Homework 2 - Sept. 22

Deadline: Monday, Oct. 6, at 11:59pm.

Submission: You need to submit through Crowdmark (you can find the link on Quercus) with your answers to Questions 1 and 2. You will upload your answers to each subquestion separately. You can produce the submissions however you like (e.g. LaTeX, Microsoft Word, scanner), as long as they are readable.

Marking: Your mark will be out of 7.5. This mark will be your mark on *either* question 1 *or* question 2. We will decide which question to mark after the deadline, and we will mark the same question for everyone in the class. To aid your learning, we will be releasing the solutions to both questions, and covering the solutions in office hours.

Neatness: We reserve the right to deduct a point for neatness, if we have a hard time reading your solutions or understanding the structure of your code.

Late Submission: 10% of the total possible marks will be deducted for each day late, up to a maximum of 3 days. After that, no submissions will be accepted.

Computing: To install Python and required libraries, see the instructions on the course web page.

Other Policies: See the syllabus[1] for detailed collaboration, generative AI, and academic integrity policies.

1. **[7.5pts] Linear Regression.** We will consider a special case of the linear regression setting that we covered in lecture. Consider the following data generating distribution in which $\mathbf{x}$ is a multivariate Gaussian in D dimensions and the label t is a linear function of $\mathbf{x}$ with one-dimensional Gaussian noise. Let $\mathbf{w}^* \in \mathbb{R}^D$ be fixed,

$$\mathbf{x} \sim \mathcal{N}(\mathbf{x} \mid \mathbf{0}, I) \text{ and } \epsilon \sim \mathcal{N}(\epsilon \mid 0, 1) \text{ independent} \tag{0.1}$$

$$t = \mathbf{x}^\top \mathbf{w}^* + \epsilon \tag{0.2}$$

Consider the hypothesis class of linear regression without a bias term. That is, any predictor

$$y(\mathbf{x}) = \mathbf{x}^\top \mathbf{w} \text{ with } \mathbf{w} \in \mathbb{R}^D. \tag{0.3}$$

   (a) **[4pt] Expected Loss.** Recall that the expected loss of linear regression is the expected squared error between a prediction and a label, where the expectation is taken over a random input and a random label (not over the predictor at this stage). We can identify a predictor with vectors $\mathbf{w} \in \mathbb{R}^D$, so we can write the expected loss as a function of $\mathbf{w}$:

$$\mathcal{R}[\mathbf{w}] = \frac{1}{2}\mathbb{E}\left[(t - \mathbf{x}^\top \mathbf{w})^2\right] \tag{0.4}$$

Prove the following, where $\|\mathbf{w}\|^2 = \sum_{d=1}^D w_d^2$ is the squared Euclidean norm.

$$\mathcal{R}[\mathbf{w}] = \frac{1}{2}\|\mathbf{w} - \mathbf{w}^*\|^2 + \frac{1}{2} \tag{0.5}$$

Hint: you can use the fact that $\mathbb{E}[x_i x_j] = 1$ *if* $i = j$ *and* 0 *otherwise.*

[1] https://www.cs.toronto.edu/~cmaddis/courses/sta314_f25/sta314_f25_syllabus.pdf

(b) **[2.5pt] Performance of Linear Regression.** In this question you will study the expected performance of linear regression. Recall from lecture that the weights that minimize the *average training loss* on a training dataset $\{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^{N}$ are given by

$$\hat{\mathbf{w}}^* = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{t} \tag{0.6}$$

where

$$\mathbf{X} = \begin{pmatrix} (\mathbf{x}^{(1)})^\top \\ (\mathbf{x}^{(2)})^\top \\ \vdots \\ (\mathbf{x}^{(N)})^\top \end{pmatrix} \text{ and } \mathbf{t} = \begin{pmatrix} t^{(1)} \\ t^{(2)} \\ \vdots \\ t^{(N)} \end{pmatrix} \tag{0.7}$$

are the design matrix and target vector, respectively. We assume that $N > D + 1$, which ensures that $\mathbf{X}^\top\mathbf{X}$ is invertible almost surely, which basically means it's always invertible.

The trained linear regression predictor in this case is given by $\hat{y}^*(\mathbf{x}) = \mathbf{x}^\top \hat{\mathbf{w}}^*$. Using the result from the previous question, we can evaluate the expected loss of $\hat{y}^*$,

$$\mathcal{R}[\hat{\mathbf{w}}^*] = \frac{1}{2}\|\hat{\mathbf{w}}^* - \mathbf{w}^*\|^2 + \frac{1}{2}. \tag{0.8}$$

Note that $\mathcal{R}[\hat{\mathbf{w}}^*]$ is random because $\hat{\mathbf{w}}^*$ is random. Now, the performance of linear regression the we can expect on average in our problem setting is the expectation $\mathbb{E}[\mathcal{R}[\hat{\mathbf{w}}^*]]$, taken over the randomness in the training set.

Prove that

$$\mathbb{E}[\mathcal{R}[\hat{\mathbf{w}}^*]] = \frac{1}{2}\frac{D}{N-D-1} + \frac{1}{2} \tag{0.9}$$

where the expectation is taken over $\hat{\mathbf{w}}^*$.

