EECE 5644: Machine Learning / Pattern Recognition

Assignment 3

Jiayun Xin

NUID: 001563582

College of Engineering

Northeastern University Boston, Massachusetts

Spring, 2022

# Q1

As described in the question, the maximum likelihood parameter estimation is used to train many multilayer perceptrons (MLP). The trained models are then used to approximate a MAP classification rule.

Six training datasets with 100, 200, 500, 1000, 2000, 5000 samples and a test dataset with 100000 samples.

Logo, company name

Description automatically generated

Figure 1

Chart, scatter chart

Description automatically generated

Figure 2

Figures 1 and 2 show the true data distributions of the training and validation datasets. Four classes were labelled as different color.

The Deep Learning Toolbox ‘patternnet’ was used to create the 2-layer MLP. 10-fold cross- validation was used to choose the number of perceptrons.

Chart, scatter chart

Description automatically generated

Figure 3

The figure 3 shows the probability of error results with different number of data points. We can see that the probability of error values decreases as the amount of training points increases. So that, the overall accuracy increases because larger amount of data is more representative.

Chart, scatter chart

Description automatically generated

Figure 4

The figure 4 shows the number of perceptrons relevant to the number of data points. When the number of data points is 500, there is a maximum perceptrons number which is 9.

A picture containing text, computer, screenshot

Description automatically generated

Figure 5

The figure 5 shows 6 cross-validation results of the training dataset. They all have same trends as the amount of perceptrons increase which is the probability of error tends to be stable. The datasets with the lowest probability of error are about ten or a smaller number of perceptrons.

# Q2

As described in the question, the Gaussian Mixture Model is selected as the true probability density function for 2-dimensional real-valued data synthesis with 4 components with different mean vectors and covariance matrices.

With different samples number, the data distribution and log likelihood estimate results are below.

Graphical user interface, application

Description automatically generated

Figure 6

The figure 6 shows the true data distribution with different datasets. As we can see, the data distributes focusing on four different parts.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 7

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 8

The figure 7 shows the log likelihood estimate results with different datasets. As GMM number increases, the estimate value tends to be a constant value but dataset equals to 10.

The figure 8 shouws the BIC estimate results with different datasets. As GMM number increases, the estimate value tends to be a constant value but dataset equals to 10.

The best GMM order would be 4 because the maximum likelihood estimate result reach the highest and constant value and BIC estimate value reach the lowest value since GMM number is 4. At the same time, it ensures the accuracy of data when GMM number equals to 4.

# Appendix A – Matlab code for Question 1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%EECE5644 Spring 2022

%Homework #3

%Problem #1

%Significant parts of this code were derived from the following sources

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;

close all;

%Switches to bypass parts 1 and 2 for debugging

dimensions=3;

numLabels=4;

Lx={'L0','L1','L2','L3'};

lossMatrix = ones(numLabels,numLabels)-eye(numLabels);

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

mu.L0=muScale\*[1 1 0]';

RandSig=SigmaScale\*rand(dimensions,dimensions);

Sigma.L0(:,:,1)=RandSig\*RandSig'+eye(dimensions);

mu.L1=muScale\*[1 0 0]';

RandSig=SigmaScale\*rand(dimensions,dimensions);

Sigma.L1(:,:,1)=RandSig\*RandSig'+eye(dimensions);

mu.L2=muScale\*[0 1 0]';

RandSig=SigmaScale\*rand(dimensions,dimensions);

Sigma.L2(:,:,1)=RandSig\*RandSig'+eye(dimensions);

mu.L3=muScale\*[0 0 1]';

RandSig=SigmaScale\*rand(dimensions,dimensions);

Sigma.L3(:,:,1)=RandSig\*RandSig'+eye(dimensions);

%Generate Data

for ind=1:length(dTypes)

D.(dTypes{ind}).x=zeros(dimensions,D.(dTypes{ind}).N); %Initialize Data

[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

%Plot Training Data

figure;

for ind=1:length(dTypes)-1

subplot(3,2,ind);

plotData(D.(dTypes{ind}).x,D.(dTypes{ind}).labels,Lx);

legend 'show';

title(['True Data Distribution ', dTypes{ind}]);

end

%Plot Validation Data

figure;

plotData(D.(dTypes{ind}).x,D.(dTypes{ind}).labels,Lx);

legend 'show';

title(['True Data Distribution ', dTypes{end}]);

%Determine Theoretically Optimal Classifier

for ind=1:length(dTypes)

[D.(dTypes{ind}).opt.PFE, D.(dTypes{ind}).opt.decisions]=...

optClass(lossMatrix,D.(dTypes{ind}).x,mu,Sigma,...

p,D.(dTypes{ind}).labels,Lx);

opPFE(ind)=D.(dTypes{ind}).opt.PFE;

fprintf('Optimal pFE, N=%1.0f: Error=%1.2f%%\n',...

