5. Prove / disprove:

For any $H_1, H_2 \neq \phi$ (for binary classification, operate on same input space)

$dvc(H_1 \cup H_2) \leq dvc(H_1) + dvc(H_2)$

pf:

Suppose $\exists H_1, H_2 \neq \phi$ s.t. $dvc(H_1 \cup H_2) > dvc(H_1) + dvc(H_2)$

Let $dvc(H_1) = k_1$, $dvc(H_2) = k_2$, then

$$\begin{cases} m_{H_1}(k_1) = 2^{k_1}, & m_{H_1}(k_1+1) < 2^{k_1+1} \\ m_{H_2}(k_2) = 2^{k_2}, & m_{H_2}(k_2+1) < 2^{k_2+1} \end{cases}$$

$\rightarrow dvc(H_1 \cup H_2) > k_1 + k_2$, which means

$\exists$ some $k_1 + k_2 + 1$ points that can be shattered by $H_1 \cup H_2$

Let $dvc(H_1 \cup H_2) = k$, $k > k_1 + k_2$

$m_{H_1 \cup H_2}(k_1 + k_2 + 1) \leq (k_1 + k_2 + 1)^k + 1 = 2^k$

$\because k_1, k_2 \in \mathbb{Z}^+ \cup \{0\}$

$\therefore (k_1, k_2) \in \{(0,1), (1,0)\} \rightarrow (k_1 + k_2 + 1)^k + 1 = 2^k + 1 > 2^k$ (↯)

Thus, $k_1 = k_2 = 0$, then $(k_1 + k_2 + 1)^k + 1 = |H| = 2$ (function)

By thm, $dvc(H) = 0 \Leftrightarrow H = \{y = a, a \in \mathbb{R}\}$ (H has exactly one hypothesis, which is a constant) ✓

Therefore, let $H_1 = \{y = a_1\}$, where $a_1 \in \mathbb{R}$, $H_2 = \{y = a_2\}$ where $a_2 \in \mathbb{R}$

$\rightarrow H_1 \cup H_2 = \{y = a_1, y = a_2\}$ where $a_1, a_2 \in \mathbb{R}$.

However, $dvc(H_1 \cup H_2) = 0 \not> 0$ ↯

Thus, $dvc(H_1 \cup H_2) \leq dvc(H_1) + dvc(H_2)$ □

6. ∵ $f_{MKT}(\vec{x}) = \text{sign}(P(y=+1 \mid \vec{x}) - \alpha)$

∴ if $\begin{cases} P(y=+1 \mid \vec{x}) > \alpha, & \text{then } f_{MKT}(\vec{x}) = +1 \\ P(y=+1 \mid \vec{x}) \le \alpha, & \text{then } f_{MKT}(\vec{x}) = -1 \end{cases}$

h \ t	+1	-1
+1	0	(1) FP.
-1	(10) FN	0

If we consider the points on the decision boundary, the expected loss for assigning any such $\vec{x}$ to either side should be the same (because the decision boundary is the place where we believe to have same probability to classify points to both sides)

Therefore, for $\vec{x}$ on the boundary, we have this equality holds:

(FN) (FP) the probability of having the point $\vec{x}$ (which is on the boundary) to have true label $y=-1$

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

$$\Rightarrow 10\,P(y=+1 \mid \vec{x}) = 1 \cdot (1 - P(y=+1 \mid \vec{x}))$$

$$\Rightarrow 10\,P(y=+1 \mid \vec{x}) = 1 - P(y=+1 \mid \vec{x})$$

$$\Rightarrow 11\,P(y=+1 \mid \vec{x}) = 1$$

$$\Rightarrow P(y=+1 \mid \vec{x}) = \frac{1}{11}$$

Therefore, we assign α as $\alpha = \frac{1}{11}$, making $f_{MKT}(\vec{x}) = \text{sign}(P(y=+1 \mid \vec{x}) - \frac{1}{11})$ □

because this means:

if $P(y=+1 \mid \vec{x}) > \frac{1}{11}$, then $f_{MKT}(\vec{x}) = 1$

→ we only need the probability to be more than $\frac{1}{11}$ (but not 0.5) to make the model classify $\vec{x}$ to be a member (+1)

→ we classify a customer to be a member more often because the cost of FN is greater.

Claim:

2 $\underset{\substack{\vec{x}\sim p(\vec{x})\\ y\sim p(y|\vec{x})}}{E}[\![h(\vec{x})\neq y]\!] \le \underset{\vec{x}\sim p(\vec{x})}{E}[\![h(\vec{x})\neq f(\vec{x})]\!] + \underset{\substack{\vec{x}\sim p(\vec{x})\\ y\sim p(y|\vec{x})}}{E}[\![f(\vec{x})\neq y]\!]$ ← naive

Suppose $\underset{\substack{\vec{x}\sim p(\vec{x})\\ y\sim p(y|\vec{x})}}{E}[\![h(\vec{x})\neq y]\!] > \underset{\vec{x}\sim p(\vec{x})}{E}[\![h(\vec{x})\neq f(\vec{x})]\!] + \underset{\substack{\vec{x}\sim p(\vec{x})\\ y\sim p(y|\vec{x})}}{E}[\![f(\vec{x})\neq y]\!]$

