Assignment 3
Juan Pablo Bernal
6/4/2020

1. Question 1

The Data was generated from 5 different 3-dimensional single Gaussians with equal priors (Figure. 1). 7 different Datasets were generated, 6 for training and 1 for validation. The training datasets respectively contained 100, 200, 500, 1000, 2000, and 5000 samples and the test set contained 10000 samples. The mean vectors and covariance matrices used to generate the data were:

$$\mu_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \Sigma_1 = \begin{bmatrix} 2 & -0.2 & 0.2 \\ -0.2 & 2 & -0.2 \\ 0.2 & -0.2 & 2 \end{bmatrix} \qquad \mu_2 = \begin{bmatrix} 3 \\ 0 \\ 3 \end{bmatrix} \quad \Sigma_2 = \begin{bmatrix} 2 & -0.4 & 0.3 \\ -0.4 & 2 & -0.4 \\ 0.3 & -0.4 & 2 \end{bmatrix}$$

$$\mu_3 = \begin{bmatrix} 0 \\ 2.8 \\ 0 \end{bmatrix} \quad \Sigma_3 = \begin{bmatrix} 2 & 0.4 & 0.3 \\ 0.4 & 2 & 0.4 \\ 0.3 & 0.4 & 2 \end{bmatrix} \qquad \mu_4 = \begin{bmatrix} -3 \\ 0 \\ -3 \end{bmatrix} \quad \Sigma_4 = \begin{bmatrix} 1 & -0.5 & 0.5 \\ -0.5 & 1 & -0.5 \\ 0.5 & -0.5 & 1 \end{bmatrix}$$

$$\mu_5 = \begin{bmatrix} 0 \\ -2.8 \\ 0 \end{bmatrix} \quad \Sigma_5 = \begin{bmatrix} 1 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 1 \end{bmatrix}$$

