Homework 3 by Jiabo Cheng

Problem 1

Generate Data

For this problem, we will generate 3-dimensional real valued data from 4 classes with uniform priors. Each class-conditional Gaussian distribution has shown below:

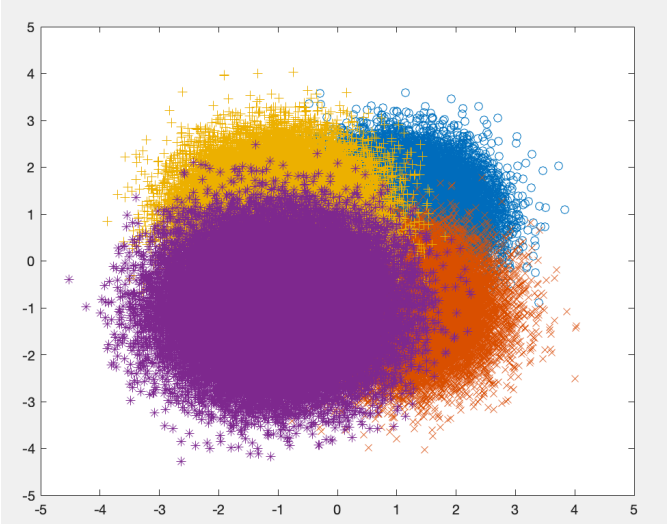
$$m_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad C_1 = \begin{bmatrix} 0.4 & 0 & 0 \\ 0 & 0.4 & 0 \\ 0 & 0 & 0.4 \end{bmatrix}$$

$$m_2 = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} \quad C_2 = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}$$

$$m_3 = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} \quad C_3 = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}$$

$$m_4 = \begin{bmatrix} -1 \\ -1 \\ 0 \end{bmatrix} \quad C_4 = \begin{bmatrix} 0.7 & 0 & 0 \\ 0 & 0.7 & 0 \\ 0 & 0 & 0.7 \end{bmatrix}$$

The visualization of data distribution is shown below:



Theoretically Minimum Probability of Error

Using the minimum-probability-of-error-rule:

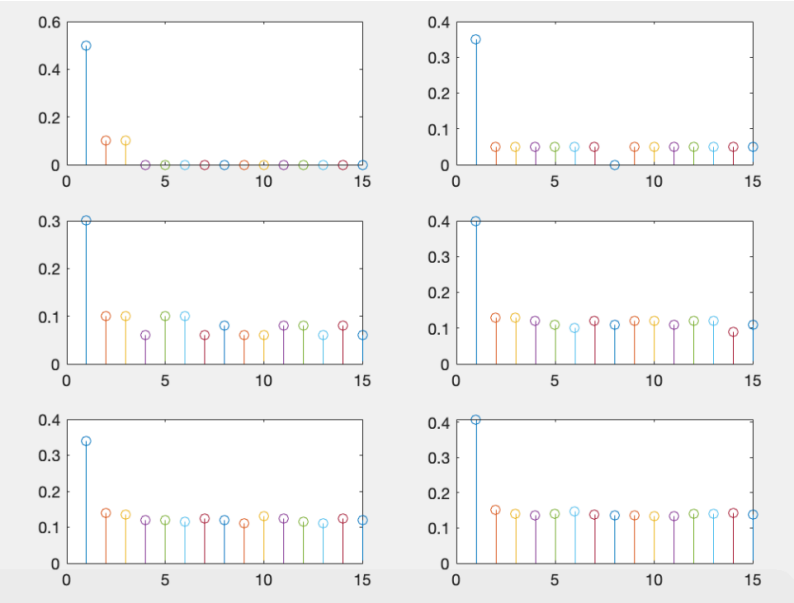
$$\begin{aligned} P(x) &= \underset{d \in \{1,\dots,A\}}{\operatorname{argmin}} \; Risk(D(x) = d \mid x) \\ &= \underset{d \in \{1,\dots,A\}}{\operatorname{argmin}} \; \sum_{l=1}^{C} \lambda_{dl} \, P(L = l \mid x) \\ &= \underset{d \in \{1,\dots,A\}}{\operatorname{argmin}} \; \sum_{l=1}^{C} \lambda_{dl} \, \frac{P(x \mid L = l) \, P(L = l)}{P(x)} \end{aligned}$$

$$\Rightarrow \begin{vmatrix} R(D=1 \mid x) \\ \vdots \\ R(D=A \mid x) \end{vmatrix} = \underbrace{\Lambda}_{\text{Loss Matrix}} \begin{bmatrix} P(L=1 \mid x) \\ \vdots \\ P(L=C \mid x) \end{bmatrix}$$

