Homework 2 – Machine Learning (CS4342, Whitehill, Spring 2023)

Linear regression for age estimation: Train an age regressor that analyzes a ($48 \times 48 = 2304$)-pixel grayscale face image and outputs a real number $\hat{y}$ that estimates how old the person is (in years). Your regressor should be implemented using linear regression. The training and testing data are available here:

- `https://s3.amazonaws.com/jrwprojects/age_regression_Xtr.npy`
- `https://s3.amazonaws.com/jrwprojects/age_regression_ytr.npy`
- `https://s3.amazonaws.com/jrwprojects/age_regression_Xte.npy`
- `https://s3.amazonaws.com/jrwprojects/age_regression_yte.npy`

Note: you must complete this problem using only linear algebraic operations in `numpy` – you may **not** use any off-the-shelf linear regression software, as that would defeat the purpose.

(a) **Analytical solution [15 points]**: Compute the optimal weights $\mathbf{w} = [w_1, \ldots, w_{2304}]^\top$ and bias term b for a linear regression model by deriving the expression for the gradient of the loss function w.r.t. $\mathbf{w}$ and b, setting it to 0, and then solving. The loss function is

$$f_{\text{MSE}}(\mathbf{w}, b) = \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)})^2$$

where $\hat{y} = g(\mathbf{x}; \mathbf{w}, b) = \mathbf{x}^\top \mathbf{w} + b$ and n is the number of examples in the training set $\mathcal{D}_{\text{tr}} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$, each $\mathbf{x}^{(i)} \in \mathbb{R}^{2304}$ and each $y^{(i)} \in \mathbb{R}$. (Note: this function is technically the *half-MSE* loss, but we omit the word "half" to avoid clutter.) After optimizing $\mathbf{w}$ and b only on the **training set**, compute and report (in the PDF file) the loss f_{MSE} on the training set $\mathcal{D}_{\text{tr}}$ and (separately) on the testing set $\mathcal{D}_{\text{te}}$. **Suggestion**: to solve for $\mathbf{w}$ and b simultaneously, use the trick shown in class whereby each image (represented as a vector $\mathbf{x}$) is appended with a constant 1 term (to yield an appended representation $\tilde{\mathbf{x}}$). Then compute the optimal $\tilde{\mathbf{w}}$ (comprising the original $\mathbf{w}$ and an appended b term) using the closed formula:

$$\tilde{\mathbf{w}} = \left(\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top\right)^{-1} \tilde{\mathbf{X}}\mathbf{y}$$

For appending, you might find the functions `np.hstack`, `np.vstack`, `np.atleast_2d` useful.

(b) **Gradient descent [20 points]**: Pick a random starting value for $\mathbf{w} \in \mathbb{R}^{2304}$ and $b \in \mathbb{R}$ and a small learning rate (e.g., $\epsilon = .001$). (In my code, I sampled each component of $\mathbf{w}$ and b from a Normal distribution with standard deviation 0.01; use `np.random.randn`). Then, using the expression for the gradient of the loss function, iteratively update $\mathbf{w}, b$ to reduce the loss $f_{\text{MSE}}(\mathbf{w}, b)$. Stop after conducting T gradient descent iterations (I suggest $T = 5000$ with a step size (aka learning rate) of $\epsilon = 0.003$). After optimizing $\mathbf{w}$ and b only on the **training set**, compute and report (in the PDF file) the loss f_{MSE} on the training set $\mathcal{D}_{\text{tr}}$ and (separately) on the testing set $\mathcal{D}_{\text{te}}$.

Note: for reasons that we will talk about in class, on this particular dataset it would take a very long time using gradient descent to reach weights as the $\mathbf{w}$ found by the analytical solution. For $T = 5000$, your training loss on part (b) will be higher than for part (a). However, the testing loss should actually be lower since the relatively small number of gradient descent steps prevents $\mathbf{w}$ from growing too large and hence acts as an implicit regularizer.

(c) **Regularization [15 points]**: Same as (b) above, but change the loss function to include a penalty for $|\mathbf{w}|^2$ growing too large:

$$f^{\text{reg}}_{\text{MSE}}(\mathbf{w}, b) = \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)})^2 + \frac{\alpha}{2n} \mathbf{w}^\top \mathbf{w}$$

where $\alpha \in \mathbb{R}^+$. Note that (1) each $\hat{y}^{(i)}$ depends implicitly on both $\mathbf{w}$ and b; and (2) this formula penalizes the squared length of $\mathbf{w}$, *not* of b. Set $\alpha = 0.1$ (this worked well for me) and then optimize $f_{\text{MSE}}^{\text{reg}}$ w.r.t. $\mathbf{w}$ and b. After optimizing $\mathbf{w}$ and b (using $f_{\text{MSE}}^{\text{reg}}$), compute and report (in the PDF file) the loss f_{MSE} (*without* the L_2 term) on the training set $\mathcal{D}_{\text{tr}}$ and (separately) the testing set $\mathcal{D}_{\text{te}}$.

Note: as mentioned above, since part (b) already provides implicit regularization by limiting the number of gradient descent steps (to $T = 5000$), you should not expect to see much (or any) difference between parts (c) and (b) *on this dataset*. In general, however, the L_2 regularization term can make a big difference.

(d) **Visualizing the machine's behavior [10 points]**: After training the regressors in parts (a), (b), and (c), create a 48×48 image representing the learned weights $\mathbf{w}$ (without the b term) from each of the different training methods. Use `plt.imshow()`. Describe in 1-2 sentences in the PDF how the weight vectors from the different methods are different. Next, using the regressor in part (c), predict the ages of all the images in the test set and report the RMSE (in years). Then, show the top 5 **most egregious errors**, i.e., the test images whose ground-truth label y is *farthest* from your machine's estimate $\hat{y}$. Include the images, along with associated y and $\hat{y}$ values, in a PDF.

Hints:

1. Make sure you are reshaping the data files I gave you correctly, so that each column of the design matrix $\tilde{\mathbf{X}}$ represents an image (as well as a "1" tacked on to the end to correspond to the bias term). To verify you have done so correctly, take the first column of the design matrix $\tilde{\mathbf{X}}$, extract just its first 2304 rows, and reshape it into a (48x48) matrix. Plot this onto the screen using `plt.imshow`. It should look like a face; otherwise, something is wrong.

2. In part (c), you will need to derive and compute the gradient of not only the MSE, but also of the L_2 regularization term $\frac{\alpha}{2n}\mathbf{w}^\top\mathbf{w}$. The gradient of the MSE was given several times in class, but the latter is something you will need to derive yourself. To figure out the general expression, suppose for the moment that $\mathbf{w}$ contains just two components, i.e., $\mathbf{w} = [w_1, w_2]^\top$. Then derive each of the two partial derivatives of the regularization term, and stack them into a column vector. What kind of structure does that column vector have? Based on the answer to that question, you should be able to deduce the expression for the gradient of the L_2 regularization term for *any* number of dimensions. Be careful: the regularization term is $\frac{\alpha}{2n}\mathbf{w}^\top\mathbf{w}$, *not* $\frac{\alpha}{2n}\tilde{\mathbf{w}}^\top\tilde{\mathbf{w}}$. In other words, the squared-magnitude of the bias term is *not* included in the regularization term.

Submission: Put your Python code in a Python file called `homework2_WPIUSERNAME.py` and put the reported accuracy information and error analysis in `homework2_errors_WPIUSERNAME.pdf`. **Important note**: when reporting MSE costs, it is ok to report either the true MSE or the half-MSE, but **make sure you say which metric you are using** at the top of your PDF file.