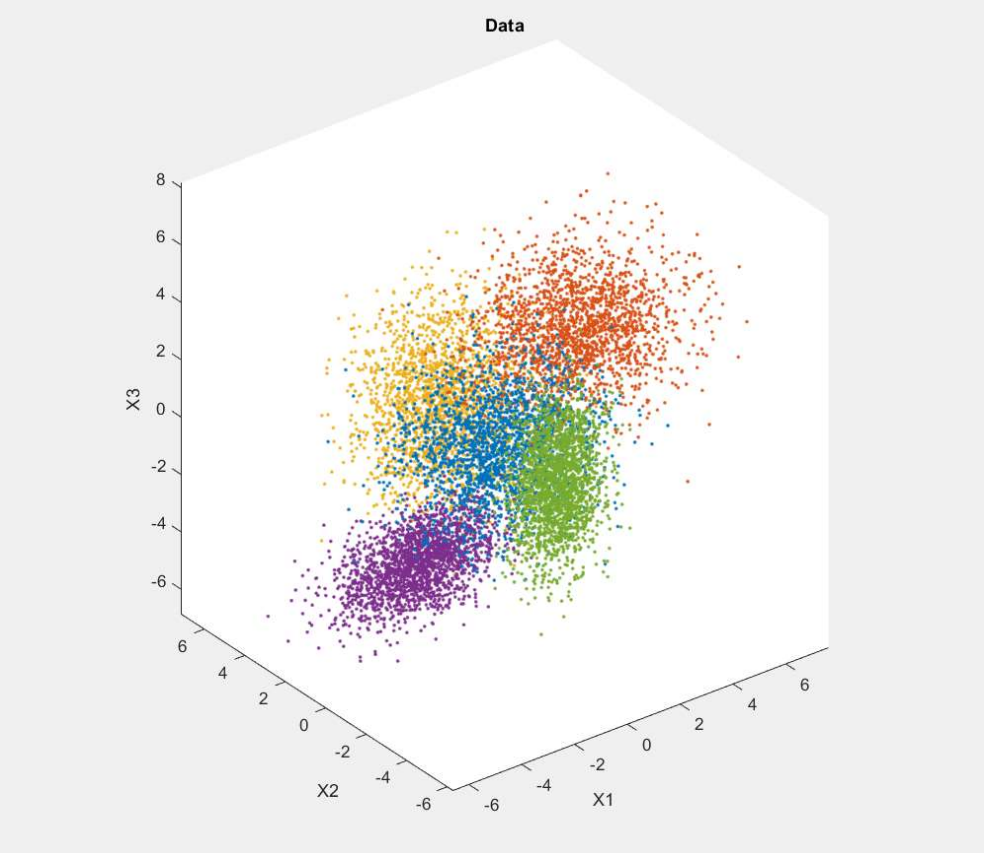

Figure. 1 10k Samples Data Distribution

Once the data was generated, the theoretical minimum probability of error was empirically calculated by using a Maximum Likelihood classifier. Since the class priors are all equal the MAP classifier reduces to an ML classifier. The ML classification rule used was: $D = argmax_l P(x \mid L = l)$. Using the true parameters, the samples in the validation dataset were classified and the minP error calculated (by using counts of decisions vs truth, how it has been done in previous assignments). It was found to be 0.1502 or 15%.

The goal was to design and train a 2-layer neural network that would approximate the class posteriors of each class. The first layer was a hidden layer of perceptrons with a non-linear activation function and the second layer was a softmax function. The function of the first layer was an ISRU function $a = \frac{x}{\sqrt{1+x^2}}$ and the softmax function was $m = \frac{e^{v_i}}{\sum_{j=1}^{5} e^{V_j}}$ thus the final NN equation was $h = m(b_2 + w_2 a\,(b_1 + w_1 x))$ where the b_s and w_s are the parameters to be estimated and h is a vector containing the class posteriors $P(L = l \mid x)$. The training of the NN was done by estimating the parameters that minimized the cross-entropy loss of the NN. This was done using the MATLab fminsearch function where the parameters were randomly initialized. To reduce the chances of getting stuck in a local minimum, the fminsearch was performed 3 times, each time the parameters randomly reinitialized and the ones that produced the minimum cross-entropy loss were chosen as the optimal solution.

To select the optimal model order (number of perceptrons to be used in the first layer) a cross-validation method using k-fold was used. First, for one perceptron, the data was shuffled (to avoid selecting all samples from one class) and divided into 10 equal subsets (10-fold). Validation data was chosen to be one of the subsets and training data was the remaining subsets. Then, by using the training set the neural network (with one perceptron) was trained. Then, the validation data was passed through the trained NN, validated using a MAP classifier $(D = argmax_l P(L = l \mid x))$ and the minimum probability of error was estimated (same way as before). This was done 10 times to use each of the subsets as validation data and the average probability of error for the 10 experiments was used as the minP error of the model. Next, then the number of perceptrons was increased by one, and the process was repeated. If the minP error of the new model was lower than the previous one, the number of perceptrons would be incremented and the algorithm ran again. This would be done until the minP error of a model was higher than the previous one. In which case the selected model order would be the current number of perceptrons - 1 (the one that produced the minimum minP error). This cross-validation algorithm was run using each of the 6 training datasets and the selected model order for each can be found in table 1.

Once the model for each training dataset was selected, a new NN was trained using the entire set (6 trained NNs). Next, each NN was tested with the 10k samples dataset. Using again the MAP classifier used in the cross-validation section. Finally, the probability of error for each model was empirically estimated in the same way as before(Table. 1).

It can be seen that as the number of samples increases the minP error decreases each time getting closer to the theoretical value found at the beginning. From 1000 samples to 2000 samples it increased a small amount maybe because in the fminsearch in 2000samples-training got stuck in a local minimum that performed worse than the minimum found in the 1000samples-training. Finally, Figure.2 graphically shows the results. The orange constant line is the theoretical minimum probability of error. The X-axis represents the semilog number of samples used to train the NN and the Y-axis represents the minimum probability of error achieved by the models. The orange circles are the exact value given by each NN and the blue line is the trendline that describes the behavior as the number of samples increases.

# of Samples	# of Perceptrons	minP error
100	3	0.2545
200	2	0.2052
500	4	0.1673
1000	5	0.1663
2000	5	0.1676
5000	6	0.1622

