Exercise 3: Regression Implementation (8 pts)

Recall that ridge regression refers to

$$\min_{\mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}} \overbrace{\underbrace{\tfrac{1}{2n}\|X\mathbf{w}+b\mathbf{1}-\mathbf{y}\|_2^2}_{\text{error}}+\lambda\|\mathbf{w}\|_2^2}^{\text{loss}}, \tag{43}$$

where $X \in \mathbb{R}^{n\times d}$ and $\mathbf{y} \in \mathbb{R}^n$ are the given dataset and $\lambda > 0$ is the regularization hyperparameter.

1. (1 pt) Show that the derivatives are

$$\frac{\partial}{\partial \mathbf{w}} = \tfrac{1}{n}X^\top(X\mathbf{w}+b\mathbf{1}-\mathbf{y})+2\lambda\mathbf{w} \tag{44}$$

$$\frac{\partial}{\partial b} = \tfrac{1}{n}\mathbf{1}^\top(X\mathbf{w}+b\mathbf{1}-\mathbf{y}). \tag{45}$$

Ans: **Answer 3.1**

2. (2 pts) Implement the gradient descent algorithm for solving ridge regression. The following incomplete pseudo-code may of help.
Test your implementation on the Boston housing dataset (to predict the median house price, i.e., y). Use the train and test splits provided on course website. Try $\lambda \in \{0, 10\}$ and report your training error, training loss and test error. [Your training loss should monotonically decrease during iteration; if not try to tune your step size η, e.g. make it smaller.]

Algorithm 1: Gradient descent for ridge regression.

```
Input: $X \in \mathbb{R}^{n\times d}$, $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{w}_0 = \mathbf{0}_d$, $b_0 = 0$, max_pass $\in \mathbb{N}$, $\eta > 0$, tol $> 0$
Output: $\mathbf{w}, b$
1 for $t = 1, 2, \ldots,$ max_pass do
2     $\mathbf{w}_t \leftarrow$
3     $b_t \leftarrow$
4     if $\|\mathbf{w}_t - \mathbf{w}_{t-1}\| \leq$ tol then            // can use other stopping criteria
5         break
6 $\mathbf{w} \leftarrow \mathbf{w}_t$, $b \leftarrow b_t$
```

Ans: **Answer 3.2**

For the next part, you may use the Python package scikit-learn.

3. (5 pts) Train (unregularized) linear regression, ridge regression, and lasso on the mystery datasets A, B, and C on the course website (using X_train and Y_train for each dataset). For ridge regression and lasso, use parameters 1 and 10 (note that you will have to divide these by n for lasso in scikit-learn -- why?). Report the average mean squared error on the test set for each method. Which approach performs best in each case? Plot the five parameter vectors obtained for each dataset on the same histogram, so they're all visible at once (change the opacity setting for the bars if necessary): specifically, for each parameter vector, plot a histogram of its value in each coordinate. Given which approach performs best, and how its parameter histogram looks, how do you suspect the true parameter vector and responses might have been generated?
Ans: **Answer 3.3**

Answer 3.1: Derivatives Derivation

Let's simplify the loss function (in Equation (43)) with some sub-group functions:

$$f_{error}(X,\mathbf{w},\mathbf{y},b) = X\mathbf{w} + b\mathbf{1} - \mathbf{y} \tag{46}$$
$$f_{reg}(\mathbf{w}) = \mathbf{w} \tag{47}$$

We will also apply partial derivative to these sub-group functions with respect to **w** and b:

$$\frac{\partial f_{error}(X,\mathbf{w},\mathbf{y},b)}{\partial \mathbf{w}} = \frac{\partial(X\mathbf{w}+b\mathbf{1}-\mathbf{y})}{\partial \mathbf{w}} \tag{48}$$
$$= \frac{\partial(X\mathbf{w})}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial(\mathbf{x}_1^T\mathbf{w})}{\partial \mathbf{w}} \\ \frac{\partial(\mathbf{x}_2^T\mathbf{w})}{\partial \mathbf{w}} \\ \vdots \\ \frac{\partial(\mathbf{x}_n^T\mathbf{w})}{\partial \mathbf{w}} \end{bmatrix} = \begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_n\end{bmatrix} = X^T \tag{49}$$
$$\frac{\partial f_{error}(X,\mathbf{w},\mathbf{y},b)}{\partial b} = \frac{\partial(X\mathbf{w}+b\mathbf{1}-\mathbf{y})}{\partial b} \tag{50}$$
$$= \frac{\partial(b\mathbf{1})}{\partial 1} = \mathbf{1}^T \tag{51}$$
$$\frac{\partial f_{reg}(\mathbf{w})}{\partial \mathbf{w}} = 1 \tag{52}$$
$$\frac{\partial f_{reg}(\mathbf{w})}{\partial b} = 0 \tag{53}$$