The Theoretical Minimum Probability of Error is **15.42%**

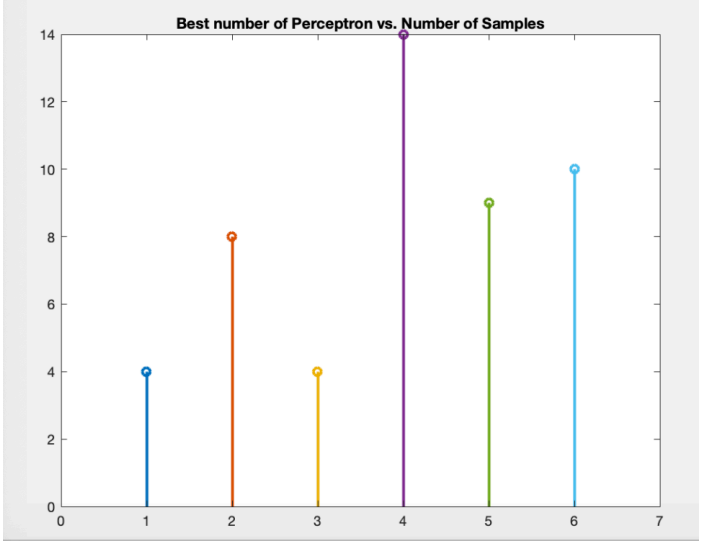
Model Order Selection

For each of the training sets with different number of samples, perform 10-fold cross-validation, and using minimum classification error probability as the objective function to select the best number of perceptrons. The plot below shows the change of minimum classification error probability vs. number of perceptions.



When number of perceptrons increase, the minimum error probability decrease.

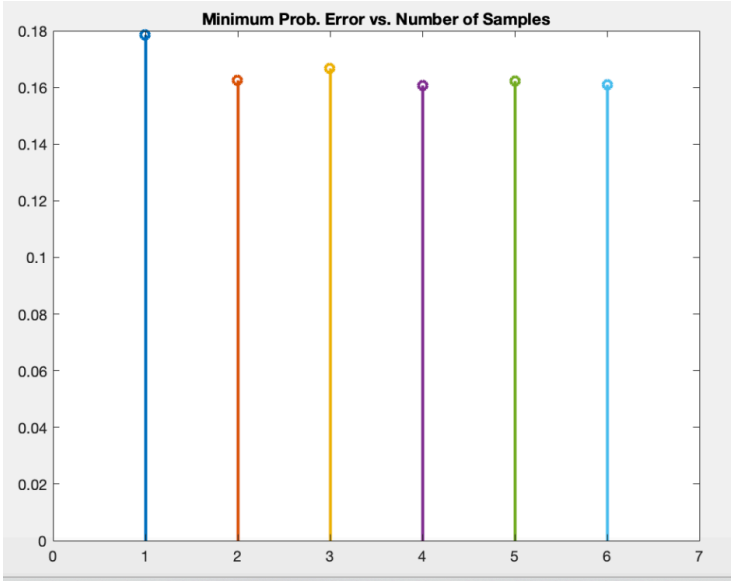
The plot shows the relationship between number of samples vs. best number of perceptrons.



With the larger training set used, the selected number of perceptrons tend to be larger. With more perceptrons, the MLP model could fit the more complex dataset than using lesser perceptrons.

Performance Assessment:
The plot below shows the minimum error probability using the final trained MLP models (Select number of perceptrons the same as the best number of perceptrons in k-fold cross validation)

The result shows when using the larger training set, minimum error probability will closer to theoretical minimum error probability, but always slightly higher than the theoretical one. Because the parameters used



in MLP are estimated based on training set, it is impossible having the same minimum error probability as theoretical.

