EEECE5644 HW2

Jing Wang

June 13 2021

Got inspiration from office hour video, 2020summer2 code, 2020spring code

1 Problem 1

1.1 Data Distribution

For numerical results, 7 iid dataset was generated according to the data distribution, shown in Figure 1 below. The first 6 datasets is for training. The last one is for testing.

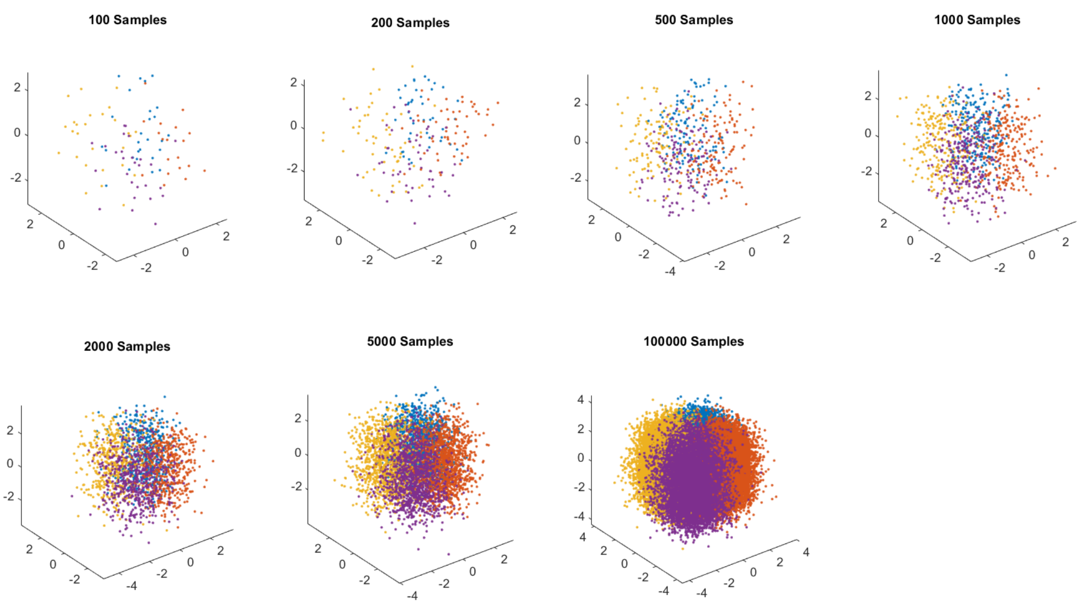


Figure 1: Data Distribution

1.2 MLP Structure

In this homework, MATLAB Neural Network Training(nntraintool) is used to create network. Pattennet is used for the network structure, which has one hidden layer.

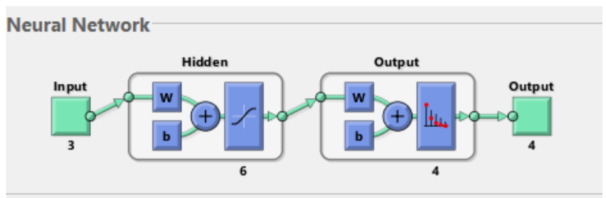


Figure 2: nntraintool-pattennet NN

For MATLAB Deeplearning Toolbox, an useful network architecture function is 'trainNetwork', which provide the feature classification or regression tasks after MATLAB 2020b. Because mine MATLAB version is 2020a, I use the old function 'train' and 'pattennet'.

1.3 Theoretically Optimal Classifier

Theoretically optimal classifier results are shown below.

Figure 3: Plots of test dataset

(a) N=100

(b) N=200

(c) N=500

(d) N=1000

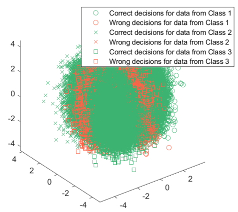
(e) N=2000

(f) N=5000

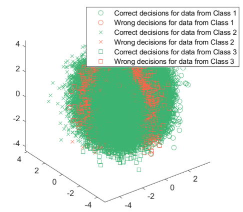
Figure 4: Plots of train datasets

Table 1: Theoretically optimal classifier result

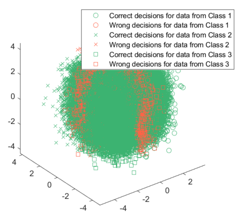
N	Theoretical pFE
100	26.00%
200	18.00%
500	18.60%
1000	19.40%
2000	20.90%
5000	19.48%
100000	20.00%