S S_1 S_2

Let S be the set that contains all $\vec{x}_i \ni h(\vec{x}_i)\neq y_i$

S_1 be the set that contains all $\vec{x}_j \ni h(\vec{x}_j)\neq f(\vec{x}_j)$

S_2 be the set that contains all $\vec{x}_k \ni f(\vec{x}_k)\neq y_k$

Suppose $\exists\, \vec{x}_t \in S$, then $h(\vec{x}_t)\neq y_t$

{ if $\vec{x}_t\in S_1$, then $f(\vec{x}_t)=y_t \to \vec{x}_t\notin S_2$
if $\vec{x}_t\in S_2$, then $f(\vec{x}_t)\neq y_t \to \vec{x}_t\notin S_1$

Thus, $S_1\cap S_2=\phi$

If $\vec{x}_t\in S$ and $\vec{x}_t\notin S_1 \cap \vec{x}_t\notin S_2$

$\to h(\vec{x}_t)\neq y_t \cap h(\vec{x}_t)\neq f(\vec{x}_t) \cap f(\vec{x}_t)\neq y_t$ ↯ // binary classification

Therefore, $\forall \vec{x}_t\in S$, either $\vec{x}_t\in S_1$ or $\vec{x}_t\in S_2$

$\therefore S\subseteq S_1\cup S_2 \Rightarrow |S| \le |S_1\cup S_2| = |S_1|+|S_2|$

// $S_1\cap S_2=\phi$

Which leads to a contradiction.

Thus, $\underset{\substack{\vec{x}\sim p(\vec{x})\\ y\sim p(y|\vec{x})}}{E}[\![h(\vec{x})\neq y]\!] \le \underset{\vec{x}\sim p(\vec{x})}{E}[\![h(\vec{x})\neq f(\vec{x})]\!] + \underset{\substack{\vec{x}\sim p(\vec{x})\\ y\sim p(y|\vec{x})}}{E}[\![f(\vec{x})\neq y]\!]$ □

8. Linear regression on dataset : $\{(\vec{x}_n, y_n)\}_{n=1}^{N}$ $\vec{x}_n \in \mathbb{R}^{d+1}$ with $x_{n0} = 1 \ \forall n \in \{1, \dots, N\}$

Assume : $X^T X$: invertible

unique sol $\vec{w}_{LIN}$ acquired on $\{(\vec{x}_n, y_n)\}_{n=1}^{N}$

+ change $x_{n0} = 1 \to b \ \forall n \in \{1, \dots, N\}$

Prove: $\vec{w}_{LIN} = D \vec{w}_{LUCKY}$ where D: diagonal matrix

---

$\because X^T X$: invertible $\therefore$ $\vec{w}_{LIN} = (X^T X)^{-1} X^T y$ (X^T)

Write:

$$X' = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1d} \\ 1 & x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & & \ddots & \\ 1 & x_{N1} & x_{N2} & \cdots & x_{Nd} \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1d} \\ 1 & x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \cdots & x_{Nd} \end{bmatrix} \begin{bmatrix} 1 \to b & & 0 \\ & 1 & \\ 0 & & \ddots \\ & & & 1 \end{bmatrix} = XD'$$

$N \times (d+1)$ $\quad$ $N \times (d+1)$ $\quad$ $D' = \det$ $(d+1) \times (d+1)$

$\to X'^T X' = (XD')^T (XD') = D'^T X^T X D' = D' X^T X D'$

$\because D'$: diagonal $\therefore D'^T = D'$

$\because \det(D') = 1 \to b \neq 0$ and $\exists (X^T X)^{-1} \therefore \det(X^T X) \neq 0$

$$\det(X'^T X') = \det(D' X^T X D') = \det(D') \det(X^T X) \det(D') = \det(D')^2 \det(X^T X) \neq 0$$

$\therefore X'^T X'$: invertible

$\because D^T = D$
$\therefore D^{-1} D^T = D^{-1} D = I$

Thus, $\vec{w}_{WLSKY} = (X'^T X')^{-1} X'^T \vec{y} = (D' X^T X D')^{-1} (XD')^T \vec{y} = D'^{-1} (X^T X)^{-1} D'^{-1} D'^T X^T \vec{y}$

(A B C)

$\because \det(D') \neq 0 \therefore D'$: invertible

$(ABC)^{-1} = C^{-1} B^{-1} A^{-1}$

$$= D'^{-1} (X^T X)^{-1} X^T \vec{y}$$

$\| $

$\vec{w}_{LSW}$

Therefore, $\vec{w}_{LSW} = D' \vec{w}_{WLSKY}$

$\rightarrow D = D' = \begin{bmatrix} 1/n & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix}$ □

9. Using a new family of sigmoid hypotheses, $\tilde{h}(\vec{x}) = \frac{1}{2}\left(\frac{\vec{w}^T\vec{x}}{\sqrt{1+(\vec{w}^T\vec{x})^2}}+1\right)$

