EECE 5644 HW3 Report

Shijin Wang

Question 1:

Description:

In this question, we are asked to train an MLP to approximate the class posterior, using maximum likelihood parameter estimation.

Process:

The overall model function is : $h(x, \theta) = m(d + C\sigma(b + Ax))$

The parameter need to be optimized : A. b. C. d.

MAP Rule:

$\text{Decision}(x) = \arg\max\{i \text{ in } \{1 \cdots C\}\}\ p(L=i \mid x)$

C is number of classes.

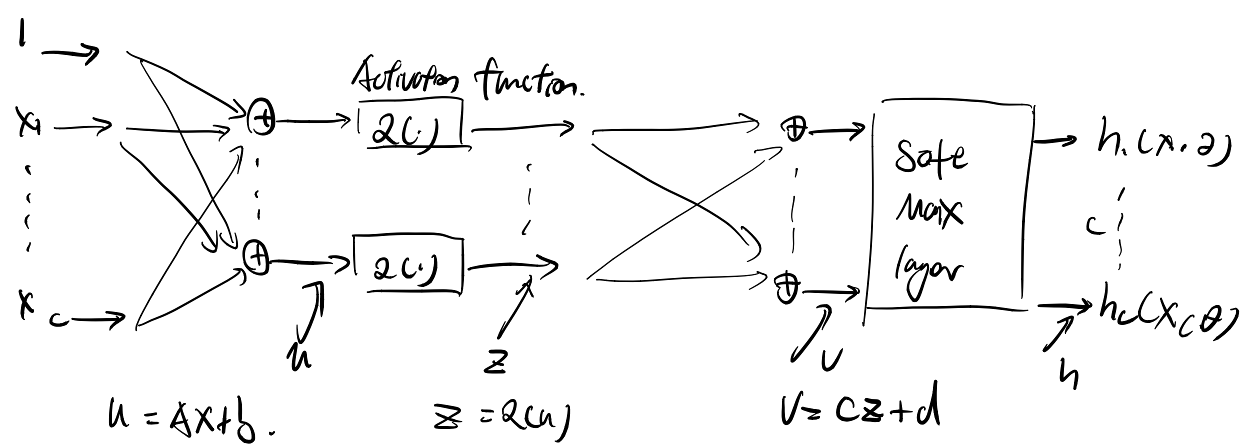

Figure 1. The structure of MLP

"Three" was picked as the number of classes in my dataset. In order to find the best combination of activation functions and perceptrons, besides ISRU and Sigmod functions, I added Soft Plus as the third function. My perceptrons range from 1 to 10. The first step is to preprocess the labeled dataset to a matrix with the same dimension as the number of classes.

Then, In order to perform the 10 fold cross-validation, I split the index of the training set and validation set to 10 blocks. For each activation function and each perceptron, 90% of the Dtrain dataset is used for training and the rest is used for validation. All the parameters for H function are randomly initialized. I choose to use the fminsearch library to achieve the objective which is to minimize the square error for parameters by comparing the input labels with the output from the softmax layer. Meanwhile, I calculated the percentage of error by applying the MAP classifier which set the estimated class as the class with the highest probability in the output of the softmax layer. I created several structs to store the values of the parameters for future parameter initialization and several array to store the percentage error of each fold process.

Lastly, after the loop has gone through 3 activation functions each with 10 perceptrons, which is 30 combinations, I select the combo with the lowest percentage of error and pass those to the final trained MLP model which is trained by the entire Dtrain dataset. Then, comparing the softmax output with the Dtest validation dataset, I achieve the final percentage of error(accuracy) for each Dtrain datasets. (100, 500, 1000)

In order to achieve a lower percentage of error with my model, I did several experiments with the relation between the initialization values of the parameters and the final percent accuracy. The default method is to initialize all the parameters to zero which results in 60% to 70% accuracy on Dtest dataset. The first method is to initialize all the parameters randomly which results in 80% to 85% accuracy on Dtest dataset.

The second method is to only randomly initialize parameters once for each activation function and perceptron, then each 10 fold iteration will use the parameters generated from the last 10 fold iteration as the initialization, meanwhile, storing the last parameters generated after the last 10 fold iteration to a 2-D struct. Then, after determining the best combo, use the parameters stored for that combo as the initialization of the final model. In this way, the model achieves 90% to 95% accuracy on Dtest dataset. The reason behind this method is that the parameters can be optimized 10 times during 10 fold cross-validation which might result in a better performance than randomly initialize the final model.

