Homework 3

School & Student ID: NYCU / 513 559 004
Course Title: Machine Learning
Announced Date: 241007
Deadline: 241021

Table of Contents

Question 5

Suppose the union $H_1 \cup H_2$ can leverage hypotheses from both sets to shatter more points than either set could alone.

$$d_{vc}(H_1 \cup H_2) > d_{vc}(H_1) + d_{vc}(H_2)$$

However, for any labeling on a set of $d_{vc}(H_1) + d_{vc}(H_2)$ points, at least one of the hypothesis sets H_1 or H_2 must fully accommodate that labeling.

Thus, the inequality above would require one of the sets to shatter more points than its individual VC dimension, which is impossible.

Therefore,

$$d_{vc}(H_1 \cup H_2) \leq d_{vc}(H_1) + d_{vc}(H_2)$$

Question 6

Given that,

$$f_{0/1}(x) = sign\left(P(y =+ 1 \,|\, x) - \frac{1}{2}\right)$$

It implies that we classify $y =+ 1$ only if $P(y =+ 1 \,|\, x) > 0.5$.

Then, we adjust the classification rule with asymmetric error costs:

$$f_{MKT}(x) = sign(P(y =+ 1 \,|\, x) - \alpha)$$

Since false negatives are 10 times more costly, it is necessary to have a threshold much more likely to classify as +1 to avoid the penalty:

$$\frac{P(y=+1 \,|\, x)}{P(y=-1 \,|\, x)} \geq 10$$

$$\Rightarrow \frac{P(y=+1 \,|\, x)}{1-P(y=+1 \,|\, x)} \geq 10$$

$$\Rightarrow P(y =+ 1 \,|\, x) \geq \frac{10}{11}$$

Hence, we can put $P(y =+ 1 \,|\, x)$ into the function $f_{MKT}(x)$, and we'll find:

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

Question 7

Given that,

$$E^{(2)}_{out}(h) = E_{x\sim P(x),\, y\sim(y|x)}[h(x) \neq y]$$

$$\Rightarrow E^{(2)}_{out}(h) = E_{x\sim P(x)}[P(h(x) \neq y \mid x)]$$

Decomposing it with the law of total probability:

$$P(h(x) \neq y \mid x) = P(h(x) \neq f(x) \ and \ f(x) = y \mid x) + P(f(x) \neq y \mid x)$$

By linearity of expectation:

$$E^{(2)}_{out}(h) = E_x[P(h(x) \neq f(x) \ and \ f(x) = y \mid x)] + E_x[P(f(x) \neq y \mid x)]$$

The first term relates to $E^{(1)}_{out}(h)$, as:

$$P(h(x) \neq f(x) \ and \ f(x) = y \mid x) \leq P(h(x) \neq f(x) \mid x)$$

$$\Rightarrow E_x[P(h(x) \neq f(x) \ and \ f(x) = y \mid x)] \leq E^{(1)}_{out}(h)$$

Combining components, therefore:

$$E^{(2)}_{out}(h) \leq E^{(1)}_{out}(h) + E^{(2)}_{out}(f)$$

Question 8

Knowing that,

$$W_{LIN} = \left(X^T X\right)^{-1} X^T y$$

The matrix X can be written as:

$$\boldsymbol{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1d} \\ 1 & x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \dots & x_{Nd} \end{pmatrix}$$

Using the diagonal matrix $X' = XD$ to scale the intercept term by 1126.

$$\mathbf{D} = \begin{pmatrix} 1126 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}$$

$$\boldsymbol{X'} = \begin{pmatrix} 1126 & x_{11} & x_{12} & \dots & x_{1d} \\ 1126 & x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1126 & x_{N1} & x_{N2} & \dots & x_{Nd} \end{pmatrix}$$

After changing x_0 to 1126, the weight vector W_{LUCKY} for X' is shown as:

$$W_{LUCKY} = \left((X')^T X'\right)^{-1} (X')^T y$$

Since $X' = XD$, we can substitute X' with XD as follows:

$$\Rightarrow W_{LUCKY} = \left(D^T X^T XD\right)^{-1} \left(D^T X^T y\right)$$

We know that $\left(D^T X^T XD\right)^{-1} = D^{-1}\left(X^T X\right)^{-1} D^{-1}$, we can replace it into the equation for W_{LUCKY} :

$$W_{LUCKY} = D^{-1}\left(X^T X\right)^{-1} X^T y$$

$$\Rightarrow W_{LIN} = D \cdot W_{LUCKY}$$

Question 9

Given that,

$$\widehat{h}(x) = \frac{1}{2}\left(\frac{w^T x}{\sqrt{1+\left(w^T x\right)^2}} + 1\right)$$

