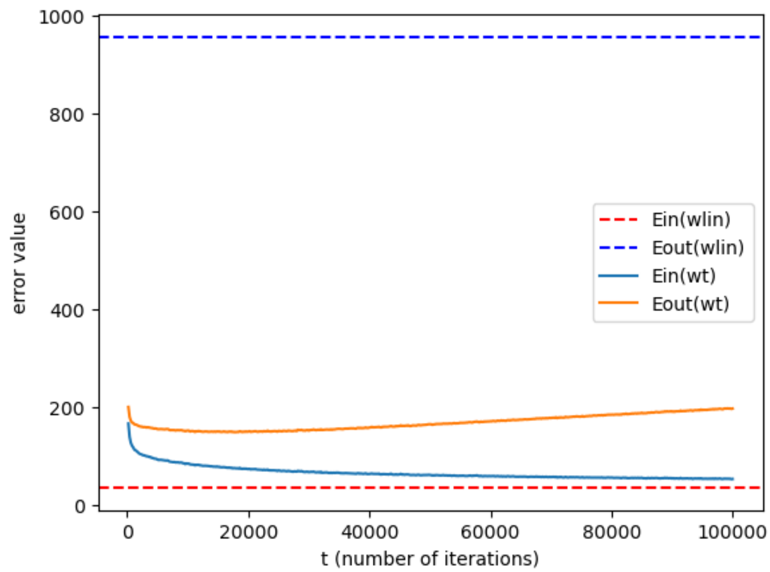
R13922136 周洛君

# ML homework 4: question 10

The resulting figure is as below:




Where the values of the in sample error and out of sample error which is not mentioned in the above plot is as below:

## Value of $E_{in}(\vec{w}_{lin})$

This value $E_{in}(\vec{w}_{lin})$ is the `avg_in_sample` in the code, and is the red dotted line in the plot.

```python
avg_in_sample
```

[521] ✓ 0.0s Python

```
35.689598963498284
```

## Value of $E_{out}(\vec{w}_{lin})$

This value $E_{out}(\vec{w}_{lin})$ is the `avg_out_sample` in the code, and is the blue dotted line in the plot.

```python
avg_out_sample
```

[522] ✓ 0.0s Python

```
956.5060504597701
```

## Meaning:

We can see that at the initial values of t, both the average in-sample error and out of sample error using SGD decreases rapidly, this dues to the fact that we initialize $w_t$ as a zero vector, which is far from the real weight vector. Therefore, at the first iterations, we make great change on the weight vector, causing great error reduction (because the gradient is large.) However, as the weight vector gets near to the optimal weight vector, the decrease of the error became smaller.

The reason why our resulting SGD has a higher in-sample error on average may due to the reason that we only use a small subset of the dataset (choose from $N = 64$ examples) to update the $100000$ iterations. This would introduce variability, and would make it hard to reach the weight vector that can have same low error as normal linear regression.

Also, another reason is that we have a closed form equation to calculate $w_{LIN}$, but we update $w_t$ randomly at each iteration, so we may not reach the exact solution as the normal regression.

However, we can see that the gap between the in-sample error and out of sample error for SGD is smaller than the normal regression, meaning that SGD provides smaller generalization error. I think it might due to the randomness we introduced in SGD, which is the reason of higher in-sample error as we mentioned. This also meet our expectation that smaller in-sample error is not better, we should not choose weight vectors that simply generates small in-sample error.

