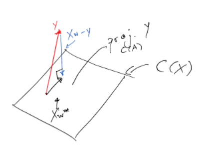
1.1 We have the following equation for linear regression:

$$Xw = y$$

where $X \in \mathbb{R}^{m \times n}$, $w \in \mathbb{R}^{n}$, $y \in \mathbb{R}^{m}$



- Suppose that the columnspace of X, $C(X)$, does not include y, then we want to make Xw as close to y

$$Xw^* = \text{proj}_{C(A)}\, y$$

$$\Rightarrow \min \|y - Xw\|$$

⇒ The least-squares approximation is just the projection of y onto $C(X)$

$$\Rightarrow Xw^* = \text{proj}_{C(X)}\, y$$

$$\Rightarrow Xw^* - y = \text{proj}_{C(X)}\, y - y$$

But $Xw^* - y \in C(X)^{\perp}$

$$\rightarrow C(X)^{\perp} = N(X^T)$$

$$\Rightarrow Xw^* - y \in N(X^T)$$

$$\Rightarrow X^T(Xw^* - y) = 0$$

$$\Rightarrow X^T X w^* - X^T y = 0$$

$$\Rightarrow X^T X w^* = X^T y$$

$$\Rightarrow \boxed{w^* = (X^T X)^{-1} X^T y \quad \text{Eq. 1.1}}$$

1.2 The residual sum-of-squares is defined as

$$RSS = \sum_{i=1}^{N} (y_i - f(x_i))^2$$

In our case, we have $y_p = X_v w^*$, where $X_v \equiv$ the validation data

$w^* \equiv$ the weights of our linear regression model

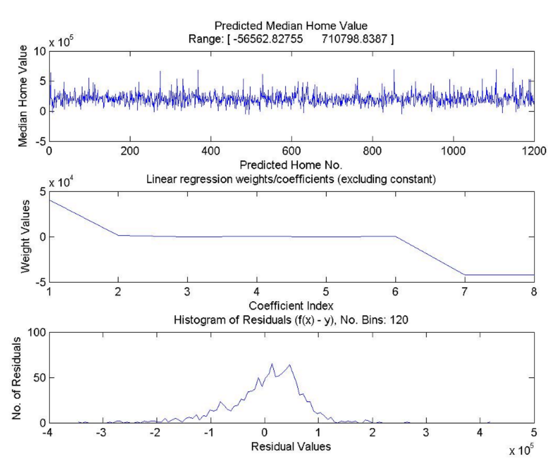
$y_p \equiv$ the predicted results

$y_v \equiv$ the validation (actual) value

$$\Rightarrow \boxed{RSS = (y_v - y_p)^T (y_v - y_p) \quad \text{Eq. 1.2}}$$

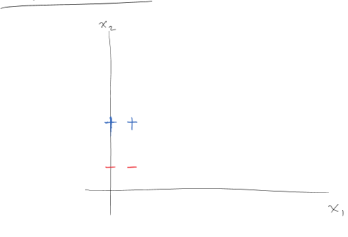
The range of the predicted median home values is [-56562.8275467875 710798.838690845]. The lower range does not make sense, because this would mean that these median home values are negative .

1.3

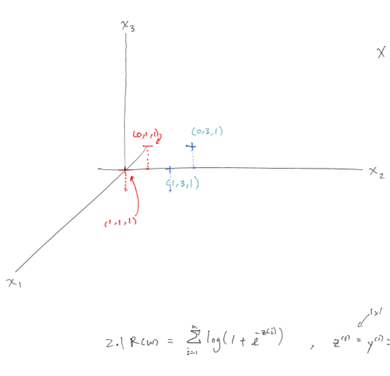


1.4 The histogram resembles the normal distribution. The histogram is plotted above.

#2



$$X = \begin{bmatrix} 0 & 3 & 1 \\ 1 & 3 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$



