Appendix A:
Problem 1

```
clear; close all;
%Got insprisation from 2020SUMMER2 code and
generateDataFromGMM.m
%evalGaussian.m

%% Generate
% 0~0.65 | 1~0.35
% 01~0.5 | 02~0.5

D.size = [20,200,2000,10000];
D.fPrior = [0.65*0.5,0.65*0.5,0.35];
D.tPrior = [0.65, 0.35];
D.mu = [3,0,2;0,3,2];
D.sigma(:,:,1) = [2,0;0,1];D.sigma(:,:,2) = [1,0;0,2];
D.sigma(:,:,3) = [1,0;0,1];

for i = 1: length(D.size)
    [D.labels{i},D.r{i}] =
myGaussian(D.size(i),D.fPrior,D.mu,D.sigma);
end

figure(1);
% Ture label & plot
for i = 1: length(D.size)
    D.labels{i}(:,D.labels{i} ==1) = 0;
    D.labels{i}(:,D.labels{i} ==2) = 0;
    D.labels{i}(:,D.labels{i} ==3) = 1;
    subplot(2,2,i);
    plot(D.r{1,i}(1,D.labels{i}
==0),D.r{1,i}(2,D.labels{i} ==0),'.b'); axis equal,hold
on;
    plot(D.r{1,i}(1,D.labels{i}
==1),D.r{1,i}(2,D.labels{i} ==1),'.r'); axis equal,hold
on;
    title(D.size(i)+" Samples");
end


%% Save Data
% save("x"+...
```

```
%     clock(4)+clock(5)+clock(6)+...
%     ".mat",'att','fx1_1','fy1_1','fz1_1');


%% Part1 D10000

% Data Import
x = D.r{4};
label = D.labels{4};
Nc = [length(find(label==0)),length(find(label==1))];
N = D.size(4);

lambda = [0 1;1 0]; % loss values
gamma = (lambda(2,1)-lambda(1,1))/(lambda(1,2)-
lambda(2,2)) * D.tPrior(1)/D.tPrior(2); %threshold
discriminantScore = -
(log(0.5*evalGaussian(x,D.mu(:,1),D.sigma(:,:,1))...
    + 0.5*evalGaussian(x,D.mu(:,2),D.sigma(:,:,2)))...
    - log(evalGaussian(x,D.mu(:,3),D.sigma(:,:,3))));%
log(gamma);
decision_ideal = (discriminantScore >= log(gamma));

ind00 = find(decision_ideal==0 & label==0); p00 =
length(ind00)/Nc(1); % probability of true negative
ind10 = find(decision_ideal==1 & label==0); p10 =
length(ind10)/Nc(1); % probability of false positive
ind01 = find(decision_ideal==0 & label==1); p01 =
length(ind01)/Nc(2); % probability of false negative
ind11 = find(decision_ideal==1 & label==1); p11 =
length(ind11)/Nc(2); % probability of true positive

% if norm(lambda-[0,1;1,0])<eps % Using 0-1 loss
indicates intent to minimize P(error)
%     Perror_MAP = [p10,p01]*Nc'/N, % probability of
error, empirically estimated
% end

figure(2); % class 0 circle, class 1 +, correct green,
incorrect red
plot(x(1,ind00),x(2,ind00),'ob'); hold on,
plot(x(1,ind10),x(2,ind10),'or'); hold on,
plot(x(1,ind01),x(2,ind01),'+r'); hold on,
plot(x(1,ind11),x(2,ind11),'+b'); hold on,
axis equal,
```

```
legend('Correct decisions for data from Class 0','Wrong
decisions for data from Class 0',...
    'Wrong decisions for data from Class 1','Correct
decisions for data from Class 1'),
title('Data and their classifier decisions versus true
labels'),
xlabel('x_1'), ylabel('x_2'),

%% Part1 Optional: Draw the decision boundary

horizontalGrid =
linspace(floor(min(x(1,:))),ceil(max(x(1,:))),101);
verticalGrid =
linspace(floor(min(x(2,:))),ceil(max(x(2,:))),91);
[h,v] = meshgrid(horizontalGrid,verticalGrid);
discriminantScoreGridValues = -
(log(0.5*evalGaussian([h(:)';v(:)'],D.mu(:,1),D.sigma(:
,:,1))...

+0.5*evalGaussian([h(:)';v(:)'],D.mu(:,2),D.sigma(:,:,2
)))...
