CS299 Machine Learning: Assign#1

Due on Thursday, March 14th, 2018

Andrew Ng 6:30 am

Bryan Zhang

Contents

Problem 1

1(a)

$H = \nabla \cdot \nabla^T J(\theta)$
So $z^T Hz = z^T(\nabla \cdot \nabla^T \cdot J(\theta))z = (z^T \cdot \nabla)^2 J(\theta) = (z^T \nabla^T \cdot J(\theta))^2 \cdot \frac{1}{J(\theta)}$.
Since the first factor of $z^T Hz$ is a square term which is definitely bigger than 0, So we focus on the factor $\frac{1}{J(\theta)}$. Because $J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} \log(h_\theta(y^{(i)}x^{(i)})$ and the hypopthesis function is a signoid function ranging from (0,1),the log of the hypothesis funcition must be negative and thus the coost function must be postive. So does the $z^T Hz$.
q.e.d.

1(b)

The optimized θ is $[-2.62042271649454, 0.760346235045246, 1.17193037252339]$.

```
close all; clear; clc;

% the rows of X is input variables
% the rows of Y is the response variables
fileIDX = fopen('logistic_x.txt', 'r');
sizeX = [2 Inf];
formatSpec = '%f';
X = fscanf(fileIDX, formatSpec, sizeX).';
% append the intercept term
X = [ones(size(X, 1), 1) X];

fileIDY  = fopen('logistic_y.txt', 'r');
Y = fscanf(fileIDY, formatSpec);

%plot the x and y
% the sub dataset for respective y is 1
Xp = X(1:50  ,:);
%the other half of sub dataset
Xn = X(51:size(X, 1)  ,:);
sz = 25;
x1range = [0 8];
x2range = [-5 4];



%the implementation of the Newton's Method for logistic regression
%serveral instance variables
THETA_INITIAL = zeros(1, size(X, 2));
ERROR_MARGINS = 0.00001;
%the size of sample space
m = size(Y, 1);




%the sigmoid function
sigmoid =  @(z)1./(1 + exp(-z));
%the cost function
```

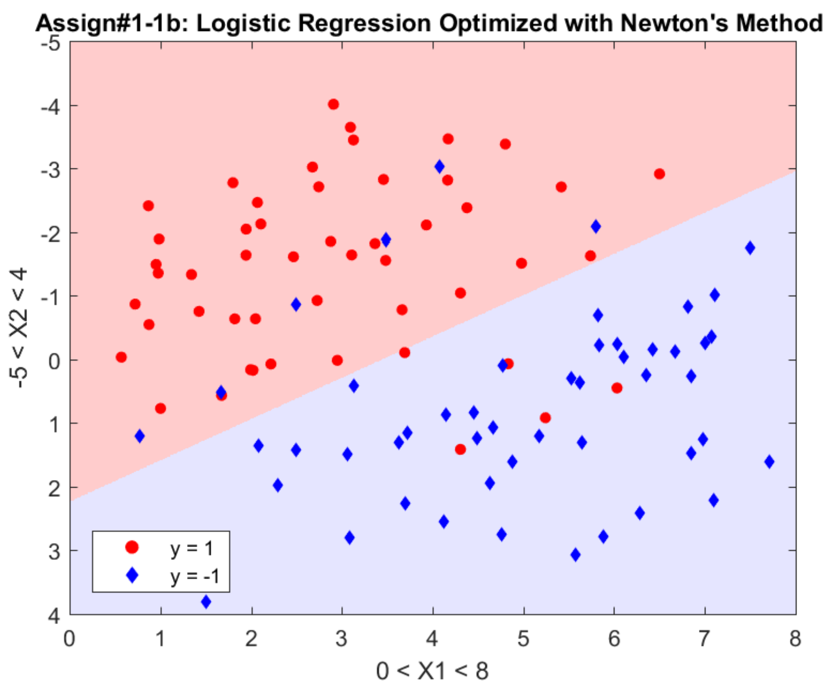


```
J = @(theta) 1 / m * sum(log(sigmoid(Y.*( X * theta'))));

thetaOptimized = getTheta(J, THETA_INITIAL,ERROR_MARGINS);
%the hypthesis funciton
h = @(X) sigmoid(X * thetaOptimized');

%plot the decision boundary
% step size for the accuracy of the boundary curve
inc = 0.01;

% generate grid coordinates
[x1, x2] = meshgrid(x1range(1):inc:x1range(2), x2range(1):inc:x2range(2));
imageSize = size(x1);

x1x2 = [x1(:) x2(:)]; % make the (x1, x2) pairs as row vectors

hypothesis = zeros(length(x1x2), 1);
for i = 1:length(x1x2)
    Xhypo= [1 x1x2(i,:)];
    htemp = h(Xhypo);
    if htemp > 0.5
        hypothesis(i) = 1;
    else
        hypothesis(i) = 0;
    end
end
%reshap the hypothesis to be positioned on each grid point
decisionMap = reshape(hypothesis, imageSize);

% plot the decision boundary
figure
imagesc(x1range, x2range, decisionMap);
hold on;
cmap = [1 0.8 0.8; 0.9 0.9 1];
colormap(cmap);


scatter(Xp(:,2), Xp(:, 3), sz, 'red', 'filled');
hold on;
scatter(Xn(:,2), Xn(:, 3), sz, 'blue', 'filled', 'd');
hold on;
title("Assign#1-1b: Logistic Regression Optimized with Newton's Method ");
xlim(x1range); ylim(x2range);
xlabel('0 < X1 < 8');
ylabel('-5 < X2 < 4');
legend('y = 1','y = -1', 'Location', 'Southwest')
hold on;

%save the image
 saveas(gcf,'1b.png')

%disp tests
disp(sum(sigmoid([1 2 3])));
```

```
disp(J([0 0 0]));
disp(getGradient(J, [0 0 0]));
disp(getHessian(J, [0 0 0]));
disp(getTheta(J, [0 0 0], 0.00001));

% get the optimized theta
function theta = getTheta(costfunc, thetaIni, errorMargins)
%% costfunc is the cost funciton for the logistic regression
%% thetaIni is the start popint to search for the optimized theta
%% return the optimized theta which can make the gradient down to zero
%% thus the cost function down to the minimal
theta = thetaIni;
grad = getGradient(costfunc, theta);
while norm(grad) > errorMargins
    grad = getGradient(costfunc, theta);
    H = getHessian(costfunc, theta);
    disp('H:'); disp(H);
    disp('grad:'); disp(grad);
    theta = theta - grad / H;
    disp('theta'); disp(theta);
end
end

%get the hessian
function H = getHessian(f, x)
%% f is a function
%% x is a input varibale
%% return the hessian for the function at x
gx = getGradient(f, x);
H = zeros(size(x, 2));
h = 0.00001;

%iterate over al indexes in x
for i = 1: size(x, 2)
    oldValues = x(i);
    x(i) = oldValues + h;
    gxh = getGradient(f, x); %get the grad f(x + h)
    x(i) = oldValues; % restore to previous value

    %compute the second partial derative
    H(:,i) = (gxh - gx)./ h;
    %iterate over to the next variable
end
end


% get the gradient
function grad = getGradient(f, x)
%% f is a function
%% x is a input varibale
%% return the gradient for the function at x

fx = f(x);
```

```
grad = zeros(size(x));
h = 0.00001;

%iterate  over all indexes in x
for i = 1:size(x, 2)
oldValues = x(i);
x(i) = oldValues + h; %increment by h
fxh  = f(x); % evaluate f(x + h)
x(i) = oldValues;%restore to the previous value for x(i)

%compute the partial derative
grad(i) = (fxh - fx) / h; %the slop
%iterate to the next index
end
end
```

