Csc 2515 Fall 2018

Homework 3

Soon Chee Loong
999295793
cheeloong.soon@mail.utoronto.ca
Last Name: Soon
FirstName: Chee Loong
cdf markus: soonchee

1. Robust Regression

↳ more robust to outliers

Huber Loss

$$L_\delta(y,t) = H_\delta(y-t)$$

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

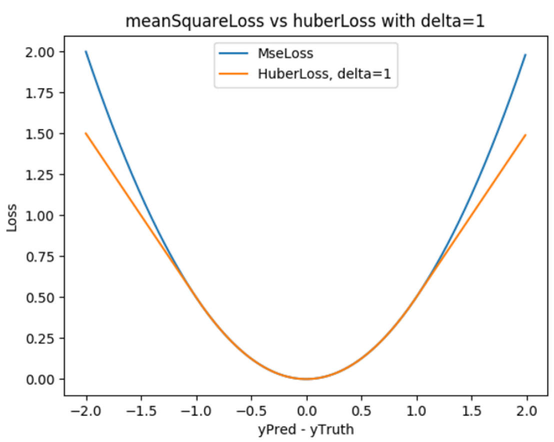
a) Sketch the Huber loss $L_\delta(y,t)$ and squared error loss $L_{SE}(y,t) = \frac{1}{2}(y-t)^2$ for $t=0$.

Based on sketch, why would you expect Huber Loss to be more robust to outliers?

Huber Loss trades off between sensitivity quadratic loss for strongly convex optimization with the robustness of absolute value loss towards outliers.

δ is a hyperparameter, $\delta \ge 0$ that becomes more robust as we reduce it since it changes to absolute value loss quicker.

I used $t=0$, $\delta=1$.



It is more robust as it is less sensitive to outliers that are far away.

b) Just as with linear regression, assume linear model

$y = w^T \cdot x + b$, $Y = \vec{X} \cdot \vec{w} + b$, $\vec{X}$ is design matrix

Calculate partial derivatives $\frac{\partial L_\delta}{\partial w}$ and $\frac{\partial L_\delta}{\partial b}$.

Hint: Find formula for derivative $H_\delta'(a)$, then give answers in terms of $H_\delta'(y-t)$.

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

$$= \begin{cases} \frac{1}{2}a & \text{if } |a| \le \delta \\ \delta\left(a - \frac{\delta}{2}\right) & \text{if } |a| > \delta,\ a \ge 0 \\ \delta\left(-a - \frac{\delta}{2}\right) & \text{if } |a| > \delta,\ a < 0 \end{cases}$$

b) Just as with linear regression, assume linear model

$y = w^T \cdot x + b$, $Y = \vec{X} \cdot \vec{w} + b$, $\vec{X}$ is design matrix

Calculate partial derivatives $\frac{\partial L_\delta}{\partial w}$ and $\frac{\partial L_\delta}{\partial b}$.

Hint: Find formula for derivative $H_\delta'(a)$, then give answers in terms of $H_\delta'(y-t)$.

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

$$= \begin{cases} \frac{1}{2}a & \text{if } |a| \le \delta \\ \delta\left(a - \frac{\delta}{2}\right) & \text{if } |a| > \delta,\ a \ge 0 \\ \delta\left(-a - \frac{\delta}{2}\right) & \text{if } |a| > \delta,\ a < 0 \end{cases}$$

$a = y - t$

$y = \vec{w}^T \cdot \vec{x} + b = \sum_{i=1}^{d} w_i x_i + b$, d = dimensionality of $\vec{x}, \vec{w}$, $\vec{x}, \vec{w} \in \mathbb{R}^d$