2.1 $R(w) = \sum_{i=1}^{n} \log(1 + e^{-z^{(i)}})$, $z^{(i)} = y^{(i)} f(x^{(i)})$ 1x1, $f(x^{(i)}) = w^T x^{(i)}$ ← 1x1

$$z = \begin{bmatrix} y_1 f_1 \\ \vdots \\ y_n f_n \end{bmatrix} = \text{diag}(y) f, \quad f = Xw, \quad \text{diag}(y) = \text{n×n square matrix with elements of vector } y \text{ on the diagonal}$$

$$\Rightarrow z = \text{diag}(y) X w \Rightarrow z \text{ is an } n \times 1 \text{ vector}, \quad Q = \text{diag}(y) X$$

$$\Rightarrow R = \sum_{i=1}^{n} \log(1 + e^{-Q_i w}), \quad \frac{\partial R}{\partial w} = \begin{bmatrix} \frac{\partial R}{\partial w_1} \\ \vdots \\ \frac{\partial R}{\partial w_d} \end{bmatrix} \qquad R = 1^T \log(1 + e^{-Qw})$$

Let $R_i = \log(1 + e^{-Q_i w})$

$$\frac{\partial R_i}{\partial w_1} = \frac{\partial R_i}{\partial w_1}\left[\log\left(1 + e^{-(Q_{i1} w_1 + \dots + Q_{id} w_d)}\right)\right]$$

$$= \frac{\partial R_i}{\partial w_1}\left[\log\left(1 + e^{-Q_{i1} w_1} \cdot e^{-Q_{i2} w_2} \cdots e^{-Q_{id} w_d}\right)\right]$$

$$= \left(\frac{-Q_{i1} e^{-Q_{i1} w_1} \cdot e^{-Q_{i2} w_2} \cdots e^{-Q_{id} w_d}}{1 + e^{-Q_{i1} w_1} \cdot e^{-Q_{i2} w_2} \cdots e^{-Q_{id} w_d}}\right)$$

$$\frac{\partial R_i}{\partial w_j} = \frac{-Q_{ij} e^{-Q_i w}}{1 + e^{-Q_i w}} = \frac{-Q_{ij}}{1 + e^{Q_i w}}$$

$$\Rightarrow \frac{\partial R}{\partial w_j} = \frac{-Q_{1j}}{1 + e^{Q_1 w}} + \dots + \frac{-Q_{nj}}{1 + e^{Q_n w}}$$

$$\Rightarrow \frac{\partial R}{\partial w} = \begin{bmatrix} \frac{-Q_{11}}{1 + e^{Q_1 w}} + \dots + \frac{-Q_{n1}}{1 + e^{Q_n w}} \\ \vdots \\ \frac{-Q_{1d}}{1 + e^{Q_1 w}} + \dots + \frac{-Q_{nd}}{1 + e^{Q_n w}} \end{bmatrix} \Rightarrow -Q^T \begin{bmatrix} \frac{1}{1 + e^{Q_1 w}} \\ \vdots \\ \frac{1}{1 + e^{Q_n w}} \end{bmatrix}$$

d×1 (d×n)(n×1) (n×1) (n×n)(n×1)

$$\boxed{\nabla_w R = \frac{\partial R}{\partial w} = -Q^T \frac{1}{1 + e^{Qw}}, \text{ where } e^{Qw} \in \mathbb{R}^n, \ Q = \text{diag}(y)X, \ \text{diag}(y) = \text{n×n square matrix with elements of vector } y \text{ on the diagonal}}$$

2.2

$$\nabla_w^2 R = \begin{bmatrix} \frac{\partial^2 R}{\partial w_1^2} & \frac{\partial^2 R}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 R}{\partial w_1 \partial w_d} \\ \frac{\partial^2 R}{\partial w_2 \partial w_1} & \frac{\partial^2 R}{\partial w_2^2} & \cdots & \frac{\partial^2 R}{\partial w_2 \partial w_d} \\ \vdots & & \ddots & \vdots \\ \frac{\partial^2 R}{\partial w_d \partial w_1} & \cdots & \cdots & \frac{\partial^2 R}{\partial w_d^2} \end{bmatrix}$$

From 1.1, we found the relation

$$\frac{\partial R}{\partial w_j} = \frac{-Q_{1j}}{1 + e^{Q_1 w}} + \dots + \frac{-Q_{nj}}{1 + e^{Q_n w}}$$

Let's find the relation $\frac{\partial^2 R}{\partial w_j \partial w_k}$,

$$\Rightarrow \frac{\partial^2 R}{\partial w_j \partial w_k} = \frac{\partial}{\partial w_k}\left(\frac{\partial R}{\partial w_j}\right)$$