Minimize the negative log-likelihood for the data $\left\{\left(x_n, y_n\right)\right\}$:

$$\widehat{E}_{in}(w) = -\frac{1}{N}\left(y_n ln\left(\widehat{h}(x)\right) + \left(1 - y_n\right) ln\left(1 - \widehat{h}(x)\right)\right)$$

Let $u = w^T x$.

$$\frac{d\widehat{h}}{du} = \frac{1}{2} \cdot \frac{\left(1+u^2\right)-u^2}{\left(1+u^2\right)^{\frac{3}{2}}} = \frac{1}{2} \cdot \frac{1}{\left(1+u^2\right)^{\frac{3}{2}}}$$

Then, the gradient $\nabla_w \widehat{h}$ we find out would be:

$$\Rightarrow \nabla_w \widehat{h} = \frac{d\widehat{h}}{du} \cdot x$$

We can calculate the gradient $\nabla \widehat{E}_{in}(w)$:

$$\nabla \widehat{E}_{in}(w) = -\frac{1}{N}\sum_{n=1}^{N}\left(\frac{y_n}{\widehat{h}\left(x_n\right)} - \frac{1-y_n}{1-\widehat{h}\left(x_n\right)}\right)\nabla_w \widehat{h}\left(x_n\right)$$

Substitute $\nabla_w \widehat{h}(x)$:

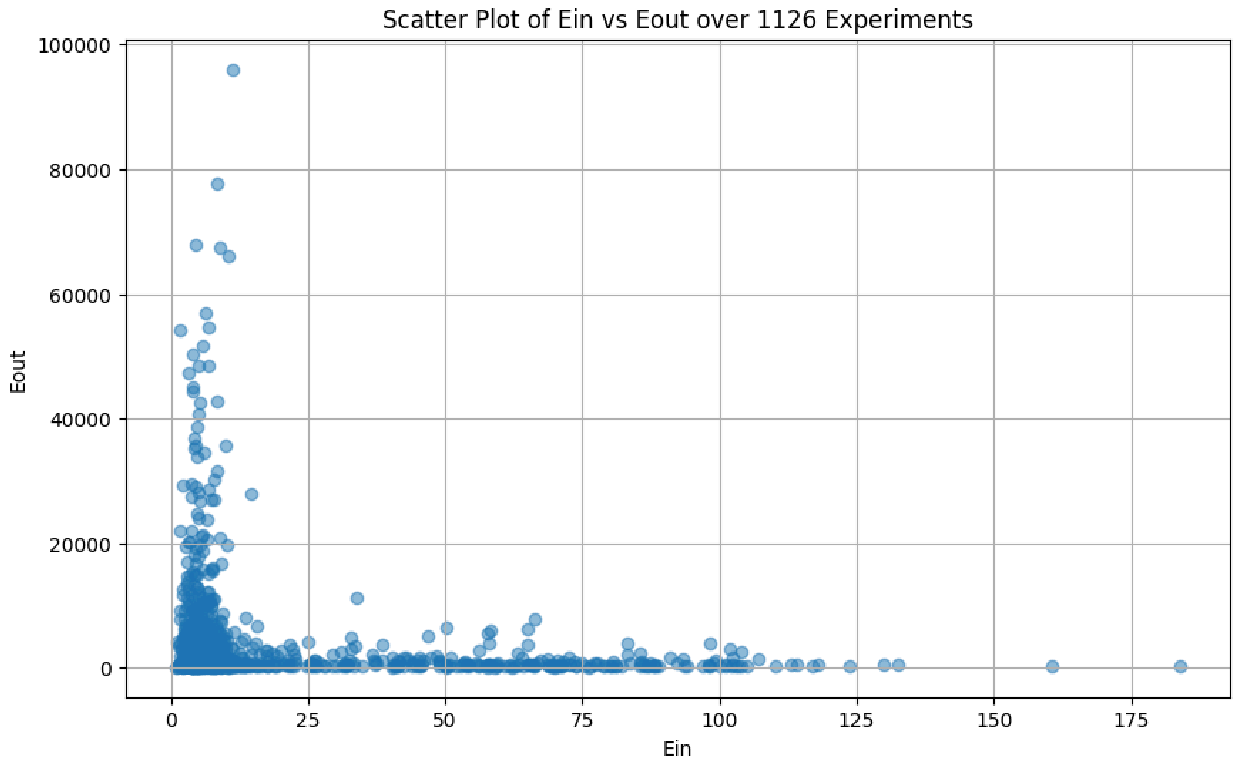
$$\Rightarrow \nabla \widehat{E}_{in}(w) = -\frac{1}{N}\sum_{n=1}^{N}\left(\frac{y_n-\widehat{h}\left(x_n\right)}{\widehat{h}\left(x_n\right)\left(1-\widehat{h}\left(x_n\right)\right)}\right) \cdot \frac{1}{2} \cdot \frac{x_n}{\left(1+\left(w^T x_n\right)^2\right)^{\frac{3}{2}}}$$