Table. 1 Results

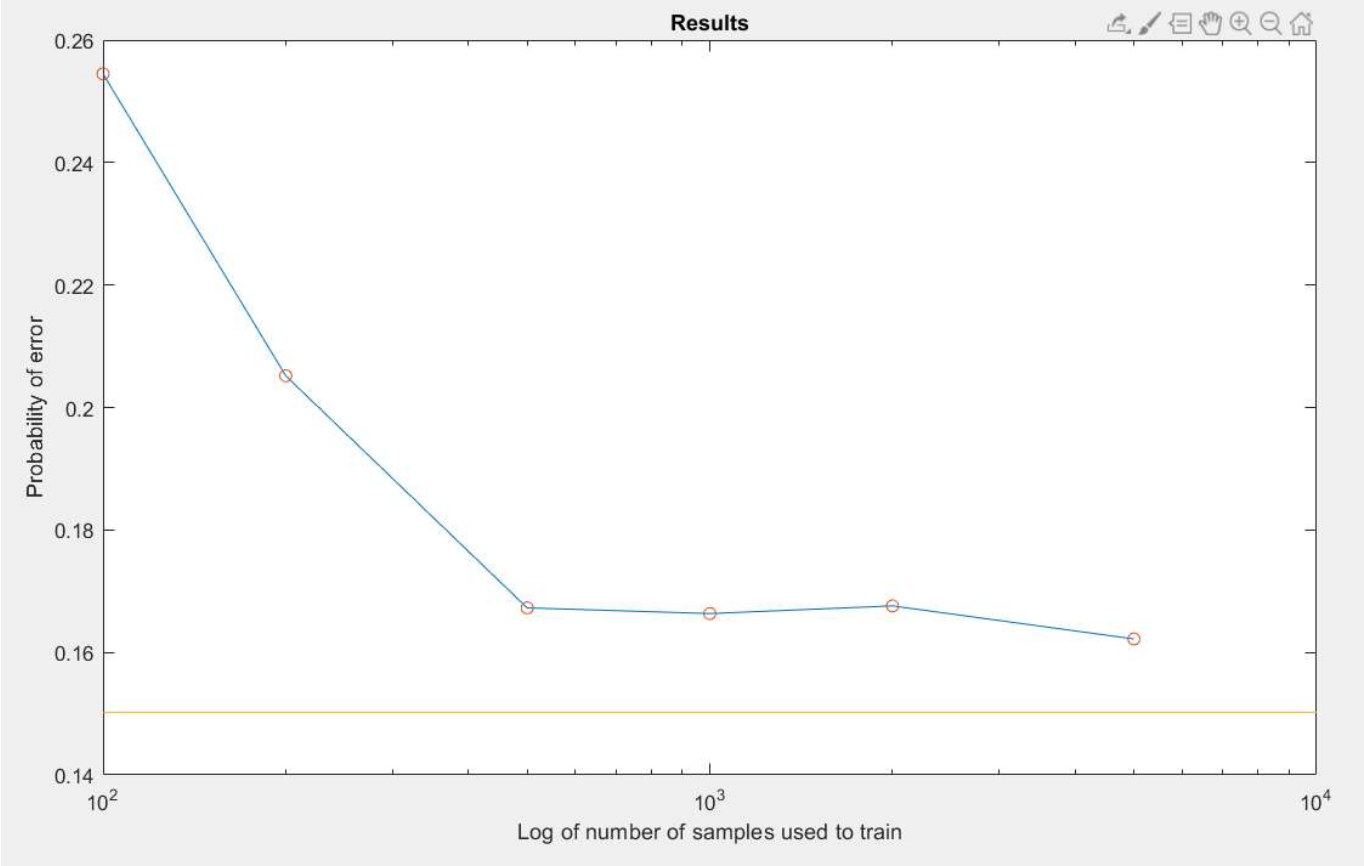

Figure. 2 Results plot of min P(error) over LOG Number of Training samples

2. Question #2

The data ((x,y) pairs) was generated using the equation $y = a^T(x + z) + v$ where a is randomly generated 10-dimensional vector, v is additive Gaussian noise with 0 mean and unit variance, z is 10-dimensional additive Gaussian noise with zero mean and αI covariance matrix (where α is the noise level). The 10-dimensional x came from a gaussian with a non-zero mean and non-diagonal covariance matrix. The mean and covariance for x, and vector a were randomly generated at the beginning. Two datasets were generated, one for training with 50 samples and one for testing with 1000 samples. The goal of having the noise level alpha value was to see the effect of additive noise in the performance of the model. As more "hidden" noise is added to the data the performance should decrease.

The model believed that generated y given x was $y = w^T x + w_0 + v$ where $w = [w_0\, w^T]$ are the parameters to be optimized and v is the additive Gaussian noise with zero mean and unit variance. Therefore, because of the noise v, y_i comes from a gaussian with mean $w^T x_i$ and unit variance ($P(y_i \mid x_i\, w) \sim \aleph(w^T x_i\,,\ 1)$). In this model, we are ignoring

the z noise that was used to generate the data but a biased term w_0 and weights w are introduced to try to account for this unknown noise. We believed that the parameter w is close to zero (to approximate the noise z) so we used a gaussian with zero mean and βI covariance matrix as the prior of the parameter w.