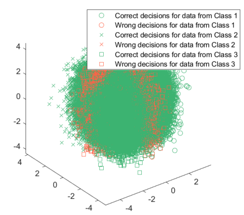
(a) N=100



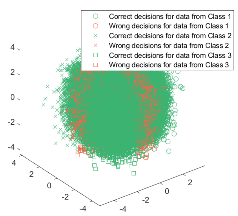
(b) N=200



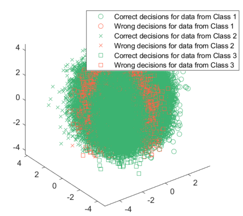
(c) N=500



(d) N=1000



(e) N=2000



(f) N=5000

Figure 5: Plots of MLP classifier results

1.4 MLP classifier

MLP optimal classifier results are shown. The max perceptrons is 15.

Table 2: MLP optimal classifier result

N	NN pFE
100	24.61%
200	21.79%
500	21.50%
1000	20.50%
2000	20.65%
5000	20.26%

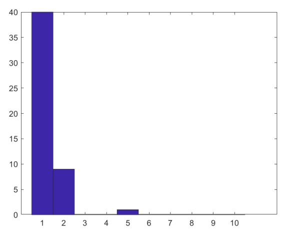
Table 3: Best Perceptrons in each datasets

N	P
100	12
200	7
500	9
1000	6
2000	13
5000	12

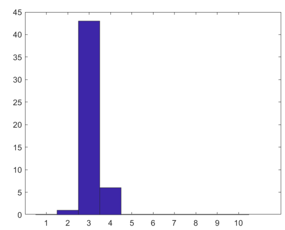
2 Problem 2

Due to the iterations taking too much time, the Repeating times E and data size may be decreased.

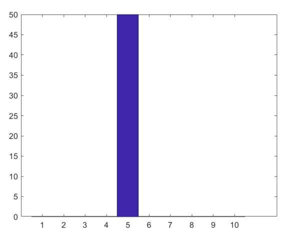
2.1 BIC



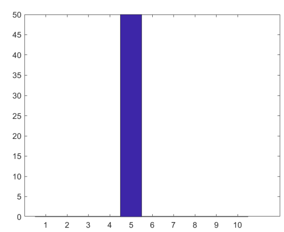
(a) N=100



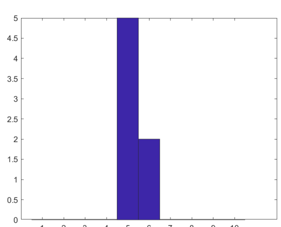
(b) N=200



(c) N=500



(d) N=1000



(e) N=2000

Figure 6: BIC histogram

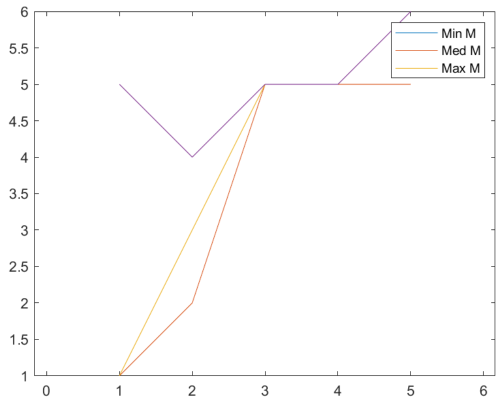
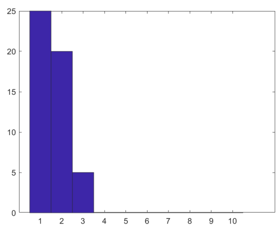
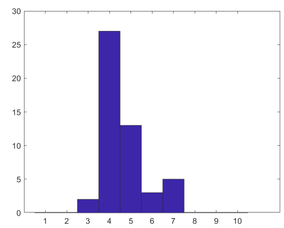


Figure 7: BIC statistic result

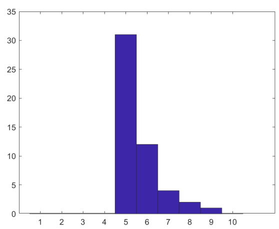
2.2 kFold



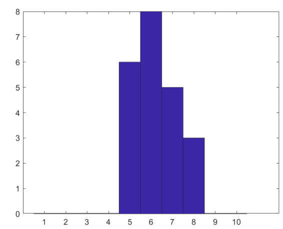
(a) N=100



(b) N=1000



(c) N=10000



(d) N=100000

Figure 8: kFold histogram

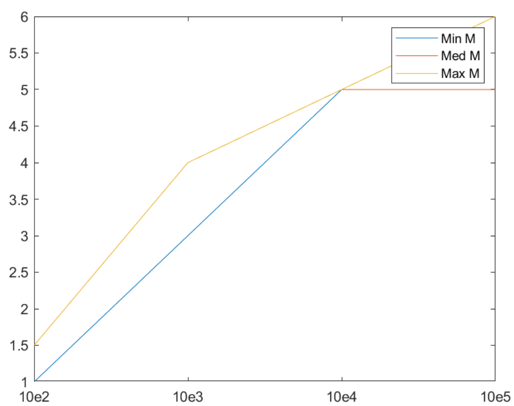


Figure 9: kfold statistic result

2.3 Results

When N increase, the result are more close to the true labels number. When N is small, both bestM are small, which is 1 or 2.