Problem2

Generate data
generate data from distributions below:
a = [2 7 2 3 7 2 7]
μ_x = [0.9293, 0.6124, 1.2296, 2.2586, 1.7968, 0.5191, 2.2734]
Σ_x =

1.1171	0.2622	0.3139	0.2012	0.2315	0.3305	0.3390
0.2622	1.2539	0.3484	0.2434	0.1702	0.3263	0.1908
0.3139	0.3484	1.4992	0.3209	0.1894	0.2973	0.3926
0.2012	0.2434	0.3209	1.0935	0.3074	0.2468	0.3587
0.2315	0.1702	0.1894	0.3074	1.4694	0.1974	0.2105
0.3305	0.3263	0.2973	0.2468	0.1974	1.0687	0.2293
0.3390	0.1908	0.3926	0.3587	0.2105	0.2293	1.4419

$\mu_z = 0$
$\Sigma_z = \alpha I$

$\mu_v = 0$
$\Sigma_v = 1$
Calculate a new random variable as $y = \alpha^T(x + z) + v$

Model parameter Estimation:
Assuming the relationship between x and y is linear: $y = w^T x + w_0 + v$, using the k-fold cross validation and MAP parameter estimation to determine β.

$$\hat{w} = \arg\max_w \log P(w|D)$$

$$= \arg\max_w \log \left(\frac{P(D|w)\cdot P(w)}{P(D)}\right)$$

$$\hat{w} = \arg\max_w \left[\log P(D|w) + \log P(w)\right]$$

$$\log P(D|w) = \log \prod \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y-w^T x}{\sigma}\right)^2} \quad \text{pdf: } \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

$$= \sum_{i=1}^{n} \log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2}\left[\frac{y-w^T x}{\sigma}\right]^2$$

$$\log P(w) = \log \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{-w^T w}{\tau}\right)}$$

$$= \log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2}\left[\frac{-w^T w}{\tau}\right]$$

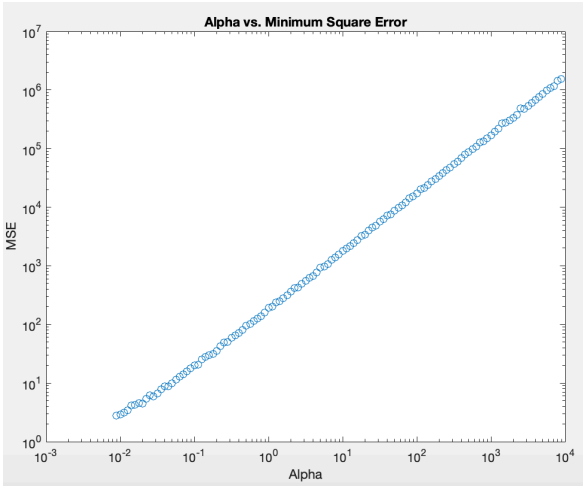
$$\hat{w} = \arg\max_w \Sigma\left(-\frac{1}{\sigma^2}(y-w^T x)^2 - \frac{w^T w}{\sigma^2}\right)$$

$$= \arg\max_w \Sigma\left(y^2 - 2yw^T x + w^T x x^T w\right) + \frac{\sigma^2}{\tau^2} w^T w$$

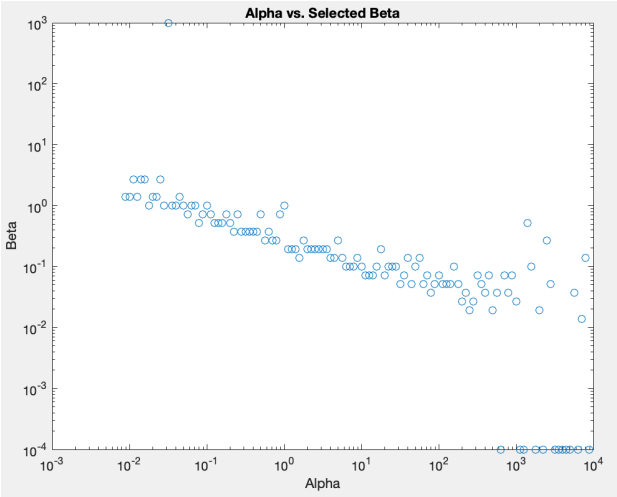
$$\frac{\partial}{\partial w} = 0$$

$$\hat{w}_{map} = \left(X X^T + \frac{\sigma^2}{\tau^2}\right)^{-1} yx$$

Hyper-parameter Optimization:
The plot below shows the relationship between alpha and MSE:



As the alpha increase, the more noise is implemented into the training set, so the performance of model will be worse, which lead to higher minimum square error.



