Homework # 3

姓名	學號
操之晴	R13922A04

5. We can assume :
   a. Hypothesis Set H_1 : This set only has one constant function, like always predicting 1. Since it can't change its output, it can't split any set of points. So, H_1 has a VC dimension of 0.
   b. Hypothesis Set H_2 : This set only has another constant function, like always predicting 0. Similarly, it can't split any set of points either, so H_2 also has a VC dimension of 0.
   c. Union of H_1 and H_2 : When we combine these two sets, the union now includes a function that predicts 1 and another that predicts 0. With these two functions together, we can correctly classify a set of one point as either 0 or 1. This means the union set can perfectly classify one point, so the VC dimension of the union is 1.

   In conclusion, $d_{vc}(H_1) = d_{vc}(H_2) = 0, \quad d_{vc}(H_1 \cup H_2) = 1$
   the VC dimension of the union is 1, which is greater than the sum of the individual VC dimensions (0 + 0). This disproves the original statement.

6. False negative is 10 times more important than a false positive means:

		h(x)	
y		+1	-1
	+1	0	1
	-1	10	0

To find the new boundary,

$$10 \cdot (1 - P(y = +1 \mid x)) = P(y = +1 \mid x)$$

$$\text{or } P(y = +1|x) = \frac{10}{11}$$

Thus, $\alpha = \frac{10}{11}$

7. Definitions:

$E^1_{out}(h) = \mathbb{E}_{x\sim P(x)}[h(x) \neq f(x)]$: the discrepancy between h(x) and f(x)

$E^2_{out}(h) = \mathbb{E}_{x\sim P(x),y\sim P(y|x)}[h(x) \neq y]$: the discrepancy between h(x) and y

$E^2_{out}(f) = \mathbb{E}_{x\sim P(x),y\sim P(y|x)}[f(x) \neq y]$: the discrepancy between f(x) and f(x)

For a fixed x, the event {h(x)≠y} can be decomposed into cases based on the target function f(x):

a. When h(x) ≠ f(x) but f(x) = y
b. When f(x) ≠ y

So, we can write

$\mathbb{P}(h(x) \neq y|x) = \mathbb{P}(h(x) \neq f(x)|x) \cdot \mathbb{P}(f(x) = y|x) + \mathbb{P}(f(x) \neq y|x)$

Then substitute into the expression, we get:

$$E^2_{out}(h) = \mathbb{E}_{x\sim P(x),y\sim P(y|x)}[\mathbb{P}(h(x) \neq f(x)|x) \cdot \mathbb{P}(f(x) = y|x) + \mathbb{P}(f(x) \neq y|x)]$$
$$= \mathbb{P}(h(x) \neq f(x)|x) \cdot \big(1 - \mathbb{P}(f(x) \neq y|x)\big) + \mathbb{P}(f(x) \neq y|x)$$
$$= E^1_{out}(h) \cdot \big(1 - E^2_{out}(f)\big) + E^2_{out}(f) \leq E^1_{out}(h) + E^2_{out}(f)$$

8. For the original dataset, the solution for w_{LIN} is $(X^TX)^{-1}X^Ty$.

Now we modify all x_0 to 1126. We call the new matrix X_{LUCKY}. Since we only change the first column, it can be seen as $X_{LUCKY} = XD$, where D is a diagonal matrix with 1126 in the first position and 1 for all others.

For w_{LUCKY}, we can write the solution as $w_{LUCKY} = (X^T_{LUCKY}X_{LUCKY})^{-1}X^T_{LUCKY}y$

Rewriting X_{LUCKY} as XD :

$$w_{LUCKY} = ((XD)^TXD)^{-1}(XD)^Ty$$
$$= (D^TX^TXD)^{-1}(XD)^Ty$$
$$= D^{-1}(X^TX)^{-1}X^Ty$$

Thus, $Dw_{LUCKY} = (X^TX)^{-1}X^Ty = w_{LIN}$.