To estimate the optimal parameter w, MAP estimation was used since we have the parameter's prior. $w_{MAP} = argmax_w \, ln(P(w \mid D))$ where D is the training data. Using Bayes rule and removing constant additive terms that do not depend on w, it becomes

$$w_{MAP} = argmax_w \sum_{i=1}^{N} ln(P(y_i \mid x_i \, w_i)) + ln(P(w))$$

Both terms are a log of a gaussian, therefore, (removing terms that do not depend on w and multiplying by $-\frac{2}{N}$)

$$w_{MAP} = argmin_w \; \frac{1}{N} \sum_{i=1}^{N} (y_i - w^T u_i)^2 + \frac{1}{N\beta} w^T w \text{ (Where } u_i = [\, 1 \;\; x_i \,]^T \text{)}$$

Expanding the first term and organizing the equation,

$$w_{MAP} = argmin_w \; \frac{1}{N} \sum_{i=1}^{N} y_i - 2\frac{1}{N} \sum_{i=1}^{N} y_i \, u_i^T * w + w^T \left(\frac{1}{N} \sum_{i=1}^{N} u_i \, u_i^T\right) \; w + w^T \left(\frac{1}{N\beta} I\right) w$$

Simplify the equation by changing variables

$$C = \frac{1}{N} \sum_{i=1}^{N} y_i \quad B = \frac{1}{N} \sum_{i=1}^{N} y_i \, u_i \qquad A = \frac{1}{N} \sum_{i=1}^{N} u_i \, u_i^T + \frac{1}{N\beta} I$$

$$w_{MAP} = argmin_w \; w^T A \, w - 2B^T w + C$$

Taking the derivative with respect to w and making it zero

$$0 = \frac{\partial}{\partial w^T} \; w^T A \, w - 2B^T w + C \quad => 0 = 2Aw - 2b \quad => \quad 0 = Aw - b$$

Thus the optimal parameter w is

$$w_{MAP} = A^{-1} B$$ (MAP parameter estimation derived during office hours)

However, this solution depends on an unknown hyperparameter β. Its optimal value was found using Cross-validation with 5-fold; which was done in the same way as to question 1. The only differences were that the range of the candidates for an optimal β was $\left[\, 10^{-5} \frac{trace(\Sigma_{estimate})}{10}, 10^{5} \frac{trace(\Sigma_{estimate})}{10} \,\right]$ and it was gradually incremented by a delta such that there were 1001 values over that rage; $\Sigma_{estimate}$ was the sample covariance of the training dataset. Moreover, instead of minP error, the selected β was the one that produced the parameter w that resulted in the maximum log-likelihood of the validation subset averaged over the 5 partitions.

Once the optimal beta was selected, the parameter w was estimated using the entire training dataset. Next, the -2Log-Likelihood of the test dataset(1000 samples) was calculated by computing the -2-times-log value of the pdf of a gaussian with mean $w^T xtest_i$ and unit variance for each y_i sample. Once this was done, the noise level alpha was incremented by a small amount and the algorithm was run again from the data generation (always using the same vector a, and mu and sigma for x). The range for the alpha values to try was $[\,10^{-3.5}\frac{trace(\Sigma)}{10}, 10^{1.5}\frac{trace(\Sigma)}{10}\,]$ and it was gradually incremented by a delta such that there were 501 values over that rage. For each alpha, the -2Log-Likelihood was plotted (Figure.3). A Gaussian model normally performs better with a higher Log-Likelihood of the data since we are using the -2Log-Likelihood, a model with better performance means lower -2Log-Likelihood. As predicted, as we increase the noise level alfa, the performance of the model decreases exponentially; in figure.3 the -2Log-Likelihood increases, thus the performance decreases. With a very small alpha (close to 10^{-3}), the -2Log-Likelihood was around 2901.116 and with a large alpha (close to 10^{3}) it went to infinity; meaning that this model is not the relationship between the (x,y) pairs. More precisely, the last value of alpha that gave a finite -2Log-Likelihood was 34.34 giving a value of 86298.147.