The plot shows the relationship between alpha and beta.

The best selected beta decreases when alpha increase(the noisy is more intense). The reason is when training set is more and more hard to predict, ridge regression would like to penalty the weight for x and increase the weigh for prior in order for better performance.

Appendix

Question1:

```
clear all;
close all;
n=3;
C=4;
N_train = [100,200,500,1000,2000,5000];


P = repmat(1/C,1,C);
Mu = [1 1 0; 1 -1 0; -1 1 0; -1 -1 0]';
cov_factor = [0.4 0.5 0.5 0.7]; %factor to change overlap
for i = 1:C
Sigma(:,:,i) = cov_factor(i)*eye(3);
end
LossM = ones(C,C)-eye(C);


[x_o, label_o] = generate_data(C,n,200000,P,Mu,Sigma,1); % generate data


ind_valid = ceil(200000*rand(1,100000)); %------valid
x_valid = x_o(:,ind_valid);
label_valid = label_o(:,ind_valid);


%theoretical p_error
for i = 1:C
pxgiven(i,:) = eval_g(x_valid,Mu(:,i),Sigma(:,:,i));

end
px = P*pxgiven;
classPosteriors = pxgiven.*repmat(P',1,length(x_valid))./repmat(px,C,1);
expectedR = LossM*classPosteriors;
[~,decisions]=min(expectedR,[],1);


count = 0;
for i= 1:length(px)
if label_valid(decisions(i),i)==1
count = count+1;
end
end
p_err_theo = 1-(count/length(px))


%for loop
Num_n = 6;%---------------
result = zeros(3,Num_n);
p_err_final = zeros(1,Num_n);
for E = 1:Num_n
%E=3

tic

ind = ceil(200000*rand(1,N_train(E))); %------train
x = x_o(:,ind);
label = label_o(:,ind);
```

```
maxnP = 15 %---------------------
p_err_nP = zeros(1,maxnP);
for nP = 1:maxnP

nPerceptrons = nP;
sizeParams = [n;nPerceptrons;C];
%kfold
K=10;%----------------
dummy = ceil(linspace(0,N_train(E),K+1));
p_err_K = zeros(1,K);
parfor k = 1:K %enable parrallel
K_valid = x(:,[dummy(k)+1:dummy(k+1)]);
label_valid_K = label(:,[dummy(k)+1:dummy(k+1)]);
train_ind = setdiff([1:N_train(E)],[dummy(k)+1:dummy(k+1)]);
K_train = x(:,train_ind);
label_K = label(:,train_ind);
% Initialize model parameters
paramsA = 1e-1*randn(nPerceptrons,n);
paramsb = 1e-1*randn(nPerceptrons,1);
paramsC = 1e-1*randn(C,nPerceptrons);
paramsd = mean(label_K,2);%zeros(nY,1); % initialize to mean of y
%params = paramsTrue;
vecParamsInit = [paramsA(:);paramsb;paramsC(:);paramsd];
%vecParamsInit = vecParamsTrue; % Override init weights with true weights


options = optimset('MaxFunEvals',1e5*length(vecParamsInit)); % Matlab default is
200*length(vecParamsInit)


% MaxFunEvals is the maximum number of function evaluations you allow


vecParams = fminsearch(@(vecParams)
(objectiveFunction(K_train,label_K,sizeParams,vecParams)),vecParamsInit,options);


%validate
paramsA = reshape(vecParams(1:n*nPerceptrons),nPerceptrons,n);
paramsb = vecParams(n*nPerceptrons+1:(n+1)*nPerceptrons);
paramsC = reshape(vecParams((n+1)*nPerceptrons+1:(n+1+C)*nPerceptrons),C,nPerceptrons);
paramsd = vecParams((n+1+C)*nPerceptrons+1:(n+1+C)*nPerceptrons+C);
H = mlpModel(K_valid,paramsA,paramsb,paramsC,paramsd);


count = 0;
for i= 1:length(H)
[~,class] = max(int8(H(:,i)));
[~,Label] = max(label_valid_K(:,i));
if class == Label
count = count+1;
end
end
p_err_K(k) = 1-(count/length(H));
end
[E,nP] %PROGRESS
p_err_nP(nP) = min(p_err_K)
figure(2)
subplot(3,2,E)
stem(nP,p_err_nP(nP)),hold on, drawnow,

end
[p_err,bestnP] =min(p_err_nP);
result(1,E) = bestnP;
result(2,E) = p_err;

% final training
%for loop
Num_iterate = 5 %---------------
p_err_f = zeros(1,Num_iterate);