We can write

$$\tilde{h}(\vec{x}) = \theta(\vec{w}^T\vec{x}), \quad \text{where } \theta(s) = \frac{1}{2}\left(\frac{s}{\sqrt{1+s^2}}+1\right)$$

Also, we will have likelihood:

$$P(y|\vec{x}) = \begin{cases} \tilde{h}(\vec{x}) & \text{for } y=+1 \\ 1-\tilde{h}(\vec{x}) & \text{for } y=-1 \end{cases}$$

substitute $\tilde{h}(\vec{x})$ by $\theta(\vec{w}^T\vec{x})$:

$$P(y|\vec{x}) = \begin{cases} \theta(\vec{w}^T\vec{x}) & \text{for } y=+1 \\ 1-\theta(\vec{w}^T\vec{x}) & \text{for } y=-1 \end{cases}$$

We can verify that:

$$\boxed{1-\theta(s)} = 1-\frac{1}{2}\left(\frac{s}{\sqrt{1+s^2}}+1\right) = 1-\frac{s}{2\sqrt{1+s^2}}-\frac{1}{2} = \frac{1}{2}-\frac{s}{2\sqrt{1+s^2}}$$

$$= \frac{1}{2}\left(\frac{-s}{\sqrt{1+(-s)^2}}+1\right) = \boxed{\theta(-s)}$$

Thus,

$$P(y|\vec{x}) = \begin{cases} \theta(\vec{w}^T\vec{x}) & \text{for } y=+1 \\ \theta(-\vec{w}^T\vec{x}) & \text{for } y=-1 \end{cases}$$

$$\Rightarrow P(y|\vec{x}) = \theta(y\cdot\vec{w}^T\vec{x})$$

We can the derive the log likelihood:

$$\ln \prod_{i=1}^{N} P(y_n | \vec{x}_n) = \sum_{i=1}^{N} \ln(P(y_n | \vec{x}_n))$$

$$= \sum_{i=1}^{N} \ln(\theta(y \cdot \vec{w}^T \vec{x}))$$

To find the maximum log likelihood is equivalent to find the minimum cross entropy error:

$$\max_{\vec{w}} \sum_{i=1}^{N} \ln(\theta(y \cdot \vec{w}^T \vec{x})) \Rightarrow \min_{\vec{w}} -\sum_{i=1}^{N} \ln(\theta(y \cdot \vec{w}^T \vec{x}))$$

By our definition of $\theta(s) = \frac{1}{2}\left(\frac{s}{\sqrt{1+s^2}} + 1\right)$,

$$\ln(\theta(y\vec{w}^T\vec{x})) = \ln\left(\frac{1}{2}\left(\frac{y\vec{w}^T\vec{x}}{\sqrt{1+(y\vec{w}^T\vec{x})^2}} + 1\right)\right)$$

$$= \ln \frac{y\vec{w}^T\vec{x} + \sqrt{1+(y\vec{w}^T\vec{x})^2}}{2\sqrt{1+(y\vec{w}^T\vec{x})^2}}$$

$\therefore$ the $\widetilde{E_{in}}$ we want to minimize is:

$$\widetilde{E_{in}}(\vec{w}) = \frac{1}{N}\left[-\sum_{i=1}^{N} \ln(\theta(y_i \vec{w}^T \vec{x}_i))\right] = \frac{1}{N}\sum_{i=1}^{N} \ln \frac{2\sqrt{1+(y\vec{w}^T\vec{x})^2}}{y\vec{w}^T\vec{x} + \sqrt{1+(y\vec{w}^T\vec{x})^2}}$$

To minimize $\tilde{E}_{in}(\vec{w})$, we need to find the place where $\nabla \tilde{E}_{in}(\vec{w}) = 0$

First we calculate $\nabla \ln(\theta(\,))$

$$\nabla \ln(\theta(y_i \vec{w}^T \vec{x}_i)) = \frac{1}{\theta(y_i \vec{w}^T \vec{x}_i)} \cdot \nabla \theta(y_i \vec{w}^T \vec{x}_i)$$

calculate $\nabla \theta(s)$:

$$\frac{d}{ds}\left(\frac{s}{\sqrt{1+s^2}}\right) = \frac{\sqrt{1+s^2} - \frac{s^2}{\sqrt{1+s^2}}}{1+s^2} = \frac{\frac{(1+s^2)-s^2}{\sqrt{1+s^2}}}{1+s^2} = \frac{\frac{1}{\sqrt{1+s^2}}}{1+s^2} = (1+s^2)^{-\frac{3}{2}}$$

$((1+s^2)^{-\frac{1}{2}})' \cdot s = \frac{1}{2}(1+s^2)^{-\frac{3}{2}} \cdot 2s \cdot s = \frac{s^2}{\sqrt{1+s^2}}$

$$\therefore \frac{d}{ds}\theta(s) = \frac{1}{2} \cdot (1+s^2)^{-\frac{3}{2}}$$

$$\rightarrow \nabla \theta(y_i \vec{w}^T \vec{x}_i) = \frac{1}{2} \frac{1}{\left[1+(y_i \vec{w}^T \vec{x}_i)^2\right]^{\frac{3}{2}}} y_i \vec{x}_i = \frac{y_i \vec{x}_i}{2\left[1+(\vec{w}^T \vec{x}_i)^2\right]^{\frac{3}{2}}}$$

