ENGN2520 Homework 2

Ming Xu (Banner ID: B01532164)

Problem1

Because x is a random variable with uniform distribution on [*a,b*], then the pmf of x is:

$$p(x) = 1/(b-a).$$

Therefore, the likelihood function L on the training set T can be written as:

$$L((a,b)|T) = P(T|(a,b)) = \prod_{i=1}^{n} p(x_i|(a,b)) = \prod_{i=1}^{n} \frac{1}{b-a} = (\frac{1}{b-a})^n$$

Denote the log likelihood as $l((a,b)|T) = log\ L((a,b)|T) = -n * log(b-a)$

$$\begin{cases} \dfrac{\partial l((a,b)|T)}{\partial a} = \dfrac{n}{b-a} \\ \dfrac{\partial l((a,b)|T)}{\partial b} = -\dfrac{n}{b-a} \end{cases}$$

According to the derivatives of the log-likelihood function, $\frac{\partial l((a,b)|T)}{\partial a}$ is monotonically increasing.

Therefore, the maximum likelihood estimator for a should be the largest possible of a.

$$a = min(x_i)$$

Similarly, the derivatives of the log-likelihood function, $\frac{\partial l((a,b)|T)}{\partial b}$ is monotonically decreasing.

Therefore, the maximum likelihood estimator for b should be the largest possible of b.

$$b = max(x_i)$$

Problem2

(a)

We know that: $w_{ML} = \underset{w \in \mathbb{R}^m}{\operatorname{argmax}} L(w)$

Let x_i^A be the ith sample from training set ***A***, and x_i^B be the ith sample from training set ***B***

The likelihood function $L(w)$ can be written as:

$$L(w) = \prod_{i=1}^{n} P(y_i^A | x_i^A, w) * \prod_{i=1}^{n} P(y_i^B | x_i^B, w)$$

$$= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma_A}} exp\{-\frac{(y_i^A - f_w(x_i^A))^2}{2\sigma_A^2}\} * \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma_B}} exp\{-\frac{(y_i^B - f_w(x_i^B))^2}{2\sigma_B^2}\}$$

Denote the log likelihood as $l(w) = log\ L(w)$,

$$l(w) = \sum_{i=1}^{n} (log \frac{1}{\sqrt{2\pi\sigma_A}} - \frac{(y_i^A - f_w(x_i^A))^2}{2\sigma_A^2}) + \sum_{i=1}^{n} (log \frac{1}{\sqrt{2\pi\sigma_B}} - \frac{(y_i^B - f_w(x_i^B))^2}{2\sigma_B^2})$$

$$w_{ML} = \underset{w\in\mathbb{R}^m}{\operatorname{argmax}}\, l(w) = \underset{w\in\mathbb{R}^m}{\operatorname{argmin}}[\sum_{i=1}^{n}\frac{(y_i^A - f_w(x_i^A))^2}{2\sigma_A^2} + \sum_{i=1}^{n}\frac{(y_i^B - f_w(x_i^B))^2}{2\sigma_B^2}]$$

$$= \underset{w\in\mathbb{R}^m}{\operatorname{argmin}}[\frac{1}{\sigma_A^2} * E_A(\mathrm{w}) + \frac{1}{\sigma_B^2} * E_B(\mathrm{w})]$$

Where $E_A(\mathrm{w})$ is the sum of squared differences over training set ***A*** and $E_B(\mathrm{w})$ is the sum of squared differences over training set ***B.***

Therefore, $E(w) = \frac{1}{\sigma_A^2} * E_A(\mathrm{w}) + \frac{1}{\sigma_B^2} * E_B(\mathrm{w})$

(b)

Because $E(w) = \frac{1}{\sigma_A^2} * E_A(\mathrm{w}) + \frac{1}{\sigma_B^2} * E_B(\mathrm{w})$,

So we search for the w such that $\nabla E(w) = 0$

This is equivalent to the set of equations $\frac{\partial E(w)}{\partial w_j} = \frac{1}{\sigma_A^2} * \frac{\partial E_A(w)}{\partial w_j} + \frac{1}{\sigma_B^2} * \frac{\partial E_B(w)}{\partial w_j}$