$L_\delta(y, t) = H_\delta(y-t) = H_\delta(a)$

From Chain Rule,

$$\frac{dL_\delta}{d\vec{w}} = \frac{d\,H_\delta(a)}{d\vec{w}} = \left(\frac{d\,H_\delta(a)}{da}\right)\left(\frac{da}{d\vec{w}}\right) = \left(\frac{d\,H_\delta(a)}{da}\right)\left(\frac{da}{dy}\right)\left(\frac{dy}{d\vec{w}}\right)$$

$$\frac{dL_\delta}{db} = \frac{d\,H_\delta(a)}{db} = \left(\frac{d\,H_\delta(a)}{da}\right)\left(\frac{da}{db}\right) = \left(\frac{d\,H_\delta(a)}{da}\right)\left(\frac{da}{dy}\right)\left(\frac{dy}{db}\right)$$

$$H_\delta'(a) = \frac{d\,H_\delta'(a)}{da} = \begin{cases} a, & \text{if } |a| \le \delta \\ \delta, & \text{if } |a| > \delta,\ a > 0 \\ -\delta, & \text{if } |a| > \delta,\ a < 0 \end{cases}$$

$$a' = \frac{da}{dy} = 1$$

$$\frac{dy}{d\vec{w}} = \begin{bmatrix} \frac{dy}{dw_1} \\ \vdots \\ \frac{dy}{dw_d} \end{bmatrix}_{(d \times 1)} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}_{(d \times 1)} = \vec{x}$$

$$\frac{dy}{db} = 1$$

$\therefore$,

$$\frac{dL_\delta}{d\vec{w}} = \left(\frac{d\,H_\delta(a)}{da}\right)\left(\frac{da}{dy}\right)\left(\frac{dy}{d\vec{w}}\right)$$

$$= \begin{cases} a(1)\,(\vec{x}), & \text{if } |a| \le \delta \\ \delta\,(1)\,(\vec{x}), & \text{if } |a| > \delta,\ a > 0 \\ -\delta\,(1)\,(\vec{x}), & \text{if } |a| > \delta,\ a < 0 \end{cases}$$

$$= \begin{cases} a(\vec{x}), & \text{if } |a| \le \delta \\ \delta\,(\vec{x}), & \text{if } |a| > \delta,\ a > 0 \\ -\delta\,(\vec{x}), & \text{if } |a| > \delta,\ a < 0 \end{cases}$$

$$\frac{dL_\delta}{db} = \left(\frac{d\,H_\delta(a)}{da}\right)\left(\frac{da}{dy}\right)\left(\frac{dy}{\ }\right)$$

$\therefore$,

$$\frac{dL_\delta}{d\vec{w}} = \left(\frac{dH_\delta(a)}{da}\right)\left(\frac{da}{dy}\right)\left(\frac{dy}{d\vec{w}}\right)$$

$$= \begin{cases} a(1)(\vec{x}) & , \text{ if } |a| \le \delta \\ \delta(1)(\vec{x}) & , \text{ if } |a| > \delta, \ a>0 \\ -\delta(1)(\vec{x}) & , \text{ if } |a| > \delta, \ a<0 \end{cases}$$

$$= \begin{cases} a(\vec{x}) & , \text{ if } |a| \le \delta \\ \delta(\vec{x}) & , \text{ if } |a| > \delta, \ a>0 \\ -\delta(\vec{x}) & , \text{ if } |a| > \delta, \ a<0 \end{cases}$$

$$\frac{dL_\delta}{db} = \left(\frac{dH_\delta(a)}{da}\right)\left(\frac{da}{dy}\right)\left(\frac{dy}{db}\right)$$

$$= \begin{cases} a(1)(1) & , \text{ if } |a| \le \delta \\ \delta(1)(1) & , \text{ if } |a| > \delta, \ a>0 \\ -\delta(1)(1) & , \text{ if } |a| > \delta, \ a<0 \end{cases}$$

$$= \begin{cases} a & , \text{ if } |a| \le \delta \\ \delta & , \text{ if } |a| > \delta, \ a>0 \\ -\delta & , \text{ if } |a| > \delta, \ a<0 \end{cases}$$ $\Rightarrow$ piecewise derivative

$$= \begin{cases} a & , \text{ if } |a| \le \delta \\ \delta & , \text{ if } |a| > \delta, \ a>0 \\ -\delta & , \text{ if } |a| > \delta, \ a<0 \end{cases}$$ $\Rightarrow$ piecewise derivative

note: Vectorized gradients were implemented directly in code as the average between the gradients $\forall$ training data

c) Write python code to perform full batch gradient descent.
Initialize w and b to zeros
Assume dataset is given as design matrix X and target vector y.
No for loops over training examples or input dimensions.
Hint: use np.where