($\because y_i^2 = 1$)

$$\rightarrow \nabla \ln(\theta(y_i \vec{w}^T \vec{x}_i)) = \frac{1}{\theta(y_i \vec{w}^T \vec{x}_i)} \cdot \frac{y_i \vec{x}_i}{2\left[1+(\vec{w}^T \vec{x}_i)^2\right]^{\frac{1}{2}}}$$

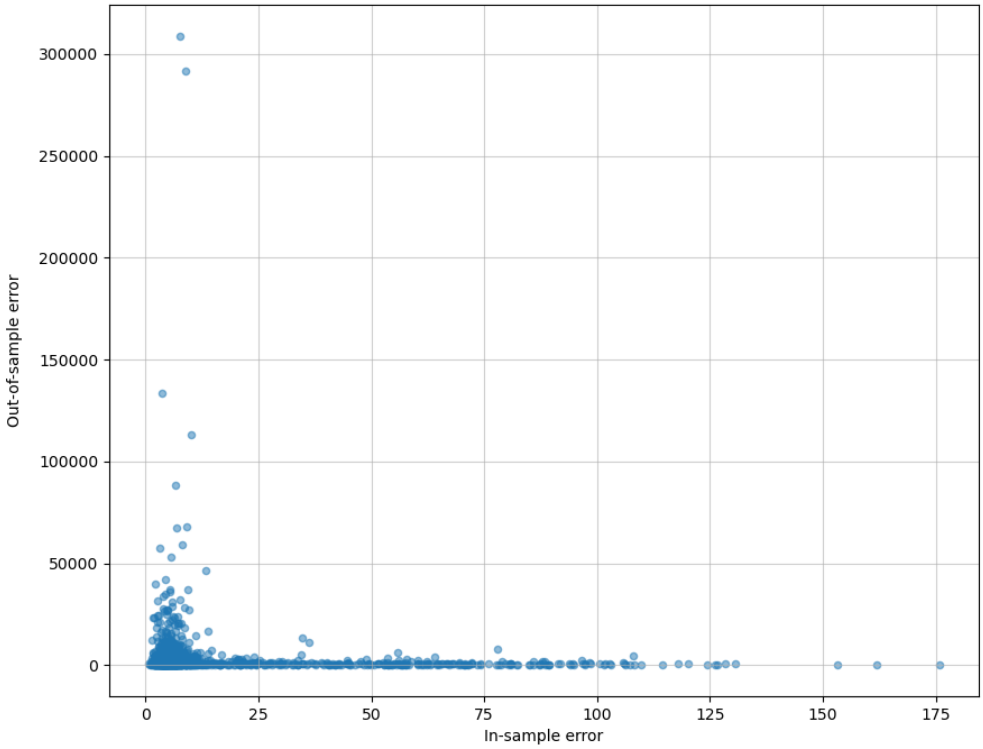
$$\rightarrow \nabla \tilde{E}_{in}(\vec{w}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2\theta(y_i \vec{w}^T \vec{x}_i)} \frac{y_i \vec{x}_i}{\left[1+(\vec{w}^T \vec{x}_i)^2\right]^{\frac{1}{2}}}$$

$$\theta(y_i \vec{w}^T \vec{x}_i) = \frac{1}{2}\left(\frac{y_i \vec{w}^T \vec{x}_i}{\sqrt{1+(y_i \vec{w}^T \vec{x}_i)^2}} + 1\right)$$

$$= \frac{1}{N} \sum_{i=1}^{N} \frac{y_i \vec{x}_i}{\left(\frac{y_i \vec{w}^T \vec{x}_i}{\sqrt{1+(y_i \vec{w}^T \vec{x}_i)^2}} + 1\right)\left[1+(\vec{w}^T \vec{x}_i)^2\right]^{\frac{1}{2}}}$$

□

Question 10: Report



Explanation:

The first thing we would probably notice is that, there's 2 points in the upper left corner of the plot, which indicates a small in-sample error, and a relatively large out of sample error. I think this is the result that I would expect, since we only choose 32 from all the 8192 examples, using such a small subset of the dataset would make the model vary greatly, depending on the subset we choose, and if the subset contains outliers, or the examples chosen are not representative enough for the entire dataset, then overfitting to such examples would cause high out of sample error, meaning that the model is nor well generalized. In addition to these 2 points, we can also see that in about 10 experiments, we have out of sample error greater than 50000, which is also

quite high. Most of the other points in the lower left corner still shows that the out of sample error is not that small, this result verifies our intuition.

