Homework 1

PENG Qiheng

Student ID 225040065

September 27, 2025

Problem 1.

(a) In supervised learning, we have a dataset of input data and labels, and the goal is to learn a mapping from inputs to labels.

In unsupervised learning, we only have input data and the goal is to discover the hidden patterns or structures in the data.

(b) Answer: 2

1. Regression is used to fit continuous labels.

3. For a given training dataset, perceptrons with different initialization may converge to different linear classifiers.

4. Least squares is a maximum likelihood estimator only if error follows a Gaussian Distribution.

(c) 1. Since we have

$$r(\boldsymbol{X}^T\boldsymbol{X}) = r(\boldsymbol{X}) = d \tag{1}$$

$$\boldsymbol{X}^T\boldsymbol{X} \in \mathbb{R}^{d\times d} \tag{2}$$

thus, $\boldsymbol{X}^T\boldsymbol{X}$ is invertible

2. As

$$(\boldsymbol{X}^T\boldsymbol{X})^T = \boldsymbol{X}^T\boldsymbol{X} \tag{3}$$

we know that $\boldsymbol{X}^T\boldsymbol{X}$ is Symmetric Matrices

3. Thus, we can conclude that $\boldsymbol{X}^T\boldsymbol{X}$ is Positive Definite

Problem 2.

(a) Let $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{\Sigma}_1\boldsymbol{U}_1^T$, $\boldsymbol{z} = \boldsymbol{U}_1^T\boldsymbol{\theta}$ and $\boldsymbol{A} := \boldsymbol{V}\boldsymbol{\Sigma}_1$ we have

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d} ||\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}||_2^2 = \min_{\boldsymbol{\theta}\in\mathbb{R}^d} ||\boldsymbol{V}\boldsymbol{\Sigma}_1\boldsymbol{U}_1^T\boldsymbol{\theta} - \boldsymbol{y}||_2^2 = \min_{\boldsymbol{z}\in\mathbb{R}^n} ||\boldsymbol{A}\boldsymbol{z} - \boldsymbol{y}||_2^2 \tag{4}$$

Since $\boldsymbol{A}$ is of full rank, we have optimal $\boldsymbol{z}$

$$\boldsymbol{z}^* = \boldsymbol{A}^{-1}\boldsymbol{y} \tag{5}$$

And we have

$$\boldsymbol{U}_1^T\boldsymbol{\theta}^* = \boldsymbol{A}^{-1}\boldsymbol{y} \tag{6}$$

$$\boldsymbol{U}_1^T\boldsymbol{\theta}^* - \boldsymbol{A}^{-1}\boldsymbol{y} = \boldsymbol{0} \tag{7}$$

Considering that $\boldsymbol{U}_1^T\boldsymbol{U}_2 = \boldsymbol{0}$

$$\boldsymbol{U}_1^T\boldsymbol{\theta}^* - \boldsymbol{A}^{-1}\boldsymbol{y} = \boldsymbol{U}_1^T\boldsymbol{U}_2 \tag{8}$$

$$\boldsymbol{\theta}^* = \boldsymbol{U}_1(\boldsymbol{V}\boldsymbol{\Sigma}_1)^{-1}\boldsymbol{y} + \boldsymbol{U}_2\boldsymbol{c}, \boldsymbol{c} \in \mathbb{R}^{d-n} \tag{9}$$

And the optimal function value is 0, if $\boldsymbol{c} = \boldsymbol{0}$

(b) Let

$$\mathcal{L}(\boldsymbol{\theta}) = ||\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}||_2^2 + \lambda||\boldsymbol{\theta}||_2^2 \tag{10}$$

Take the gradient

$$\nabla\mathcal{L}(\boldsymbol{\theta}) = 2\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}) + 2\lambda\boldsymbol{\theta} \tag{11}$$

$$(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})\boldsymbol{\theta}^* = \boldsymbol{X}^T\boldsymbol{y} \tag{12}$$

Since $\boldsymbol{X}^T\boldsymbol{X}$ is semi-positive definite and $\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}$ is positive definite. Thus, we have

$$\boldsymbol{\theta}^* = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{13}$$

Problem 3.

