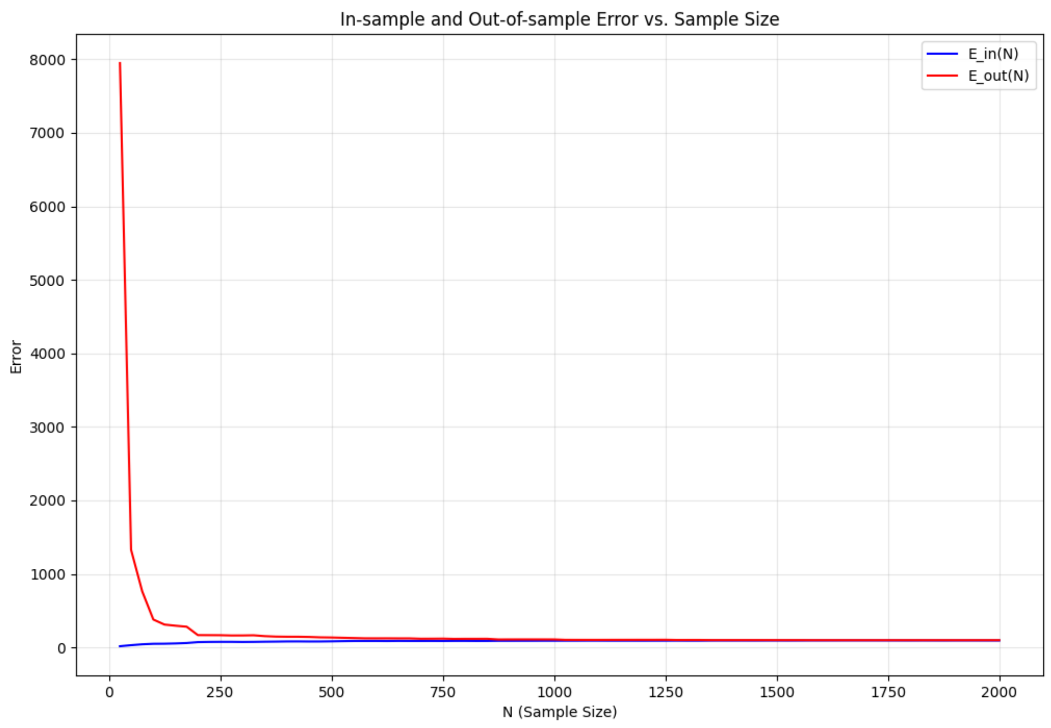
R13922136 周洛君

Question 11: Report



Explanations:

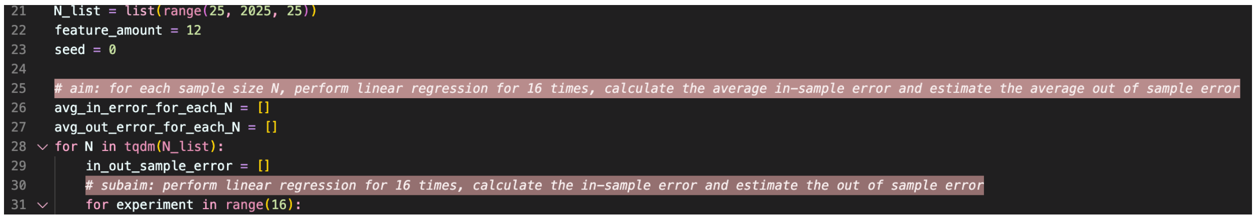
In this problem, we gradually increase the sample size (until 2000, which is nearly $\frac{1}{4}$ of our dataset), and calculate the average in sample error and out of sample error over only 16 experiments.

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.

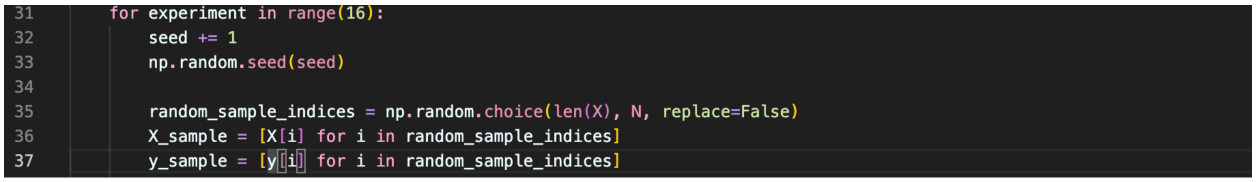
Code:

(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.