Next, we can see that in the lower middle part, and the lower right corner, there are experiments having nearly 0 out of sample error but higher in sample errors (compared to those having only in sample errors less than 25.) For this part, this result actually matches that in theory, higher in sample error (due to underfitting) would often end up with lower out of sample error, because models like this generalize better than those that overfit. However, I still think that it's surprising to use 32 points to generalize well on the remaining points, but the amount of experiments like this is small, so I think maybe it's reasonable due to probability.

Last, observe the whole plot, we can see that the distribution varies a lot over all of the experiments. Even though we don't know the correlations between the 8192 examples, randomly choosing 32 from them still results in a high variability (since there would be C(8192, 32) possibilities.) As mentioned before, this would cause our distribution to depend highly on the 32 samples we choose, so I don't think we can tell more by only conducting 1126 experiments.

Code:

Read in the data:

```
def read_libsvm_format(file_path):
    X, y = [], []
    with open(file_path, 'r') as f:
        for line in f:
            parts = line.strip().split()
            y.append(float(parts[0]))
            features = {}
            for item in parts[1:]:
                index, value = item.split(":")
                features[int(index)] = float(value)
            X.append(features)
    return X, y

# aim: read the data from the file 'cpusmall_scale'
X, y = read_libsvm_format('cpusmall_scale')

# explain: the data X returned is in the form of a list of dictionaries, where each dictionary contains the features of a data point
# ex: X[0] = {1: -0.993496, 2: -0.993043, 3: -0.850291, 4: -0.963479, 5: -0.960727, 6: -0.900596, 7: -0.96642, 8: -0.863996, 9: -0.606175, 10: -0.999291, 11: 0.08118

# explain: the label y is a list of floats
# ex: y[0] = 90.0

```

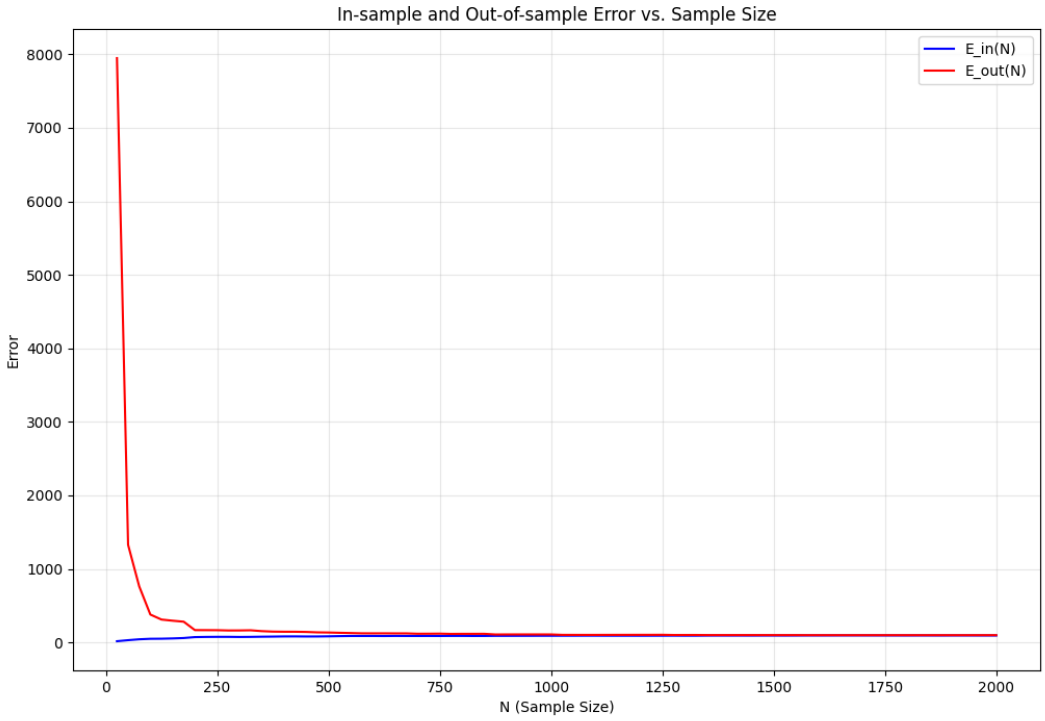
Calculate w_{LIN} and E_{in}:

```python
N = 32
feature_amount = 12
in_out_sample_error = []
# aim: perform linear regression for 1126 times, calculate the in-sample error and estimate the out of sample error
for experiment in tqdm(range(1126)):
    np.random.seed(experiment)

    random_sample_indices = np.random.choice(len(X), N, replace=False)
    X_sample = [X[i] for i in random_sample_indices]
    y_sample = [y[i] for i in random_sample_indices]

    # subaim: convert each sample data points (which is a dictionary) to a ndarray and save in X_sample_array
    X_sample_array = []
    for sample in X_sample:
        input_vector = np.concatenate((np.array([1]), np.zeros(feature_amount)))
        # explain: originally we need subtract 1 because indices in libsvm format start from 1, but now we add the 0-th element (1) to each input vector
        for index, value in sample.items():
            input_vector[index] = value
        X_sample_array.append(input_vector)

    # subaim: convert X_sample_array to a matrix, and y_sample to an array
    X_mat = np.array(X_sample_array)
    y_array = np.array(y_sample)

    # subaim: perform linear regression
    # explain: we do linear regression by using the normal equation (w = (X^TX)^(-1)X^Ty)
    # explain: we use np.linalg.inv() calculate the inverse of X^TX, and use @ to denote matrix multiplication
    w = np.linalg.inv(X_mat.T @ X_mat) @ X_mat.T @ y_array

    # subaim: calculate the in-sample error
    # explain: the in-sample error is the mean squared error of the N points
    in_sample_error = np.mean((X_mat @ w - y_array) ** 2)