D.(dTypes{ind}).N,100\*D.(dTypes{ind}).opt.PFE);

end

%Train and Validate Data

numPerc=15; %Max number of perceptrons to attempt to train

k=10; %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}).decisions-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

for ind=1:length(dTypes)-1

[~,select]=min(valData.(dTypes{ind}).stats.mPFE);

M(ind)=(valData.(dTypes{ind}).stats.M(select));

N(ind)=D.(dTypes{ind}).N;

end

%Plot number of perceptrons vs. pFE for the cross validation runs

for ind=1:length(dTypes)-1

figure;

stem(valData.(dTypes{ind}).stats.M,valData.(dTypes{ind}).stats.mPFE);

xlabel('Number of Perceptrons');

ylabel('pFE');

title(['Probability of Error vs. Number of Perceptrons for ' dTypes{ind}]);

end

%Number of perceptrons vs. size of training dataset

figure,semilogx(N(1:end-1),M(1:end-1),'o','LineWidth',2)

grid on;

xlabel('Number of Data Points')

ylabel('Optimal Number of Perceptrons')

ylim([0 10]);

xlim([50 10^4]);

title('Optimal Number of Perceptrons vs. Number of Data Points');

%Prob. of Error vs. size of training data set

figure,semilogx(outpFE(1:end-1,1),outpFE(1:end-1,2),'o','LineWidth',2)

xlim([90 10^4]);

hold all;

semilogx(xlim,[opPFE(end) opPFE(end)],'r--','LineWidth',2)

legend('NN pFE','Optimal pFE')

grid on

xlabel('Number of Data Points')

ylabel('pFE')

title('Probability of Error vs. Data Points in Training Data');

# Appendix B – Matlab code for Question 2

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%EECE5644 Spring 2022

%Homework #3

%Problem #2

%Significant parts of this code were derived from the following sources

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;

close all;

%Counting variables to ensure the program is not stuck

countN = 0;

%All four sample sizes

for i = 1:4

countN = countN+1

%Tolerance for EM stopping criterion

delta = 1e-4;

%Regularization parameter for covariance estimates

regWeight = 1e-10;

%K-Fold Cross Validation

K = 10;

%Number of samples

N = [10,100,1000,10000];

%Generate samples from a 4-component GMM

alpha\_true = [0.17,0.22,0.28,0.33];

mu\_true = [10 -10 -10 10;-10 10 -10 10];

Sigma\_true(:,:,1) = [25 1;1 20];

Sigma\_true(:,:,2) = [27 4;4 5];

Sigma\_true(:,:,3) = [15 -9;-9 15];

Sigma\_true(:,:,4) = [4 1;1 22];

x = randGMM(N(i),alpha\_true,mu\_true,Sigma\_true);

%Plotting data

figure(i), clf,

plot(x(1,:),x(2,:),'ob')

xlabel('x1'); ylabel('x2');

title(strcat('True Data Distribution, N=',num2str(N(i))));

%To determine dimensionality of samples and number of GMM components

[d,MM] = size(mu\_true);

%Divide the data set into 10 approximately-equal-sized partitions

dummy = ceil(linspace(0,N(i),K+1));

for k = 1:K

indPartitionLimits(k,:) = [dummy(k)+1,dummy(k+1)];

end

%Allocate space

loglikelihoodtrain = zeros(K,6);

loglikelihoodvalidate = zeros(K,6);

Averagelltrain = zeros(1,6);

Averagellvalidate = zeros(1,6);

countM = 0;

%Try all 6 mixture options

for M = 1:6

countM = countM+1

countk = 0;

%10-fold cross validation

for k = 1:K

countk = countk+1

indValidate = [indPartitionLimits(k,1):indPartitionLimits(k,2)];

%Using folk k as validation set

x1Validate = x(1,indValidate);

x2Validate = x(2,indValidate);

if k == 1

indTrain = [indPartitionLimits(k,2)+1:N(i)];

elseif k == K

indTrain = [1:indPartitionLimits(k,1)-1];

else

indTrain = [1:indPartitionLimits(k-1,2),indPartitionLimits(k+1,2):N(i)];

end

%Using all other folds as training set

x1Train = x(1,indTrain);

x2Train = x(2,indTrain);

xTrain = [x1Train; x2Train];

xValidate = [x1Validate; x2Validate];

Ntrain = length(indTrain); Nvalidate = length(indValidate);

%Train model parameters (EM)

%Initialize the GMM to randomly selected samples

alpha = ones(1,M)/M;

shuffledIndices = randperm(Ntrain);