    -
log(evalGaussian([h(:)';v(:)'],D.mu(:,3),D.sigma(:,:,3)
)) - log(gamma));
minDSGV = min(discriminantScoreGridValues);
maxDSGV = max(discriminantScoreGridValues);
discriminantScoreGrid =
reshape(discriminantScoreGridValues,91,101);
figure(2),
contour(horizontalGrid,verticalGrid,discriminantScoreGr
id,[minDSGV*[0.9,0.6,0.3],0,[0.3,0.6,0.9]*maxDSGV]); %
plot equilevel contours of the discriminant function
% including the contour at level 0 which is the
decision boundary
legend('Correct decisions for data from Class 0','Wrong
decisions for data from Class 0','Wrong decisions for
data from Class 1','Correct decisions for data from
Class 1','Equilevel contours of the discriminant
function' ),
title('Data and their classifier decisions versus true
labels'),
xlabel('x_1'), ylabel('x_2'),
```

```
%% Part1 ROC Curve
%Estimate Minimum Error
sortDS=sort(discriminantScore);
%Generate vector of gammas for parametric sweep
logGamma=[min(discriminantScore)-eps
sort(discriminantScore)+eps];
for ind=1:length(logGamma)
decision=discriminantScore> logGamma(ind);
Num_pos(ind)=sum(decision);
pFP(ind)=sum(decision==1 & label==0)/Nc(1);
pTP(ind)=sum(decision==1 & label==1)/Nc(2);
pFN(ind)=sum(decision==0 & label==1)/Nc(2);
pTN(ind)=sum(decision==0 & label==0)/Nc(1);
pFE(ind)=(sum(decision==0 & label==1) + sum(decision==1
& label==0))/N;
end

%If multiple minimums are found choose the one closest
to the theoretical
%minimum
[min_pFE, min_pFE_ind]=min(pFE);
if length(min_pFE_ind)>1

[~,minDistTheory_ind]=min(abs(logGamma(min_pFE_ind)-
logGamma_ideal));
    min_pFE_ind=min_pFE_ind(minDistTheory_ind);
end
%Find minimum gamma and corresponding false and true
positive rates
minGAMMA=exp(logGamma(min_pFE_ind));
min_FP=pFP(min_pFE_ind);
min_TP=pTP(min_pFE_ind);

figure(3);
plot(pFP,pTP,'DisplayName','ROC Curve','LineWidth',2);
hold all;
plot(min_FP,min_TP,'o','DisplayName','Estimated Min.
Error','LineWidth',2);
plot(p10,p11,'+','DisplayName',...
'Theoretical Min. Error','LineWidth',2);
xlabel('Prob. False Positive');
ylabel('Prob. True Positive');
title('Mininimum Expected Risk ROC Curve');
legend 'show';
```

```
grid on; box on;
fprintf('Theoretical for ERM: Gamma=%1.2f,
Error=%1.2f%%\n',...
gamma,(length(ind10)+length(ind01))/100);
fprintf('Estimated for ERM: Gamma=%1.2f,
Error=%1.2f%%\n',...
    minGAMMA,100*min_pFE);

%% Part1 Probability of Error vs. Gamma
figure(4);
plot(logGamma,pFE,'DisplayName','Errors','LineWidth',2)
;
hold on;
plot(logGamma(min_pFE_ind),pFE(min_pFE_ind),...
    'o','DisplayName','Estimated Minimum
Error','LineWidth',2);
plot(log(gamma),(length(ind10)+length(ind01))/N,...