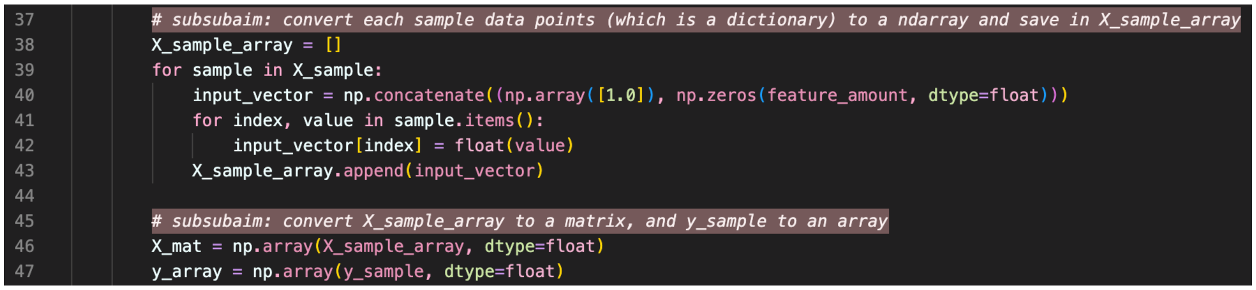
```python
N_list = list(range(25, 2025, 25))
feature_amount = 12
seed = 0

# 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:

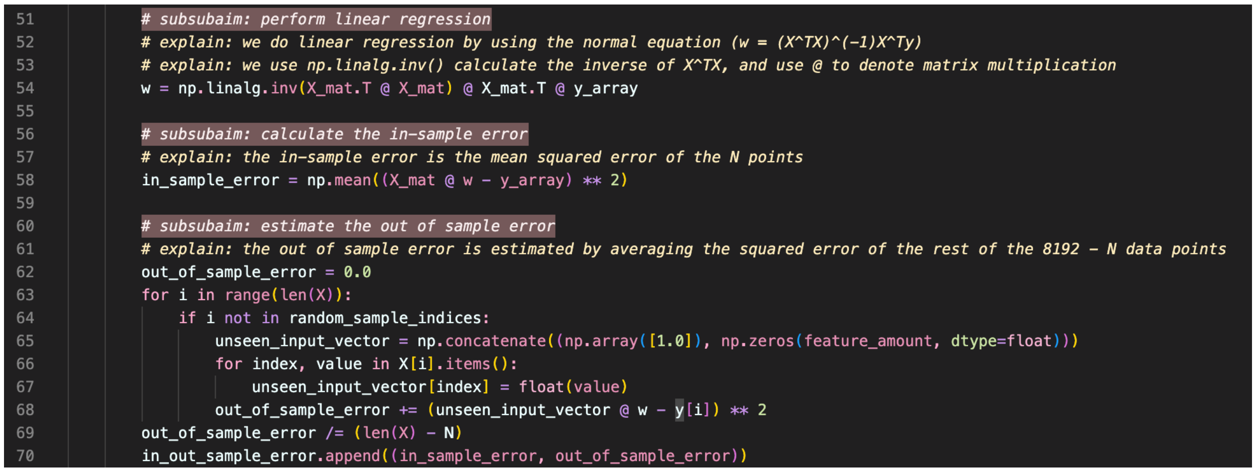


```python
    for experiment in range(16):
        seed += 1
        np.random.seed(seed)

        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:

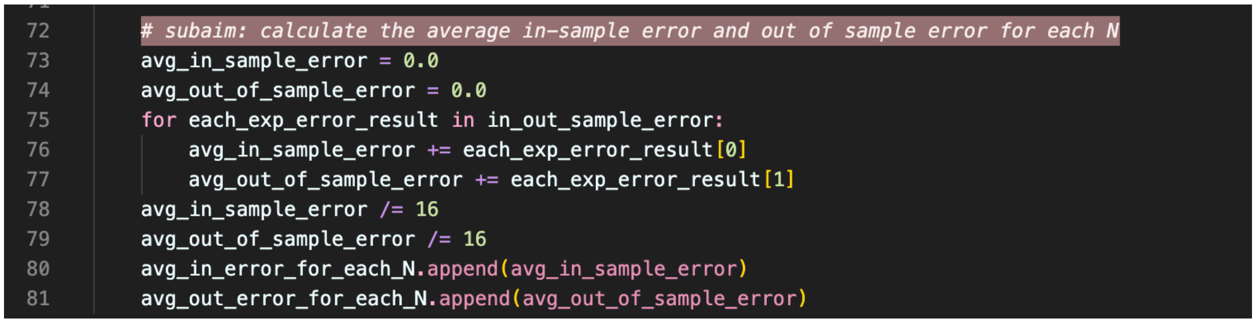

```python
        # subsubaim: convert each sample data points (which is a dictionary) to a ndarray and save in X_sample_array
        X_sample_array = []
        for sample in X_sample:
            input_vector = np.concatenate((np.array([1.0]), np.zeros(feature_amount, dtype=float)))
            for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


```python
        # 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 $\overline{E}_{in}(N)$ and $\overline{E}_{out}(N)$:


```python
    # 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.