%Pick M random samples as initial mean estimates (this led to good initial estimates (better log likelihoods))

mu = xTrain(:,shuffledIndices(1:M));

%Assign each sample to the nearest mean (better initialization)

[~,assignedCentroidLabels] = min(pdist2(mu',xTrain'),[],1);

%Use sample covariances of initial assignments as initial covariance estimates

for m = 1:M

Sigma(:,:,m) = cov(xTrain(:,find(assignedCentroidLabels==m))') + regWeight\*eye(d,d);

end

t = 0;

%Not converged at the beginning

Converged = 0;

while ~Converged

for l = 1:M

temp(l,:) = repmat(alpha(l),1,Ntrain).\*evalGaussian(xTrain,mu(:,l),Sigma(:,:,l));

end

plgivenx = temp./sum(temp,1);

clear temp

alphaNew = mean(plgivenx,2);

w = plgivenx./repmat(sum(plgivenx,2),1,Ntrain);

muNew = xTrain\*w';

for l = 1:M

v = xTrain-repmat(muNew(:,l),1,Ntrain);

u = repmat(w(l,:),d,1).\*v;

%Adding a small regularization term

SigmaNew(:,:,l) = u\*v' + regWeight\*eye(d,d);

end

Dalpha = sum(abs(alphaNew-alpha));

Dmu = sum(sum(abs(muNew-mu)));

DSigma = sum(sum(abs(abs(SigmaNew-Sigma))));

%Check if converged

Converged = ((Dalpha+Dmu+DSigma)<delta);

alpha = alphaNew;

mu = muNew;

Sigma = SigmaNew;

t = t+1;

end

%Validation

loglikelihoodtrain(k,M) = sum(log(evalGMM(xTrain,alpha,mu,Sigma)));

loglikelihoodvalidate(k,M) = sum(log(evalGMM(xValidate,alpha,mu,Sigma)));

end

%Average Performance Variables

Averagelltrain(1,M) = mean(loglikelihoodtrain(:,M));

BICtrain(1,M) = -2\*Averagelltrain(1,M)+M\*log(N(i));

Averagellvalidate(1,M) = mean(loglikelihoodvalidate(:,M));

%Sometimes the log likelihoods for N=10 are zero, leading to %negative infinity results.

% I assume that this is instead the %lowest log likelihood value instead (so it is possible to graph).

if isinf(Averagellvalidate(1,M))

Averagellvalidate(1,M) = (min(Averagellvalidate(find(isfinite(Averagellvalidate)))));

end

BICvalidate(1,M) = -2\*Averagellvalidate(1,M)+M\*log(N(i));

%Recording values

TotBICValidate(i,M) = BICvalidate(1,M);

TotBICTrain(i,M) = BICtrain(1,M);

TotAvgllValidate(i,M) = Averagellvalidate(1,M);

TotAvgllTrain(i,M) = Averagelltrain(1,M);

end

%Recording Best Outcomes

[LowestBIC orderB] = min(BICvalidate)

[Lowestll orderl] = max(Averagellvalidate)

figure(i+4), clf, plot(Averagelltrain,'.b');

xlabel('GMM Number');

ylabel(strcat('Log likelihood estimate with ',num2str(K),'- fold cross-validation'));

title(strcat('Training Log-Likelihoods for N=',num2str(N(i))));

grid on

figure(i+8), clf, plot(Averagellvalidate,'rx');

xlabel('GMM Number');

ylabel(strcat('Log likelihood estimate with ',num2str(K),'- fold cross-validation'));

title(strcat('Log-Likelihoods Estimate Result, N=',num2str(N(i))));

grid on

figure(i+12), clf, plot(BICtrain,'.b');

xlabel('GMM Number');

ylabel(strcat('BIC estimate with ',num2str(K),'-fold cross- validation'));

title(strcat('Training BICs for N=',num2str(N(i))));

grid on

figure(i+16), clf, plot(BICvalidate,'rx');

xlabel('GMM Number');

ylabel(strcat('BIC estimate with ',num2str(K),'-fold cross- validation'));

title(strcat('Validation BICs, N=',num2str(N(i))))

grid on

%Saving values

BICorder(i) = orderB;

BIClow(i) = LowestBIC;

lorder(i) = orderl;

lllow(i) = Lowestll;

end

# Appendix C – Matlab code function used

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Function Definitions

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function x = randGMM(N,alpha,mu,Sigma)

d = size(mu,1); % dimensionality of samples

cum\_alpha = [0,cumsum(alpha)];

u = rand(1,N); x = zeros(d,N); labels = zeros(1,N);

for m = 1:length(alpha)

ind = find(cum\_alpha(m)<u & u<=cum\_alpha(m+1));

x(:,ind) = randGaussian(length(ind),mu(:,m),Sigma(:,:,m));

end

end

function x = randGaussian(N,mu,Sigma)