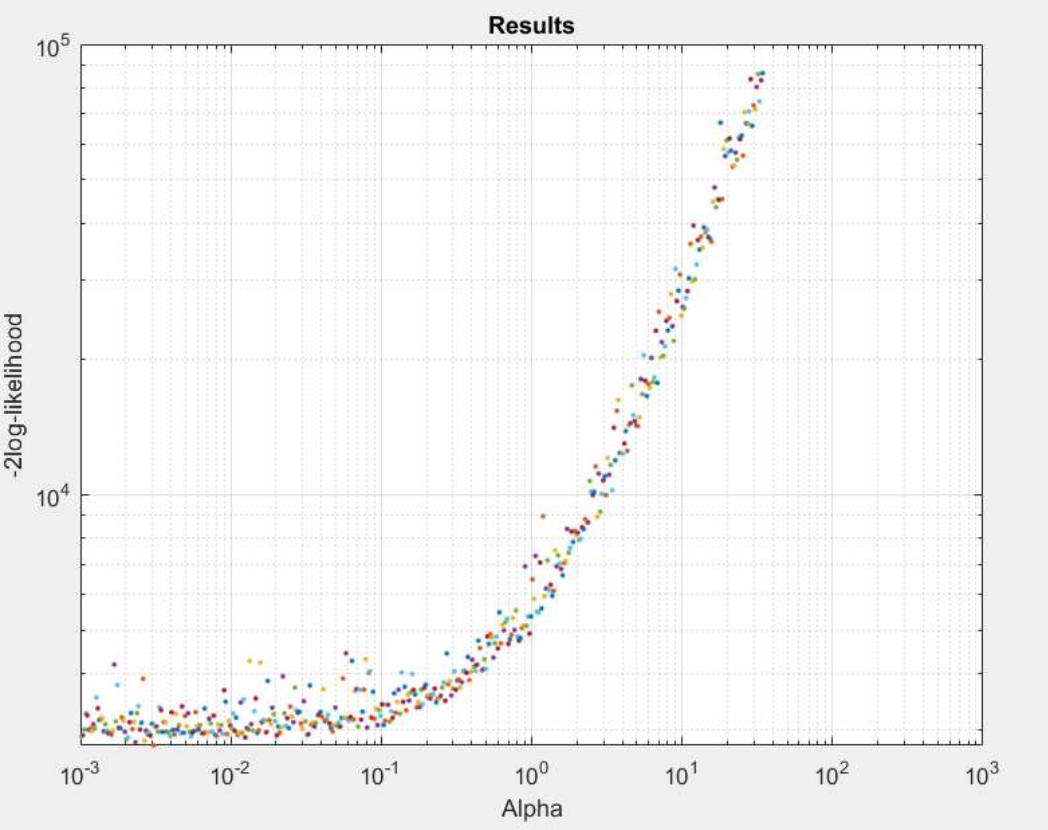

Figure.3 Alpha VS -2Log-Likelihood

Citations

For question 1 the format of the code and the overall solution were based on the given code mleMLPwAWGN.m but nothing was directly copied from it.

For question 2, the derivation of the MAP estimator was taken from the office hours from June 2nd at 5 pm. Its code and the function to generate the data were taken from the given code Exam3_Q2starter.m and modified to fit my code.