## Code:

# linear regression

In the linear regression function, we do the following steps:

1. calculate the weight vector $w_{LIN}$ by using the normal equation $w_{LIN} = (X^T X)^{-1} X^T y$

> the inverse of $X^T X$ is calculated by using `np.linalg.inv()`, and the matrix multiplication is done by using `@`

2. calculate the in-sample error

> $E_{in}(\vec{w}_{LIN}) = \frac{1}{N}(X^T \vec{w}_{LIN} - \vec{y})^2$

3. estimate the out of sample error

> calculated similarly to the in-sample error, with the data matrix and real value array changed.

```python
def linear_regression(X_in_mat, y_in_arr, X_out_mat, y_out_arr):

    w_lin = np.linalg.inv(X_in_mat.T @ X_in_mat) @ X_in_mat.T @ y_in_arr

    in_sample_error = np.mean((X_in_mat @ w_lin - y_in_arr) ** 2)
    out_of_sample_error = np.mean((X_out_mat @ w_lin - y_out_arr) ** 2)

    in_out_sample_error.append((in_sample_error, out_of_sample_error))

```

✓ 0.0s

# stochastic gradient descent (SGD)

In each iteration, we do the followings:

1. choose a random index from the list of sample indices, and save the corresponding data in `x_i` (a dict), `y_i` (a float)
2. convert the dictionary to a proper form and save in `input_vector`
3. calculate $-\nabla err(\vec{w}_t, \vec{x}_n, y_n)$ and save in variable `negative_stochastic_gradient`
4. update $w_t$

Each time after the above things are done, we check if t is a multiple of 200, if so, calculate the in sample error and out of sample error

```python
def sgd(sample_ind, X_in_mat, y_in_arr, X_out_mat, y_out_arr):

    w_t =np.zeros(12 + 1)

    for iteration in tqdm(range(1, 100001)):
        random_index = np.random.choice(sample_ind)
        x_i = X[random_index]
        y_i = y[random_index]

        input_vector = np.concatenate((np.array([1]), np.zeros(12)))
        for index, value in x_i.items():
            input_vector[index] = value

        update_direction = 2 * (y_i - w_t @ input_vector) * input_vector

        w_t += eta * update_direction

        if iteration % 200 == 0 and iteration != 0:
            error_record_index = (iteration // 200) - 1
            mult_200_E_in[error_record_index] += (np.mean((X_in_mat @ w_t - y_in_arr) ** 2))
            mult_200_E_out[error_record_index] += (np.mean((X_out_mat @ w_t - y_out_arr) ** 2))

```

✓ 0.0s

## main function

```python
for experiment in tqdm(range(1126), leave = True):

    seed = experiment
    random_sample_indices = generate_random_sample(seed)
    X_sample = [X[i] for i in random_sample_indices]
    y_sample = [y[i] for i in random_sample_indices]

    out_ind = generate_out_of_sample_ind(random_sample_indices)
    X_out_of_sample = [X[i] for i in out_ind]
    y_out_of_sample = [y[i] for i in out_ind]

    X_sample_mat = convert_dtype(X_sample)
    X_out_of_sample_mat = convert_dtype(X_out_of_sample)

    y_sample_array = np.array(y_sample)
    y_out_of_sample_array = np.array(y_out_of_sample)

    linear_regression(X_sample_mat, y_sample_array, X_out_of_sample_mat, y_out_of_sample_array)
    sgd(random_sample_indices, X_sample_mat, y_sample_array, X_out_of_sample_mat, y_out_of_sample_array)

avg_in_sample = np.mean([error[0] for error in in_out_sample_error])
avg_out_sample = np.mean([error[1] for error in in_out_sample_error])
avg_in_sample_200 = mult_200_E_in / 1126
avg_out_sample_200 = mult_200_E_out / 1126

t_values = np.arange(200, 100200, 200)
plt.axhline(y=avg_in_sample, color='r', linestyle='--', label='Ein(wlin)')
plt.axhline(y=avg_out_sample, color='b', linestyle='--', label='Eout(wlin)')
plt.plot(t_values, avg_in_sample_200, label='Ein(wt)')
plt.plot(t_values, avg_out_sample_200, label='Eout(wt)')
plt.xlabel('t (number of iterations)')
plt.ylabel('error value')
plt.legend()
plt.show()

```

[518] ✓ 59m 20.1s