% Generates N samples from a Gaussian pdf with mean mu covariance Sigma

n = length(mu);

z = randn(n,N);

A = Sigma^(1/2);

x = A\*z + repmat(mu,1,N);

end

function gmm = evalGMM(x,alpha,mu,Sigma)

gmm = zeros(1,size(x,2));

for m = 1:length(alpha) % evaluate the GMM on the grid

gmm = gmm + alpha(m)\*evalGaussian(x,mu(:,m),Sigma(:,:,m));

end

end

function g = evalGaussian(x,mu,Sigma)

% Evaluates the Gaussian pdf N(mu,Sigma) at each column 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 [x,labels,N\_l,p\_hat]= genData(N,p,mu,Sigma,Lx,d)

%Generates data and labels for random variable x from multiple gaussian %distributions

numD = length(Lx);

cum\_p = [0,cumsum(p)];

u = rand(1,N);

x = zeros(d,N);

labels = zeros(1,N);

for ind=1:numD

pts = find(cum\_p(ind)<u & u<=cum\_p(ind+1));

N\_l(ind)=length(pts);

x(:,pts) = mvnrnd(mu.(Lx{ind}),Sigma.(Lx{ind}),N\_l(ind))';

labels(pts)=ind-1;

p\_hat(ind)=N\_l(ind)/N;

end

end

function plotData(x,labels,Lx)

%Plots data

for ind=1:length(Lx)

pindex=labels==ind-1;

plot3(x(1,pindex),x(2,pindex),x(3,pindex),'.','DisplayName',Lx{ind});

hold all;

end

grid on;

xlabel('x1');

ylabel('x2');

zlabel('x3');

end

function [minPFE,decisions]=optClass(lossMatrix,x,mu,Sigma,p,labels,Lx)

% Determine optimal probability of error

symbols='ox+\*v';

numLabels=length(Lx);

N=length(x);

for ind = 1:numLabels

pxgivenl(ind,:) = evalGaussian(x,mu.(Lx{ind}),Sigma.(Lx{ind})); % Evaluate p(x|L=l)

end

px = p\*pxgivenl; % Total probability theorem

classPosteriors = pxgivenl.\*repmat(p',1,N)./repmat(px,numLabels,1); %P(L=1|x)

% Expected Risk for each label (rows) for each sample (columns)

expectedRisks =lossMatrix\*classPosteriors;

% Minimum expected risk decision with 0-1 loss is the same as MAP

[~,decisions] = min(expectedRisks,[],1);

decisions=decisions-1; %Adjust to account for L0 label

fDecision\_ind=(decisions~=labels);%Incorrect classificiation vector

minPFE=sum(fDecision\_ind)/N;

%Plot Decisions with Incorrect Results

figure;

for ind=1:numLabels

class\_ind=decisions==ind-1;

plot3(x(1,class\_ind & ~fDecision\_ind),...

x(2,class\_ind & ~fDecision\_ind),...

x(3,class\_ind & ~fDecision\_ind),...

symbols(ind),'Color',[0.39 0.83 0.07],'DisplayName',...

['Class ' num2str(ind) ' Correct Classification']);

hold on;

plot3(x(1,class\_ind & fDecision\_ind),...

x(2,class\_ind & fDecision\_ind),...

x(3,class\_ind & fDecision\_ind),...

['r' symbols(ind)],'DisplayName',...

['Class ' num2str(ind) ' Incorrect Classification']);

hold on;

end

xlabel('x1');

ylabel('x2');

grid on;

title('X Vector with Incorrect Classifications');

legend 'show';

if 0

%Plot Decisions with Incorrect Decisions

figure;

for ind2=1:numLabels

subplot(3,2,ind2);

for ind=1:numLabels

class\_ind=decisions==ind-1;

plot3(x(1,class\_ind),x(2,class\_ind),x(3,class\_ind),...

'.','DisplayName',['Class ' num2str(ind)]);

hold on;

end

plot3(x(1,fDecision\_ind & labels==ind2),...

x(2,fDecision\_ind & labels==ind2),...

x(3,fDecision\_ind & labels==ind2),...

'kx','DisplayName','Incorrectly Classified','LineWidth',2);

ylabel('x2');

grid on;

title(['X Vector with Incorrect Decisions for Class ' num2str(ind2)]);

if ind2==1

legend 'show';

elseif ind2==4

xlabel('x1');

end

end

end

end

function [outputNet,outputPFE, optM,stats]=kfoldMLP\_NN(numPerc,k,x,labels,numLabels)

%Assumes data is evenly divisible by partition choice

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);

% net.layers{1}.transferFcn = 'softplus';%didn't work

%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