    '+','DisplayName','Theoretical Minimum
Error','LineWidth',2);
xlabel('Gamma');
ylabel('Proportion of Errors');
title('Probability of Error vs. Gamma')
grid on;
legend 'show';


%% Part 2a : Linear
for i = 1:length(D.size)-1


tx = D.r{i}';
tlabel = D.labels{i}';
tNc =
[length(find(tlabel==0)),length(find(tlabel==1))];
tN = D.size(i);
dimension = length(D.mu(:,i));

zX = [ones(tN,1),tx];%Training Data
zVX = [ones(N,1),x'];%Validate Data
iTheta = zeros(dimension+1,1);

%cost min
[theta,cost] =
fminsearch(@(t)(costFunction(t,zX,tlabel,tN)),iTheta);
```

```
%plot
plotX(1,:) = [min(zX(:,2))-2 , max(zX(:,2))+2];
plotX(2,:) = (-
1./theta(3)).*(theta(2).*plotX(1,:)+theta(1));
figure(5); plotTrainData(tlabel,i,zX,plotX,1);

%valid
decision = zVX*theta>=0;
figure(6);plotValidationData(x,tN,N,Nc,label,decision',
plotX,i,1);

end
%% Part 2b : quadratic
for i = 1:length(D.size)-1


tx = D.r{i}';
tlabel = D.labels{i}';
tNc =
[length(find(tlabel==0)),length(find(tlabel==1))];
tN = D.size(i);
dimension = length(D.mu(:,i));

tx= tx';
zX =
[ones(tN,1),tx(1,:)',tx(2,:)',(tx(1,:).^2)',(tx(1,:).*t
x(2,:))',(tx(2,:).^2)'];
tx = tx';
zVX =
[ones(N,1),x(1,:)',x(2,:)',(x(1,:).^2)',(x(1,:).*x(2,:)
)',(x(2,:).^2)'];%Validate Data

iTheta = zeros(size(zX,2),1);

%cost min
[theta,cost] =
fminsearch(@(t)(costFunction(t,zX,tlabel,tN)),iTheta);

%plot
figure(7);subplot(2,3,i);
hGrid = linspace(min(zX(:,2))-6,max(zX(:,2))+6);
vGrid = linspace(min(zX(:,3))-6,max(zX(:,3))+6);
```

```
score = boundaryScore(hGrid,vGrid,theta); % score = 0
is decision boundary level
plotTrainData(tlabel,i,zX,[hGrid;vGrid;score],0);

%valid
decision = zVX*theta>=0;
figure(8);plotValidationData(x,tN,N,Nc,label,decision',
[hGrid;vGrid;score],i,0);
end
%% Functions
%Cost Function
function cost = costFunction(theta,zx,label,N)
h = 1./(1+exp(-zx*theta)); %Sigmoid Function
cost = (-1/N)*((sum(label'*log(h)))+(sum((1-
label)'*log(1-h))));
end

function plotTrainData(label,fig,x,bound,type)
% Plot original class labels and boundary
% type 1~linear 0~quadtic
subplot(1,3,fig);
%axis([min(x(2))-2,max(x(2))+2,min(x(3))-
2,max(x(3))+2]);
axis equal,
axis square,hold on;
plot(x(label == 0,2),x(label==0,3),'o',x(label ==
1,2),x(label==1,3),'x');

if type
    plot(bound(1,:),bound(2,:));
else
    contour(bound(1,:),bound(2,:),bound(3:end,:),[0
0]);
end

legend('Class 0','Class 1','Classifier');
xlabel('x1');ylabel('x2');hold on;
end

function score = boundaryScore(hGrid, vGrid, theta)
% score = 0 is decision boundary level
z = zeros(length(hGrid), length(vGrid));
for i = 1:length(hGrid)
    for j =1:length(vGrid)
```

```
        xBound =
[1,hGrid(i),vGrid(j),hGrid(i)^2,hGrid(i)*vGrid(j),vGrid
(j)^2];
        z(i,j)=xBound*theta;
    end
end
score = z';
end

function
plotValidationData(x,tN,N,Nc,label,decision,bound,fig,t
ype)
subplot(1,3,fig);
%axis([min(x(2))-2,max(x(2))+2,min(x(3))-
2,max(x(3))+2]);

%keyboard,
ind00 = find(decision==0 & label==0); p00 =
length(ind00)/Nc(1); % probability of true negative
ind10 = find(decision==1 & label==0); p10 =
length(ind10)/Nc(1); % probability of false positive
ind01 = find(decision==0 & label==1); p01 =
length(ind01)/Nc(2); % probability of false negative
ind11 = find(decision==1 & label==1); p11 =
length(ind11)/Nc(2); % probability of true positive
plot(x(1,ind00),x(2,ind00),'ob'); hold on,
plot(x(1,ind10),x(2,ind10),'or'); hold on,
plot(x(1,ind01),x(2,ind01),'+r'); hold on,
plot(x(1,ind11),x(2,ind11),'+b'); hold on,
if type
    plot(bound(1,:),bound(2,:));axis equal,hold on,
else
    contour(bound(1,:),bound(2,:),bound(3:end,:),[0
0]);
end
fprintf('Theoretical for ERM: Train Dataset Size=%1.0f,
Error=%1.2f%%\n',...