9. When $h(x) = \frac{1}{1+\exp(-w^T x)}$ $\theta(s) = \frac{1}{1+\exp(-s)}$ where $s = y_n w^T x_n$

Now $\tilde{h}(x) = \frac{1}{2}\left[\frac{w^T x}{\sqrt{1+(w^T x)^2}} + 1\right]$ $\tilde{\theta}(s) = \frac{s+\sqrt{1+s^2}}{2\sqrt{1+s^2}}$

$$\widetilde{E_{in}} = \frac{1}{N}\sum_{n=1}^{N} -\ln \tilde{\theta}(s) = \frac{1}{N}\sum_{n=1}^{N} -\ln \frac{s+\sqrt{1+s^2}}{2\sqrt{1+s^2}} = \frac{1}{N}\sum_{n=1}^{N} \ln \frac{2\sqrt{1+s^2}}{s+\sqrt{1+s^2}}$$

$$\nabla \widetilde{E}_{in} = \frac{\partial E_{in}}{\partial w} = \frac{1}{N}\sum_{n=1}^{N} \frac{\partial \ln \frac{2\sqrt{1+s^2}}{s+\sqrt{1+s^2}}}{\partial \frac{2\sqrt{1+s^2}}{s+\sqrt{1+s^2}}} \cdot \frac{\partial \frac{2\sqrt{1+s^2}}{s+\sqrt{1+s^2}}}{\partial s} \cdot \frac{\partial s}{\partial w}$$

$$= \frac{1}{N}\sum_{n=1}^{N} \left(\frac{s+\sqrt{1+s^2}}{2\sqrt{1+s^2}}\right)\left(-\frac{2}{(s+\sqrt{1+s^2})^2\sqrt{1+s^2}}\right)(y_n x_n)$$

$$= \frac{1}{N}\sum_{n=1}^{N} \frac{(2s(\sqrt{1+s^2}-s)-1)}{(1+s^2)(\sqrt{1+s^2}-s)} \cdot y_n x_n \qquad \text{where } s = y_n w^T x_n$$

10.

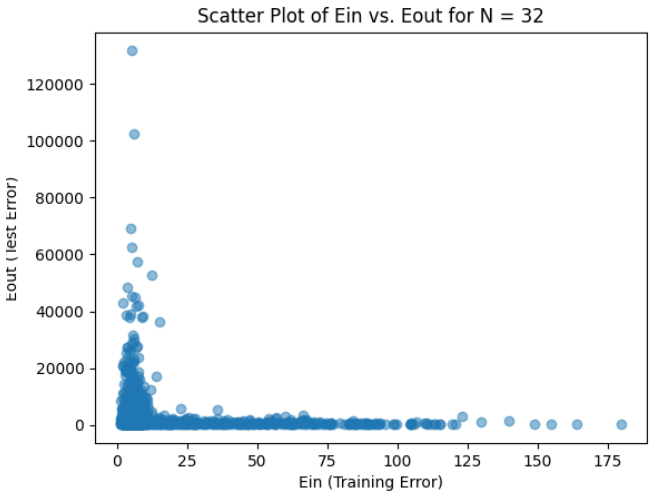


Figure 10a. scatter plot of (Ein, Eout)

```
HW3 > hw3_src > q10.py > ...
import numpy as np
from sklearn.datasets import load_svmlight_file
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# parameters
N = 32
EXPERIMENTS = 1126

# loading data
x, y = load_svmlight_file('cpusmall_scale')
x = x.toarray()

x = np.hstack((np.ones((x.shape[0], 1)), x))    # adding x0 = 1

Ein = []
Eout = []

for i in range(EXPERIMENTS):

    # randomly take N samples for training
    train_indices = np.random.choice(len(x), N, replace=False)
    x_train, y_train = x[train_indices], y[train_indices]

    # use the rest for testing
    test_indices = np.setdiff1d(np.arange(len(x)), train_indices)
    x_test, y_test = x[test_indices], y[test_indices]

    w_lin = np.linalg.pinv(x_train.T @ x_train) @ x_train.T @ y_train

    y_train_pred = x_train @ w_lin
    y_test_pred = x_test @ w_lin

    Ein.append(mean_squared_error(y_train, y_train_pred))
    Eout.append(mean_squared_error(y_test, y_test_pred))

plt.scatter(Ein, Eout, alpha=0.5)
plt.xlabel('Ein (Training Error)')
```