Question 10

The first page of the snapshot of my ***code is on the next page***.

My findings:

1. The scatter plot shows the relationship between Ein and Eout for different experiments.
2. Generally, Ein and Eout are ***positively correlated***, meaning that when the model performs well on training data, it also tends to perform well on unseen data.
3. However, some variability indicates that a low Ein does not always guarantee a low Eout. This suggests that ***overfitting can still occur*** in some cases, depending on the specific training samples.



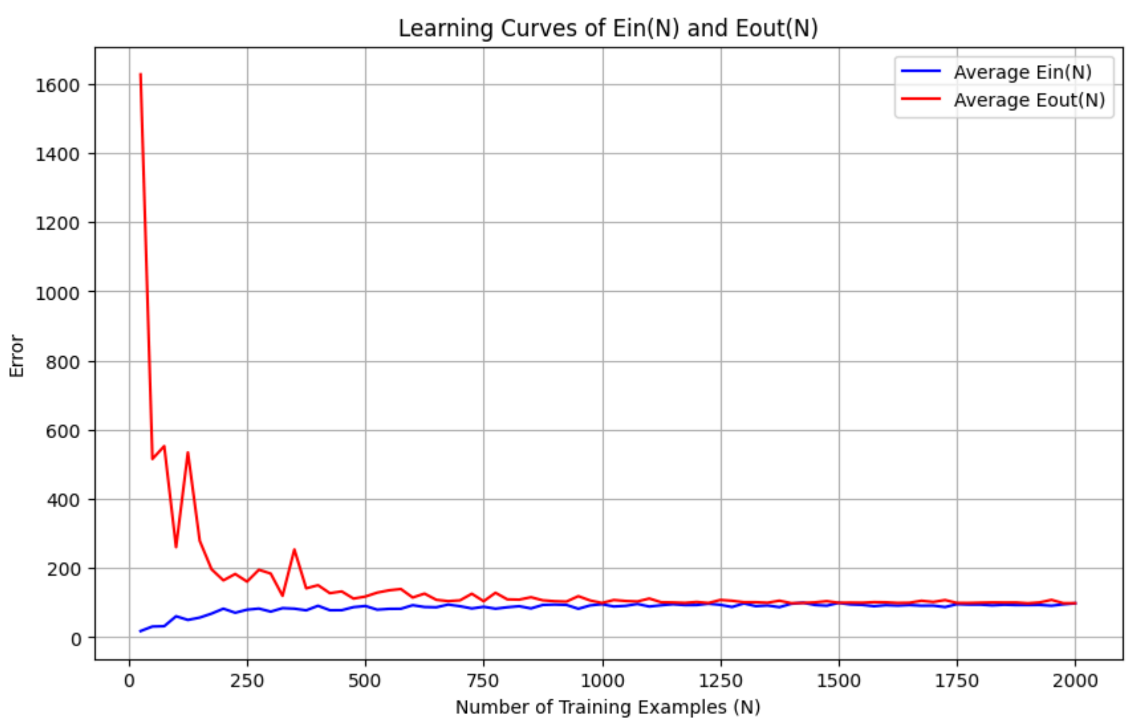
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from google.colab import drive

# Load the dataset
drive.mount('/d')
file_path = '/d/MyDrive/ML_hw3_attachment-cpusmall_scale.txt'
data = []
with open(file_path, 'r') as f:
    for line in f:
        values = line.strip().split()
        y = float(values[0])
        x = [float(v.split(':')[1]) for v in values[1:]]
        data.append((x, y))

X_full = np.array([x for x, y in data])
y_full = np.array([y for x, y in data])

# Parameters
N = 32
num_experiments = 1126
Ein_list = []
Eout_list = []

# Run the experiments
for _ in range(num_experiments):
    # Randomly sample N examples for training
    indices = np.random.choice(len(X_full), N, replace=False)
    X_train = X_full[indices]
    y_train = y_full[indices]