    tN,(length(ind10)+length(ind01))/N*100);

axis([min(x(1,:))-2, max(x(1,:))+2, min(x(2,:))-2,
max(x(2,:))+2]);
end
```

Appendix B:
Problem 2

```
clear;close all;
%Got insprisation from office hour video, 2020spring
code and hw2q2.m

%%
Ntrain = 100;
Nvalidate = 1000;
[xTrain,yTrain,xValidate,yValidate] =
hw2q2(Ntrain,Nvalidate);%

cXTrain = cubicX(Ntrain,xTrain);% cubic poly
cXValidate = cubicX(Nvalidate,xValidate);
%% ML

%ML train w
wML =
((cXTrain*cXTrain')/Ntrain)\((cXTrain*yTrain')/Ntrain);

%ML average-squared error
errorML = mean((yValidate-wML'*cXValidate).^2);

%% MAP
gammaArray = 10.^[-7:0.1:4];
sigma=.1;

%MAP w & average-squared error
for i = 1:length(gammaArray)
    gamma = gammaArray(i);
    lambdaSize = size(cXTrain,1);
    lambda = (sigma^2/gamma)*eye(lambdaSize);
    wMAP(:,i) = ((cXTrain*cXTrain')/Ntrain +
lambda)\((cXTrain*yTrain')/Ntrain);
    errorMAP(:,i) = mean((yValidate-
wMAP(:,i)'*cXValidate).^2);
end

%figure
figure;
plot(log10(gammaArray),errorMAP,'b.');hold on,
plot(log10(gammaArray),errorMAP,'b-');hold on,
```

```
plot(log10(gammaArray),repmat(errorML,length(gammaArray
)),'r-');
xlabel('log(gamma)'),ylabel('Error'), title('MAP error
vs. ML error'),

legend('MAP Error Dot','MAP Error Line','ML Error')
%% functions
% Most From hw2q2.m
function r = cubicX(N,xTrain)
r = [ones(1,N);...
    xTrain(1,:);xTrain(2,:);...
    xTrain(1,:).^2; xTrain(1,:).*xTrain(2,:);
xTrain(2,:).^2;...
    xTrain(1,:).^3; (xTrain(1,:).^2).*xTrain(2,:);...
    xTrain(1,:).*(xTrain(2,:).^2); xTrain(2,:).^3];
end
function [xTrain,yTrain,xValidate,yValidate] =
hw2q2(Ntrain,Nvalidate)

%Ntrain = 100;
data = generateData(Ntrain);
figure(1), plot3(data(1,:),data(2,:),data(3,:),'.'),
axis equal,
xlabel('x1'),ylabel('x2'), zlabel('y'), title('Training
Dataset'),
xTrain = data(1:2,:); yTrain = data(3,:);

%Nvalidate = 1000;
data = generateData(Nvalidate);
figure(2), plot3(data(1,:),data(2,:),data(3,:),'.'),
axis equal,
xlabel('x1'),ylabel('x2'), zlabel('y'),
title('Validation Dataset'),
xValidate = data(1:2,:); yValidate = data(3,:);
end

%%
function x = generateData(N)
gmmParameters.priors = [.3,.4,.3]; % priors should be a
row vector
gmmParameters.meanVectors = [-10 0 10;0 0 0;10 0 -10];
gmmParameters.covMatrices(:,:,1) = [1 0 -3;0 1 0;-3 0
15];
```

```
gmmParameters.covMatrices(:,:,2) = [8 0 0;0 .5 0;0
0 .5];
gmmParameters.covMatrices(:,:,3) = [1 0 -3;0 1 0;-3 0
15];
[x,labels] = generateDataFromGMM(N,gmmParameters);
end
%%
function [x,labels] =
generateDataFromGMM(N,gmmParameters)
% Generates N vector samples from the specified mixture
of Gaussians
% Returns samples and their component labels
% Data dimensionality is determined by the size of
mu/Sigma parameters
priors = gmmParameters.priors; % priors should be a row
vector
meanVectors = gmmParameters.meanVectors;
covMatrices = gmmParameters.covMatrices;
n = size(gmmParameters.meanVectors,1); % Data
dimensionality
C = length(priors); % Number of components
x = zeros(n,N); labels = zeros(1,N);
% Decide randomly which samples will come from each
component
u = rand(1,N); thresholds = [cumsum(priors),1];
for l = 1:C
    indl = find(u <= thresholds(l)); Nl = length(indl);
    labels(1,indl) = l*ones(1,Nl);
    u(1,indl) = 1.1*ones(1,Nl); % these samples should
not be used again
    x(:,indl) =
mvnrnd(meanVectors(:,l),covMatrices(:,:,l),Nl)';
end
end
```