for M = 1:Num_iterate
nPerceptrons = bestnP;
```

```
sizeParams = [n;nPerceptrons;C];
paramsA = 1e-1*randn(nPerceptrons,n);
paramsb = 1e-1*randn(nPerceptrons,1);
paramsC = 1e-1*randn(C,nPerceptrons);
paramsd = mean(label,2);%zeros(nY,1); % initialize to mean of y
%params = paramsTrue;
vecParamsInit = [paramsA(:);paramsb;paramsC(:);paramsd];
%vecParamsInit = vecParamsTrue; % Override init weights with true weights


options = optimset('MaxFunEvals',1e6*length(vecParamsInit));


vecParams = fminsearch(@(vecParams)
(objectiveFunction(x,label,sizeParams,vecParams)),vecParamsInit,options);


%validate
paramsA = reshape(vecParams(1:n*nPerceptrons),nPerceptrons,n);
paramsb = vecParams(n*nPerceptrons+1:(n+1)*nPerceptrons);
paramsC = reshape(vecParams((n+1)*nPerceptrons+1:(n+1+C)*nPerceptrons),C,nPerceptrons);
paramsd = vecParams((n+1+C)*nPerceptrons+1:(n+1+C)*nPerceptrons+C);
H = mlpModel(x_valid,paramsA,paramsb,paramsC,paramsd);


count = 0;
for i= 1:length(H)
[~,class] = max(int8(H(:,i)));
[~,Label] = max(label_valid(:,i));
if class == Label
count = count+1;
end
end
p_err_f(M) = 1-(count/length(H))
end
p_err_final(E)= min(p_err_f);
result(3,E) = p_err_final(E)
toc
figure(3)% size vs np
stem(E,bestnP,'LineWidth',2),hold on, drawnow,
xlim([0 7]);
title('Best number of Perceptron vs. Number of Samples');


figure(4) % size vs pe
stem(E,p_err_final(E),'LineWidth',2),hold on, drawnow,
xlim([0 7]);
title('Minimum Prob. Error vs. Number of Samples');
end


function Value = objectiveFunction(X,Y,sizeParams,vecParams)
N = size(X,2); % number of samples
nX = sizeParams(1);
nPerceptrons = sizeParams(2);
nY = sizeParams(3);
paramsA = reshape(vecParams(1:nX*nPerceptrons),nPerceptrons,nX);
paramsb = vecParams(nX*nPerceptrons+1:(nX+1)*nPerceptrons);
paramsC = reshape(vecParams((nX+1)*nPerceptrons+1:
(nX+1+nY)*nPerceptrons),nY,nPerceptrons);
paramsd = vecParams((nX+1+nY)*nPerceptrons+1:(nX+1+nY)*nPerceptrons+nY);
H = mlpModel(X,paramsA,paramsb,paramsC,paramsd);
% Change the objective function appropriately
%objFncValue = sum(sum((Y-H).*(Y-H),1),2)/N; % MSE for regression under AWGN model
Value = sum(-sum(Y.*log(H),1),2)/N; % CrossEntropy for ClassPosterior approximation
end


function H = mlpModel(X,paramsA,paramsb,paramsC,paramsd)
N = size(X,2);                          % number of samples
nY = length(paramsd);                  % number of outputs
U = paramsA*X + repmat(paramsb,1,N);
```

```
Z = U./sqrt(1+U.^2);     %activation function
V = paramsC*Z + repmat(paramsd,1,N);


H = exp(V)./repmat(sum(exp(V),1),nY,1); % softmax nonlinearity for second/last layer

end


function [x, label] = generate_data(C,n, N, P, Mu, Sigma,i)
x = zeros(n,N);
label = zeros(C,N);


cumP= [0 cumsum(P)];
u = rand(1,N);
for i=1:length(P)
    ind = find(u>=cumP(i) & u <cumP(i+1));
    label(i,ind)=1;
    x(:,ind) = mvnrnd(Mu(:,i),Sigma(:,:,i),length(ind))';
end
%subplot(3,2,i);
plot(x(1,label(1,:)==1),x(2,label(1,:)==1),'o',x(1,label(2,:)==1),x(2,label(2,:)==1),'x'
,x(1,label(3,:)==1),x(2,label(3,:)==1),'+',x(1,label(4,:)==1),x(2,label(4,:)==1),'*');

end


function g = eval_g(x,mu,Sigma)
% Evaluates the Gaussian pdf N(mu,Sigma) at each coumn of X
[n,N] = size(x);
invSigma = inv(Sigma);
C = (2*pi)^(-n/2) * det(invSigma)^(1/2);
E = -0.5*sum((x-repmat(mu,1,N)).*(invSigma*(x-repmat(mu,1,N))),1);
g = C*exp(E);
end
```