The third method is the one that I use for this assignment. Firstly, randomly initializing the parameters in each 10 fold cross-validation iteration, then, storing the perimeters with the lowest error of percentages for each activation function and perceptron, finally, after determining the best combo, use the parameters stored for that combo as the initialization of the final model. The percent accuracy of the final model can be around 96% to 97% in this way.

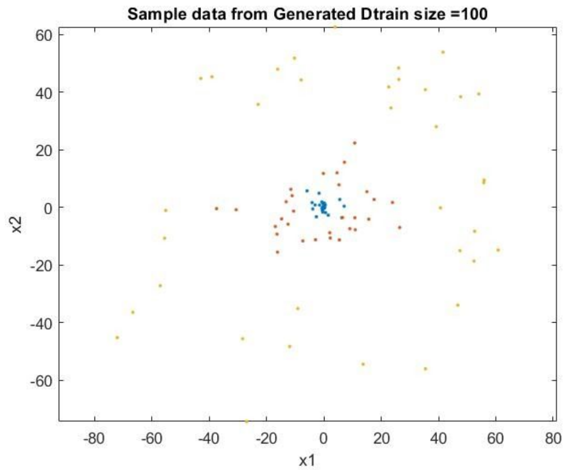

Figure 2. The 100 iid training samples

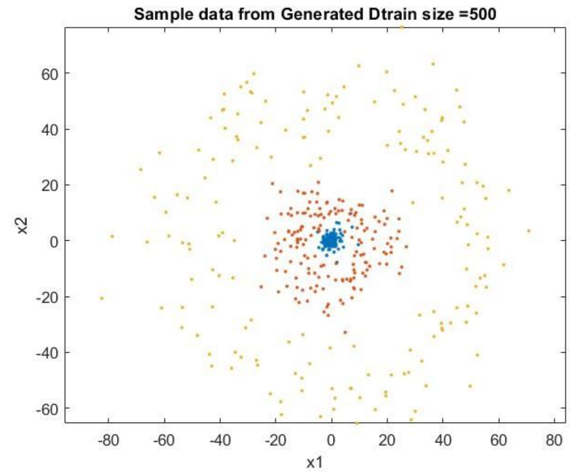

Figure 3. The 500 iid training samples