Let $\frac{\partial R_i}{\partial w_j} = \frac{-Q_{ij}}{1 + e^{Q_i w}} = -Q_{ij}(1 + e^{Q_i w})^{-1}$

$$\Rightarrow \frac{\partial}{\partial w_k}\left(\frac{\partial R_i}{\partial w_j}\right) = -Q_{ij}\left[(-1)(1 + e^{Q_i w})^{-2}(Q_{ik} e^{Q_i w})\right]$$

$$= Q_{ij} Q_{ik} e^{Q_i w}(1 + e^{Q_i w})^{-2}$$

$$= Q_{ij} Q_{ik} e^{Q_i w}(1 + e^{Q_i w})^{-2}$$

$$\Rightarrow \frac{\partial^2 R}{\partial w_j \partial w_k} = \frac{\partial}{\partial w_k}\left(\frac{\partial R}{\partial w_j}\right) = \frac{Q_{1j} Q_{1k} e^{Q_1 w}}{(1 + e^{Q_1 w})^2} + \dots + \frac{Q_{nj} Q_{nk} e^{Q_n w}}{(1 + e^{Q_n w})^2}$$

$$= \sum_{i=1}^{n} \frac{Q_{ij} Q_{ik} e^{Q_i w}}{(1 + e^{Q_i w})^2}$$

Let $a_i = \frac{e^{Q_i w}}{(1 + e^{Q_i w})^2}$

$$\Rightarrow \frac{\partial^2 R}{\partial w_j \partial w_k} = \sum_{i=1}^{n} Q_{ij} Q_{ik} a_i = [Q_{1j} Q_{1k} \cdots Q_{nj} Q_{nk}] \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}$$

$$= Q_{ij}^T \text{diag}(Q_{ik}) a, \text{ where } Q_{ij} = \begin{bmatrix} Q_{1j} \\ \vdots \\ Q_{nj} \end{bmatrix}, \ Q_{ik} = \begin{bmatrix} Q_{1k} \\ \vdots \\ Q_{nk} \end{bmatrix}, \ a = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}$$

$$\Rightarrow \nabla_w^2 R = \begin{bmatrix} Q_{i1}^T \text{diag}(Q_{i1}) a & Q_{i1}^T \text{diag}(Q_{i2}) a & \cdots & Q_{i1}^T \text{diag}(Q_{id}) a \\ Q_{i2}^T \text{diag}(Q_{i1}) a & Q_{i2}^T \text{diag}(Q_{i2}) a & \cdots & Q_{i2}^T \text{diag}(Q_{id}) a \\ \vdots & & & \vdots \\ Q_{id}^T \text{diag}(Q_{i1}) a & Q_{id}^T \text{diag}(Q_{i2}) a & \cdots & Q_{id}^T \text{diag}(Q_{id}) a \end{bmatrix}$$

could not simplify

2.3

(a) For logistic regression, we use the following probability function:

$$\mu^{(i)} = P(Y = 1 | X = x) = \frac{1}{1 + e^{-z_i}} = \frac{1}{1 + e^{-Qw_i}}, Q = diag(y)X$$

u0 =
0.9526
0.7311
0.2689
0.7311

(b) According to wikipedia, To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point.

The next point can be computed as follows:

