = np.where((xb @ w) >= 0, 1, -1)
    Ein = np.mean(y_pred != y)
    return w, Ein

def main():
    PATH = "/Users/jack/Desktop/CEE554/HW1/Bridge_Condition.txt"
    data = np.loadtxt(PATH)
    dataset1 = data[0:20, :]
    x1 = dataset1[:, 0:2]
    y1 = dataset1[:, 2]
    w, Ein = linear_regression(x1, y1)
    print(f"Linear Regression Weights: {w}, In-sample Error: {Ein}")

    if __name__ == "__main__":
        main()

I have write the linear regression classification , you give me the plot and visuali

\subsection*{prompt 2}
Randomly select x sample indices from 0-N
\subsection*{prompt 3}
Returns the index of the minimum value in v.


\section*{appendix(b)}
import numpy as np
import matplotlib.pyplot as plt

def linear_regression(x, y):
    N = x.shape[0]
    xb = np.c_[np.ones(N), x]
    w = np.linalg.pinv(xb) @ y
    y_pred = np.where((xb @ w) >= 0, 1, -1)
```

```
    Ein = np.mean(y_pred != y)
    return w, Ein

def plot_scatter_with_boundary(x, y, w, title="Linear Regression Classification (D
    fig, ax = plt.subplots(figsize=(8, 6))
    ax.scatter(x[y == 1, 0], x[y == 1, 1], c="red", marker="o", label="y=+1")
    ax.scatter(x[y == -1, 0], x[y == -1, 1], c="green", marker="x", label="y=-1")

    w0, w1, w2 = w
    xs = np.linspace(x[:, 0].min(), x[:, 0].max(), 200)

    if abs(w2) > 1e-12:
        ys = -(w0 + w1 * xs) / w2
        ax.plot(xs, ys, linewidth=2, label="decision boundary")
    elif abs(w1) > 1e-12:
        ax.axvline(-w0 / w1, linewidth=2, label="decision boundary")
    else:
        ax.text(0.5, 0.5, "Degenerate boundary (w1=0 and w2=0)", transform=ax.trans
                ha="center", va="center")

    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_title(title)
    ax.grid(alpha=0.3)
    ax.legend()
    plt.show()
def main():
    PATH = "/Users/jack/Desktop/CEE554/HW1/Bridge_Condition.txt"
    data = np.loadtxt(PATH)
    dataset1 = data[0:20, :]
    x1 = dataset1[:, 0:2]
    y1 = dataset1[:, 2]
    w, Ein = linear_regression(x1, y1)
    print(f"Linear Regression Weights: {w}, In-sample Error: {Ein}")

    plot_scatter_with_boundary(x1, y1, w)

if __name__ == "__main__":
        main()
```

```
import numpy as np
```

```
from pathlib import Path

def pla_with_init(x, y, w_init, max_iter=1000):
    N = x.shape[0]
    xb = np.c_[np.ones(N), x]
    w = np.array(w_init, dtype=float).copy()
    updates = 0
    for _ in range(max_iter):
        changed = False
        for i in range(N):
            if y[i] * (w @ xb[i]) <= 0:
                w += y[i] * xb[i]
                updates += 1
                changed = True
        if not changed:
            break
    return w, updates

def main():
    path = Path("/Users/jack/Desktop/CEE554/HW1/Bridge_Condition.txt")
    data = np.loadtxt(path)
    datasetA = data[:20, :]
    x1 = datasetA[:, 0:2]
    y1 = datasetA[:, 2].astype(int)

    w_lr = np.array([-2.06799143, 0.17494643, 0.20105202], dtype=float)
    w_final_lr, updates_lr = pla_with_init(x1, y1, w_lr)

    rng = np.random.default_rng(0)
    w_rand = rng.standard_normal(3)
    w_final_rand, updates_rand = pla_with_init(x1, y1, w_rand)
    print("Init = Linear Regression:", w_lr)
    print("PLA updates to converge:", updates_lr)
    print("Final w:", w_final_lr)
    print()
    print("Init = Random (seed=0):", w_rand)
    print("PLA updates to converge:", updates_rand)
    print("Final w:", w_final_rand)

if __name__ == "__main__":
    main()
```

```python
import numpy as np

def run_once(N=1000, noise_rate=0.1, rng=None):
    if rng is None:
        rng = np.random.default_rng()