```

Estimate E_{out}:

```python
    # subaim: estimate the out of sample error
    # explain: the out of sample error is estimated by averaging the squared error of the rest of the 8192 - N data points
    out_of_sample_error = 0
    for i in range(len(X)):
        if i not in random_sample_indices:
            unseen_input_vector = np.concatenate((np.array([1]), np.zeros(feature_amount)))
            for index, value in X[i].items():
                unseen_input_vector[index] = value
            out_of_sample_error += (unseen_input_vector @ w - y[i]) ** 2
    out_of_sample_error /= (len(X) - N)
    in_out_sample_error.append((in_sample_error, out_of_sample_error))
```

Question 11: Report



Explanations:

In this problem, we gradually increase the sample size (until 2000, which is nearly $\frac{1}{4}$ of our dataset), and calculate the average in sample error and out of sample error over only 16 experiments.

From the graph we can see that when the sample size is under about 100, we cannot generalize well since this is represented by the high out of sample error. But after we met a certain sample size (which is maybe 100 or 150 or some amount around them), we have enough data to represent the whole dataset by the sample, so that we have enough information to construct our model.

Also, if we really look at the examples, we could find that the number of features is 12, which is enough to provide sufficient degrees of freedom to fit the training data. In a geometrical view, we can think of finding a 12 dimensional hyperplane to split the N data points, in a 13 dimensional space (since we add 1 to the input vector.) In lower dimensional spaces, like 2D or 3D spaces, the degree of freedom is smaller, therefor it is hard to find a line or a 2D plane that can pass through all the points, but for a 12 dimensional hyperplane, it is easier to do so. Thus, it is possible to find the “optimal” hyperplane to fit best to the sample points, producing nearly 0 in sample error in all cases.

Code:

(The reading data part is ignored in this report since it’s the same in the previous question.)
The first thing to do is to define a list of all possible Ns that we need to iterate through (25, 50, 75,…,2000), then for each value of N, we conduct the experiment 16 times.

```
N_list = list(range(25, 2025, 25))
feature_amount = 12
seed = 0

# aim: for each sample size N, perform linear regression for 16 times, calculate the average in-sample error and estimate the average out of sample error
avg_in_error_for_each_N = []
avg_out_error_for_each_N = []
for N in tqdm(N_list):
    in_out_sample_error = []
    # subaim: perform linear regression for 16 times, calculate the in-sample error and estimate the out of sample error
    for experiment in range(16):
```

The random choosing process is done without replacement. We save the chosen examples in X_sample and their corresponding y in y_sample:

```
    for experiment in range(16):
        seed += 1
        np.random.seed(seed)

        random_sample_indices = np.random.choice(len(X), N, replace=False)
        X_sample = [X[i] for i in random_sample_indices]
        y_sample = [y[i] for i in random_sample_indices]
```

Then convert data type:

```
        # subsubaim: convert each sample data points (which is a dictionary) to a ndarray and save in X_sample_array
        X_sample_array = []
        for sample in X_sample:
            input_vector = np.concatenate((np.array([1.0]), np.zeros(feature_amount, dtype=float)))
            for index, value in sample.items():
                input_vector[index] = float(value)
            X_sample_array.append(input_vector)

        # subsubaim: convert X_sample_array to a matrix, and y_sample to an array
        X_mat = np.array(X_sample_array, dtype=float)
        y_array = np.array(y_sample, dtype=float)
```

Calculate in sample error and out of sample error:

```
        # subsubaim: perform linear regression
        # explain: we do linear regression by using the normal equation (w = (X^TX)^(-1)X^Ty)
        # explain: we use np.linalg.inv() calculate the inverse of X^TX, and use @ to denote matrix multiplication
        w = np.linalg.inv(X_mat.T @ X_mat) @ X_mat.T @ y_array

        # subsubaim: calculate the in-sample error
        # explain: the in-sample error is the mean squared error of the N points
        in_sample_error = np.mean((X_mat @ w - y_array) ** 2)

        # subsubaim: estimate the out of sample error
        # explain: the out of sample error is estimated by averaging the squared error of the rest of the 8192 - N data points
        out_of_sample_error = 0.0
        for i in range(len(X)):
            if i not in random_sample_indices:
                unseen_input_vector = np.concatenate((np.array([1.0]), np.zeros(feature_amount, dtype=float)))
                for index, value in X[i].items():
                    unseen_input_vector[index] = float(value)
                out_of_sample_error += (unseen_input_vector @ w - y[i]) ** 2
        out_of_sample_error /= (len(X) - N)
        in_out_sample_error.append((in_sample_error, out_of_sample_error))
```