Appendix A

Code Question #1

```
close all, clear all

%Generate Datasets
[x100,label100,sigma,mu,N100] = genData(100);
[x200,label200,~,~,N200] = genData(200);
[x500,label500,~,~,N500] = genData(500);
[x1000,label1000,~,~,N1000] = genData(1000);
[x2000,label2000,~,~,N2000] = genData(2000);
[x5000,label5000,~,~,N5000] = genData(5000);
[x10k,label10k,~,~,N10k] = genData(10000);

%Theoretically Optimal Classifier
TError = MLval(x10k,label10k,mu,sigma,N10k);

%Model Order Selection
Np100 = C_V(x100,label100,N100);
Np200 = C_V(x200,label200,N200);
Np500 = C_V(x500,label500,N500);
Np1000 = C_V(x1000,label1000,N1000);
Np2000 = C_V(x2000,label2000,N2000);
Np5000 = C_V(x5000,label5000,N5000);

%NN Model Training
p100 = NNtrain(x100,label100,Np100,100);
p200 = NNtrain(x200,label200,Np200,200);
p500 = NNtrain(x500,label500,Np500,500);
p1000 = NNtrain(x1000,label1000,Np1000,1000);
p2000 = NNtrain(x2000,label2000,Np2000,2000);
p5000 = NNtrain(x5000,label5000,Np5000,5000);
```

```
%Testing Performance Assessment
PE100 = NNtest(p100,x10k,label10k,N10k);
PE200 = NNtest(p200,x10k,label10k,N10k);
PE500 = NNtest(p500,x10k,label10k,N10k);
PE1000 = NNtest(p1000,x10k,label10k,N10k);
PE2000 = NNtest(p2000,x10k,label10k,N10k);
PE5000 = NNtest(p5000,x10k,label10k,N10k);

%Results
TError
Modelorder = [Np100 Np200 Np500 Np1000 Np2000 Np5000]
PE = [PE100 PE200 PE500 PE1000 PE2000 PE5000]
xaxis = [100 200 500 1000 2000 5000];
%Plot
figure(2), semilogx(xaxis,PE),hold on, semilogx(xaxis,PE,'o'),hold on,
plot(xlim, TError*[1 1]),xlabel('Log of number of samples used to train'),
ylabel('Probability of error'), title('Results');

%Shuffle the Dataset to increase the efficiency of the Cross validation
function [newx,newlabel] = shuffle(x,label)
    for i = 1:length(x)
       ind(i) = i;
    end
    ind = randperm(length(ind));

    for i = 1:length(x)
       newx(i,:) = x(ind(i),:);
       newlabel(i,:) = label(ind(i),:);
    end
end
%Cross Validation function with 10-fold
function order = C_V(x,label,N)
    s = length(x)/10;
    [x,label] = shuffle(x,label);
    done = 0;
    Np = 0;
    currentScore = 1;

    while ~done
        Np = Np+1;
        orderPE = 0;
        %10-fold
```

```
        for k = 1:10
            x_validate = [];
            x_train = [];
            label_validate = [];
            label_train = [];
            if k == 1
               x_validate = x([1:(k*s)],:);
               x_train = x([(k*s+1):end],:);
               label_validate = label([1:(k*s)],:);
               label_train = label([(k*s+1):end],:);

            elseif k == 10
               x_validate = x([(length(x)-s+1):end],:);
               x_train = x([1:(length(x)-s)],:);
               label_validate = label([(length(x)-s+1):end],:);
               label_train = label([1:(length(x)-s)],:);

            else
                x_train = x([1:(((k-1)*s))],:);
                x_validate = x([(((k-1)*s)+1):(k*s)],:);
                x_train = [x_train ; x([(k*s+1):end],:)];
                label_train = label([1:(((k-1)*s))],:);
                label_validate = label([(((k-1)*s)+1):(k*s)],:);
                label_train = [label_train ; label([(k*s+1):end],:)];
            end

            N_validate = sum(label_validate,1);
            p = NNtrain(x_train,label_train,Np,length(x_train));
            orderPE = orderPE +
NNtest(p,x_validate,label_validate,N_validate);
        end
        NewScore  = (1/10)*orderPE;
        if NewScore > currentScore
            done = 1;
        else
            currentScore = NewScore;
        end
        Np
    end
    order = Np-1;
end
% test the NN estimated with a new 10k samples dataset
```

```
function TPE = NNtest(p,x,label,N)

    h = L2MLP(x,p);

    [~,decision] = max (h',[],2);
    [~,label] = max(label,[],2);
    pi1 = zeros(1,5);pi2 = zeros(1,5);pi3 = zeros(1,5);pi4 = zeros(1,5);
    pi5 = zeros(1,5);