Question2

```
close all;
clear all;


n = 7;
N_train = 100;
N_valid = 10000;


a = ceil(n*rand(n,1));
Mu = 3*rand(n,1); %adjust
Sigma = eye(n)+ 0.2*rand(n,n); %adujust
Sigma = Sigma'*Sigma;

alpha_loop = 10.^(-3:0.05:3)*trace(Sigma);

Mu_z = zeros(n,1);

Mu_v = 0;
Sigma_v = 1;

beta_loop =logspace(-4,3);

Mu_w = 0;

%loop
for i = 1:length(alpha_loop)
Sigma_z = alpha_loop(i) * eye(n);
x_train = mvnrnd(Mu,Sigma, N_train)';
z_train = mvnrnd(Mu_z,Sigma_z, N_train)';
v_train = Mu_v + Sigma_v*randn(N_train,1);
```

```
y_train = (a'*(x_train+z_train))'+v_train;

x_valid = mvnrnd(Mu,Sigma, N_valid)';
z_valid = mvnrnd(Mu_z,Sigma_z, N_valid)';
v_valid = Mu_v + Sigma_v*randn(N_valid,1);

y_valid = (a'*(x_valid+z_valid))'+v_valid;

for j =1:length(beta_loop)

%start k fold
K = 10;
dummy = ceil(linspace(0,N_train,K+1));
for k = 1:K
K_x_valid = x_train(:,[dummy(k)+1:dummy(k+1)]);
K_y_valid = y_train([dummy(k)+1:dummy(k+1)]);
train_ind = setdiff([1:N_train],[dummy(k)+1:dummy(k+1)]);
K_x_train = x_train(:,train_ind);
K_y_train = y_train(train_ind);

A = [ones(1,length(K_x_train)); K_x_train];

            w_est = inv((A*A')+ Sigma_v^2/
beta_loop(j)^2*eye(size(A,1)))*(K_y_train'*A')';

            valid_A = [ones(1,length(K_x_valid)); K_x_valid];

            MSE_K(k) = mean((K_y_valid'-w_est'*valid_A).^2);

        end
        avgMSE(j) = mean(MSE_K);

    end
    [minMSE,min_ind] = min(avgMSE)

    A=[ones(1,N_train); x_train];
    w_est = inv((A*A')+ Sigma_v^2/beta_loop(min_ind)^2*eye(size(A,1)))*(y_train'*A')';
    log_MSE_train = (mean((y_train'-w_est'*A).^2));

    A_valid = [ones(1,N_valid); x_valid]
    MSE_valid = mean((y_valid'-w_est'*A_valid).^2);

    pxgivenwx = -2*log(eval_g(y_valid',(w_est'*A_valid),Sigma_v));

    %result
    MSE(i) = MSE_valid
    log_MSE_train(i) = log_MSE_train
    beta_s(i) = beta_loop(min_ind)
    alpha_a(i) = alpha_loop(i)


end


figure(1)
loglog(alpha_a,beta_s,'o')
xlabel('Alpha');
ylabel('Beta');
title('Alpha vs. Selected Beta')


figure(2)
loglog(alpha_a,MSE,'o')
xlabel('Alpha');
ylabel('MSE');
title('Alpha vs. Minimum Square Error')


figure(3)
loglog(1:length(beta_loop),avgMSE,'o')

function g = eval_g(x,mu,Sigma)
```

```
% Evaluates the Gaussian pdf N(mu,Sigma) at each coumn of X
[n,N] = size(x);
invSigma = inv(Sigma);
C = (2*pi)^(-n/2) * det(invSigma)^(1/2);
E = -0.5*sum((x-mu).*(invSigma*(x-mu)),1);
g = C*exp(E);
end
```