Hint: you can use the following result without proof. Let

$$\mathbf{X} = \begin{pmatrix} (\mathbf{x}^{(1)})^\top \\ (\mathbf{x}^{(2)})^\top \\ \vdots \\ (\mathbf{x}^{(N)})^\top \end{pmatrix} \textit{ and } \mathbf{e} = \begin{pmatrix} \epsilon^{(1)} \\ \epsilon^{(2)} \\ \vdots \\ \epsilon^{(N)} \end{pmatrix} \tag{0.10}$$

be such that $\mathbf{x}^{(i)} \sim \mathcal{N}(\mathbf{x} \mid \mathbf{0}, I)$ *and* $\epsilon^{(i)} \sim \mathcal{N}(\epsilon \mid 0, 1)$ *are all mutually independent. Then*

$$\mathbb{E}\left[\left\|(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{e}\right\|^2\right] = \frac{D}{N-D-1}. \tag{0.11}$$

(c) **[1pt] The Curse of Dimensionality Revisited.** One way to improve a prediction is to add a dimension to the input $\mathbf{x}$. For example if you were trying to predict lifespan (the label) from height (a one-dimensional input) using linear regression, you might find that your predictions are OK but not great. So, you may decide to add a dimension to the input and instead predict lifespan (the label) from height and weight (now a two-dimensional input). It seems like this can only help because now you have more information in the input that might tell you something about the label.

This gives you an idea: you decide to very aggressively add dimensions to the input until $D = N - 2$, where N is the size of your training set. Explain why this might be a bad idea.

2. **[7.5pts] Robust Regression.** One problem with linear regression using squared error loss is that it can be sensitive to outliers. This can be a problem if the training set does not have the same distribution as the validation or testing sets. We will explore this scenario in this question.

   We can use different loss functions to make training robust to outliers. Recall that an outlier is a data point that differs significantly from other observations. In our context, we will consider a few targets $t^{(i)}$ that are outliers in the sense they are drawn from a conditional $p(t|\mathbf{x})$ that is distinct from one used in the validation set and potentially with much larger variance. To cope with this, we will use the *Huber loss*, parameterized by a hyperparameter $\delta > 0$:

$$\begin{aligned} L_\delta(y,t) &= H_\delta(y-t) \\ H_\delta(a) &= \begin{cases} \frac{1}{2}a^2 & \text{if } |a| \le \delta \\ \delta(|a| - \frac{1}{2}\delta) & \text{if } |a| > \delta \end{cases} \end{aligned}$$

   (a) **[1pt]** Sketch the Huber loss $L_\delta(y,t)$ and squared error loss $L_{\mathrm{SE}}(y,t) = \frac{1}{2}(y-t)^2$ for $t = 0$, either by hand or using a plotting library. Based on your sketch, why would you expect the Huber loss to be more robust to a label $t^{(i)}$ that is an outlier?

   (b) **[2pt]** Just as with linear regression, assume a linear model *without a bias*:

$$y(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}.$$

   As usual, the cost is the average loss over the training set:

$$\hat{\mathcal{R}} = \frac{1}{N}\sum_{i=1}^{N} L_\delta(y^{(i)}, t^{(i)}).$$

   Derive a sequence of vectorized mathematical expressions for the gradients of the cost (averaged over a training set) with respect to $\mathbf{w}$. Recall that the inputs are organized into a design matrix

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)\top} \\ \vdots \\ \mathbf{x}^{(N)\top} \end{pmatrix}$$

   with one row per training example and recall *there is no bias term.* The expressions should be something you can translate into a Python program without requiring a `for`-loop. Your answer should look like:

$$\begin{aligned} \mathbf{y} &= \cdots \\ \frac{\partial \hat{\mathcal{R}}}{\partial \mathbf{y}} &= \cdots \\ \frac{\partial \hat{\mathcal{R}}}{\partial \mathbf{w}} &= \cdots \end{aligned}$$

   We recommend you find a formula for the derivative $H'_\delta(a)$. Then give your answers in terms of $H'_\delta(\mathbf{y} - \mathbf{t})$, where we assume that H'_δ is applied point-wise to the vector $\mathbf{y} - \mathbf{t}$.

Remember that $\partial\hat{\mathcal{R}}/\partial\mathbf{w}$ denotes the gradient vector,

$$\frac{\partial\hat{\mathcal{R}}}{\partial\mathbf{w}} = \begin{pmatrix} \frac{\partial\hat{\mathcal{R}}}{\partial w_1} \\ \vdots \\ \frac{\partial\hat{\mathcal{R}}}{\partial w_D} \end{pmatrix}$$

(c) **[2pt]** We have provided Python starter code to perform gradient descent on this model and you need to write two functions. Complete the function `robust_regression_grad`, which computes the gradients of the robust regression model for a weight vector $\mathbf{w}$. You should be able to read the expected parameters and return values from the docstring. You will want find the functions `np.where`, `np.abs`, `np.dot`, `np.shape`, and `np.sign`. You may submit your code as a screenshot or a PDF with the body of your function.

(d) **[2pt]** Complete the function `optimization`. This function initializes a weight vector at 0 and runs gradient descent for `num_iterations`. There is no need to modify `num_iterations` nor the initialization of `w`. You should use your function `robust_regression_grad` in this function. You may submit your code as a screenshot or a PDF with the body of your function.

(e) **[0.5pt]** We provided a script that tries 5 different δ values of the Huber loss for training and reports validation losses. For this experiment, we generated a dataset in which the training set has target values $t^{(i)}$ that are outliers, i.e., some small subset of training points are *not* i.i.d. with the validation set and the noise that we add these is much larger. You can see how we generated the data in `q2_data.py`.

Run the script `q2.py` (or the equivalent notebook). The model in the script is *trained on the Huber loss using the training set*, which is robust to these outliers, but we report the *standard squared error loss on the validation and training sets.* In sum we report:

i. the average squared error on the validation set of a linear regression model
ii. for each δ, the average squared error on the validation set of a robust regression model trained with the Huber loss
iii. for each δ, the average squared error on the training set of a robust regression model trained with the Huber loss

If you implemented your functions correctly, you should see that the training squared error of the robust model goes down as δ increases and approaches the loss of the linear regression (not robust) model. On the other hand, you should see that there is an optimal δ value for the validation squared error. Why do you think this is? Answer this question briefly in a few sentences.