Hence, the loss function now becomes:

$$\ell(X,\mathbf{w},\mathbf{y},b) = \frac{1}{2n}\|f_{error}(X,\mathbf{w},\mathbf{y},b)\|_2^2 + \lambda\|f_{reg}(\mathbf{w})\|_2^2 \tag{54}$$

Now, we may apply 1st order derivatives with Chain Rule:

$$\frac{\partial \ell(X,\mathbf{w},\mathbf{y},b)}{\partial \mathbf{w}} = \frac{\partial \frac{1}{2n}\|f_{error}(X,\mathbf{w},\mathbf{y},b)\|_2^2 + \lambda\|f_{reg}(\mathbf{w})\|_2^2}{\partial \mathbf{w}} \tag{55}$$
$$= \frac{1}{2n}\cdot 2\frac{\partial f_{error}}{\partial \mathbf{w}}\cdot f_{error} + \lambda\cdot 2\cdot\frac{\partial f_{reg}}{\partial \mathbf{w}}\cdot f_{reg} \tag{56}$$
$$= \frac{1}{n}\cdot X^T\cdot(X\mathbf{w}+b\mathbf{1}-\mathbf{y}) + 2\lambda\cdot 1\cdot\mathbf{w} \tag{57}$$
$$= \tfrac{1}{n}X^\top(X\mathbf{w}+b\mathbf{1}-\mathbf{y}) + 2\lambda\mathbf{w} \tag{58}$$

$$\frac{\partial \ell(X,\mathbf{w},\mathbf{y},b)}{\partial b} = \frac{\partial \frac{1}{2n}\|f_{error}(X,\mathbf{w},\mathbf{y},b)\|_2^2 + \lambda\|f_{reg}(\mathbf{w})\|_2^2}{\partial b} \tag{59}$$
$$= \frac{1}{2n}\cdot 2\cdot\frac{\partial f_{error}}{\partial b}\cdot f_{error} + 0 \tag{60}$$
$$= \frac{1}{n}\cdot\mathbf{1}^T\cdot(X\mathbf{w}+b\mathbf{1}-\mathbf{y}) \tag{61}$$
$$= \tfrac{1}{n}\mathbf{1}^\top(X\mathbf{w}+b\mathbf{1}-\mathbf{y}) \tag{62}$$

Q.E.D.

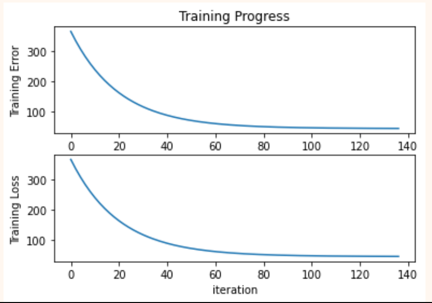
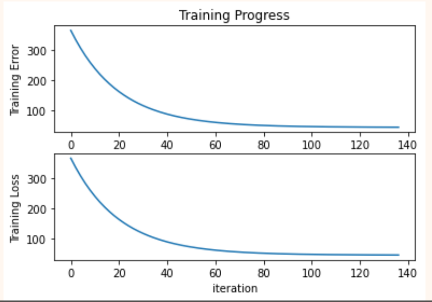
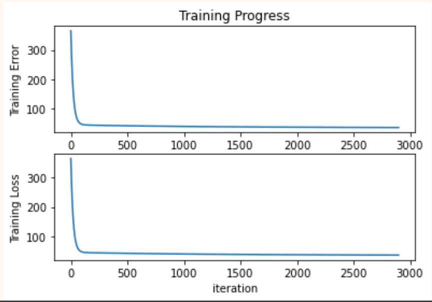
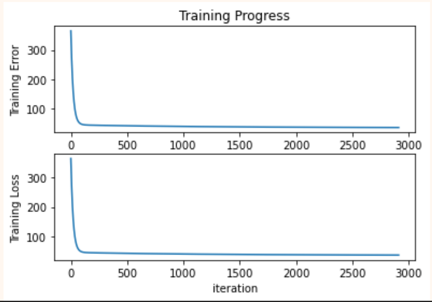
Answer 3.2