Since gradients for Huber Loss are piecewise,
np.where() is useful for figuring out which piece each training instance is in.

Submitted as q1.py

2. Locally Weighted Regression

a) Given $\{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\}$ and positive weights $a^{(1)}, \dots, a^{(N)}$.
Show that the solution to the weighted least square problem

$$\vec{w}^* = \underset{\vec{w}}{\arg\min} \ \frac{1}{2}\sum_{i=1}^{N} a^{(i)}(y^{(i)} - \vec{w}^T\vec{x}^{(i)})^2 + \frac{\lambda}{2}\|\vec{w}\|^2$$

is given by formula

$$\vec{w}^* = (\vec{X}^T \cdot \vec{A} \cdot \vec{X} + \lambda I)^{-1}\vec{X}^T\vec{A}y$$

$X =$ Design Matrix

2. Locally Weighted Regression

a) Given $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$ and positive weights $a^{(1)}, \ldots, a^{(N)}$.

Show that the solution to the weighted least square problem

$$\vec{w}^* = \underset{w}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)})^2 + \frac{\lambda}{2} \|\vec{w}\|^2$$

is given by formula

$$\vec{w}^* = (\vec{X}^T \cdot \vec{A} \cdot \vec{X} + \lambda I)^{-1} \vec{X}^T \vec{A} y$$

$X =$ Design Matrix

$\vec{A} = \operatorname{diag}(\vec{a})$, $\vec{A}_{ii} = a^{(i)}$

Can ignore constant of $\frac{1}{2}$ during minimization

The loss function to minimize is

$$L_1(w) = \frac{1}{2} \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)})^2 + \frac{\lambda}{2} \|\vec{w}\|^2$$

$$= \frac{1}{2} \left(\sum_{i=1}^{N} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)})^2 + \lambda \|\vec{w}\|^2 \right)$$

Since $\underset{w}{\text{minimize}}(c L'(w)) = \underset{w}{\text{minimize}}(L'(w))$ for any loss function, $L'(w)$ multiplied by a constant, c, we can ignore the constant of $\frac{1}{2}$ by minimizing $2 \times L_1(w)$ instead.

Let $L = 2 \times L_1$

$$= \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)})^2 + \lambda \|\vec{w}\|^2$$

where

$$\vec{w}^* = \underset{w}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)}) + \frac{\lambda}{2} \|\vec{w}\|^2$$

$$= \underset{w}{\operatorname{argmin}} (L_1(w))$$

$$= \underset{w}{\operatorname{argmin}} (2 \times L_1(w))$$

$$= \underset{w}{\operatorname{argmin}} (L(w))$$

$$\therefore \vec{w}^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)})^2 + \lambda \|\vec{w}\|^2$$

Re-write Loss function, $L(w)$ using Matrices instead of summations

Let

Design Matrix, $\vec{X}$

$$\vec{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_d^{(1)} \\ x_1^{(2)} & \cdots & & x_d^{(2)} \\ \vdots & & \ddots & \vdots \\ x_1^{(N)} & x_2^{(N)} & \cdots & x_d^{(N)} \end{bmatrix}_{(N \times d)} = \begin{bmatrix} (\vec{x}^{(1)})^T \\ (\vec{x}^{(2)})^T \\ \vdots \\ (\vec{x}^{(N)})^T \end{bmatrix}$$

$$Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix}, \quad \vec{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