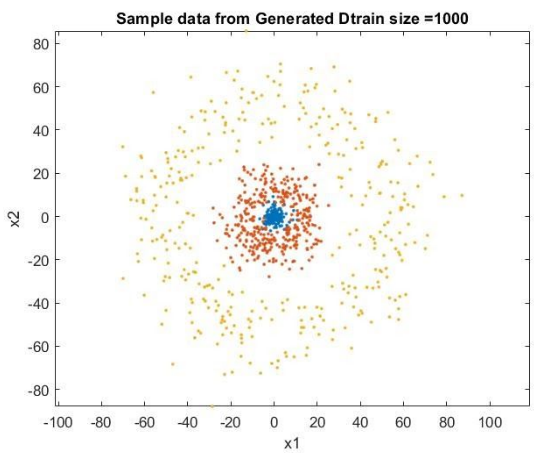

Figure 4. The 1000 iid training samples

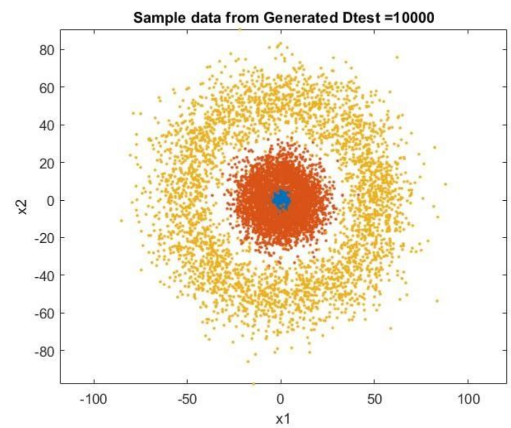

Figure 5. The 10000 iid testing samples

Best combo: Activation Function= Sigmoid, Perceptron = 10

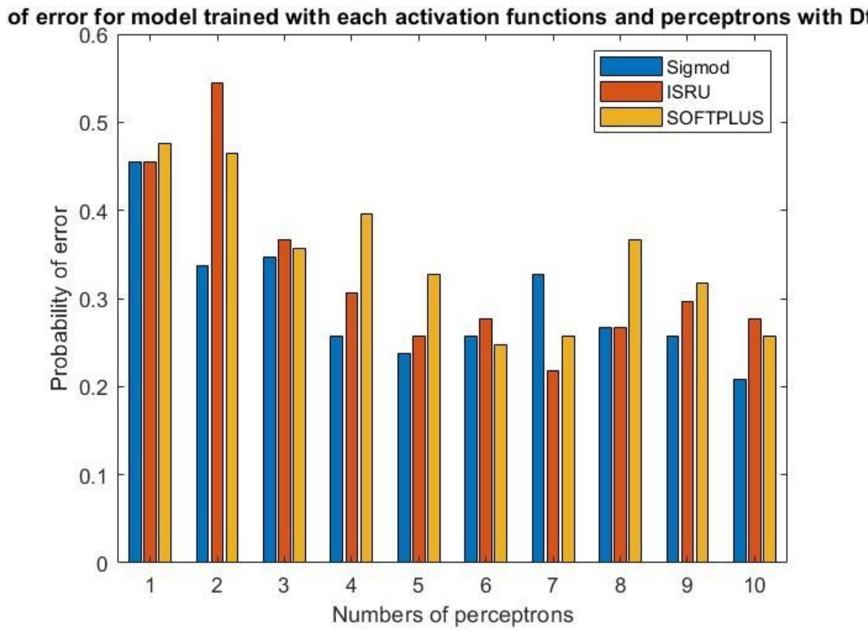

Figure 6. Percentage of error for a model trained with each activation functions and perceptrons with Dtrain size = 100

Best combo: Activation Function= SoftPlus, Perceptron = 10

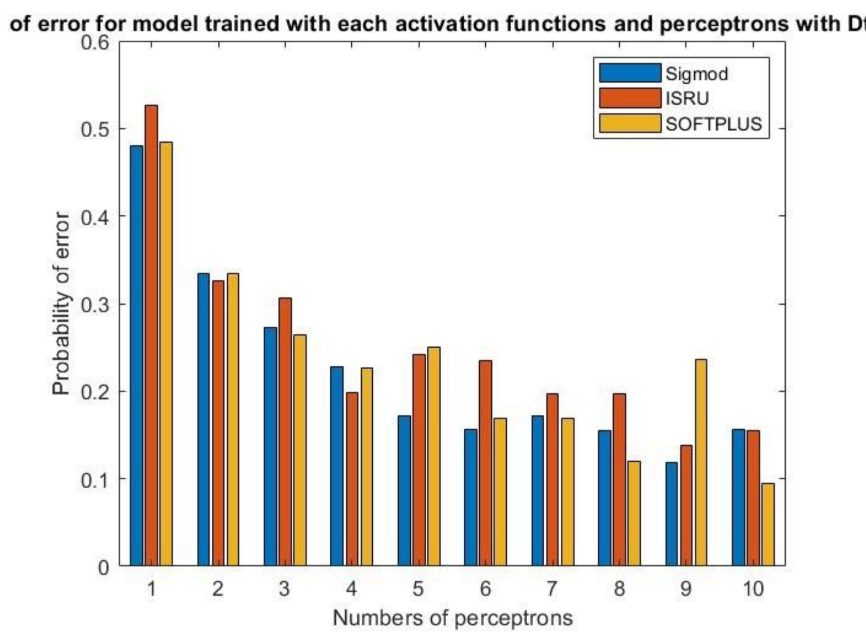

Figure 7. Percentage of error for a model trained with each activation functions and perceptrons with Dtrain size = 500