(a) As the assumption, we have

$$\epsilon_i = y_i - \boldsymbol{X}_i\boldsymbol{\theta} \tag{14}$$

$$P(D|\boldsymbol{\theta}) = \Pi_{i=1}^{n}(\frac{1}{2b}e^{-\frac{|y_i-\boldsymbol{X}_i\boldsymbol{\theta}|}{b}}) = 2b^{-n}e^{-\frac{|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\theta}|}{b}} \tag{15}$$

Let $\mathcal{L}(\boldsymbol{\theta}) = \log P(D|\boldsymbol{\theta})$, we have

$$\mathcal{L}(\boldsymbol{\theta}) = -n\log(2b) - \frac{1}{b}|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\theta}| \tag{16}$$

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} |\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\theta}| \tag{17}$$

(b) According to the definition of $h_\mu(z)$, we have

$$\frac{\partial h_\mu(z)}{\partial z} = \begin{cases} sign(z), & |z| \geq \mu \\ \frac{z}{\mu}, & |z| \leq \mu \end{cases} \tag{18}$$

And we can rewrite $h'_\mu(z)$ as

$$h'_\mu(z) = \frac{z}{\max(|z|,\mu)} \tag{19}$$

Thus, we have

$$H'_\mu(\boldsymbol{z}) = \frac{\partial \Sigma_{j=1}^{n} h_\mu(z_j)}{\partial \boldsymbol{z}} = \begin{bmatrix} h'_\mu(z_1) \\ \vdots \\ h'_\mu(z_n) \end{bmatrix} = h'_\mu(\boldsymbol{z}) \tag{20}$$

Finally, we have

$$\nabla\mathcal{L}(\boldsymbol{\theta}) = \frac{\partial(\boldsymbol{X}\boldsymbol{\theta}-\boldsymbol{y})}{\partial\boldsymbol{\theta}}\frac{\partial H_\mu(\boldsymbol{z})}{\partial\boldsymbol{z}} = \boldsymbol{X}^T h'_\mu(\boldsymbol{X}\boldsymbol{\theta}-\boldsymbol{y}) \tag{21}$$

(c)

After calculating, output $||\hat{\theta}_{LS} - \boldsymbol{\theta}^*|| = 59.5136$

code as follows:

```python
import numpy as np
import matplotlib.pyplot as plt
d = 50  # feature dimension
X = np.load("data/X.npy")
y = np.load("data/y.npy")
print("data shape: ", X.shape, y.shape)
theta_star = np.load("data/theta_star.npy")
###### part (1): least square estimator ########
theta_hat = np.linalg.inv(X.T @ X) @ X.T @ y
Error_LS = np.linalg.norm(theta_hat - theta_star, 2)
print("Estimator approximated by LS:", Error_LS)
###### part (2): L1 estimator ########
mu = 1e-5  # smoothing parameter
alpha = 0.001  # stepsize
T = 1000  # iteration number
# random initialization
theta = np.random.randn(d, 1)
Error_huber = []
for _ in range(1, T):
    # calculate the l2 error of the current iteration
    Error_huber.append(np.linalg.norm(theta - theta_star, 2))
    # calculate gradient
    z = X @ theta - y
    h_prime = z / np.maximum(np.abs(z), mu)
    grad = X.T @ h_prime
    # gradient descent update
    theta = theta - alpha * grad
#######    plot the figure    #########
plt.figure(figsize=(10, 5))
plt.yscale("log", base=2)
plt.plot(Error_huber, "r-")
plt.title(r"$\ell_1$ estimator approximated by Huber")
plt.ylabel(r"$\theta$")  # set the label for the y axis
plt.xlabel("Iteration")  # set the label for the x axis
plt.show()
```

The Huber smoothing error is shown in Figure 1.

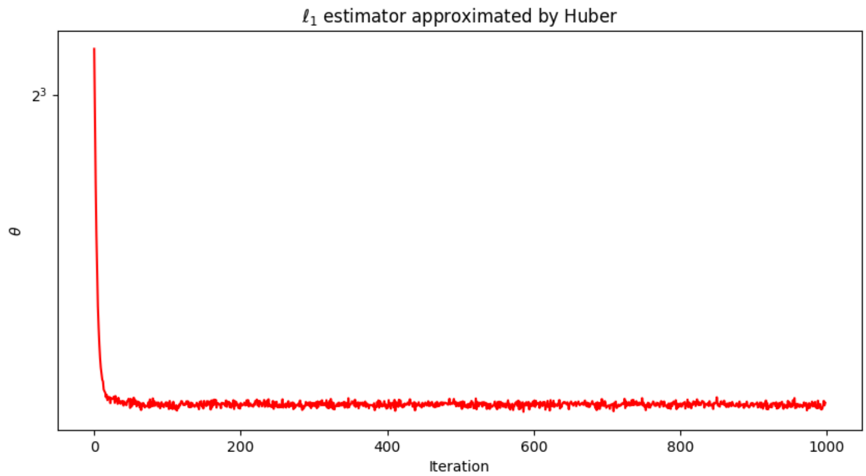

Figure 1: Huber smoothing error

Problem 4.