Since, for $\frac{\partial E_A(w)}{\partial w_j}$, the matrix ***M*** and vector ***Z*** are:

$$M_{k,j}^A = \sum_{i=1}^{n} f_k(x_i^A) * f_j(x_i^A)$$

$$Z_j^A = \sum_{i=1}^{n} y_i^A * f_j(x_i^A)$$

Similarly, for $\frac{\partial E_B(w)}{\partial w_j}$, the matrix ***M*** and vector ***Z*** are:

$$M_{k,j}^B = \sum_{i=1}^{n} f_k(x_i^B) * f_j(x_i^B)$$

$$Z_j^B = \sum_{i=1}^{n} y_i^B * f_j(x_i^B)$$

Combining the result above, the final matrix ***M*** and vector ***Z*** are:

$$M_{k,j} = \frac{1}{\sigma_A^2} M_{k,j}^A + \frac{1}{\sigma_B^2} M_{k,j}^B = \sum_{i=1}^{n} \frac{1}{\sigma_A^2} f_k(x_i^A) * f_j(x_i^A) + \frac{1}{\sigma_B^2} f_k(x_i^B) * f_j(x_i^B)$$

$$Z_j = \frac{1}{\sigma_A^2} Z_j^A + \frac{1}{\sigma_B^2} Z_j^B = \sum_{i=1}^{n} \frac{1}{\sigma_A^2} y_i^A * f_j(x_i^A) + \frac{1}{\sigma_B^2} y_i^B * f_j(x_i^B)$$

Therefore, w_{ML} can be computed by solving the linear system.

(c)

From the mathematical model above, they shouldn't ignore Bob's measurements in estimating their function. They can incorporate by combining their test results using a weighted sum method.

(d)

If we don't know the σ_A and σ_B, we can suppose $\sigma_A = \sigma_B$. Then use the mean of $M_{k,j}^A$ and $M_{k,j}^B$ to calculate $M_{k,j}$, and use the mean of Z_j^A and Z_j^B to calculate Z_j.

Problem3

(a)

The degree 2 polynomials that estimated by least square regression and least absolute deviations regression are shown in the figure below.

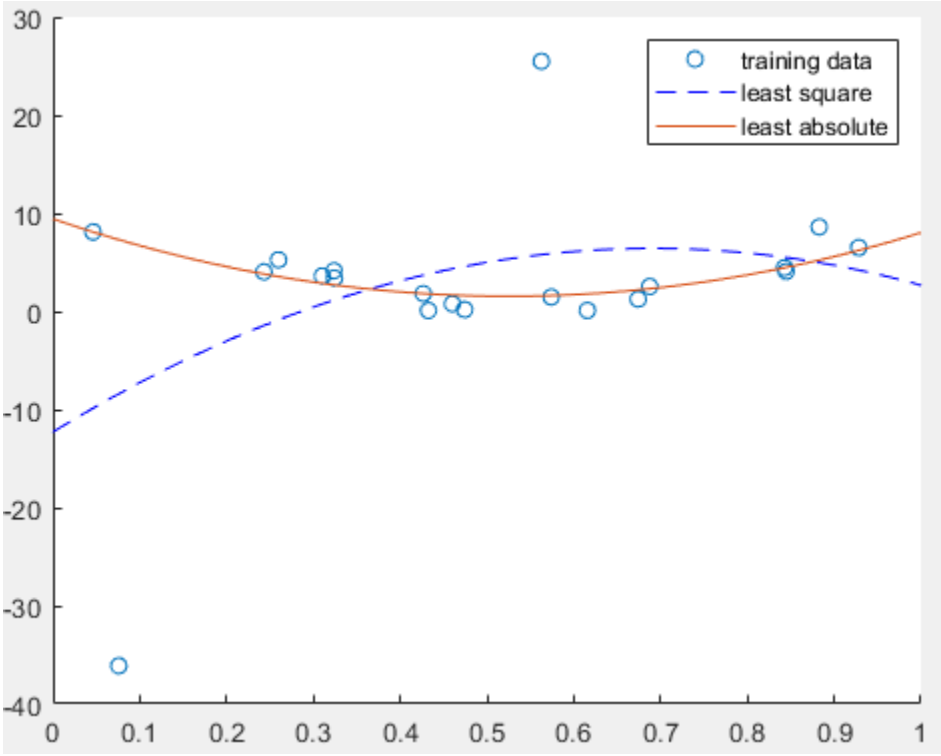

Fig.1. The degree 2 polynomials that estimated from training data

In the figure, the orange solid line indicates the degree 2 polynomials estimated by least absolute deviations regression, while the blue dash line indicates the degree 2 polynomials estimated by least square regression.

(b)

The degree 4 polynomials that estimated by least square regression and least absolute deviations regression are shown in the figure below.