Re-write Loss function, $L(w)$ using Matrices instead of summations

Let

Design Matrix, $\vec{X}$

$$\vec{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_d^{(1)} \\ x_1^{(2)} & \cdots & & x_d^{(2)} \\ \vdots & & \ddots & \vdots \\ x_1^{(N)} & x_2^{(N)} & \cdots & x_d^{(N)} \end{bmatrix}_{(N\times d)} = \begin{bmatrix} (\vec{x}^{(1)})^T \\ (\vec{x}^{(2)})^T \\ \vdots \\ (\vec{x}^{(N)})^T \end{bmatrix}$$

$$Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix}_{(N\times 1)}, \quad \vec{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}_{(d\times 1)}$$

$$A = \mathrm{diag}(\vec{a}) = \begin{bmatrix} a^{(1)} & 0 & \cdots & 0 \\ 0 & a^{(2)} & 0 \cdots & 0 \\ 0 & & \ddots & 0 \\ 0 \; 0 & \cdots & 0 & a^{(N)} \end{bmatrix}_{(N\times N)}$$

$$B = \mathrm{diag}(\sqrt{\vec{a}}) = \begin{bmatrix} \sqrt{a^{(1)}} & 0 & \cdots & 0 \\ 0 & \sqrt{a^{(2)}} & 0 \cdots & 0 \\ 0 & & \ddots & 0 \\ 0 \; 0 & \cdots & 0 & \sqrt{a^{(N)}} \end{bmatrix}_{(N\times N)} \Rightarrow B^T \cdot B = B \cdot B^T = A$$

$$\therefore L(\vec{w}) = \sum_{i=1}^{N} a^{(i)} \left(y^{(i)} - w^T x^{(i)}\right)^2 + \lambda \|w\|^2$$

$$= \sum_{i=1}^{N} \left(\sqrt{a^{(i)}} \left(y^{(i)} - w^T x^{(i)}\right)\right)^2 + \lambda \|w\|^2$$

$$= B(Y - XW))^2 + \lambda \|w\|_F^2, \quad \|w\|_F^2 = \vec{w}^T \cdot \vec{w}$$

$\therefore$

$$\vec{w}^* = \underset{w}{\mathrm{argmin}} \left(B(Y - XW))^2 + \lambda \|w\|_F^2\right)$$

Matrix Derivatives and proof:

Since we are dealing with Matrix Algebra, will prove some Matrix Differentiation formula before using these formula to come up with closed-form solution for Locally Weighted Regression.

F independent of $\vec{w}$, $\vec{w} \in \mathbb{R}^{d\times 1}$, $F \in \mathbb{R}^{m\times d}$

$g = F \cdot \vec{w}$

$$g = \begin{bmatrix} F_{11} & \cdots & F_{1d} \\ \vdots & \ddots & \vdots \\ F_{m1} & \cdots & F_{md} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \Rightarrow g_i = \sum_{j=1}^{d} F_{ij} W_j, \quad i \in \{1, 2, \ldots, m\}$$

$$\frac{\partial g}{\partial \vec{w}} = \begin{bmatrix} \frac{\partial g_1}{\partial \vec{w}_1} & \cdots & \frac{\partial g_m}{\partial \vec{w}_1} \end{bmatrix} \Rightarrow \frac{\partial g_i}{\partial W_k} = \frac{\partial}{\partial w_k} \sum_{j=1}^{d} F_{ij} W_j = F_{ik}$$

Matrix Derivatives and proof:

Since we are dealing with Matrix Algebra, will prove some Matrix Differentiation formula before using these formula to come up with closed-form solution for Locally Weighted Regression.