    # Add bias term (x_0 = 1)
    X_train = np.hstack((np.ones((X_train.shape[0], 1)), X_train))
    X_full_bias = np.hstack((np.ones((X_full.shape[0], 1)), X_full))

    # Fit linear regression model
    model = LinearRegression(fit_intercept=False)
    model.fit(X_train, y_train)
    w_lin = model.coef_

    # Calculate Ein (in-sample error)
    y_train_pred = X_train @ w_lin
    Ein = np.mean((y_train - y_train_pred) ** 2)
    Ein_list.append(Ein)
```

Question 11

The first page of the snapshot of my ***code is on the next page***.

My findings:

1. As N increases,
   a. both Ein and Eout tend to ***decrease***, indicating improved model generalization with more training data.
   b. The gap between Ein and Eout ***narrows***, suggesting reduced overfitting with larger training sizes.
   c. Both ***converge*** to similar values, indicating that the model is better at generalizing to new data, reducing bias and variance.
2. For smaller values of N, there is a significant difference between Ein and Eout, which indicates ***overfitting. The model fits the training data well but struggles to generalize to unseen data.***
3. The learning curve shows that ***adding more training data*** helps the model improve its performance, especially for smaller training set sizes where the model initially suffers from high variance.



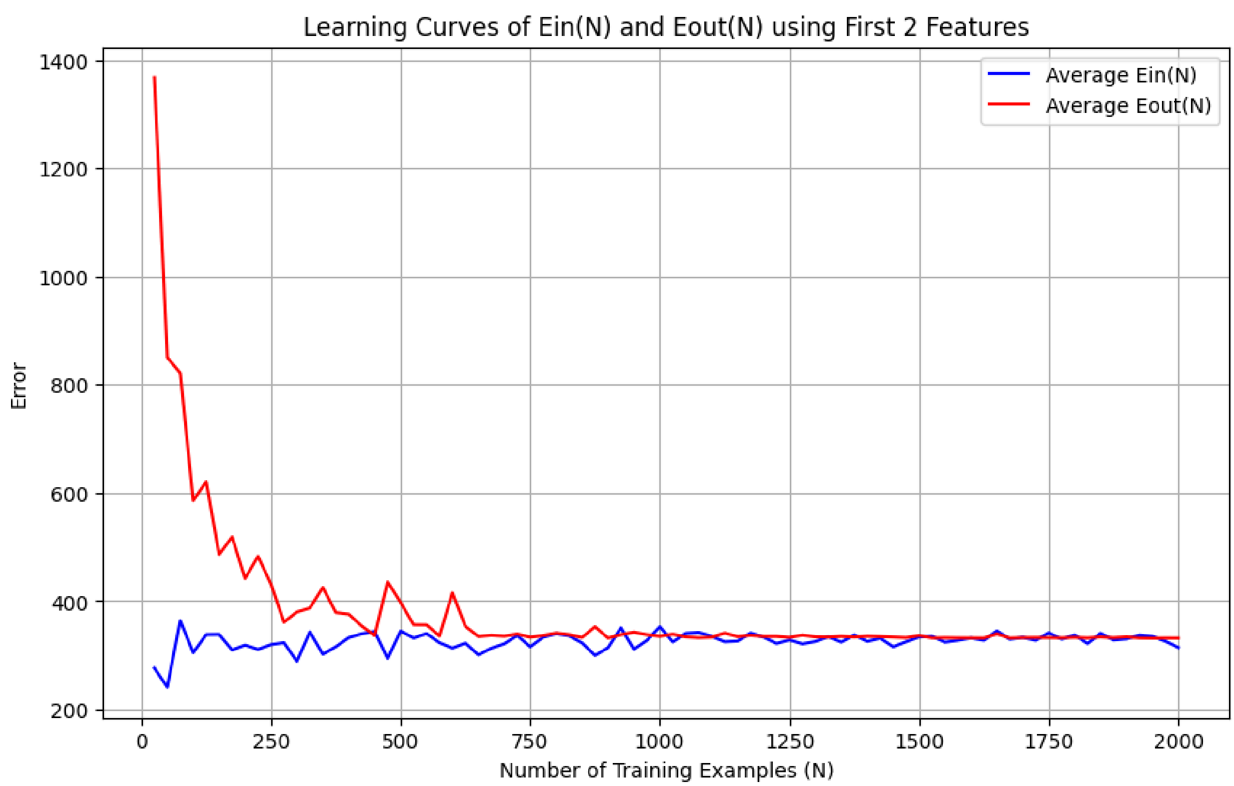
```python
with open(file_path, 'r') as f:
    for line in f:
        values = line.strip().split()
        y = float(values[0])
        x = [float(v.split(':')[1]) for v in values[1:]]
        data.append((x, y))