(a) Since θ^* is a classifier that correctly separates all the data points, we can get

$$y_i(\boldsymbol{\theta^*}^T \boldsymbol{x}_i) > 0, \forall i = 1, \ldots, n$$

We can easily have

$$\rho \geq y_i(\boldsymbol{\theta^*}^T \boldsymbol{x}_i) > 0$$

(b) According to update (1), we have

$$\boldsymbol{\theta}_k^T \theta^* = \boldsymbol{\theta}_{k-1}^T \theta^* + y_{k-1} \boldsymbol{x}_{k-1}^T \boldsymbol{\theta}^* \geq \boldsymbol{\theta}_{k-1}^T \theta^* + \rho$$

After recursion, we have

$$\boldsymbol{\theta}_k^T \theta^* \geq k\rho$$

(c) According to update (1), we have

$$\begin{aligned} ||\boldsymbol{\theta}_k||^2 &= ||\theta_{k-1} + y_{k-1} \boldsymbol{x}_{k-1}^T||^2 \\ &= ||\boldsymbol{\theta}_{k-1}||^2 + ||\boldsymbol{x}_{k-1}||^2 + 2y_{k-1} \boldsymbol{\theta}_{k-1}^T \boldsymbol{x}_{k-1} \end{aligned}$$

Since $\boldsymbol{x}_{k-1}$ is misclassified by $\boldsymbol{\theta}_{k-1}$, we have $y_{k-1}\boldsymbol{\theta}_{k-1}^T\boldsymbol{x}_{k-1} \leq 0$

$$||\boldsymbol{\theta}_k||^2 \leq ||\boldsymbol{\theta}_{k-1}||^2 + ||\boldsymbol{x}_{k-1}||^2$$

(d) According to the problem, we have $R \geq ||\boldsymbol{x}_i||, \forall i = 1, \ldots, n$ And we have

$$||\boldsymbol{\theta}_k||^2 \leq ||\boldsymbol{\theta}_{k-1}||^2 + ||x_{k-1}||^2 \leq ||\boldsymbol{\theta}_{k-1}||^2 + R^2$$

After recursion, we have

$$||\boldsymbol{\theta}_k||^2 \leq kR^2$$

(e) According to steps 2 and 4, we have $\boldsymbol{\theta}_k^T\boldsymbol{\theta}^* \geq k\rho$ and $||\boldsymbol{\theta}_k||^2 \leq kR^2$ we can get

$$\frac{\boldsymbol{\theta}_k^T\boldsymbol{\theta}^*}{||\boldsymbol{\theta}_k||} \geq \sqrt{k}\frac{\rho}{R}$$

By using Cauchy-Schwarz inequality, we have

$$k^2\rho^2 \leq ||\boldsymbol{\theta}_k||^2||\boldsymbol{\theta}^*||^2 \leq kR^2||\theta^*||^2$$

Thus, we have

$$\bar{k} \leq \frac{R^2||\theta^*||^2}{\rho^2}$$

Problem 5. Main code as follows:

```
# train
iters = 2000
d = 2
num_sample = X.shape[0]
threshold = 1e-4
theta = np.zeros((d + 1, 1))

X = np.hstack([X, np.ones((num_sample, 1))])
X_test = np.hstack([X_test, np.ones((X_test.shape[0], 1))])
Er_in_perceptron = []
Er_out_perceptron = []
Er_in_pocket = []
Er_out_pocket = []
pocket = theta.copy()
best_error = cal_error(theta, X, y)

for iterate in range(iters):
    for i in random.sample(range(num_sample), num_sample):
        if np.sign(X[i] @ theta)[0] != y[i]:
            theta += (y[i] * X[i]).reshape(-1, 1)
            break

    cur_error = cal_error(theta, X, y)
    if cur_error < best_error:
        best_error = cur_error
        pocket = theta.copy()

    Er_in_perceptron.append(cur_error)
    Er_out_perceptron.append(cal_error(theta, X_test, y_test))
    Er_in_pocket.append(best_error)
    Er_out_pocket.append(cal_error(pocket, X_test, y_test))

# print(f"theta (perceptron): {theta}")
# print(f"pocket (pocket): {pocket}")

# plot Er_in and Er_out
fig, axs = plt.subplots(1, 2, figsize=(14, 6))

# perceptron
axs[0].plot(Er_in_perceptron, label="Perceptron Train Error")
```

```
    axs[0].plot(Er_in_pocket, label="Pocket Train Error")
    axs[0].set_xlabel("Iteration")
    axs[0].set_ylabel("Error Rate")
    axs[0].set_title("Training Error")
    axs[0].legend()
    # pocket
    axs[1].plot(Er_out_perceptron, label="Perceptron Test Error")
    axs[1].plot(Er_out_pocket, label="Pocket Test Error")
    axs[1].set_xlabel("Iteration")
    axs[1].set_ylabel("Error Rate")
    axs[1].set_title("Test Error")
    axs[1].legend()

    plt.tight_layout()
    plt.show()

    # plot decision boundary
    fig, axs = plt.subplots(1, 2, figsize=(14, 6))

    ax0 = plot_feature(X, y, plot_num=500, ax=axs[0], classes=np.unique(y))
    x1 = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
    x2_perceptron = -(theta[0] * x1 + theta[2]) / theta[1]
    x2_pocket = -(pocket[0] * x1 + pocket[2]) / pocket[1]
    ax0.plot(x1, x2_perceptron, "r--", label="Perceptron Boundary")
    ax0.plot(x1, x2_pocket, "b-", label="Pocket Boundary")
    ax0.set_title("Decision Boundaries (Train)")
    ax0.legend()

    ax1 = plot_feature(X_test, y_test, plot_num=500, ax=axs[1], classes=np.unique(y_test))
    x1_test = np.linspace(X_test[:, 0].min(), X_test[:, 0].max(), 100)
    x2_perceptron_test = -(theta[0] * x1_test + theta[2]) / theta[1]
    x2_pocket_test = -(pocket[0] * x1_test + pocket[2]) / pocket[1]
    ax1.plot(x1_test, x2_perceptron_test, "r--", label="Perceptron Boundary")
    ax1.plot(x1_test, x2_pocket_test, "b-", label="Pocket Boundary")
    ax1.set_title("Decision Boundaries (Test)")
    ax1.legend()

    plt.tight_layout()
    plt.show()
```