For the test, we have setup tuning parameters as following:

```
COMMON_MAX_PASS      = 10000
COMMON_ETA           = 1e-7 # >0
COMMON_TOLERANCE     = 5e-5 # >0
COMMON_ERR_TOL       = 35.8 # Disable: None
```

With the default termination criterion (which terminates when the slope of w below certain threshold), the algorithm seems to perform similarly as Trial 1 and 2 in the table below.

Hence, we modified the stopping criterion so that it terminates when training error reaches below the given tolerance. As Trial 3 and 4 in the table below shown, the trial with $\lambda = 10$ takes longer passes with better performance on test dataset with lower test error. We do observer both trial 2 and 4 has better performance than without regularization term ($\lambda = 0$).

Trial	λ	Passes	Training Progress	Training Error	Training Loss	Test Error
1	0	136		44.42208	44.42208	108.42167
2	10	136		44.42367	44.44802	108.39676
3	0	2899		35.79958	35.79958	49.89885
4	10	2911		35.79973	35.84981	49.88967

The complete algorithm is completed as shown below:

Algorithm 2: Completed Pseudocode Gradient descent for ridge regression.

```
Input: $X \in \mathbb{R}^{n\times d}$, $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{w}_0 = \mathbf{0}_d$, $b_0 = 0$, max_pass $\in \mathbb{N}$, $\eta > 0$, tol $> 0$
Output: $\mathbf{w}, b$
1 for $t = 1, 2, \ldots,$ max_pass do
2     $\mathbf{w}_t \leftarrow \frac{1}{n} X^\top (X\mathbf{w} + b\mathbf{1} - \mathbf{y}) + 2\lambda\mathbf{w}$
3     $b_t \leftarrow \frac{1}{n}\mathbf{1}^\top (X\mathbf{w} + b\mathbf{1} - \mathbf{y})$
4     if $\|\mathbf{w}_t - \mathbf{w}_{t-1}\| \le$ tol then          // can use other stopping criteria
5         break
6 $\mathbf{w} \leftarrow \mathbf{w}_t,\ b \leftarrow b_t$
```

The equivalent code implementation is as shown below:

```python
def gradient_descent_ridge_regression_training(
    X: List[List[float]],
    y: List[float],
    # Configuration with Default Settings
    max_pass: int    = 500,
    eta: float       = 0.3, # >0
    tol: float       = 0.1, # >0
    lamb: float      = 0,   # Try \in {0,10}
    error_tol: Optional[float] = None,
)-> [List[float], float, Dict]:
    """
    @param          X: \in R^{nxd}
    @param          y: \in R^n
    @param   max_pass: \in N
    @param        eta: step size
    @param        tol: tolerance
    @param       lamb: regulation weight '\lambda'
    """
    XT = X
    X = np.transpose(XT)
    y = np.array(y)
    [n, d] = np.shape(X)
    w = np.zeros(d) # w = 0_d
    b = 0
    mistake = []
    # optimization: pre-compile
    div = 1/n
    # logger
    training_log = {
        "t" : [],
        "w" : [],
        "b" : [],
        "training_error": [],
        "training_loss": [],
    }
    # training
    for t in range(0, max_pass): # max passes / iterations
        pw = copy.deepcopy(w)
        # update:
        f_err = ( np.dot(X, w) + b - y ) # pred - y
        dw = div * np.dot(XT, f_err) + 2 * lamb * w
        db = div * np.sum(f_err) # 1^T f_err
        w = w - eta * dw
        b = b - eta * db
        # compute loss and error:
        error = div / 2 * (np.linalg.norm(f_err) ** 2)
        loss = error + lamb * (np.linalg.norm(w) ** 2)
        # log progress:
        training_log["t"].append(t)
        training_log["w"].append(w)
        training_log["b"].append(b)
```

```
            training_log["training_error"].append(error)
            training_log["training_loss"].append(loss)
            # stopping criteria:
            if error_tol is None:
                if np.linalg.norm(pw - w) <= tol: # can use other stopping criteria
                    break # STOPPING
            else:
                if error <= error_tol:
                    break # STOPPING

        return w, b, training_log

```