Appendix A:
Problem 1

```
run('hw3_1_1_generate.m');
clear;
close all;

%Got significant information and code example from
2020SUMMER2 code
load('hw3data.mat','D2');


dimensions=3; %Dimension of data
numLabels=4;
Lx={'L0','L1','L2','L3'};
muScale=2.5;
SigmaScale=0.2;
%Define data
D.d100.N=100;
D.d200.N=200;
D.d500.N=500;
D.d1k.N=1e3;
D.d2k.N=2e3;
D.d5k.N=5e3;
D.d100k.N=100e3;
dTypes=fieldnames(D);
%Define Statistics
p=ones(1,numLabels)/numLabels; %Prior
%Label data stats

%% Generate Data

for ind=1:length(dTypes)
    %D.(dTypes{ind}).x=zeros(dimensions,D.(dTypes{ind})
.N); %Initialize Data
    ddd=1;
    if ddd == 1
    D.(dTypes{ind}).x=D2.r{ind};
    D.(dTypes{ind}).labels=D2.labels{ind}-1;
    else
        [D.(dTypes{ind}).x,D.(dTypes{ind}).labels,...

D.(dTypes{ind}).N_l,D.(dTypes{ind}).p_hat]=...
```

```
genData(D.(dTypes{ind}).N,p,mu,Sigma,Lx,dimensions);
    end
end

%% Determine Theoretically Optimal Classifier
lossMatrix = ones(D2.classN,D2.classN)-
eye(D2.classN,D2.classN);
fprintf('Theoretically optimal classifier:\n');
for i = 1:length(D2.size)
    fig = figure;
    pFEIndex(i)=
optClass(D2.size(i),D2.r{i},D2.priors,D2.labels{i},loss
Matrix,D2.mu,D2.Sigma);
    fprintf('Theoretical pFE, N=%1.0f:
Error=%1.2f%%\n',...
        D2.size(i),100*pFEIndex(i));
    saveas(fig,append('Barchart',string(i),'.png'));
end
%% kFold & MLP
fprintf('MLP classifier:\n');
numPerc=15; %Max number of perceptrons to attempt to
train
k=5; %number of folds for kfold validation
for ind=1:length(dTypes)-1
    %kfold validation is in this function
    [D.(dTypes{ind}).net,D.(dTypes{ind}).minPFE,...

D.(dTypes{ind}).optM,valData.(dTypes{ind}).stats]=...
        kfoldMLP_NN(numPerc,k,D.(dTypes{ind}).x,...
        D.(dTypes{ind}).labels,numLabels);
    %Produce validation data from test dataset

valData.(dTypes{ind}).yVal=D.(dTypes{ind}).net(D.d100k.
x);


[~,valData.(dTypes{ind}).decisions]=max(valData.(dTypes
{ind}).yVal);

valData.(dTypes{ind}).decisions=valData.(dTypes{ind}).d
ecisions-1;
    %Probability of Error is wrong decisions/num data
points
```

```
    valData.(dTypes{ind}).pFE=...

sum(valData.(dTypes{ind}).decisions~=D.d100k.labels)/D.
d100k.N;
    outpFE(ind,1)=D.(dTypes{ind}).N;
    outpFE(ind,2)=valData.(dTypes{ind}).pFE;
    outpFE(ind,3)=D.(dTypes{ind}).optM;
    fprintf('NN pFE, N=%1.0f: Error=%1.2f%%\n',...

D.(dTypes{ind}).N,100*valData.(dTypes{ind}).pFE);
end

%% Functions
function g = evalGaussian(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

function pFE =
optClass(N,x,priors,labels,lossMatrix,mu,Sigma)
symbols = 'oxsv';
n = 4;
for ind = 1:8
    pxgivenl(ind,:) =...
        evalGaussian(x,mu(:,ind),Sigma(:,:,ind)); %
Evaluate p(x|L=l)
end
for ind = 1:length(priors)
    pxgivenl(ind,:)=0.5*pxgivenl(2*ind-
1,:)+0.5*pxgivenl(2*ind,:);
end
pxgivenl(ind+1:end,:)=[];

px = priors*pxgivenl; % Total probability theorem
classPosteriors =
pxgivenl.*repmat(priors',1,N)./repmat(px,4,1); %
P(L=l|x)
```

```
expectedRisks = lossMatrix*classPosteriors; % Expected
Risk for each label (rows) for each sample (columns)
[~,decisions] = min(expectedRisks,[],1); % Minimum
expected risk decision with 0-1 loss is the same as MAP

for i = 1:4

    plot3(x(1,labels == i&decisions == i),x(2,labels ==
i&decisions == i),x(3,labels == i&decisions == i),...

'Marker',symbols(i),'MarkerEdgeColor','#3CB371',...

'MarkerFaceColor','none','LineStyle','none');axis
equal,hold on;
    plot3(x(1,labels == i&decisions ~= i),x(2,labels ==
i&decisions ~= i),x(3,labels == i&decisions ~= i),...

'Marker',symbols(i),'MarkerEdgeColor','#FF6347',...

'MarkerFaceColor','none','LineStyle','none');axis
equal,hold on;
    pFEIndex(i) = sum(labels == i & decisions ~= i);
end
legend('Correct decisions for data from Class 1','Wrong
decisions for data from Class 1',...
    'Correct decisions for data from Class 2','Wrong
decisions for data from Class 2',...
    'Correct decisions for data from Class 3','Wrong
decisions for data from Class 3', ...
    'Correct decisions for data from Class 4','Wrong
decisions for data from Class 4', ...
    'Location','northeast');
xlabel('x1');
ylabel('x2');
zlabel('x3');
title('Data and their classifier decisions versus true
labels')
pFE = sum(pFEIndex)/N;
% fprintf('Theoretical pFE =%1.2f%% \n',...
%     N,pFE*100);
end

function [outputNet,outputPFE, optM,
stats]=kfoldMLP_NN(numPerc,k,x,labels,numLabels)
```

```
%Assumes data is evenly divisible by partition choice
which it should be
N=length(x);
numValIters=10;
%Create output matrices from labels
y=zeros(numLabels,length(x));
for ind=1:numLabels
    y(ind,:)=(labels==ind-1);
end
%Setup cross validation on training data
partSize=N/k;
partInd=[1:partSize:N length(x)];
%Perform cross validation to select number of
perceptrons
for M=1:numPerc
    for ind=1:k
        index.val=partInd(ind):partInd(ind+1);
        index.train=setdiff(1:N,index.val);
        %Create object with M perceptrons in hidden
layer
        net=patternnet(M);%patternet
        % The featureInputLayer function is provided
after MATLAB 2020b.
        % I am not able to use trainNetwork function
and set the layers because mine is 2020a.
        % First  parameter of layer should be
'featureInputLayer'.
        % Secondly parameter is  RELU 'relyLayers'
        % Final parameter is softmax 'softmaxLayer'
        %Train using training data

net=train(net,x(:,index.train),y(:,index.train));
        %Validate with remaining data
        yVal=net(x(:,index.val));
        [~,labelVal]=max(yVal);
        labelVal=labelVal-1;

pFE(ind)=sum(labelVal~=labels(index.val))/partSize;
    end
    %Determine average probability of error for a
number of perceptrons
    avgPFE(M)=mean(pFE);
    stats.M=1:M;
    stats.mPFE=avgPFE;
```

```
end
%Determine optimal number of perceptrons
[~,optM]=min(avgPFE);
%Train one final time on all the data
for ind=1:numValIters
    netName(ind)={['net' num2str(ind)]};
    finalnet.(netName{ind})=patternnet(optM);
    % finalnet.layers{1}.transferFcn = 'softplus';%Set
to RELU
    finalnet.(netName{ind})=train(net,x,y);
    yVal=finalnet.(netName{ind})(x);
    [~,labelVal]=max(yVal);
    labelVal=labelVal-1;
    pFEFinal(ind)=sum(labelVal~=labels)/length(x);
end
[minPFE,outInd]=min(pFEFinal);
stats.finalPFE=pFEFinal;
outputPFE=minPFE;
outputNet=finalnet.(netName{outInd});
end
```