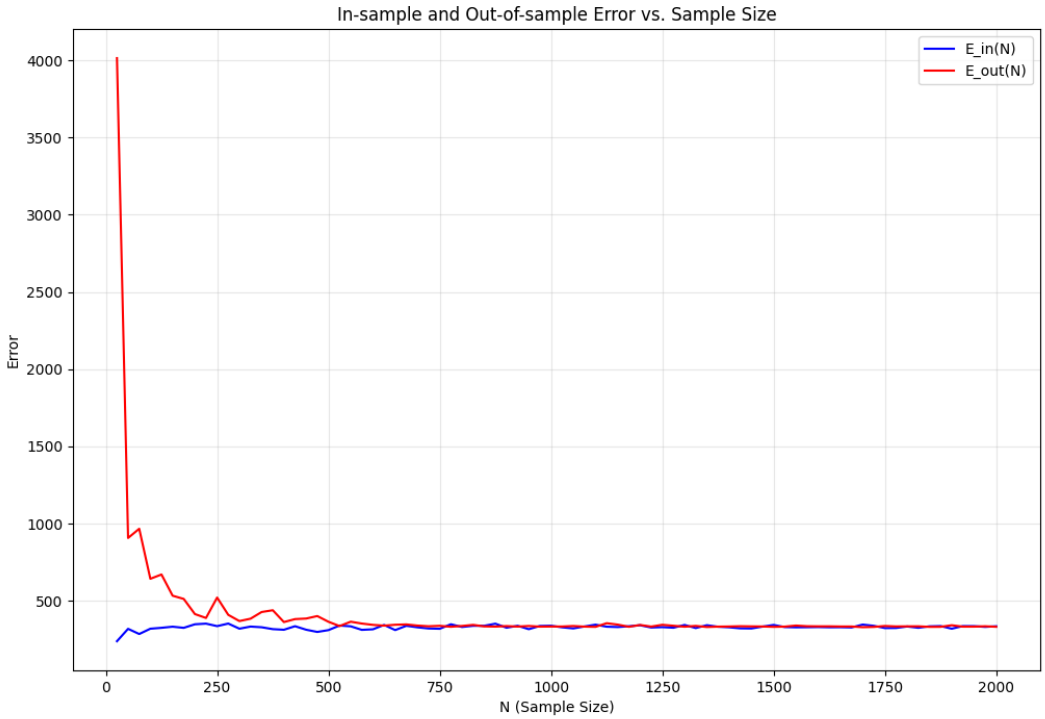
The in_out_sample_error is a list containing the result of the 16 experiments for any given N.

We then calculate $\overline{\mathrm{E}}_{\mathrm{in}}(\mathrm{N})$ and $\overline{\mathrm{E}}_{\mathrm{out}}(\mathrm{N})$:

```
        # subaim: calculate the average in-sample error and out of sample error for each N
        avg_in_sample_error = 0.0
        avg_out_of_sample_error = 0.0
        for each_exp_error_result in in_out_sample_error:
            avg_in_sample_error += each_exp_error_result[0]
            avg_out_of_sample_error += each_exp_error_result[1]
        avg_in_sample_error /= 16
        avg_out_of_sample_error /= 16
        avg_in_error_for_each_N.append(avg_in_sample_error)
        avg_out_error_for_each_N.append(avg_out_of_sample_error)
```

The last part is to plot the result, but it’s similar to the previous question.

Question 12: Report



Explanations:

The difference between this question and the previous question 11 is that we only preserve the first 2 features instead of using all the 12 features. We could see that in the resulting figure, the highest out of sample value is lower comparing to the previous question. Although they both gradually decrease as the sample size increases.

This is because when we're in a higher dimensional space, we would have to face the problem of generalization. Although using all the 12 features results in a space that can form more flexible models, meaning that is more possible to find a hyperplane that can separate our training data, making the in sample error to be about 0, when the model is used to evaluate the

unseen data, over-parameterized models may cause overfitting. In this question we decrease the amount of features to 2, indicates a choice toward generalization than approximation.

However, we can see that the in sample error is a nonzero value, even though the value is low for over all possible N. This is also the result as we reduce the dimension of the space to 3 (including the fixed zeroth coordinate), our free parameter is reduced to 2, so that it is hard to make the training data linearly separable.

Nonetheless, reducing the number of features would make $\mathrm{d_{VC}}$ small, and as the amount of data needed is proportional to the VC dimension, we could reach the same level of generalization error using less examples, and this is reflected in the drastic decrease in the out of sample error in a low value of N.