$$w_{n+1} = w_n - \eta \nabla_w R$$

$$w_{n+1} = w_n - \eta\left[-Q^T \frac{1}{1 + e^{Qw_n}}\right]$$

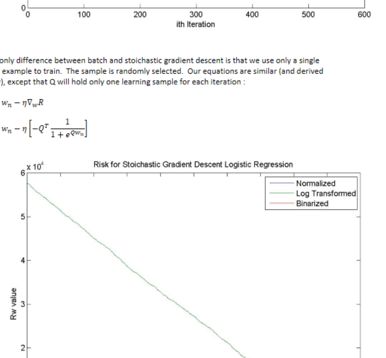
w1 =
-2
0.9491
-0.68363

(c)
u1 =
0.89694
0.54083
0.43402
0.84999

(d)
w2 =
-1.6908
1.9198
-0.83739

u2 =
0.99277
0.96199
0.25305
0.64758

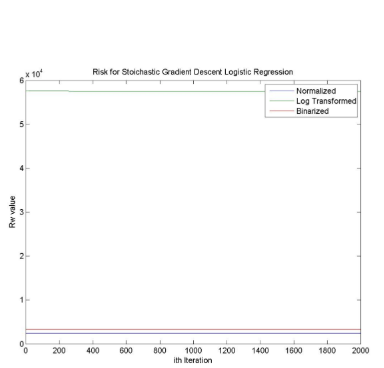
3.1



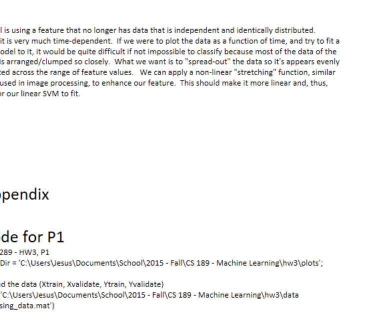
3.2 The only difference between batch and stoichastic gradient descent is that we use only a single learning example to train. The sample is randomly selected. Our equations are similar (and derived similarly), except that Q will hold only one learning sample for each iteration :

$$w_{n+1} = w_n - \eta \nabla_w R$$

$$w_{n+1} = w_n - \eta\left[-Q^T \frac{1}{1 + e^{Qw_n}}\right]$$



3.3



4. Daniel is using a feature that no longer has data that is independent and identically distributed. Indeed, it is very much time-dependent. If we were to plot the data as a function of time, and try to fit a linear model to it, it would be quite difficult if not impossible to classify because most of the data of the feature is arranged/clumped so closely. What we want is to "spread-out" the data so it's appears evenly distributed across the range of feature values. We can apply a non-linear "stretching" function, similar to ones used in image processing, to enhance our feature. This should make it more linear and, thus, easier for our linear SVM to fit.

Appendix

Code for P1

```
%CS 289 - HW3, P1
plotsDir = 'C:\Users\Jesus\Documents\School\2015 - Fall\CS 189 - Machine Learning\hw3\plots';

%Load the data (Xtrain, Xvalidate, Ytrain, Yvalidate)
load('C:\Users\Jesus\Documents\School\2015 - Fall\CS 189 - Machine Learning\hw3\data
\housing_data.mat')

%% 1.1 Linear Regression

%assign the data to our equation variables
a = ones(numel(Ytrain),1);
X = [a Xtrain];
y = Ytrain;

%compute the weights using Equation 1.1
w = inv(X'*X)*X'*y;

%% 1.2 Residual sum-of-squares (RSS)

%assign data to our equation variables
b = ones(numel(Yvalidate),1);
Xv = [b Xvalidate]; %
yv = Yvalidate;

%compute the predicted value
yp = Xv*w;

%compute the RSS
RSS = (yv - yp)'*(yv-yp);

%plot the predicted values and get the range
ypMax = max(yp);
ypMin = min(yp);
ypRange = [ypMin ypMax];
disp(ypRange);
h = figure('visible', 'on','units', 'normalized','outerposition',[0 0 1 1]);
subplot(3,1,1), plot(yp);
title({'Predicted Median Home Value'; ['Range: [ ' num2str(ypRange) ' ]']});
xlabel('Predicted Home No.');
ylabel('Median Home Value');

%% 1.3 Plot w as function of its index
subplot(3,1,2), plot(w(2:end));
title('Linear regression weights/coefficients (excluding constant)')
xlabel('Coefficient Index');
ylabel('Weight Values');

%% Plot a histogram of the residuals (f(x) - y)
rezids = yp - yv;
nBins = numel(rezids)/10;

%get the range of the residuals for plotting
rezMax = max(rezids);
rezMin = min(rezids);
rezRangeDiff = rezMax - rezMin;

%Create a vector for the x dimension of the histogram
xHistStep = rezRangeDiff/(nBins-1);
xHist = rezMin:xHistStep:rezMax;

[myHist, histCen] = hist(rezids, nBins);
subplot(3,1,3), plot(xHist, myHist);
title(['Histogram of Residuals (f(x) - y), No. Bins: ' num2str(nBins)]);
xlabel('Residual Values');
ylabel('No. of Residuals');


saveas(h, [plotsDir '\P1.jpg']);
```