X_full = np.array([x for x, y in data])
y_full = np.array([y for x, y in data])

# Parameters
N_values = np.arange(25, 2001, 25)
num_experiments = 16
average_Ein = []
average_Eout = []

# Run experiments for each value of N
for N in N_values:
    Ein_total = 0
    Eout_total = 0

    for _ in range(num_experiments):
        # Randomly sample N examples for training
        indices = np.random.choice(len(X_full), N, replace=False)
        X_train = X_full[indices]
        y_train = y_full[indices]

        # Add bias term (x_0 = 1)
        X_train = np.hstack((np.ones((X_train.shape[0], 1)), X_train))
        X_full_bias = np.hstack((np.ones((X_full.shape[0], 1)), X_full))

        # Fit linear regression model
        model = LinearRegression(fit_intercept=False)
        model.fit(X_train, y_train)
        w_lin = model.coef_

        # Calculate Ein (in-sample error)
        y_train_pred = X_train @ w_lin
        Ein = np.mean((y_train - y_train_pred) ** 2)
        Ein_total += Ein

        # Calculate Eout (out-of-sample error)
        y_full_pred = X_full_bias @ w_lin
```

Question 12

The first page of the snapshot of my ***code is on the next page***.

My findings:

1. With only the first 2 features, both Ein and Eout are ***generally higher compared to using all 12 features***, indicating a decrease in model performance.
2. The gap between Ein and Eout ***remains more significant for smaller values of N***, suggesting that the reduced feature set leads to increased bias and potentially higher variance.
3. As N increases, the errors decrease, but they ***do not reach as low values as when using all features***, demonstrating that the model benefits from having more features to learn from.



```python
X_full = np.array([x[:2] for x, y in data])  # Use only the first 2 features
y_full = np.array([y for x, y in data])

# Parameters
N_values = np.arange(25, 2001, 25)
num_experiments = 16
average_Ein = []
average_Eout = []

# Run experiments for each value of N
for N in N_values:
    Ein_total = 0
    Eout_total = 0

    for _ in range(num_experiments):
        # Randomly sample N examples for training
        indices = np.random.choice(len(X_full), N, replace=False)
        X_train = X_full[indices]
        y_train = y_full[indices]

        # Add bias term (x_0 = 1)
        X_train = np.hstack((np.ones((X_train.shape[0], 1)), X_train))
        X_full_bias = np.hstack((np.ones((X_full.shape[0], 1)), X_full))

        # Fit linear regression model
        model = LinearRegression(fit_intercept=False)
        model.fit(X_train, y_train)
        w_lin = model.coef_

        # Calculate Ein (in-sample error)
        y_train_pred = X_train @ w_lin
        Ein = np.mean((y_train - y_train_pred) ** 2)
        Ein_total += Ein

        # Calculate Eout (out-of-sample error)
        y_full_pred = X_full_bias @ w_lin
        Eout = np.mean((y_full - y_full_pred) ** 2)
        Eout_total += Eout

    # Calculate average Ein and Eout
    average_Ein.append(Ein_total / num_experiments)
    average_Eout.append(Eout_total / num_experiments)

# Plot learning curves
```

Question 13

Knowing that $B(N,\ k)$ covers every possible subset size from 0 up to $k\ -\ 1$.

$$B(N,\ k)\ \leq\ \sum_{i=0}^{k-1} C_i^N$$

Precisely, $B(N,\ k)$ should count all subsets that can be formed up to k elements. Such that, it appears $B(N,\ k)$ is at least large enough to include all such subsets.

$$B(N,\ k)\ =\ \sum_{i=0}^{k} C_i^N \geq \sum_{i=0}^{k-1} C_i^N$$

$$\Rightarrow B(N,\ k)\ \geq\ \sum_{i=0}^{k-1} C_i^N$$

Since $B(N,\ k)$ includes all subsets up to that size, it cannot be less than $\sum_{i=0}^{k-1} C_i^N$. In addition, we have already known $B(N,\ k)\ \leq\ \sum_{i=0}^{k-1} C_i^N$, we can eventually establish:

$$B(N,\ k)\ =\ \sum_{i=0}^{k-1} C_i^N$$