1(c)



Problem 2

2(a)

The possion distribution, $p(y;\lambda) = \frac{e^{-\lambda}\lambda^y}{y!} = \frac{1}{y!}e^(log(\lambda)y - \lambda)$. Compared with the expotential family $p(y;\eta) = b(y)exp(n^T T(y) - a(\eta))$, we can get

$$\begin{aligned} b(y) &= \frac{1}{y!} \\ \eta &= log(\lambda) \\ T(y) &= y \\ \alpha(\eta) &= \lambda = e^\eta \end{aligned}$$

2(b)

The canonical response function gives the mean of the distribution , which in this case, a Possion distribution, is λ , in terms of the natural parameter η. So the canonical response funciton for the Possion distribution is $g(\eta) = e^\lambda$.

2(c)&2(d)

Here we directly look at the all GLMs instead of the possion case. Our goal is first to derivate the $p(y^{(i)}|x^{(i)};\theta)$ with repect to θ_j .
For exponential family, $p(y;\eta) = b(y)exp(n^T T(y) - a(\eta))$. After using assumption 3, we subttitute the natural parameter for $\theta^T X$. For training set

$$\{(X^{(i)}, Y^{(i)}); i = 1......m\},$$

the distribution is

$$p(y^{(i)}|x^{(i)};\theta) = b(y)exp(\theta^T X^{(i)} - a(\theta^T X^{(i)})). \tag{1}$$

Then we take derivative of the log-likelyhood above with respect to θ_j, which shows,

$$\begin{aligned} &\frac{d}{d\theta_j} log(p(y^{(i)}|x^{(i)};\theta)) \\ &= \frac{1}{p(y^{(i)}|x^{(i)};\theta)} \cdot \frac{d}{d\theta_j} p(y^{(i)}|x^{(i)};\theta) \\ &= \frac{1}{p(y^{(i)}|x^{(i)};\theta)} \cdot (p(y^{(i)}|x^{(i)};\theta)[T(Y^{(i)}) - \frac{d}{d\eta}a(\eta)] \cdot x_j \\ &= (T(Y^{(i)}) - \frac{d}{d\eta}a(\eta)) \cdot x_j \end{aligned} \tag{2}$$

Using the assumption 2, we can get $T(Y^{(i)} = Y^{(i)}$. Because of this assumption and α being only a hyper-parameter, we need only to prove $\frac{d}{d\eta}a(\eta)$ is the reponse function when natural parameter η is replaced by $\theta^T X^{(i)}$ to prove the stochastic gradient ascent has the update rule

$$\theta_j = \theta_j - \alpha(h(x) - y)x_j$$

for every j and looping i until m .
$a(\eta)$ is the log parition function, which is to make sure the priobability funciton can integrate to unity at last. So

$$a(\eta) = log[\int b(Y)exp(\eta^T T(Y)v(dY)]$$

$$\begin{aligned}
&\frac{d}{d\eta^T}a(\eta) = \frac{\int b(Y)T(Y)exp(\eta^T T(Y)v(dY)}{\int b(Y)exp(\eta^T T(Y)v(dY)} \\
&= \int T(Y)[b(Y)exp(\eta^T T(Y)]v(dY) \cdot exp(-log[\int exp(\eta^T T(Y)b(Y)v(dY)]) \\
&= \int T(Y)[b(Y)exp(\eta^T T(Y)]v(dY) \cdot exp(-a(\eta)) \\
&= \int T(Y)[b(y)exp(\eta^T T(Y) - a(\eta))]v(dY) \\
&= \int T(Y)p(Y|X;\theta)V(dY) \\
&= E(T(Y)|X)
\end{aligned} \tag{3}$$

Since the response function give the estimation of Y (the T(Y) in most cases) in terms of input features X, so $\frac{d}{d\eta^T}a(\eta)$ is $h_\theta(X)$. Thus, we can conclude that the derivative of log-likelyhood given $(X^{(i)}, Y^{(i)})$ with respect to θ_j is the update rule for stochastic gradient ascent. Since 2(c) is only a special case for 2(d), q.e.d.