F independent of $\vec{w}$, $\vec{w} \in \mathbb{R}^{d\times 1}$, $F \in \mathbb{R}^{m\times d}$

$g = F \cdot \vec{w}$

$$g = \begin{bmatrix} F_{11} & \cdots & F_{1d} \\ \vdots & \ddots & \vdots \\ F_{m1} & \cdots & F_{md} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \Rightarrow g_i = \sum_{j=1}^{d} F_{ij} W_j, \quad i \in \{1, 2, \dots, m\}$$

$$\frac{\partial g}{\partial \vec{w}} = \begin{bmatrix} \frac{\partial g_1}{\partial \vec{w}_1} & \cdots & \frac{\partial g_m}{\partial \vec{w}_1} \\ \vdots & \ddots & \\ \frac{\partial g_1}{\partial \vec{w}_d} & & \frac{\partial g_m}{\partial \vec{w}_d} \end{bmatrix} \Rightarrow \frac{\partial g_i}{\partial W_k} = \frac{\partial}{\partial w_k} \sum_{j=1}^{d} F_{ij} W_j = F_{ik}$$

i comes from g

k " " W

$$= \begin{bmatrix} F_{11} & F_{21} & \cdots & F_{m1} \\ F_{12} & F_{22} & & \vdots \\ \vdots & & \ddots & \\ F_{1m} & \cdots & & F_{md} \end{bmatrix} = F^T$$

$\therefore$, $\frac{d\, F\cdot\vec{w}}{d\vec{w}} = F^T$ $\Rightarrow$ Take transpose of what's on left of $\vec{w}$

Let $h = \vec{w}^T \cdot F^T$

$$\frac{\partial h}{\partial \vec{w}} = \frac{\partial \vec{w}^T \cdot F^T}{\partial \vec{w}}$$

Let $p = h^T = F \cdot w$

$\Rightarrow \frac{\partial h}{\partial \vec{w}} = \frac{\partial h^T}{\partial \vec{w}}$ since $h \in \mathbb{R}$ $\Rightarrow$ scalars are always symmetric

$\frac{\partial h^T}{\partial w} = F^T$ as proven above

$\therefore$, $\frac{\partial\, \vec{w}^T \cdot F^T}{\partial \vec{w}} = F^T$ $\Rightarrow$ just rewrite what's on right of $\vec{w}$

Quadratic Form, independent of x, $A \in \mathbb{R}^{n\times n}$, $x \in \mathbb{R}^n$

$f = \vec{w}^T \cdot H \cdot \vec{w}$, H independent of w, $H \in \mathbb{R}^{d\times d}$, $w \in \mathbb{R}^d$

$$f = \sum_{j=1}^{d} \left(\sum_{i=1}^{d} w_i \cdot H_{ij} \right) x_j \quad \in \mathbb{R}$$

$$\frac{\partial f}{\partial \vec{w}} = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \vdots \\ \frac{\partial f}{\partial w_d} \end{bmatrix}, \quad \frac{\partial f}{\partial w_k} = \frac{\partial}{\partial w_k} \left(\sum_{j=1}^{d} \left(\sum_{i=1}^{d} w_i \cdot H_{ij} W_j \right) \right)$$

$$= \sum_{i=1}^{d} H_{ik} W_i + \sum_{j=1}^{d} H_{kj} W_j, \text{ Product Rule}$$

$\therefore$, $\dfrac{\partial \vec{w}^T \cdot F^T}{\partial \vec{w}} = F^T$ $\Rightarrow$ just rewrite what's on right of $\vec{w}$

Quadratic Form, independent of x, $A \in \mathbb{R}^{n \times n}$, $x \in \mathbb{R}^n$