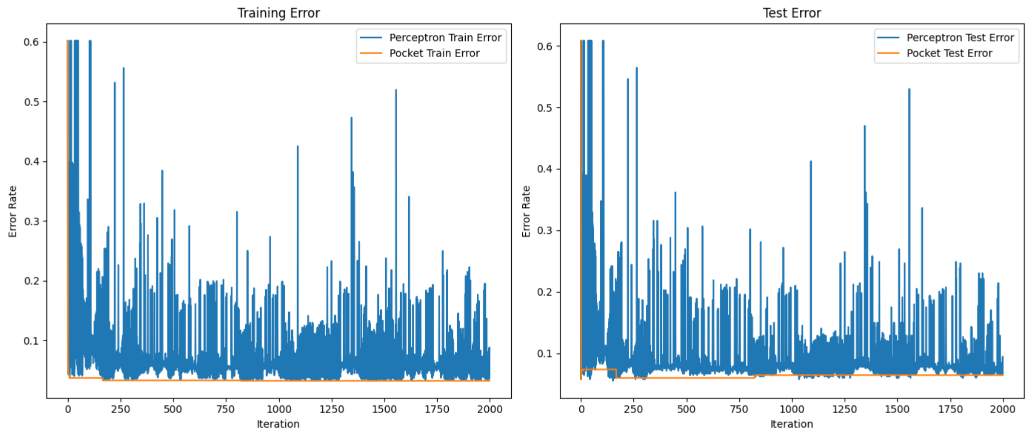

Figure 2: Training and Test Error Rate

From the Figure 2, we observe that:

The error of the Perceptron algorithm fluctuates more, showing that the Perceptron is sensitive to noisy or non-separable samples.

The Pocket algorithm maintains and uses the best solution found so far, so its test error curve is smoother and lower than that of the Perceptron. The error of pocket algorithms decreases as the number of iterations increases.

And the out-of-sample error is generally higher than the in-sample error as we study in lectures.

Overall, the Pocket algorithm is more robust to outliers and non-separable data.

The decision boundaries of the two algorithms are shown in Figure 3.

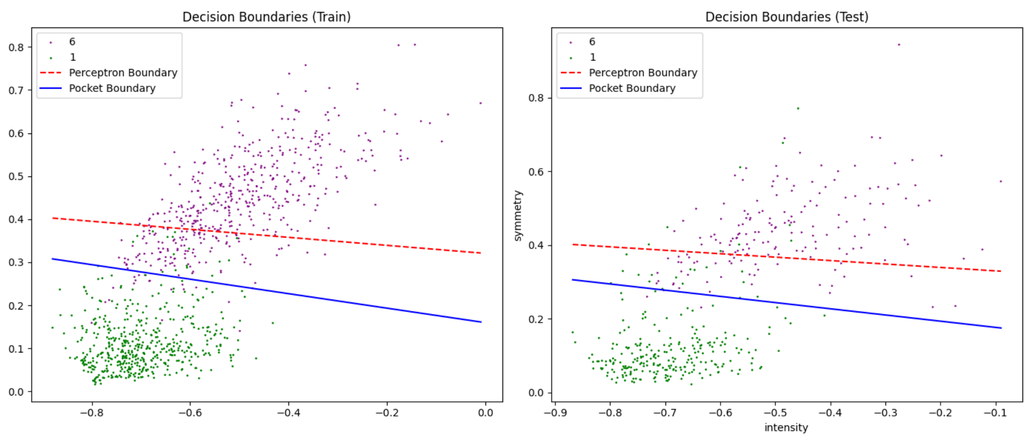

Figure 3: Decision Boundaries