Problem 3

3(a)

Our goal [1] is to prove $p(y=1|x;\phi,\sum,\mu_{-1},\mu_1) = \frac{1}{1+exp(-\theta^T x-\theta_0)}$. and the similiar form for y = -1.
We can use Bayes's Formula to prove

$$\begin{aligned}
&p(y=1|x;\phi,\Sigma,\mu{-}1,\mu_1) \\
&= \frac{p(x|y=1)\cdot p(y=1)}{p(x)} \\
&= \frac{p(x|y=1)\cdot p(y=1)}{p(x|y=1)\cdot p(y=1) + p(x|y=-1)\cdot p(y=-1)} \\
&= \frac{1}{1+\frac{p(x|y=-1)\cdot p(y=-1)}{p(x|y=1)\cdot p(y=1)}}
\end{aligned} \tag{4}$$

The form is already similiar to our goal. Next we only need to prove the fraction $\frac{p(x|y=-1)\cdot p(y=-1)}{p(x|y=1)\cdot p(y=1)} = exp(\theta^T x + \theta_0)$.

$$\begin{aligned}
&\frac{p(x|y=-1)\cdot p(y=-1)}{p(x|y=1)\cdot p(y=1)} \\
&= exp[-1/2(x-\mu_{-1})^T\Sigma^{-1}(x-\mu_{-1}) + 1/2(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)] \cdot \frac{1-\phi}{\phi} \\
&= exp[-1/2(x-\mu_{-1})^T\Sigma^{-1}(x-\mu_{-1}) + 1/2(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)] \cdot exp(log(\frac{1-\phi}{\phi}) \\
&= exp[\frac{(\mu_1-\mu_{-1})^T x}{\Sigma} - \frac{\mu_{-1}^2-\mu_1^2}{2\Sigma} + log(\frac{1-\phi}{\phi})]
\end{aligned} \tag{5}$$

Campared with $exp(-\theta^T x + \theta_0)$, we can get

$$\theta = \frac{\mu_1 - \mu{-}1}{\Sigma},$$

[1] I got this section down by referrencing
https://duphan.wordpress.com/2016/10/27/gaussian-discriminant-analysis-and-logistic-regression/

Problem 3 continued on next page...

$$\theta_0 = log(\frac{1-\phi}{\phi}) - \frac{\mu_{-1}^2 - {\mu_1}^2}{2\Sigma}$$

Since the fraction is just the inverse in case y = -1, so the form satisfies in both case; q.e.d.

3(b) & 3(c)

Our goal [2]is to perform log-likelyhood maximum estimation on the log-likelyhood. By Bayes's rule[3],

$$\begin{aligned}
&l(\phi, \mu_1, \mu_{-1}, \Sigma) \\
&= log \prod_{i=1}^{m} p(x^{(i)}|y^{(i)}; \mu_{-1}, \mu_1, \Sigma) \cdot p(y^{(i)}) \\
&= \sum_{i=1}^{m} log\{[\phi \cdot N(x^{(i)}|\mu_1, \Sigma)]^{1\{y^{(i)}=1\}} \cdot [(1-\phi) \cdot N(x^{(i)}|\mu_{-1}, \Sigma)]^{1\{y^{(i)}=-1\}}\} \\
&= \sum_{i=1}^{m} \{1\{y^{(i)}=1\}log\phi + 1\{y^{(i)}=1\}logN(x^{(i)}|\mu_1, \Sigma) + 1\{y^{(i)}=-1\}log(1-\phi) + 1\{y^{(i)}=-1\}logN(x^{(i)}|\mu_{-1}, \Sigma)\}
\end{aligned} \tag{6}$$

Let's first estimate ϕ . Consider the parts that contains ϕ, we have:

$$\sum_{i=1}^{m} [1\{y^{(i)=1}\}log\phi + 1\{y^{(i)}=1\}log(1-\phi)]$$

Take the derivative over ϕ:

$$\sum_{i=1}^{m} [1\{y^{(i)=1}\}\frac{1}{\phi} - 1\{y^{(i)}=1\}\frac{1}{1-\phi}]$$

Setting it to zero, we have

$$\phi = \frac{\sum_{i=1}^{m} 1\{y^{(i)}=1\}}{\sum_{i=1}^{m} 1\{y^{(i)}=1\} + \sum_{i=1}^{m} 1\{y^{(i)=-1}\}}$$

Since y = either 1 or -1,

$$\phi = \frac{1}{m}\sum_{i=1}^{m} 1\{y^{(i)}=1\}$$

We now move onto estimating μ_1 or μ_{-1}. Since both have identical form, we only need to consider one of mean vectors. Considering the parts contain μ_1, we have:

$$\begin{aligned}
&\sum_{i=1}^{m} 1\{y^{(i)}=1\}logN(x^{(i)}|\mu_1, \Sigma) \\
&= \sum_{i=1}^{m} 1\{y^{(i)}=1\}\frac{-1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) + const
\end{aligned} \tag{7}$$

Take the derivative over μ_1 and set it to zero, we have:

$$\sum_{i=1}^{m} 1\{y^{(i)}=1\}\Sigma^{-1}(x^{(i)}-\mu_1) = 0$$

$$\mu_1 = \frac{\sum_{i=1}^{m} 1\{y^{(i)}=1\}x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)}=1\}}$$

[2]I got this section down by referrencing http://web.engr.oregonstate.edu/~xfern/classes/cs534/notes/LDA.pdf
[3]1{y = i} is the notation introduced in the lecture notes 1 page 27.

Similiarly we have:

$$\mu_{-1} = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = -1\}x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = -1\}}$$

Finally, we will estimate Σ. Taking the part that contains Σ we have [4] :

$$\begin{aligned}
&\sum_{i=1}^{m}[1\{y^{(i)} = 1\}logN(x^{(i)}|\mu_1, \Sigma) + 1\{y^{(i)} = -1\}logN(x^{(i)}|\mu_{-1}, \Sigma)] \\
&= \sum_{i=1}^{m}[1\{y^{(i)} = 1\}[\frac{(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)}{-2} - \frac{1}{2}log|\Sigma|] + 1\{y^{(i)} = -1\}[\frac{(x-\mu_{-1})^T\Sigma^{-1}(x-\mu_{-1})}{-2} - \frac{1}{2}log|\Sigma|]] \\
&= -\frac{m}{2}log|\Sigma| - \frac{1}{2}\sum_{i=1}^{m}1\{y^{(i)} = 1\}Tr(\Sigma^{-1}((x^{(i)}-\mu_1)^T(x^{(i)}-\mu_1)) - \frac{1}{2}\sum_{i=1}^{m}1\{y^{(i)} = -1\}Tr(\Sigma^{-1}((x^{(i)}-\mu_{-1})^T(x^{(i)}-\mu_{-1})) \\
&= -\frac{m}{2}log|\Sigma| - \frac{m}{2}Tr(\frac{1}{m}\sum_{i=1}^{m}1\{y^{(i)} = 1\}\Sigma^{-1}((x^{(i)}-\mu_1)^T(x^{(i)}-\mu_1)) - \frac{m}{2}Tr(\sum_{i=1}^{m}1\{y^{(i)} = -1\}\Sigma^{-1}((x^{(i)}-\mu_{-1})^T(x^{(i)}-\mu_{-1}) \\
&= -\frac{m}{2}log|\Sigma| - \frac{m}{2}Tr(\Sigma^{-1}\frac{1}{m}(\sum_{i=1}^{m}1\{y^{(i)} = 1\}((x^{(i)}-\mu_1)^T(x^{(i)}-\mu_1) + \sum_{i=1}^{m}1\{y^{(i)} = -1\}((x^{(i)}-\mu_{-1})^T(x^{(i)}-\mu_{-1}))
\end{aligned} \tag{8}$$