Figure 10b. snapshot of my code

Most of the points are concentrated in a low-error range for both Ein and Eout. This indicates that in most experiments, the model performs quite stably on both the training and test sets. However, there are still some points have very high Eout values while their Ein remains low. This shows that in these experiments, the model overfits the training set, causing very high errors on the test set. This could be due to having too few training samples N=32, which limits the model's ability to learn the data's characteristics.

Overall, most points are clustered in a small error area, showing that in most cases, the model learns the data well and has reasonable generalization.

For problems 10 to 12, the full code has been made available on GitHub.
Feel free to check it out if required or interested!

11.

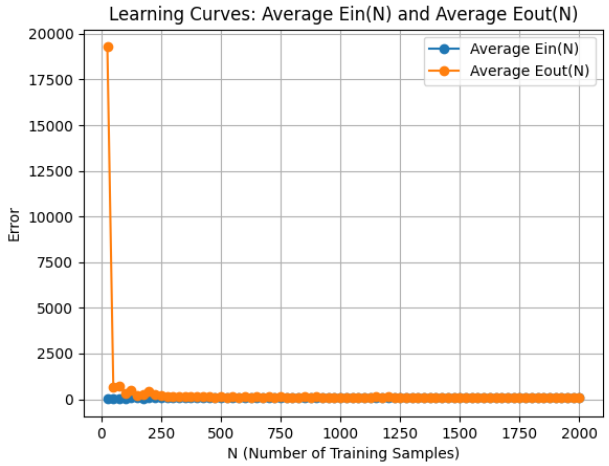
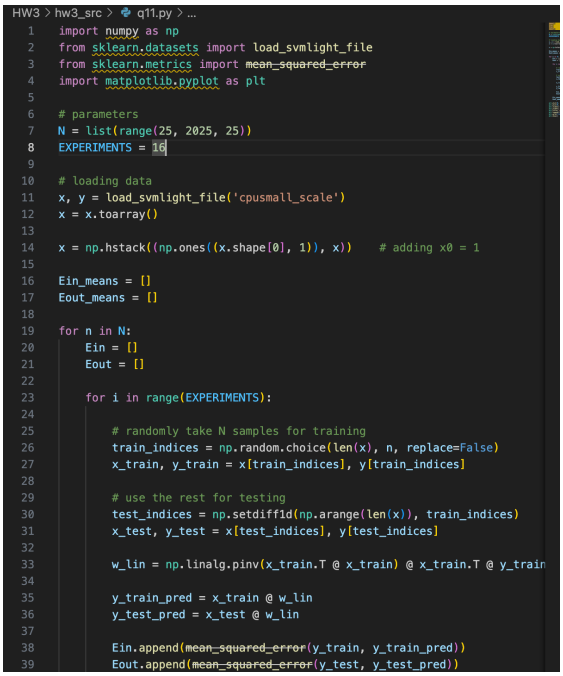