     for i = 1:5
        indi1(i) = length(find(decision==i & label==1)); if N(1) ~= 0
pi1(i) = indi1(i)/N(1);end
        indi2(i) = length(find(decision==i & label==2)); if N(2) ~= 0
pi2(i) = indi2(i)/N(2);end
        indi3(i) = length(find(decision==i & label==3)); if N(3) ~= 0
pi3(i) = indi3(i)/N(3);end
        indi4(i) = length(find(decision==i & label==4)); if N(4) ~= 0
pi4(i) = indi4(i)/N(4);end
        indi5(i) = length(find(decision==i & label==5)); if N(5) ~= 0
pi5(i) = indi5(i)/N(5);end
    end
%Total Probability of error
    TPE = 1 - (pi1(1)*0.2 + pi2(2)*0.2 + pi3(3)*0.2 + pi4(4)*0.2 +
pi5(5)*0.2);

end
%Train the NN by estimating its parameters
function p = NNtrain(x,label,Np,N)

    for t = 1:3
        %initialize the parameters randomly
        p.w1 = rand(Np,3);
        p.b1 = rand(Np,1);
        p.w2 = rand(5,Np);
        p.b2 = rand(5,1);
        vecPinit = [p.w1(:);p.b1(:);p.w2(:);p.b2(:)];
        options = optimset('MaxFunEvals',1e6*length(vecPinit));
        [Ep, val] = fminsearch
(@(Ep)(CEL(x,Ep,label,N,Np)),vecPinit,options);

        p.w1 = reshape(Ep(1:3*Np),Np,3);
        p.b1 = Ep((3*Np+1):(3*Np+Np));
        p.w2 = reshape(Ep((4*Np+1):(9*Np)),5,Np);
```

```
        p.b2 = Ep((9*Np+1):end);
        scores(t) = val;
        allp(t) = p;
    end
    [~,Min] = min(scores);

    p = allp(Min);
end
%2 layer MLP
function h = L2MLP(D, p)
    %First Layer ISRU activation function
    ISRU = @(z) z./(sqrt(1+z.^2));

    x = p.w1*D' + p.b1;
    z = ISRU(x);

    v = p.w2*z + p.b2;
    %Soft Max layer
    for i = 1:size(v,1)
        h(i,:) = (exp(v(i,:)))./(sum(exp(v),1));
    end
end
%Cross Entropy Calculator
function entropy = CEL(D, vecp, label,N,Np)
    p.w1 = reshape(vecp(1:3*Np),Np,3);
    p.b1 = repmat(vecp((3*Np+1):(3*Np+Np)),1,N);
    p.w2 = reshape(vecp((4*Np+1):(9*Np)),5,Np);
    p.b2 = repmat(vecp((9*Np+1):end),1,N);
    h = L2MLP(D,p);
    %Cross Entropy Loss Function
    entropy = (1/N)*sum(sum(-(label.*log(h'))));
end
%Theoretical min probability of error using ML (MAP with equal priors)
function  TPE = MLval(x,label,mu,sigma,N)
    score = [];
    for i = 1:5
        xi = mvnpdf(x, mu(:,:,i), sigma(:,:,i));
        score = [score xi];
    end
    [~,decision] = max(score,[],2);
    [~,label] = max(label,[],2);
    for i = 1:5
```

```
        indi1(i) = length(find(decision==i & label==1)); pi1(i) =
indi1(i)/N(1);
        indi2(i) = length(find(decision==i & label==2)); pi2(i) =
indi2(i)/N(2);
        indi3(i) = length(find(decision==i & label==3)); pi3(i) =
indi3(i)/N(3);
        indi4(i) = length(find(decision==i & label==4)); pi4(i) =
indi4(i)/N(4);
        indi5(i) = length(find(decision==i & label==5)); pi5(i) =
indi5(i)/N(5);
    end
    TPE = 1 - (pi1(1)*0.2 + pi2(2)*0.2 + pi3(3)*0.2 + pi4(4)*0.2 +
pi5(5)*0.2);
end
%Generate Datasets
function [x,label,sigma,mu, Nc] = genData (N)

    mu (:,:,1) = [0 0 0];
    mu (:,:,2) = [3 0 3];
    mu (:,:,3) = [0 2.8 0];
    mu (:,:,4) = [-3 -0 -3];
    mu (:,:,5) = [-0 -2.8 -0];

    sigma(:,:,1) = [2 -.2 .2;-.2 2 -.2;.2 -.2 2];
    sigma(:,:,2) = [2 -0.4 0.3;-0.4 2 -0.4;0.3 -0.4 2];
    sigma(:,:,3) = [2 0.4 0.3;0.4 2 0.4;0.3 0.4 2];
    sigma(:,:,4) = [1 -.5 .5;-.5 1 -.5;.5 -.5 1];
    sigma(:,:,5) = [1 .5 .5;.5 1 .5;.5 .5 1];

    u = rand(1,N);
    Nc(1) = length(find(u <=0.2));
    Nc(2) = length(find((u > 0.2) & (u <=0.4)));
    Nc(3) = length(find((u > 0.4) & (u <=0.6)));
    Nc(4) = length(find((u > 0.6) & (u <= 0.8)));
    Nc(5) = length(find((u > 0.8)));
    x = [];
    label = [];
    clf;
    for i = 1:5

        xi = mvnrnd(mu(:,:,i), sigma(:,:,i), Nc(i));
        labeli = zeros(5,Nc(i));
```

```
        labeli(i,:) = 1;
        x = [x; xi];
        label = [label ; labeli'];
        figure(1), plot3(xi(:,1),xi(:,2),xi(:,3),'.'); axis equal,hold on;
        title ('Data'),xlabel('X1'),ylabel('X2'),zlabel('X3');
    end
end
```