Taking derivative over Σ [5] and set it to zero, we have

$$(\Sigma^{-1})^T + \frac{1}{m}(\sum_{i=1}^{m}1\{y^{(i)} = 1\}((x^{(i)}-\mu_1)^T(x^{(i)}-\mu_1)) + \sum_{i=1}^{m}1\{y^{(i)} = -1\}((x^{(i)}\mu_{-1})^T(x^{(i)}-\mu_{-1})))^T(-(\Sigma^{-1})^T)^2 = 0$$

Thus, we can get

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T.$$

Problem 4

4(a)

We will use induction to prove linear invariance of Newton's method. We already have $z^{(0)} = A^{-1}x^{(0)}$. Applying Newton's method to function f, we can have

$$x^{(i+1)} = x^{(i)} - H^{-1}\nabla_x(x^{(i)})$$

Applying Newton's method to function g, we can have

$$z^{(i+1)} = z^{(i)} - H^{-1}\nabla_z(z^{(i)})$$

Assuming and substutting $z^{(i)} = A^{-1}x^{(i)}$, we have

$$z^{(i+1)} = A^{-1}x^{(i)} - H^{-1}\nabla_x(A^{-1}x^{(i)})$$

[4] To get A'BA = Tr(BA'A), in which A is a n × 1 vector and B is a $n \times n$ square matrix, A'BA = $\sum_{i=1}^{n}\sum_{j=1}^{n} A_iB_{ij}A_j = \sum_{q=1}^{n}\sum_{p=1}^{n} B_{xq}A_pA_x = Tr(B(A'A))$.

[5] To figure out how to take derivative of a determinant, I read the lecture notes, Matrix Cookbook,https://www.ics.uci.edu/~welling/teaching/KernelsICS273B/MatrixCookBook.pdf .

Here we prove that $\nabla_x(Ax) = A\nabla_x x$,as long as A is independent with x. Since $\nabla_x(Ax)_p = \frac{\partial}{\partial x_p}\sum_{q=1}^n A_q A_p x_p = \sum_{q=1}^n A_q A_p \frac{\partial}{\partial x_p} x_p = (A\nabla_x(x))_p$, we can prove our conclusion. The similiar way for the hessian matrix can make it egligiable to move the constant matrix above the hessian.

$$\begin{aligned} z^{(i+1)} &= A^{-1}x^{(i)} - H^{-1}\nabla_x(A^{-1}x^{(i)}) \\ &= A^{-1}(x^{(i)} - H^{-1}\nabla_x(x^{(i)})) \\ &= A^{-1}x^{(i+1)} \end{aligned} \tag{9}$$

q.e.d. Similiar procedure can be applied to gradient optimazition to prove its linear invariance.

Problem 5

5(a)

5(a), i

We want to prove $\frac{1}{2}\sum_{i=1}^m w^{(i)}(\theta^T x^{(i)} - y^{(i)})^2 = (X\theta - \vec{y})^T W(X\theta - \vec{y})$, we choose W is a diagonal matrix by $m \times m$,

$$W = \begin{bmatrix} \frac{1}{2}w_1 & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{2}w_2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \frac{1}{2}w_m \end{bmatrix}$$

$$\begin{aligned} LHS &= J(\theta) \\ &= \frac{1}{2}\sum_{i=1}^m w^{(i)}(\theta^T x^{(i)} - y^{(i)})^2 \\ RHS &= (X\theta - \vec{y})^T W(X\theta - \vec{y})^T \\ &= [\dots \frac{1}{2}((x^{(i)})^T\theta - y^{(i)})w(i)\dots] \cdot \begin{bmatrix} \dots \\ ((x^{(i)})^T\theta - y^{(i)}) \\ \dots \end{bmatrix} \\ &= LHS \end{aligned} \tag{10}$$

q.e.d

5(a), ii

From i, we know that $J(\theta) = (X\theta - \vec{y})^T W(X\theta - \vec{y})$ under this weight setting. First, we will prove $\nabla_A(XA - Y)^T W(XA - Y) = 2W(X\theta - Y)X^T$, is W is symmetric.
Taking the derivative of the cost funciton with respect to θ, [6]

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta(X\theta - \vec{y})^T W(X\theta - \vec{y}) \\ &= \nabla_\theta(\theta^T X^T W X\theta - \theta^T X^T W\vec{y} - \vec{y}^T W X\theta + \vec{y}^T W\vec{y}) \\ &= \nabla_\theta Tr[\theta^T X^T W X\theta - \theta^T X^T W\vec{y} - \vec{y}^T W X\theta + \vec{y}^T W\vec{y}] \\ &= \nabla_\theta(Tr[\theta^T X^T W X\theta] - 2Tr[\vec{y}^T W X\theta]) \\ &= 2[X^T W^T X\theta - X^T W^T\vec{y}] \end{aligned} \tag{11}$$

Setting it to zero, we have

$$\theta = (X^T W^T X)^{-1} X^T W^T \vec{y}$$

[6] $\nabla_A(XA - Y)^T W(XA - Y) = 2W(X\theta - Y)X^T$, I refer to Matrix Cookbook at https://www.ics.uci.edu/ welling/teaching/KernelsICS273B/MatrixCookBook.pdf