Appendix B:
Problem 2

```
clear; close all;
%Got significant information and code example from
2020SUMMER2 code, office hour and BICforGMM.m


%E = 100; %Repeat times
E = 50;
K = 10; %kfolds
maxM = 10;

%% Samples settings
% C = 5

D.classN = 5;
D.size = 10.^(2:6);
D.priors = repmat(1/5,1,5);

temp = linspace(0.5,0.8,8);% variance
for ind = 1:D.classN
    D.mu(:,ind) = [ind*2.3;0];
    D.Sigma(:,:,ind) = temp(ind)*eye(2);
end

for i = 1:length(D.size)
    for e = 1:E

        [D.labels{i},D.r{i}] =
myGaussian(D.size(i),D.priors,D.mu,D.Sigma);
        [meanEM(e,:,i),EM,bestMEM(i,e)] =
Kfold(K,D.r{i});
        [bestM(i,e),BIC]=bic(D.r{i},maxM);
        fprintf('The %1d-%1dth iterations: Kfold-%1d
\n',i,e,bestMEM(i,e))
    end
    fig=figure();
    hist(bestMEM(i,:),[1:1:maxM]);

saveas(fig,append('itsNOTBarchart',string(i),'.png'));
        fig=figure();
    hist(bestM(i,:),[1:1:maxM]);
```

```
saveas(fig,append('realNOTBarchart',string(i),'.png'));
end

%%
function [meanEM,EM,bestM] = Kfold(K,x)
[~,N]=size(x);
partSize = N/K;
partInd = [1:partSize:N N];
maxM =10;
for m=1:maxM
for ind = 1:K
    testXInd = partInd(ind):partInd(ind+1);
    trainXInd = setdiff(1:N,testXInd);
options = statset('Maxiter',1000); %max iterations
gm{ind}=fitgmdist(x(:,trainXInd)',m,'Replicates',5,'Reg
ularizationValue',1e-10,'Options',options);

EM{ind} = -sum(log(pdf(gm{ind},x(:,testXInd)')));
%logLikelihood(:,m) =
sum(log(evalGMM(gm{ind},alpha,mu,Sigma)));
end
meanEM(:,m) = sum([EM{:}])/(K-1);
end
[~,bestM] = min(meanEM);
end

%from BICforGMM.m
function [bestM,BIC]=bic(x,maxM)
[d,N] = size(x); %

for M = 1:maxM  %try m

    nParams(1,M) = (M-1) + d*M + M*(d+nchoosek(d,2));
    % (M-1) is the degrees of freedomg for alpha
parameters
    % d*M is the derees of freedomg for mean vectors of
M Gaussians
    % M*(d+nchoosek(d,2)) is the degrees of freedom in
cov matrices
    % For cov matrices, due to symmetry, only count
diagonal and half of
% off-diagonal entries.
    options = statset('Maxiter',1000); %max iterations
```

```
gm{M}=fitgmdist(x',M,'Replicates',5,'RegularizationValu
e',1e-10,'Options',options);
    %keyboard,
    neg2logLikelihood(1,M) = -
2*sum(log(pdf(gm{M},x')));
    BIC(1,M) =  neg2logLikelihood(1,M) +
nParams(1,M)*log(d*N);
end
[~,bestM] = min(BIC);
end
```