$f = \vec{w}^T \cdot H \cdot \vec{w}$, H independent of w, $H \in \mathbb{R}^{d \times d}$, $w \in \mathbb{R}^d$

$$f = \sum_{j=1}^{d} \left(\sum_{i=1}^{d} w_i \cdot H_{ij} \right) x_j \in \mathbb{R}$$

$$\frac{\partial f}{\partial \vec{w}} = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \vdots \\ \frac{\partial f}{\partial w_d} \end{bmatrix}, \quad \frac{\partial f}{\partial w_k} = \frac{\partial}{\partial w_k} \left(\sum_{j=1}^{d} \left(\sum_{i=1}^{d} w_i \cdot H_{ij} w_j \right) \right)$$

$$= \sum_{i=1}^{d} H_{ik} W_i + \sum_{j=1}^{d} H_{kj} W_j \text{, Product Rule}$$

$$= \begin{bmatrix} \sum_{i=1}^{d} H_{i1} W_i + \sum_{j=1}^{d} H_{1j} W_j \\ \vdots \\ \sum_{i=1}^{d} H_{id} W_i + \sum_{j=1}^{d} H_{dj} W_j \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{d} H_{i1} W_i \\ \vdots \\ \sum_{i=1}^{d} H_{id} W_i \end{bmatrix} + \begin{bmatrix} \sum_{j=1}^{d} H_{1j} W_j \\ \vdots \\ \sum_{j=1}^{d} H_{dj} W_j \end{bmatrix}$$

$$= H^T \cdot \vec{w} + H \cdot \vec{w}$$

$$= (H^T + H) \cdot \vec{w}$$

$\therefore$, $\dfrac{\partial (\vec{w}^T \cdot H \cdot \vec{w})}{\partial \vec{w}} = (H^T + H) \cdot \vec{w}$, if $H^T = H \Rightarrow 2 H \cdot \vec{w}$

Closed-Form Solution for Locally Weighted Regression

Take derivative and set to 0 to solve for w^*

$$L(\vec{w}) = (\vec{B}(Y - \vec{X}\vec{w}))^2 + \lambda \|\vec{w}\|_F^2$$

$$= (\vec{B}(\vec{Y} - \vec{X}\vec{w}))^T (\vec{B}(\vec{Y} - \vec{X}\vec{w})) + \lambda \vec{w}^T \vec{w}$$

$$= \underbrace{((\vec{Y} - \vec{X}\vec{w})^T B^T)}_{(1 \times m)} \cdot \underbrace{(\vec{B}\vec{Y} - \vec{B}\vec{X}\vec{w})}_{(m \times 1)} + \lambda \vec{w}^T \vec{w}$$

$$= ((\vec{Y}^T - \vec{w}^T \vec{X}^T) \vec{B}^T) \cdot (\vec{B}\vec{Y} - \vec{B}\vec{X}\vec{w}) + \lambda \vec{w}^T \vec{w}$$

$$= (\vec{Y}^T \vec{B}^T - \vec{w}^T \vec{X}^T \vec{B}^T)(\vec{B}\vec{Y} - B\vec{X}\vec{w}) + \lambda \vec{w}^T \vec{w}$$

$$= \vec{Y}^T \vec{B}^T \vec{B} Y - \vec{Y}^T \vec{B}^T \vec{B} \vec{X} \vec{w} - \vec{w}^T \vec{X}^T \vec{B}^T \vec{B} \vec{Y} + \vec{w}^T \vec{X}^T \vec{B}^T \vec{B} \vec{X} w + \lambda \vec{w}^T \vec{w}$$

Closed-Form Solution for Locally Weighted Regression

Take derivative and set to 0 to solve for w^*

$$L(\vec{w}) = (\vec{B}(Y - \vec{X}\vec{w}))^2 + \lambda \|\vec{w}\|_F^2$$

$$= (\vec{B}(\vec{Y} - \vec{X}\vec{w}))^T \cdot (\vec{B}(\vec{Y} - \vec{X}\vec{w})) + \lambda \vec{w}^T\vec{w}$$

$$= \underbrace{((\vec{Y} - \vec{X}\vec{w})^T B^T)}_{(1\times m)} \cdot \underbrace{(\vec{B}\vec{Y} - \vec{B}\vec{X}\vec{w})}_{(m\times 1)} + \lambda \vec{w}^T\vec{w}$$