5(a), iii

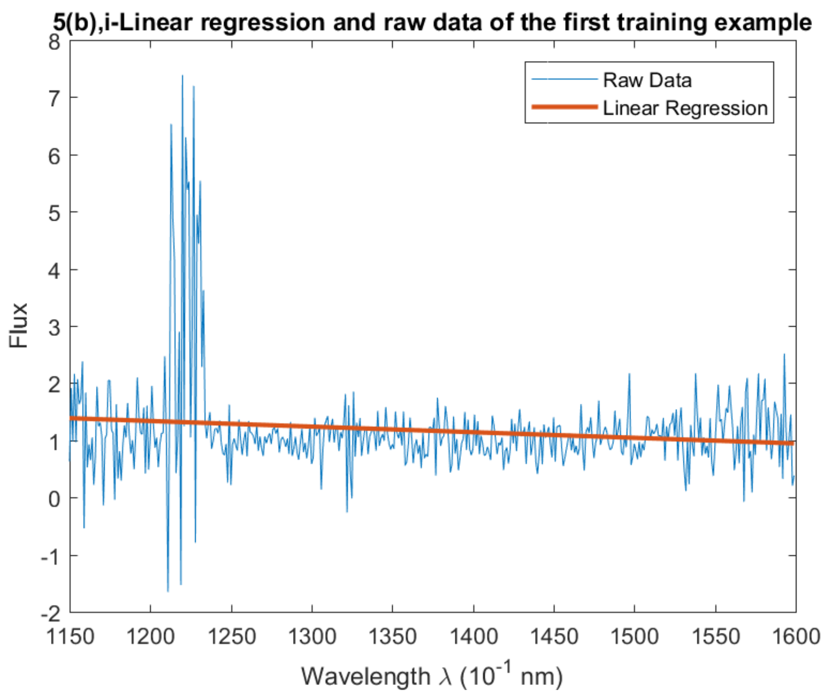
$$
\begin{aligned}
l(\theta) &= log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma^{(i)}} exp(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2}) \\
&= -\sum_{i=1}^{m} \frac{1}{2(\sigma^{(i)})^2} (y^{(i)} - \theta^T x^{(i)})) + z
\end{aligned} \tag{12}
$$

So we only need to optimize the first term. Compared with cost function, we can get $w^{(i)} = \frac{1}{2(\sigma^{(i)})^2}$.

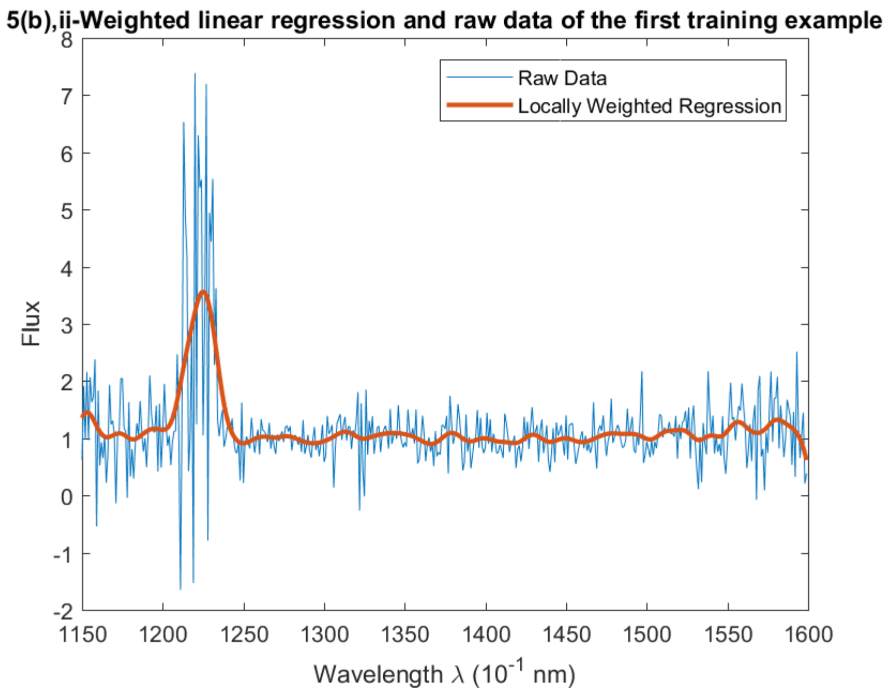
5(b)

5(b), i

The optimized θ is $[2.513 - 0.0009]^T$.