Figure 11a. learning curves of avg(Ein, Eout)



```
HW3 > hw3_src > q11.py > ...
import numpy as np
from sklearn.datasets import load_svmlight_file
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# parameters
N = list(range(25, 2025, 25))
EXPERIMENTS = 16

# loading data
x, y = load_svmlight_file('cpusmall_scale')
x = x.toarray()

x = np.hstack((np.ones((x.shape[0], 1)), x))    # adding x0 = 1

Ein_means = []
Eout_means = []

for n in N:
    Ein = []
    Eout = []

    for i in range(EXPERIMENTS):

        # randomly take N samples for training
        train_indices = np.random.choice(len(x), n, replace=False)
        x_train, y_train = x[train_indices], y[train_indices]

        # use the rest for testing
        test_indices = np.setdiff1d(np.arange(len(x)), train_indices)
        x_test, y_test = x[test_indices], y[test_indices]

        w_lin = np.linalg.pinv(x_train.T @ x_train) @ x_train.T @ y_train

        y_train_pred = x_train @ w_lin
        y_test_pred = x_test @ w_lin

        Ein.append(mean_squared_error(y_train, y_train_pred))
        Eout.append(mean_squared_error(y_test, y_test_pred))
```

Figure 11b. snapshot of my code

When N is small, like 25 or 50, the model might overfit the training data, causing a huge increase in test error. This shows that when there are too few training samples, the model is hard to learn accurately. As N increases to around 200, the test error drops sharply and then stays stable. This indicates that as the training data size grows, the model's generalization improves, the model can generally learn and apply the data well in most cases.

Overall, the training error remains in a low and stable range, indicating that the model fits the training data well.

12.

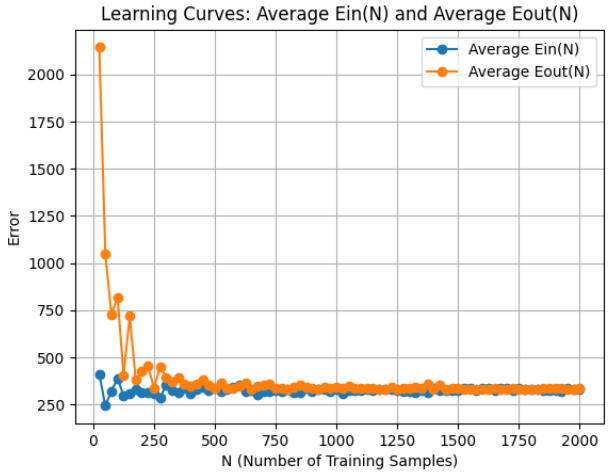


Figure 11a. learning curves of avg(Ein, Eout)

```
HW3 > hw3_src > q12.py > ...
import numpy as np
from sklearn.datasets import load_svmlight_file
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# parameters
N = list(range(25, 2025, 25))
EXPERIMENTS = 16

# loading data
x, y = load_svmlight_file('cpusmall_scale')
x = x.toarray()
x = x[:, :2]

x = np.hstack((np.ones((x.shape[0], 1)), x))    # adding x0 = 1

Ein_means = []
Eout_means = []

for n in N:
    Ein = []
    Eout = []

    for i in range(EXPERIMENTS):

        # randomly take N samples for training
        train_indices = np.random.choice(len(x), n, replace=False)
        x_train, y_train = x[train_indices], y[train_indices]

        # use the rest for testing
        test_indices = np.setdiff1d(np.arange(len(x)), train_indices)
        x_test, y_test = x[test_indices], y[test_indices]

        w_lin = np.linalg.pinv(x_train.T @ x_train) @ x_train.T @ y_train

        y_train_pred = x_train @ w_lin
        y_test_pred = x_test @ w_lin

        Ein.append(mean_squared_error(y_train, y_train_pred))
        Eout.append(mean_squared_error(y_test, y_test_pred))
```

Figure 11b. snapshot of my code

In problem 12, the figure is similar to the one in problem 11. However, we only used the first two features, which means the model had significantly less information, leading to higher prediction errors. We can see the test error still stabilizes, but the overall error values are higher compared to problem 11. This is because the model lacks information from the other 10 features, making it less able to fully learn from the data.

In summary, this plot shows the error trend when using only two features, as the number of training samples increases. Compared to using all features, the overall error is higher, but the model does not show signs of overfitting.