Appendix C:
hw3_1_1_generate.m

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

%% Generate samples
% 111 cube
% C = 4

D.classNF = 8;
D.classN = 4;
D.size = [100,200,500,1000,2000,5000,100000];
D.priorsF = repmat(1/8,1,8);
D.priors = repmat(1/4,1,4);

D.mu(:,1) = [1;1;1];D.mu(:,2) = [1;1;-1];
D.mu(:,3) = [1;-1;-1];D.mu(:,4) = [1;-1;1];
D.mu(:,5) = [-1;1;1];D.mu(:,6) = [-1;1;-1];
D.mu(:,7) = [-1;-1;-1];D.mu(:,8) = [-1;-1;1];

temp = linspace(0.5,0.8,8);% viarance
for i = 1:8
    D.Sigma(:,:,i) = temp(i)*eye(3);
end

%generate function
for i = 1: length(D.size)
    [D.labelsF{i},D.r{i}] =
myGaussian(D.size(i),D.priorsF,D.mu,D.Sigma);
end


%% Ture label & plot
figure(1);
for i = 1: length(D.size)
    for j = 1:length(D.labelsF)
    D.labels{i}(:,D.labelsF{i} ==(2*j-1)) = j;
    D.labels{i}(:,D.labelsF{i} ==(2*j)) = j;
    subplot(2,4,i);
    plot3(D.r{1,i}(1,D.labels{i}
==j),D.r{1,i}(2,D.labels{i} ==j),D.r{1,i}(3,D.labels{i}
==j),'.'); axis equal,hold on;
```

```
    end
    title(D.size(i)+" Samples");
end

D2 = D;
%% Save Data
save("hw3data"+".mat",'D','D2');
```