Code:

The initialization part (getting the dataset, and generate random indices is the same as the previous 2 questions), so the first difference occurs when we save the input vectors, as we only save the first 2 features instead of 12:

```
38          # subsubaim: convert each sample data points (which is a dictionary) to a ndarray and save in X_sample_array
39          X_sample_array = []
40          for sample in X_sample:
41              input_vector = [1, sample.get(1, 0), sample.get(2, 0)]
42              X_sample_array.append(input_vector)
```

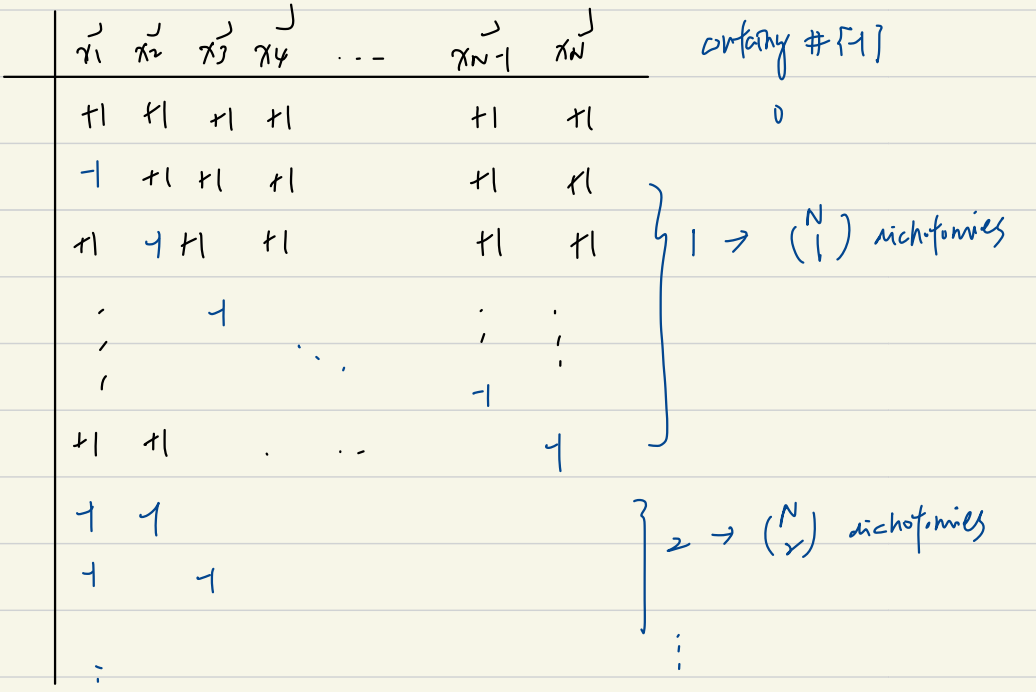
The following part is almost the same as the previous question, except modifying it to be more clear and simple, but the concept behind is all the same:

```
        # subsubaim: convert X_sample_array to a matrix, and y_sample to an array
        X_mat = np.array(X_sample_array)
        y_array = np.array(y_sample)

        # subsubaim: perform linear regression
        w = np.linalg.inv(X_mat.T @ X_mat) @ X_mat.T @ y_array

        # subsubaim: calculate the in-sample error
        in_sample_error = np.mean((X_mat @ w - y_array) ** 2)

        # subsubaim: estimate the out of sample error
        out_sample_indices = list(set(range(len(X))) - set(random_sample_indices))
        X_out = np.array([[1, X[i].get(1, 0), X[i].get(2, 0)] for i in out_sample_indices])
        y_out = np.array([y[i] for i in out_sample_indices])
        out_of_sample_error = np.mean((X_out @ w - y_out) ** 2)
        in_out_sample_error.append((in_sample_error, out_of_sample_error))

    # subaim: calculate the average in-sample error and out of sample error for each N
    avg_in_sample_error = np.mean([error[0] for error in in_out_sample_error])
    avg_out_of_sample_error = np.mean([error[1] for error in in_out_sample_error])

    avg_in_error_for_each_N.append(avg_in_sample_error)
    avg_out_error_for_each_N.append(avg_out_of_sample_error)
```

13. By the definition of $B(N,k)$, it is the max number of dichotomies on N points,

$\rightarrow$ no subset of size k of the N points can be shattered by these dichotomies

If we consider the case that we have a set S, containing dichotomies of N points that have number of $[-1]$s $\leq k-1$



Similarly, it is easy to see that we will have $\sum_{i=0}^{k-1} \binom{N}{i}$ dichotomies in S

Suppose $\exists$ subset of size k of the N points:

$$X' = \{\vec{x}_j, \vec{x}_{j+1}, \cdots, \vec{x}_{j+k-1}\} \subseteq \{\vec{x}_1, \cdots, \vec{x}_N\}$$

$\downarrow$ size k

$\nrightarrow$ X' can be shattered by these $\sum_{i=0}^{N} \binom{N}{i}$ dichotomies,

$\rightarrow$ then $[\vec{x}_j, \vec{x}_{j+1}, \cdots, \vec{x}_{j+k-1}] = [\underbrace{-1, -1, \cdots -1}_{k\{-1\}s}] \in S$ ~~$\times$~~

↑ "By our assumption, S has dichotomies with number of $\{-1\}$s $\leq k-1$"

Thus, we obtain the result:

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