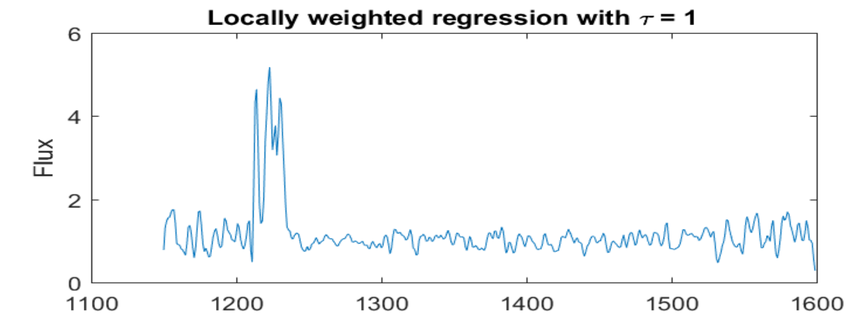


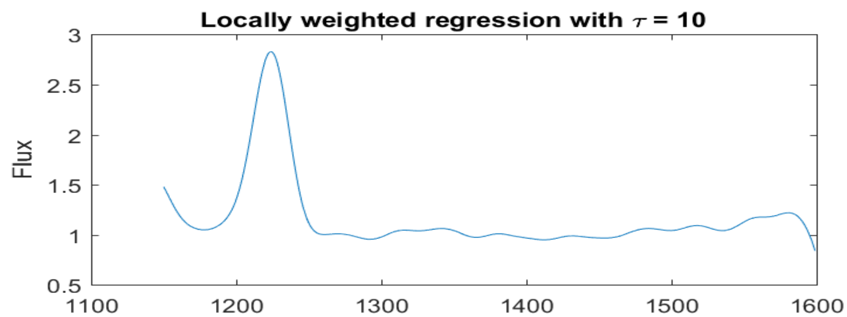
5(b), ii

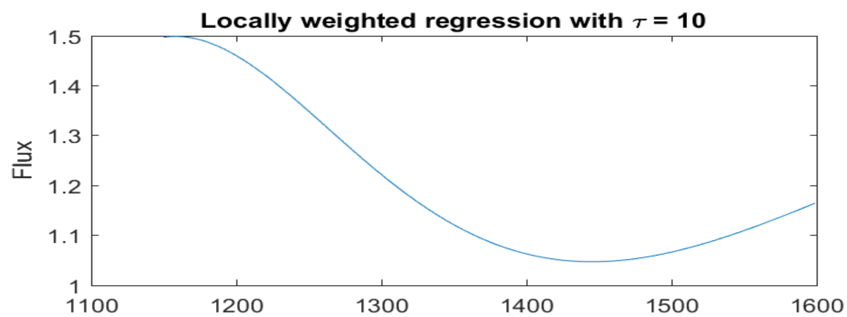


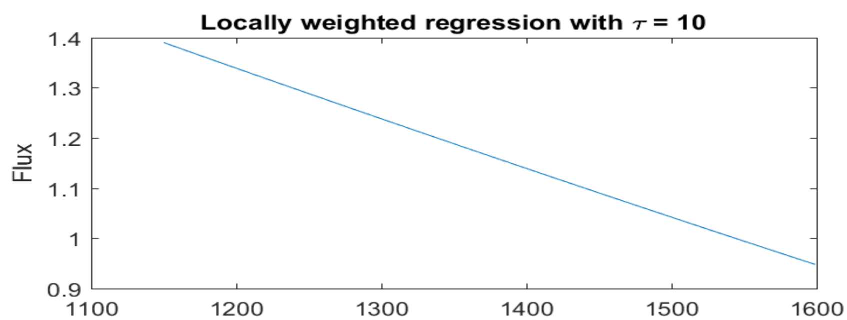
5(b),ii-Weighted linear regression and raw data of the first training example
Raw Data
Locally Weighted Regression
Flux
Wavelength λ (10^-1 nm)
8
7
6
5
4
3
2
1
0
-1
-2
1150
1200
1250
1300
1350
1400
1450
1500
1550
1600

5(b), iii









Below is the code for the section, 5(b),i,ii and iii. The function are seperated into another file for convienience.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  5(b) visualize the data
%%% i. demands a linear regression with
%%% visualization and optimized parameter;
%%% ii. visualize raw data and weighted
%%% linear regression;
%%% iii. vary the bandwidth parameter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all; clear; clc;

% define the query point
run 'load_quasar_data.m';
%load the first training example
trainEx1 = train_qso(1,:)';

        %%%%%%%%%% linear regression for the first training example %%%%%%%%%
%plot the raw data
figure
plot( lambdas, trainEx1);
title('5(b),i-Linear regression and raw data of the first training example');
xlabel('Wavelength \lambda (10^{-1} nm)');
ylabel('Flux');
hold on;

%compute theta using normal equation
X = [ones(size(lambdas)) lambdas];
theta = ( X' * X)^(-1) * X' * trainEx1;
disp(theta);

%plot the linear regression line
YHypoLin = X * theta;
p1 = plot(lambdas, YHypoLin);
legend('Raw Data', 'Linear Regression');
p1(1).LineWidth = 2;

saveas(gcf, '5b(i).png')


                  %%%%%%%%%    weighted linear regression %%%%%%%%%%
%plot the raw data
figure
plot( lambdas, trainEx1);
title('5(b),ii-Weighted linear regression and raw data of the first training example')
    ;
xlabel('Wavelength \lambda (10^{-1} nm)');
ylabel('Flux');
hold on;

% for each lambda compute the predict value
LWRX = X;
%the demension of sample space
m = size(LWRX, 1);
% the bandwidth parameter
```



```
tau = 5;
LWRY = smoothLWR(X, trainEx1, LWRX, tau);

%plot the locally weighted regression
p2 = plot(lambdas, LWRY);
ylim([-2 8]);
legend('Raw Data', 'Locally Weighted Regression')
p2(1).LineWidth = 2;
saveas(gcf, '5b(ii).png')



                    %%%%% vary the bandwidthparameter %%%%%

% when tau is 1
tau = 1;
LWRY = smoothLWR(X, trainEx1, LWRX, tau);

variedTau = figure(3);
set(variedTau, 'Position', [100, 100, 512, 1200]);
ax1 = subplot(4, 1, 1);
plot(ax1, lambdas, LWRY);
title(ax1, 'Locally weighted regression with \tau = 1');
ylabel('Flux');

% when tau is 10
tau = 10;
LWRY = smoothLWR(X, trainEx1, LWRX, tau);

ax2 = subplot(4, 1, 2);
plot(ax2, lambdas, LWRY);
title(ax2, 'Locally weighted regression with \tau = 10');
ylabel('Flux');

% when tau is 100
tau = 100;
LWRY = smoothLWR(X, trainEx1, LWRX, tau);

ax3 = subplot(4, 1, 3);
plot(ax3, lambdas, LWRY);
title(ax3, 'Locally weighted regression with \tau = 10');
ylabel('Flux');

% when tau is 100
tau = 1000;
LWRY = smoothLWR(X, trainEx1, LWRX, tau);

ax4 = subplot(4, 1, 4);
plot(ax4, lambdas, LWRY);
title(ax4, 'Locally weighted regression with \tau = 10');
ylabel('Flux');