    X = rng.uniform(-1.0, 1.0, size=(N, 2))
    x1, x2 = X[:, 0], X[:, 1]

    y = sign(x1**2 + x2**2 - 0.6)

    n_flip = int(noise_rate * N)
    flip_idx = rng.choice(N, size=n_flip)
    y[flip_idx] *= -1

    xb = np.c_[np.ones(N), x1, x2]
    w = np.linalg.pinv(xb) @ y
    y_pred = sign(xb @ w)
    Ein = np.mean(y_pred != y)

    return w, Ein

def sign(z):
    return np.where(z >= 0, 1, -1)

def experiment(n_runs=1000, N=1000, noise_rate=0.1, seed=0):
    rng = np.random.default_rng(seed)
    Eins = np.empty(n_runs)

    for r in range(n_runs):
        w, Ein = run_once(N=N, noise_rate=noise_rate, rng=rng)
        Eins[r] = Ein

    return Eins.mean()

"""
# Different each run (good): one RNG, reused inside the loop
rng = np.random.default_rng(0)
for r in range(n_runs):
    x = rng.uniform()
    idx = rng.choice(10)

# Same every run (bad): RNG is re-created each loop iteration
for r in range(n_runs):
    rng = np.random.default_rng(0)
    x = rng.uniform()
"""
if __name__ == "__main__":
```

```
    avg_Ein = experiment(n_runs=1000, N=1000, noise_rate=0.1, seed=0)
    print("Average Ein over 1000 runs =", avg_Ein)
```

```
import numpy as np


def run_once(N=1000, noise_rate=0.1, rng=None):
    if rng is None:
        rng = np.random.default_rng()

    X = rng.uniform(-1.0, 1.0, size=(N, 2))
    x1, x2 = X[:, 0], X[:, 1]

    y = sign(x1**2 + x2**2 - 0.6)

    n_flip = int(noise_rate * N)
    flip_idx = rng.choice(N, size=n_flip)
    y[flip_idx] *= -1

    z = np.c_[np.ones(N), x1, x2, x1*x2, x1**2, x2**2]
    w = np.linalg.pinv(z) @ y

    y_pred = sign(z @ w)
    Ein = np.mean(y_pred != y)

    return w, Ein


def sign(z):
    return np.where(z >= 0, 1, -1)

def experiment(n_runs=1000, N=1000, noise_rate=0.1, seed=0):
    rng = np.random.default_rng(seed)
    Eins = np.empty(n_runs)
    Ws = np.empty((n_runs, 6))

    for r in range(n_runs):
        w, Ein = run_once(N=N, noise_rate=noise_rate, rng=rng)
        Eins[r] = Ein
        Ws[r] = w
```

```
    return Eins.mean(), Ws.mean(axis=0)

if __name__ == "__main__":
    avg_Ein, avg_w = experiment(n_runs=1000, N=1000, noise_rate=0.1, seed=3)
    print("Average Ein over 1000 runs =", avg_Ein)
    print("Average W over 1000 runs =", avg_w)
```

```
import numpy as np

def experiment(n_runs=100000, n_coins=1000, n_flips=10):
    rng = np.random.default_rng()

    v1 = np.empty(n_runs)
    vrand = np.empty(n_runs)
    vmin = np.empty(n_runs)

    for r in range(n_runs):
        flips = rng.integers(0, 2, size=(n_coins, n_flips))
        heads = flips.sum(axis=1)
        v = heads / n_flips

        v1[r] = v[0]

        crand = rng.integers(0, n_coins)
        vrand[r] = v[crand]

        cmin = int(np.argmin(v))
        vmin[r] = v[cmin]

    return v1.mean(), vrand.mean(), vmin.mean()

if __name__ == "__main__":
    mean_v1, mean_vrand, mean_vmin = experiment(n_runs=100000, n_coins=1000, n_flip
    print("Average v1    =", mean_v1)
    print("Average vrand =", mean_vrand)
    print("Average vmin  =", mean_vmin)
```