Code for P2

```
%% 2.3 (a)

%Define our variables
X = [0 3 1; 1 3 1; 0 1 1; 1 1 1]; %make sure to append a third dimension to X
y = [1; 1; -1; -1];
w0 = [-2; 1; 0];
Q = diag(y)*X;

%compute u0
u0 = 1./(1 + exp(-Q*w0));
disp('u0 = ');
disp(u0);

wc = w0;

for ii = 1:1:10
    %compute the next set of w's
    n = 1; %nabla
    wn = wc - n*(- Q'*(1./(1 + exp(Q*wc))));

    %Compute the probabilities u
    u = 1./(1 + exp(-Q*wn));

    if (ii == 1) || (ii == 2)
       %display the weights w
       disp(['w' num2str(ii) ' = ' ])
       disp(num2str(wn));
       disp(' ');

       %display the probabilities u
       disp(['u' num2str(ii) ' = ' ]);
       disp(num2str(u));
       disp(' ');
    end


    %set wc
    wc = wn;
end
```

Code for P3

```
%CS 289 - HW3, P3
plotsDir = 'C:\Users\Jesus\Documents\School\2015 - Fall\CS 189 - Machine Learning\hw3\plots';


%Load the data (Xtrain, Xtest, Ytrain)
load('C:\Users\Jesus\Documents\School\2015 - Fall\CS 189 - Machine Learning\hw3\data\spam.mat')

[mX, nX] = size(Xtrain);

%Standardize each column so they each have mean 0 and unit variace.  This
%is the same thing as getting the "z" value, which can be computed as
%follows
%
%     zi = (xi - xm)/SD,
%                    where xi is the ith sample, xm is the mean, and
%                    SD is the standard deviation
Xm = repmat(mean(Xtrain), mX,1);
Xsd = repmat(std(Xtrain), mX,1);
Xz = (Xtrain - Xm)./Xsd;

%Transform the features using x(i,j) <==  log(x(i,j) + 1)
Xlog = log(Xtrain + 1);

% Binarize the features x(i,j) <== I(x(i,j) > 0) I denotes an indicator
% variable.  Let's make the condition the mean of each feature.
XB = Xtrain;
XB(Xtrain>= Xm) = 1;
XB(Xtrain<Xm) = 0;


Xcell = {Xz; Xlog; XB};
RWcell = cell(size(Xcell));

%% 3.1
%{
Y = Ytrain;


for ii = 1:1:numel(Xcell);
    X = Xcell{ii};

    nIts = 2000;
    [ wc, u, RWcell{ii} ] = log_reg_batch(X, Y, nIts );

end

inds = 1:1:nIts;
h = figure('visible', 'on','units', 'normalized','outerposition',[0 0 1 1]);
plot(inds, RWcell{1}, inds, RWcell{2}, inds, RWcell{3}),
title('Risk for Batch Gradient Descent Logistic Regression');
xlabel('ith Iteration');
ylabel('Rw value');
legend('Normalized', 'Log Transformed', 'Binarized');
saveas(h, [plotsDir '\BatchGradDescent.jpg']);
%}


%% 3.2
%
Y = Ytrain;
nablaBool = false;

for ii = 1:1:numel(Xcell);
    X = Xcell{ii};

    nIts = 2000;
    [ wc, u, Rwtemp ] = log_reg_stoic(X, Y, nIts, nablaBool);
    RWcell{ii} = Rwtemp;

end

inds = 1:1:nIts;
h = figure('visible', 'on','units', 'normalized','outerposition',[0 0 1 1]);
plot(inds, RWcell{1}, inds, RWcell{2}, inds, RWcell{3}),
title('Risk for Stoichastic Gradient Descent Logistic Regression');
xlabel('ith Iteration');
ylabel('Rw value');
legend('Normalized', 'Log Transformed', 'Binarized');
saveas(h, [plotsDir '\StoichGradDescent.jpg']);
%}

%% 3.3
Y = Ytrain;
nablaBool = true;

for ii = 1:1:numel(Xcell);
    X = Xcell{ii};

    nIts = 2000;
    [ wc, u, Rwtemp ] = log_reg_stoic(X, Y, nIts, nablaBool);
    RWcell{ii} = Rwtemp;

end

inds = 1:1:nIts;
h = figure('visible', 'on','units', 'normalized','outerposition',[0 0 1 1]);
plot(inds, RWcell{1}, inds, RWcell{2}, inds, RWcell{3}),
title('Risk for Stoichastic Gradient Descent Logistic Regression');
xlabel('ith Iteration');
ylabel('Rw value');
legend('Normalized', 'Log Transformed', 'Binarized');
saveas(h, [plotsDir '\StoichNabCorrGradDescent.jpg']);