saveas(gca, '5b(iii).png')
```

```
%%% comments on varied tau %%%
% When tau get bigger, the weight decrease quickly. Thus no input will have
% much difference on the regression model.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The function file for the section, 5(b),i,ii and iii is here.

```
function outputs = smoothLWR(trainInputs, trainOutputs, queryPoints, tau)
%%
% return smoothed curve
%-'trainInputs': training input data
%-'trainOutputs': training output data
%-'tau': also called as bandwidth parameter for the gaussian kernel
%%
outputs = zeros([size(queryPoints, 1) 1]);
% the demension of the sample size
m = size(trainInputs, 1);
for i = 1:m
    outputs(i) = LWRPredict(trainInputs, trainOutputs, queryPoints(i,:), tau);
end
end

function Output = LWRPredict(trainInputs, trainOutputs, queryPoint, tau)
%%
% return the predicted value for the query point
%-'trainInputs': training input data
%-'trainOutputs': training output data
%- 'queryPoints': the data points we want to predict on
%-'tau': also called as bandwidth parameter for the gaussian kernel
%%
W = getWeight(trainInputs, queryPoint, tau);
theta = (trainInputs' * W * trainInputs)^(-1)* trainInputs' * W * trainOutputs;
Output = queryPoint * theta;
end

function W = getWeight(trainInputs, queryPoint, tau)
%%
% get the diagonal weighted matrix
%-'trainInputs': training input data
%- 'queryPoints': the data points we want to predict on
%-'tau': also called as bandwidth parameter for the gaussian kernel
%%

% the demension of the training input
m = size(trainInputs, 1);
W = eye(m);
for i = 1: m
    distance = trace((trainInputs(i,:) - queryPoint)' * (trainInputs(i,:) - queryPoint
        ));
   W(i, i) = exp(- distance / (2* tau^2));
end
end
```

5(c)

5(c), i

Below is the matlab code for the section 5c, i. The function file are listed in section 5(b).

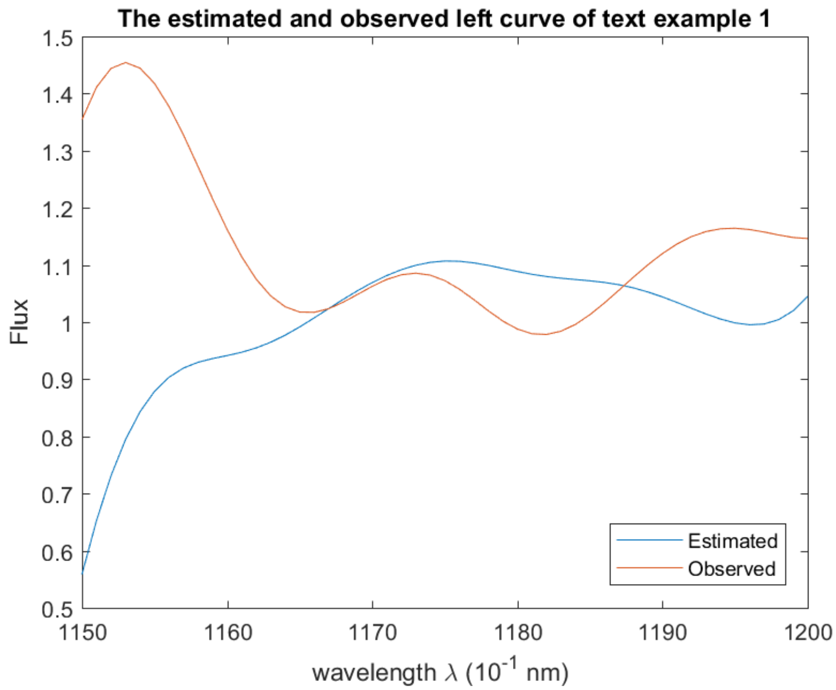
```
close all; clear; clc;
run 'load_quasar_data.m';

      %%%%% 5(c),i-smooth the entire dataset %%%%%%
% smooth the test and trian data set to get rid of random noise
smoothed_qso_train = smoothTrainSet(train_qso, lambdas);
smoothed_qso_test = smoothTrainSet(test_qso, lambdas);

save('smoothed');

function  smoothed_qso_train = smoothTrainSet(train_qso, lambdas)
%%
% return the smoothed training data set
%-'train_qso': the original training data set that contains random  noised
%-'lambdas': the training input wavelengths
%%
train_inputs = [ones(size(lambdas)) lambdas];
% bandwidth parameter
tau = 5;
%sample size
m = size(train_qso, 1);
smoothed_qso_train = zeros(size(train_qso));

for i = 1:m
    smoothed_qso_train(i,:) = smoothLWR(train_inputs, train_qso(i,:)', train_inputs,
        tau)';
end
end
```

5(c), ii

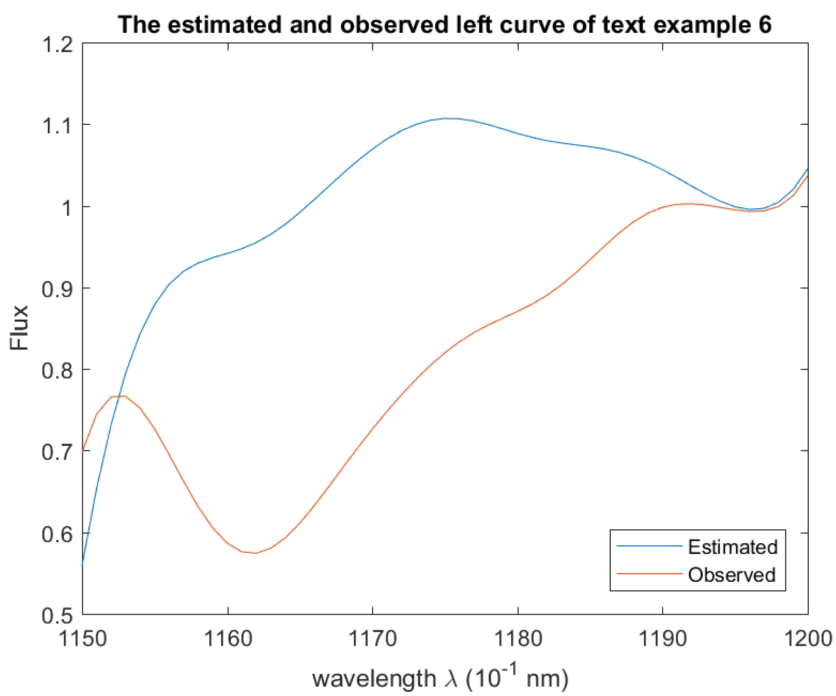
The average training error is 2.3084.
The average test error is 2.1080.

