Homework 2 - V2 Oct. 3

Deadline: Thursday, Oct. 14, at 11:59pm.

Submission: You need to submit one or two files through Quercus with our answers to Questions 1, 2, and 3 as well as code requested for Question 2. You can submit a single PDF titled `hw2_writeup.pdf` with screenshots of your code. You may also upload PDF titled `hw2_writeup.pdf` and a Python file title `q2.py`. You can produce the PDF file however you like (e.g. LaTeX, Microsoft Word, scanner), as long as it is readable.

Neatness Point: One point will be given for neatness. You will receive this point as long as we don't 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.

Homeworks are to be done alone or in pairs. See the Course Information handout[1] for detailed policies.

1. **[2pts] Linear Models.** Recall that a function $f : \mathbb{R}^D \to \mathbb{R}$ is linear if both of the following conditions hold.

   - For all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^D$, $f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$.
   - For all $\mathbf{x} \in \mathbb{R}^D$ and $a \in \mathbb{R}$, $f(a\mathbf{x}) = af(\mathbf{x})$.

   Consider the hypothesis class of linear regression. That is, any predictor

   $$y(x) = \mathbf{w}^\top \mathbf{x} \text{ with } \mathbf{w} \in \mathbb{R}^D. \tag{0.1}$$

   Prove that every linear regression predictor is a linear function. You may use the definition of $\mathbf{w}^\top \mathbf{x} = \sum_j w_j x_j$ as well as any basic fact about arithmetic without proof.

2. **[8pts] 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 $\delta > 0$:

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

[1] https://www.cs.toronto.edu/~cmaddis/courses/sta314_f21/sta314_f21_syllabus.pdf

(a) **[1pt]** Sketch the Huber loss $L_\delta(y,t)$ and squared error loss $L_{\text{SE}}(y,t) = \frac{1}{2}(y-t)^2$ 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:

$$\mathbf{y} = \cdots$$
$$\frac{\partial \hat{\mathcal{R}}}{\partial \mathbf{y}} = \cdots$$
$$\frac{\partial \hat{\mathcal{R}}}{\partial \mathbf{w}} = \cdots$$

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 the completed script `q2.py`.

(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 the completed script `q2.py`.

(e) **[1pt]** 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.

3. **[4pts] Feature Maps.** A 1-D binary classification training set $\{(x^{(i)}, t^{(i)})\}_{i=1}^{N}$ with $x^{(i)} \in \mathbb{R}$ and $t^{(i)} \in \{0, 1\}$ is linear separable if there exists a threshold $a \in \mathbb{R}$ such that

$$x^{(i)} < a \text{ for all } t^{(i)} = 0$$
$$x^{(i)} \geq a \text{ for all } t^{(i)} = 1$$

(a) **[2pts]** Suppose we have the following 1-D dataset for binary classification:

$x^{(i)}$	$t^{(i)}$
-1	1
1	0
3	1

Argue briefly (at most a few sentences) that this dataset is not linearly separable. *Hint: Consider any threshold a that correctly classifies both $t^{(i)} = 1$ examples and derive a contradiction. Your argument can resemble the one we used in lecture to prove XOR is not linearly separable.*

(b) **[2pts]** Now suppose we apply the feature map

$$\boldsymbol{\psi}(x) = \begin{pmatrix} \psi_1(x) \\ \psi_2(x) \end{pmatrix} = \begin{pmatrix} x \\ x^2 \end{pmatrix}.$$

Assume we have no bias term, so that the parameters are w_1 and w_2. Write down the constraint on w_1 and w_2 corresponding to each training example, and then find a pair of values (w_1, w_2) that correctly classify all the examples. Remember that there is no bias term.