CSC2515: Homework #3

Due on October 12

Roger Grosse

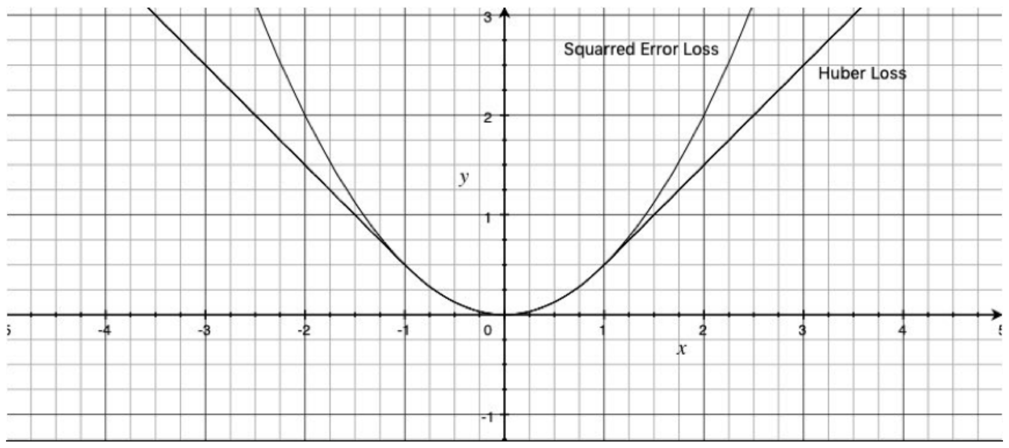
Thomas Hollis

Problem 1

Sketch the Huber loss $L_\delta(y, t)$ and squared error loss $L_{SE}(y, t) = \frac{1}{2}(y - t)^2$ 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 outliers?

Solution

The graph for Squared Error loss and Huber loss ($\delta = 1$) is shown below. In both cases, t is set to 0.



From the above graph we can easily see that Huber loss is resistant to outliers. Indeed, this is because as the predictions get increasingly extreme (i.e. increasingly close to positive or negative infinity) the penalty does not increase as fast as for other loss functions, such as squared error loss. This is because, for outliers, penalty increases linearly for Huber loss but it increases quadratically for squared error loss.

Problem 2

Just as with linear regression, assume a linear model: $y = \mathbf{w}^T\mathbf{x} + b$. Give formulas for the partial derivatives $\frac{\partial L_\delta}{\partial \mathbf{w}}$ and $\frac{\partial L_\delta}{\partial b}$.

Solution

We know that:

$$y = \mathbf{w}^T\mathbf{x} + b \tag{1}$$

Hence:

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

$$H_\delta^{'}(a) = \begin{cases} a & |a| \le \delta \\ \delta \operatorname{sgn}(a) & |a| > \delta \end{cases} \tag{3}$$

Since we have taken the derivative of a piecewise function we must evaluate the limits of the function at the crossover points ($\pm\delta$) to ensure that it is indeed defined:

$$\lim_{a\to\delta} a = \delta \tag{4}$$

$$\lim_{a\to-\delta} a = -\delta \tag{5}$$

$$\lim_{a\to\delta} \delta \operatorname{sgn}(a) = \delta \tag{6}$$

$$\lim_{a\to-\delta} \delta \operatorname{sgn}(a) = -\delta \tag{7}$$

Therefore, as it is defined at the crossover points, we can say:

$$H_\delta^{'}(y-t) = \begin{cases} y-t & |y-t| \le \delta \\ \delta \operatorname{sgn}(y-t) & |y-t| > \delta \end{cases} \tag{8}$$

Therefore:

$$\frac{\partial L_\delta}{\partial \mathbf{w}} = \frac{\partial L_\delta}{\partial y}\frac{\partial y}{\partial \mathbf{w}} = H_\delta^{'}(y-t)\cdot\mathbf{x} \tag{9}$$

$$\frac{\partial L_\delta}{\partial b} = \frac{\partial L_\delta}{\partial y}\frac{\partial y}{\partial b} = H_\delta^{'}(y-t) \tag{10}$$

Problem 3

Write Python code to perform (full batch mode) gradient descent on this model.

Solution

See the code in `q1.py`.

Problem 4

Show that the solution to the weighted least squares problem:

$$\mathbf{w}^* = \arg\min \frac{1}{2}\sum_{i=1}^{N} a^{(i)}\left(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\right)^2 + \frac{\lambda}{2}||\mathbf{w}||^2 \tag{11}$$

is given by the formula:

$$\mathbf{w}^* = (\mathbf{W}^T\mathbf{A}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{A}\mathbf{y} \tag{12}$$

Solution

The solution to the weighted least squares problem (11) is found by differentiating the summation inside the argmin expression for $\mathbf{w}^*$ in (11) and setting it to zero. This is because the minimum point will occur at a part of the function where the gradient is zero. Thus:

$$\frac{\partial}{\partial w_j}\left[\frac{1}{2}\sum_{i=1}^{N} a^{(i)}\left(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\right)^2 + \frac{\lambda}{2}||\mathbf{w}||^2\right] \tag{13}$$

Since the derivative of the sum is the same as the sum of derivatives, thus (13) simplifies to:

$$\frac{1}{2}\sum_{i=1}^{N}\left[2a^{(i)}(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}) \cdot -\mathbf{x}^{(i)}\right] + \lambda w_j \tag{14}$$