5(c), iii

Plot for test example 1:



Plot for test example 6:



Below is the matlab code for the section 5c,ii and iii.

```
close all; clear; clc;
load('smoothed');


%     %%%%% 5(c), ii - Function Estimator %%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%TEST FOR FUNCITON%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % test for getDistance method
% distance = getDistance(smoothed_qso_test(2,:), smoothed_qso_test(3,:), true);
% disp(distance);
%
% % test for getNeighbors and getDistances
% distances = getDistances(smoothed_qso_test, smoothed_qso_test(2,:));
% disp(distances);
% neighbors = getNeighbors(distances, 3);
% disp(neighbors);
%
% % test for the kerl function
% temp = ker(2);
% disp(temp);
%
% temp = ker(0.45);
% disp(temp);
%
% % test for estimateLeft method
% lambdasLeft = lambdas(1:51);
% lambdasRight = lambdas(151:end);
% f = smoothed_qso_train(3,:);
% fright = f(:,151:end);
% fleft =  f(:,1: 51);
% fleftEstimated = estimateLeft(smoothed_qso_train, fright, 3);
% plot(lambdasLeft, fleftEstimated);
% hold on;
% plot(lambdasLeft, fleft);
% disp(getDistance(fleftEstimated, fleft, false));
%
%%%%%%%%%%%%%%%%%%%%%END OF TEST%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% compute the training error
averageTrainError = getAverageTrainError(smoothed_qso_train, 3);
disp(averageTrainError);

% compute the test error
averageTestError = getAverageTestError(smoothed_qso_train, smoothed_qso_test, 3);
disp(averageTestError);

% plot for test example 1
figure
lambdasLeft = lambdas(1:51);
lambdasRight = lambdas(151:end);
f = smoothed_qso_train(1,:);
fright = f(:,151:end);
fleft =  f(:,1: 51);
fleftEstimated = estimateLeft(smoothed_qso_train, fright, 3);
plot(lambdasLeft, fleftEstimated);
hold on;
plot(lambdasLeft, fleft);
title('The estimated and observed left curve of text example 1');
xlabel('wavelength \lambda (10^{-1} nm) ');
```

```
ylabel('Flux');
legend('Estimated','Observed','Location', 'southeast');
saveas(gcf,'5c(iii)-testEx1.png');

%plot for the test example 6
figure
lambdasLeft = lambdas(1:51);
lambdasRight = lambdas(151:end);
f = smoothed_qso_train(6,:);
fright = f(:,151:end);
fleft =  f(:,1: 51);
fleftEstimated = estimateLeft(smoothed_qso_train, fright, 3);
plot(lambdasLeft, fleftEstimated);
hold on;
plot(lambdasLeft, fleft);
title('The estimated and observed left curve of text example 6');
xlabel('wavelength \lambda (10^{-1} nm) ');
ylabel('Flux');
legend('Estimated','Observed','Location', 'southeast');
saveas(gcf,'5c(iii)-testEx6.png');

function averageTestError = getAverageTestError(trainSet, testSet, k)
%%
% return the average test error for the entire test data set
% -'trainSet'; the training data set
% -'testSet': the test data set
% -'k': the number of nearest neighbors we want to focous on
%%
% the test size
m = size(testSet, 1);
testSetRight = trainSet(:,151: end);
testSetLeft = trainSet(:,1: 51);
testEstimated = zeros(size(testSetLeft));
testErrors = zeros(size(testSet, 1), 1);
for i = 1: m
    testEstimated(i,:) = estimateLeft(trainSet, testSetRight(i,:), k);
    testErrors(i,:) = getDistance(testSetLeft(i,:), testEstimated(i,:), false);
end
averageTestError = sum(testErrors) / m;
end

function averageTrainError = getAverageTrainError(trainSet, k)
%%
% return the estimated left funtion for the entire training right funciton;
% -'trainSet': the training data set;
% -'k': the number of nearest neighbors we want to foucus on.
%%
trainSetRight = trainSet(:,151: end);
trainSetLeft = trainSet(:,1: 51);
trainEstimated = zeros(size(trainSetLeft));
errors = zeros(size(trainSet, 1), 1);
% the size of training set
m = size(trainSet, 1);
```

```
for i = 1:m
    trainEstimated(i,:) = estimateLeft(trainSet, trainSetRight(i,:), k);
    errors(i) = getDistance(trainSetLeft(i,:), trainEstimated(i,:), false);
end
averageTrainError = sum(errors) / m;
end

function fleft = estimateLeft(trainSet, fright, k)
%%
% return the estimated left function
% -'trainSet'; the training data set
% -'fright': the observed right function
% -'k': the number of nearest neighbors
%%
% cut the training set into left and right part
trainSetRight = trainSet(:,151: end);
trainSetLeft = trainSet(:,1: 51);

distances = getDistances(trainSetRight, fright);
neighbors = getNeighbors(distances, k);
h = max(distances);

%comopute the numerator and xc
numerator = zeros(1, size(trainSetLeft, 2));
denominator = 0;
for i = 1:k
neighbor = neighbors(k);
numerator = numerator + ker(distances(neighbor) / h) * trainSetLeft(i,:);
denominator = denominator +ker(distances(neighbor) / h);

end

fleft = numerator / denominator;
end

function  temp = ker(t)
temp = max([1-t; 0]);
end

function neighbors = getNeighbors(distances, k)
%%
% return the K neighbors that is closet to our query funciton
%-'distances': is the vector containing distance of each training example
%-'K': the numbers of neighbors we want to find, must be a postitive
%integers
%%
neighbors = zeros(k, 1);
for i = 1:k
    [~, neighbors(i)] = min(distances,  [], 'omitnan');
    % update the the min to NaN for the next closet neighbor
    distances(neighbors(i)) = NaN;
end
end
```

```
function distance = getDistance(f1, f2, right)
diff = (f1 - f2).^2;
if right == true
    % 151 is index of the wavelength 1300 among the lambdas vector
    distance = sum(diff(151:end));
else
    % 51 is the index of the wavelength 1200 among the lambdas vector
    distance = sum(diff(1:51));
end
end

function distances = getDistances(trainSet, f)
%%
% return the all samples' distances relative to the function f.
%-'trainSet': the training set that contains both left and right examples
%-'f': the smoothed funciton
%%
diff = trainSet - f;
% only cumulate the right function distance
diff = diff(:,151:end).^2;
temp = cumsum(diff, 2);
distances = temp(:, end);
end
```