%%%%%%%%%%%FUNCTIONS%%%%%%%%%%%%%%%%%%%%%%%%
function [ wc, u, Rw ] = log_reg_batch(X, Y, nIts )
%This function uses batch gradient descent to obtain parameters to the
%logistic regression model:
%
%p(Y = 1 | X = x) = 1./(1 + exp(-z))

numSamples = size(X,1);
numFeats = size(X, 2);

%make sure that our variables are doubles
X = double(X);
Y = double(Y);

%Make w0 the mean of our features
w0 = mean(X)';%rand(numFeats,1); %

%Define our variables
Q = diag(Y)*X;

%compute u0
u0 = 1./(1 + exp(-Q*w0));


wc = w0;

zThreshLow = -100;
zThreshHigh = 200;

%Preallocate variables
Rw = zeros(nIts,1);
OneT = ones(1, numSamples);
for ii = 1:1:nIts

    %Compute the current z
    zc = Q*wc;
    zc(zc<=zThreshLow) = zThreshLow; %Limit the range
    zc(zc>=zThreshHigh) = zThreshHigh;

    %make sure to restrict the values of z

    %compute the next set of w's
    n = 1; %nabla
    wn = wc - n*(- Q'*(1./(1 + exp(zc))));

    %Compute the probabilities u
    zn = Q*wn;
    zn(zn<=zThreshLow) = zThreshLow;
    zn(zn>=zThreshHigh) = zThreshHigh;

    u = 1./(1 + exp(-zn));

    %{
    if (ii == 1) || (ii == 2)
       %display the weights w
      disp(['w' num2str(ii) ' = ' ])
      disp(num2str(wn));
      disp(' ');

      %display the probabilities u
      disp(['u' num2str(ii) ' = ' ]);
      disp(num2str(u));
      disp(' ');
    end
    %}

    %Compute the Risk for each iteration and store in Rw
    Rw(ii) = OneT*log(1 + exp(-zn));

    %set wc
    wc = wn;
end


end


function [ wc, u, Rw  ] = log_reg_stoic(X, Y, nIts, nablaBool )
%This function uses stoichastic gradient descent to obtain parameters to
%the logistic regression model:
%
%p(Y = 1 | X = x) = 1./(1 + exp(-z))
%
%The only difference between this and batch gradient descent is that we use
%only one learning example for each iteration, chosen at random at each
%iteration as well.
%
%nablaBool if this is high, then nabla = 1/t, where t is the ith iteration
numExamples = size(X,1);
numFeats = size(X, 2);

%make sure that our variables are doubles
X = double(X);
Y = double(Y);

%Make w0 the mean of our features.
w0 = mean(X)';%rand(numFeats,1); %

%Define our variables
Q = diag(Y)*X;



%compute u0
u0 = 1./(1 + exp(-Q*w0));


wc = w0;

zThreshLow = -100;
zThreshHigh = 200;

%Preallocate variables
Rw = zeros(nIts,1);
OneT = ones(1, size(Q,1));

numSamples = 1; %floor(numExamples/100);
for ii = 1:1:nIts

    %Randomly sample learning examples from Q

    Qt = Q(randsample(numExamples,numSamples), :);


    %Compute the current z
    zc = Qt*wc;
    zc(zc<=zThreshLow) = zThreshLow; %Limit the range
    zc(zc>=zThreshHigh) = zThreshHigh;

    %make sure to restrict the values of z

    %Compute nabla
    if nablaBool
       n = .001/ii;
    else
       n = .001;
    end

    %compute the next set of w's


    wn = wc - n*(- Qt'*(1./(1 + exp(zc))));

    %Compute the probabilities u
    zn = Q*wn;
    zn(zn<=zThreshLow) = zThreshLow;
    zn(zn>=zThreshHigh) = zThreshHigh;

    u = 1./(1 + exp(-zn));

    %{
    if (ii == 1) || (ii == 2)
       %display the weights w
      disp(['w' num2str(ii) ' = ' ])
      disp(num2str(wn));
      disp(' ');

      %display the probabilities u
      disp(['u' num2str(ii) ' = ' ]);
      disp(num2str(u));
      disp(' ');
    end
    %}

    %Compute the Risk for each iteration and store in Rw
    Rw(ii) = OneT*log(1 + exp(-zn));

    %set wc
    wc = wn;
end


end
```