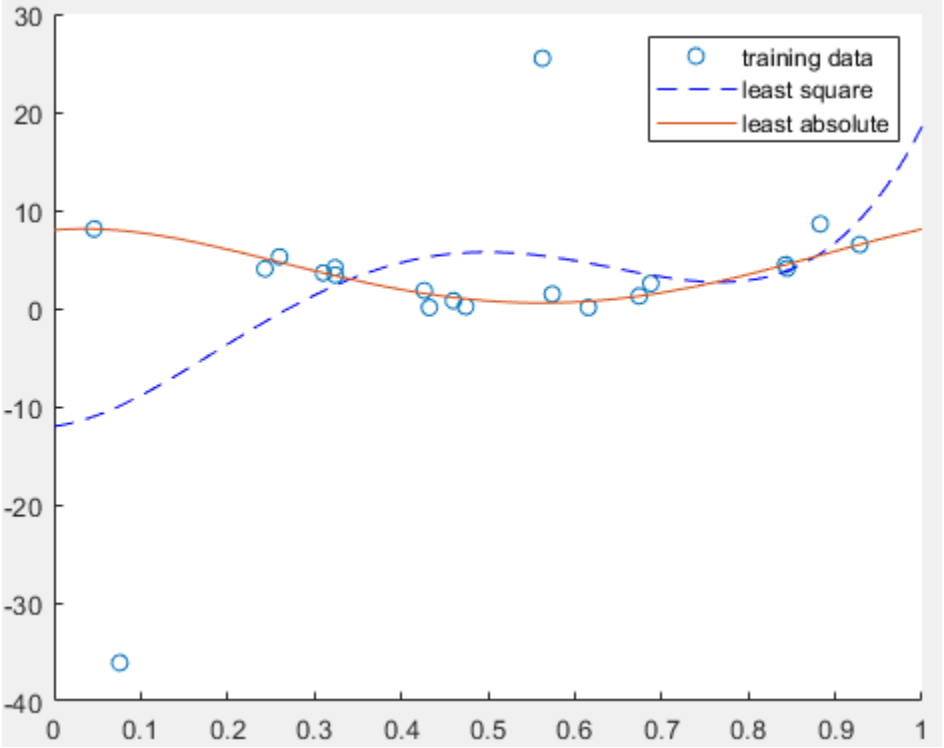

Fig.2. The degree 4 polynomials that estimated from training data

In the figure, the orange solid line indicates the degree 4 polynomials estimated by least absolute deviations regression, while the blue dash line indicates the degree 4 polynomials estimated by least square regression.

(c)

Based on the given experiments, least absolute deviations regression is more robust than the least square regression when outliners exist.

(d) Source code

1. Function to calculate w from training set by least square regression: "solveW.m"

```
function [w] = solveW(x,y,deg)
    %initialization
    M = zeros(deg+1,deg+1);
    Z = zeros(deg+1,1);
    %contruct matrix M
    for row = 1:deg+1
       for col = 1:deg+1
           M(row,col) = sum((x.^(row-1)).*(x.^(col-1)));
       end
    end
    %construct matrix Z
    for row = 1:deg+1
        Z(row,1) = sum(y.*(x.^(row-1)));
    end
```

```
    %calculate w
    w = M\Z;
end
```

2. Function to calculate fx by giving x and w: “calculateFxByGivingW.m”

```
function [fx] = calculateFxByGivingW(x,w)
    %get degree
    [deg, col] = size(w);
    %get size
    [rowNum, col] = size(x);
    %initialize fx
    fx = zeros(rowNum, col);
    %calculate fx by usuing polynomial function defined by w
    for row = 1:rowNum
        for i = 1:deg
            fx(row,1) = fx(row,1) + w(i)*x(row,1)^(i-1);
        end
    end
end
```

3. Function to calculate w from training set by least absolute derivations regression: “solveWByLinearProgramming.m”

```
function [w] = solveWByLinearProgramming(x,y,deg)
    %get size
    [rowNum, col] = size(x);

    deg = deg + 1;
    %initialization
    A = zeros(2*rowNum, rowNum+deg);
    b = zeros(2*rowNum,1);
    C = zeros(1,rowNum+deg);

    %generate C
    index = deg+1:rowNum+deg;
    C(index) = 1;

    %generate b
    index = 1:2:2*rowNum-1;
    b(index) = y;
    index = 2:2:2*rowNum;
    b(index) = -y;
```

```
    %generate A
    for row = 1:2:2*rowNum-1
       for col = 1:deg
          A(row,col) = x(ceil(row/2),1).^(col-1);
       end

       A(row,deg+ceil(row/2))=-1;
    end

    for row = 2:2:2*rowNum
        A(row,:) = -A(row-1,:);
        A(row,deg+ceil(row/2))=-1;
    end
    %calculate w by linear program
    result = linprog(C,A,b);
    w = result(1:deg);
end
```

4. Main function: “main.m”

```
%loal data
load Xtrain
load Ytrain

%plot showing the training data and degree 2 polynomial estimated from
the
%data
w1 = solveW(Xtrain,Ytrain,4);
w2 = solveWByLinearProgramming(Xtrain,Ytrain,4);
scatter(Xtrain,Ytrain,'DisplayName','training data');
hold on;
x = 0:0.01:1;
x = x';
fx1 = calculateFxByGivingW(x,w1);
fx2 = calculateFxByGivingW(x,w2);
plot(x,fx1,'b--','DisplayName','least square');
plot(x,fx2,'DisplayName','least absolute');
legend;
```