Expanding out (14) yields:

$$\frac{1}{2}\sum_{i=1}^{N}\left[-2a^{(i)}y^{(i)}\mathbf{x}^{(i)} + 2a^{(i)}\mathbf{w}^T\left(\mathbf{x}^{(i)}\right)^2\right] + \lambda w_j \tag{15}$$

Simplifying and splitting the sum of (15) yields:

$$\sum_{i=1}^{N}\left[a^{(i)}\mathbf{w}^T\left(\mathbf{x}^{(i)}\right)^2\right] - \sum_{i=1}^{N}\left[a^{(i)}y^{(i)}\mathbf{x}^{(i)}]\right] + \lambda w_j \tag{16}$$

Thus generalising (16) for the full vector $\mathbf{w}$ and setting the derivative to 0 yields:

$$0 = \sum_{i=1}^{N}\left[a^{(i)}\mathbf{w}^T\left(\mathbf{x}^{(i)}\right)^2\right] - \sum_{i=1}^{N}\left[a^{(i)}y^{(i)}\mathbf{x}^{(i)}]\right] + \lambda\mathbf{w} \tag{17}$$

Thus, the $\mathbf{w}$ that satisfies equation (17), denoted here $\mathbf{w}^*$, is the solution to equation (11). Therefore by vectorising equation (17), we get:

$$0 = \mathbf{X}^T\mathbf{A}\mathbf{X}\mathbf{w}^* - \mathbf{X}^T\mathbf{A}\mathbf{y} + \lambda\mathbf{I}\mathbf{w}^* \tag{18}$$

Therefore factorising and rearranging (18) for $\mathbf{w}^*$ yields:

$$\mathbf{w}^* = (\mathbf{W}^T\mathbf{A}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{A}\mathbf{y} \tag{19}$$

Since the least squares function is convex, the second derivative of (13) will be positive so this value of $\mathbf{w}^*$ is indeed the local minimum. Hence, we have shown that equation (12) is indeed the solution of the weighted least squares problem shown in (11).

Problem 5

Complete the implementation of locally reweighted least squares by providing the missing parts for `q2.py`.

Solution

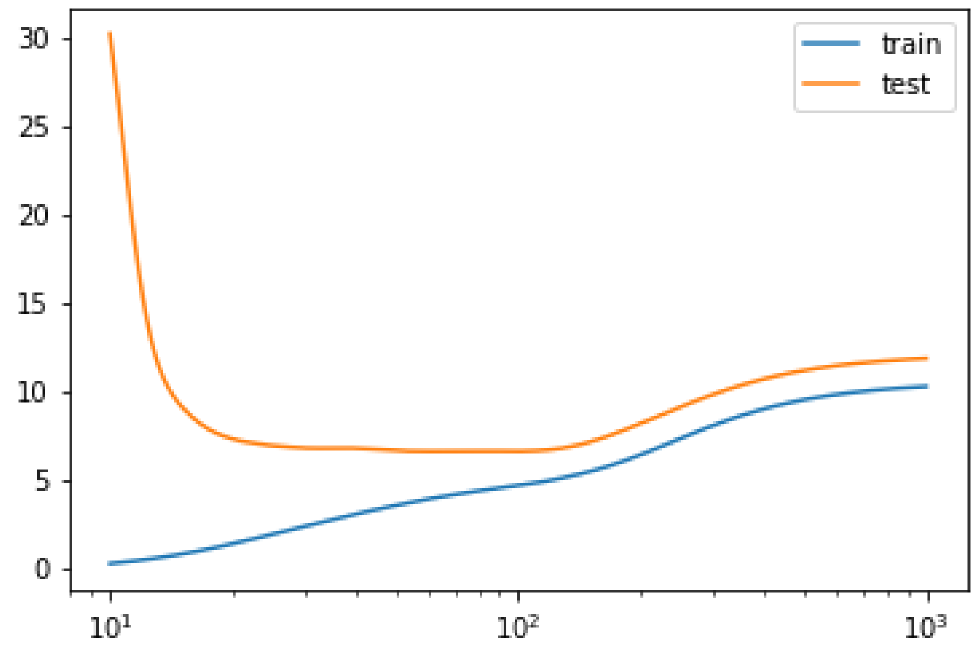
See the code in `q2.py`.

Problem 6

Randomly hold out 30% of the dataset as a validation set. Compute the average loss for different values of τ in the range [10,1000] on both the training set and the validation set. Plot the training and validation losses as a function of τ (using a log scale for τ).

Solution

See the code in `q2.py`. The plot of training and validation losses as a function of τ is show below.



Problem 7

How would you expect this algorithm to behave as $\tau \to \infty$? When $\tau \to 0$? Is this what actually happened?

Solution

I expect that this algorithm would severely underfit as τ tends to infinity (fail to fit in both test and training data) and severely overfit as τ tends to 0 (i.e. perfect performance in training data but will not generalise to validation data). This is indeed what was observed in the graph of the previous question.

This is because as τ increases to infinity, all the weights a decrease to $1/n$. Thus all the weights are worth the same for all the points being compared to the test point. So the performance is poor as each iteration is worth the same when clearly the closer iterations should be weighed more heavily.

Similarly, as τ tends to 0, the first closest point is given an excessively large weight and the subsequent points decrease in weight so much that they become essentially meaningless. Thus the algorithm will predict perfectly the training points but fail to generalise for validation points.