$$= ((\vec{Y}^T - \vec{w}^T\vec{X}^T)\vec{B}^T) \cdot (\vec{B}\vec{Y} - \vec{B}\vec{X}\vec{w}) + \lambda \vec{w}^T\vec{w}$$

$$= (\vec{Y}^T\vec{B}^T - \vec{w}^T\vec{X}^T\vec{B}^T)(\vec{B}\vec{Y} - B\vec{X}\vec{w}) + \lambda \vec{w}^T\vec{w}$$

$$= \vec{Y}^T\vec{B}^T\vec{B}Y - \vec{Y}^T\vec{B}^T\vec{B}\vec{X}\vec{w} - \vec{w}^T\vec{X}^T\vec{B}^T\vec{B}\vec{Y} + \vec{w}^T\vec{X}^T\vec{B}^T\vec{B}\vec{X}w + \lambda \vec{w}^T\vec{w}$$

Take Matrix derivative set to 0, applying closed form derivatives derived above

$$\frac{dL(\vec{w})}{d\vec{w}} = -\vec{X}^T\vec{B}^T\vec{B}\vec{Y} - \vec{X}^T\vec{B}^T\vec{B}\vec{Y} + 2\vec{X}^T\vec{B}^T\vec{B}\vec{X}\vec{w} + \lambda 2\vec{w} = 0$$

$$\Rightarrow (2\vec{X}^T\vec{B}^T\vec{B}\vec{X} + \lambda \cdot I)\vec{w} = 2\vec{X}^T\vec{B}^T\vec{B}\vec{Y}$$

$$\Rightarrow (\vec{X}^T\vec{B}^T\vec{B}\vec{X} + \lambda \cdot I)\vec{w} = \vec{X}^T\vec{B}^T\vec{B}\vec{Y}$$

$$\Rightarrow \vec{w} = (\vec{X}^T\vec{B}^T\vec{B}\vec{X} + \lambda I)^{-1}\vec{X}^T\vec{B}^T\vec{B}\vec{Y}$$

Since $\vec{B}^T\vec{B} = \vec{A}$,

$$\Rightarrow \vec{w} = (\vec{X}^T\vec{A}\vec{X} + \lambda I)^{-1}\vec{X}^T\vec{A}\vec{Y} \quad \therefore \text{ proven}$$

b) Locally reweighted least squares combines ideas from kNN and linear regression. For each new test example $\vec{x}$, we compute distance-based weights for each training example

$$a^{(i)} = \frac{e^{\frac{-\|x - x^{(i)}\|^2}{2\tau^2}}}{\sum_j e^{\frac{-\|x - x^{(j)}\|^2}{2\tau^2}}}$$

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

$$\hat{y} = x^T \cdot w^*$$

Complete implementation in q2.py.

Notes for implementation.

- Do not invert any matrix. Use np.linalg.solve
- $\dfrac{\exp(A_i)}{\sum_j \exp(A_j)} = \dfrac{\exp(A_i - B)}{\sum_j \exp(A_j - B)} = \dfrac{\exp(A_i - \max_j(A_j))}{\sum_j \exp(A_j - \max_j(A_j))} \rightarrow$ more numerically stable

$\Rightarrow$ Use scipy.misc.logsumexp.

b) Locally reweighted least squares combines ideas from kNN and linear regression.
For each new test example $\vec{x}$, we compute distance-based weights for each training example

$$a^{(i)} = \frac{e^{\frac{-\|x-x^{(i)}\|^2}{2\tau^2}}}{\sum_j e^{\frac{-\|x-x^{(j)}\|^2}{2\tau^2}}}$$

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

$$\hat{y} = x^T \cdot w^*$$

Complete implementation in q_2.py.

Notes for implementation.

↳ Do not invert any matrix. Use np.linalg.solve

↳ $\frac{\exp(A_i)}{\sum_j \exp(A_j)} = \frac{\exp(A_i - B)}{\sum_j \exp(A_j - B)} = \frac{\exp(A_i - \max_j(A_j))}{\sum_j \exp(A_j - \max_j(A_j))}$ → more numerically stable

⇒ Use scipy.misc.logsumexp.