Appendix B

Code Question #2

```
clear all, close all

%calculated the values that are random
n = 10; Ntrain = 50; Ntest = 1000;
a = rand(1,n);
mu = round(10*rand(1,n));
A = rand(n); sigma = A*A'+1e-3*eye(n,n);
alphaSet= 10.^linspace(-3.5,1.5,501)*trace(sigma)/n;

%vary the noise level alpha
for EX = 1:length(alphaSet)

    alpha = alphaSet(EX);
    %generate data
    [xtrain,ytrain] = genData(a,mu,sigma,n,Ntrain,alpha);
    [xtest,ytest] = genData(a,mu,sigma,n,Ntest,alpha);
    %estimate the optimal Beta with Cross-Validation
    beta = C_V(xtrain,ytrain',n);
    %estimate the parameters w with the optimal beta
    p = mapPE(xtrain,ytrain',beta);
    %calculate the -2log-likelihood of the data with current alpha
    utest = [ones(1,length(xtest));xtest];
    mutest = p'*utest;
    LL = 0;

    for i = 1:length(ytest)
        LL = LL - 2*log(mvnpdf(ytest(i),mutest(i),1));
    end
```

```
    %plot the results
    figure(1), loglog(alpha,LL,'.'), grid on, title('Results'),
    xlabel('Alpha'),ylabel('-2log-likelihood'),xlim([10^-3,10^3]), hold on;

    %stop once the -2log-likelihood goes to infinity
    if LL == inf
        disp ('-2log-likelihood has gone to inf');
        alpha
        LL
        break;
    end

    results(:,EX) = [alpha;LL];
end

%Cross-validation with 5-fold to find optimal beta
function Ebeta = C_V(x,y,n)

    s = length(x)/5;
    done = 0;
    b = 0;
    currentScore = -inf;

    SigmaHat = cov(x');
    betaCandidates = 10.^linspace(-5,5,1001)*trace(SigmaHat)/n;

    while ~done & b < 1001
        b = b+1;
        LL= 0;
        beta = betaCandidates(b);

        %5-fold
        for k = 1:5
            x_validate = [];
            x_train = [];
            y_validate = [];
            y_train = [];
            if k == 1
               x_validate = x(:,[1:(k*s)]);
               x_train = x(:,[(k*s+1):end]);
               y_validate = y([1:(k*s)]);
               y_train = y([(k*s+1):end]);
```

```
            elseif k == 10
               x_validate = x(:,[(length(x)-s+1):end]);
               x_train = x(:,[1:(length(x)-s)]);
               y_validate = y([(length(x)-s+1):end]);
               y_train = y([1:(length(x)-s)]);

            else
                x_train = x(:,[1:(((k-1)*s))]);
                x_validate = x(:,[(((k-1)*s)+1):(k*s)]);
                x_train = [x_train  x(:,[(k*s+1):end])];
                y_train = y([1:(((k-1)*s))]);
                y_validate = y([(((k-1)*s)+1):(k*s)]);
                y_train = [y_train ; y([(k*s+1):end])];
            end
            u_validate = [ones(1,length(x_validate));x_validate];
            p = mapPE(x_train,y_train,beta);
            LL = 0;
            for i = 1:length(y_validate)
                mu = p'*u_validate;
                LL = LL + log(mvnpdf(y_validate(i),mu(i),1));
            end
        end

        MLL  = (1/5)*LL;
        if MLL < currentScore
            done = 1;
        else
            currentScore = MLL;
        end
    end
    Ebeta = betaCandidates(b-1);
End

%MAP estimator for the parameters w
function wMAP = mapPE(xTrain,yTrain,beta)
   [n,N] = size(xTrain);
   u = [ones(1,N);xTrain];
   b = u*yTrain/N;
   A = u*u'/N + eye(n+1,n+1)/(N*beta);
   wMAP = inv(A)*b;
end
```

```
%Generate Data
function [x,y] = genData(a,mu,sigma,n,N,alpha)
   x = mvnrnd(mu,sigma,N);
   z = mvnrnd(zeros(1,n),alpha*eye(n),N);
   v = mvnrnd(0,1,N);
   y = a*(x' + z')+ v';
   x = x';
end
```