Let $A_i = -\frac{1}{2\tau^2}\|x - x^{(i)}\|^2 \in \mathbb{R}^{n \times n}$

⇒ pairwise distance between ∀ pair of points

↳ A_i in hint function above

$$a^{(i)} = \frac{e^{A_i}}{\sum_j e^{A_j}}, \quad \log(a^{(i)}) = A_i - \log\left(\sum_j e^{A_j}\right)$$

$= A_i$ − scipy.misc.logsumexp (A)

⇒ use $\log(a^{(i)})$ as weights instead

$\vec{A}$ = np.diagflat(a)

↳ $\vec{A}$ as weight for each training point

$$\vec{w} = (\vec{X}^T \vec{A} \vec{X} + \lambda I)^{-1} \vec{X}^T \vec{A} \vec{Y}$$

$$\Rightarrow \quad (\vec{x}^T \cdot \vec{A} \cdot \vec{x} + \lambda \cdot I)\,\vec{w} = \vec{x}^T \vec{A} \cdot \vec{Y}$$

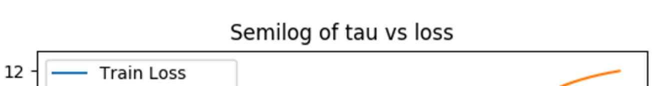
$a\,x = b$ in linear algebra

⇒ np.linalg.solve$((\vec{x}^T \cdot \vec{A} \cdot \vec{x} + \lambda \cdot I), \vec{x}^T \cdot \vec{A} \cdot \vec{Y})$

Submitted under q_2.py

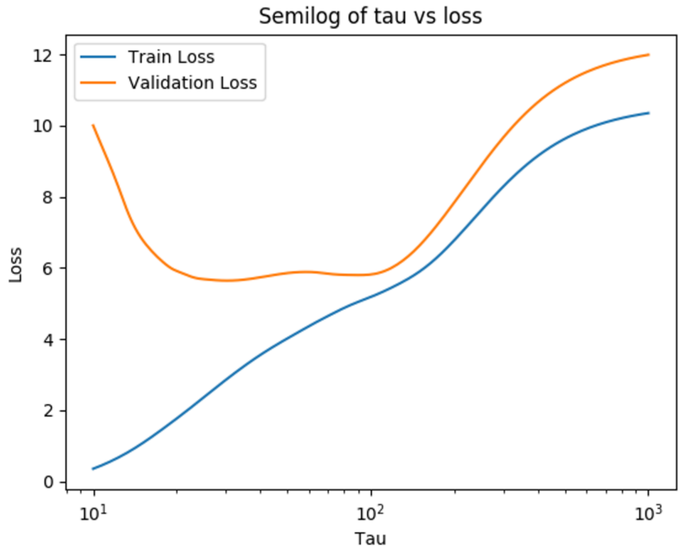
c) Randomly hold out 30% of dataset as validation set.
Compute the average loss for different values of τ in the range [10, 1000] on both the training set and validation set.
Plot the training and validation losses as a function of τ.
(using a log scale for τ)

Submitted under q_2.py



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

Submitted under q2.py



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

As $\tau \to \infty \Rightarrow$ all training point has equal weight
$\Rightarrow$ algorithm fits a single global unweighted linear regression for each training point.
$\Rightarrow$ underfitting $\Rightarrow$ high bias
$\tau \to 0 \Rightarrow$ closest training point has most weight,
algorithm fits separate local linear regression for every training point.
$\Rightarrow$ overfitting to training point $\Rightarrow$ high variance
Yes, this occurs in the plot shown in Q2. c) above.
As $\tau \to 0$, training loss is minimized, but test loss is high
As we sweep τ, train loss should increase as no longer overfitting
Test loss shows a nice minimum due to
bias-variance decomposition of sweeping through τ)