Best combo: Activation Function= SoftPlus, Perceptron = 8

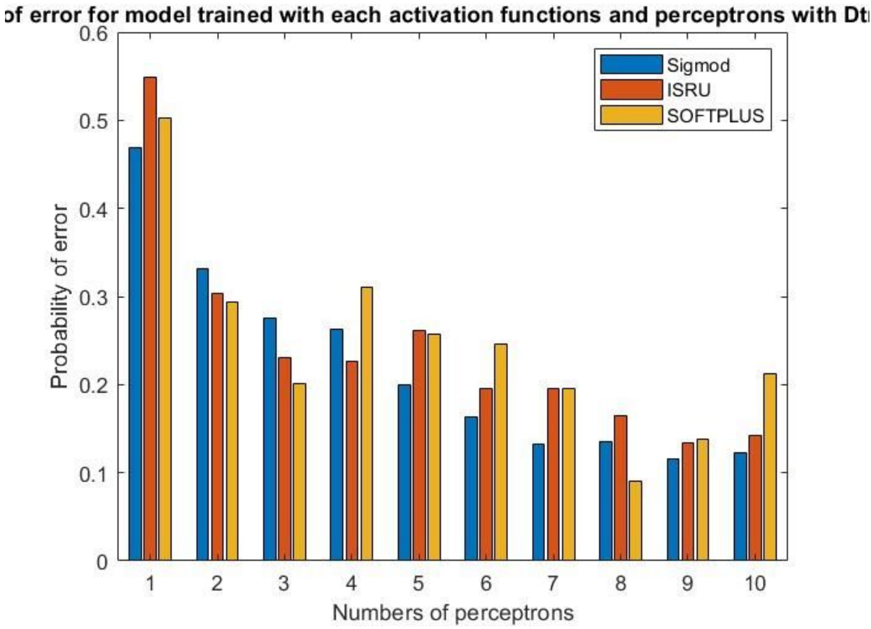

Figure 8. Percentage of error for a model trained with each activation functions and perceptrons with Dtrain size = 1000

Percent accuracy: 0.5091 0.9602 0.9613

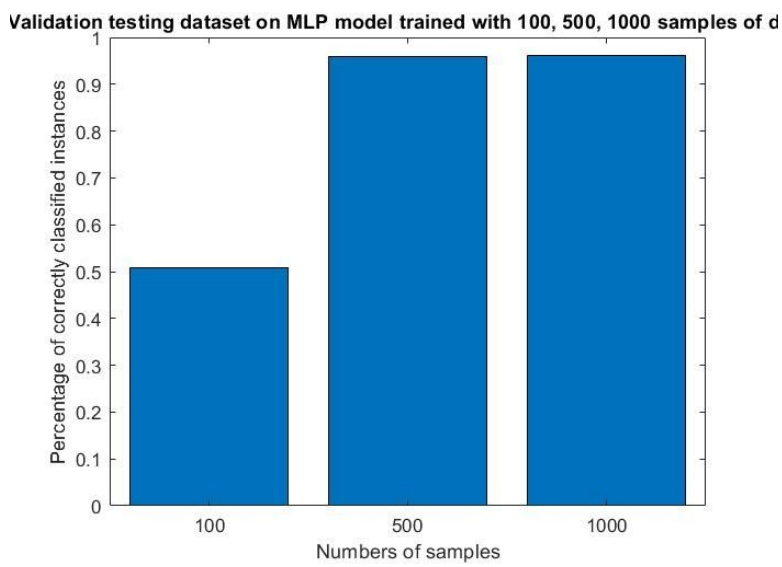

Figure 9. Percent accuracy of models trained by 100, 500, and 1000 training samples on testing samples

Conclusion:

In figure 6, figure 7 and figure 8, we can see that the different combinations of activation functions and perceptrons have a greater impact on the model trained with a larger Dtrain dataset. As we can see, there is not much difference in terms of probability of error in figure 6 (100 Dtrain datasets), however, it is obvious that the probability of error gets lower around 8 to 10 perceptrons in figure 7 and figure 8. On the other hand, with the same numbers of perceptrons, sigmoid performs the best, ISRU and Softplus perform identically in a smaller size dataset. However, as the dataset becomes larger, Softplus performs better than sigmoid and ISRU in terms of probability of error.

Overall, the final trained model performs the best when it is trained with a larger dataset. However, the percent accuracy does not have a huge difference between the dataset with 500 samples and 1000 samples.

Question 2:

Description:

In this exercise, we are asked to train an alternative approximate MAP classifier for the same datasets in Question 1, using GMM for each class conditional PDF.

Process:

MAP: $\text{argmax}\{i \text{ in } \{1 \cdots C\}\}\ P(x|L=i)\, p(L=i)$ rule. C is number of classes.

In this problem, the training dataset needs to be split into the number of classes. For each class and for each GMM order (1-6), I applied 10-fold cross-validation steps that are the same as question 1. Then, I used EM algorithm which was provided by Professor Deniz which is to initialize the alpha value to the same weights for the number of components, then randomly pick values from the training dataset for mu and Sigma. Those parameters were recalculated and updated each time until converged. In order to solve the problem of not converging, I set up a limitation of the maximum iteration times which is 10000.

After the validation, an average sum log-likelihood was calculated for each component. The best GMM order can be selected by the component with the highest log-likelihood. Then, passing this GMM order to the EM algorithm again, we can have an estimated alpha, mu, and Sigma value for each class. Applying the MAP classifier which is using the evalGMM function to generate the class-conditional pdf and times the prior of each class (P(x | L=i) p(L=i)). The estimation result can be determined by the class with the highest score.

Best GMM order for Class 1: 3 Class 2: 1 Class3: 2

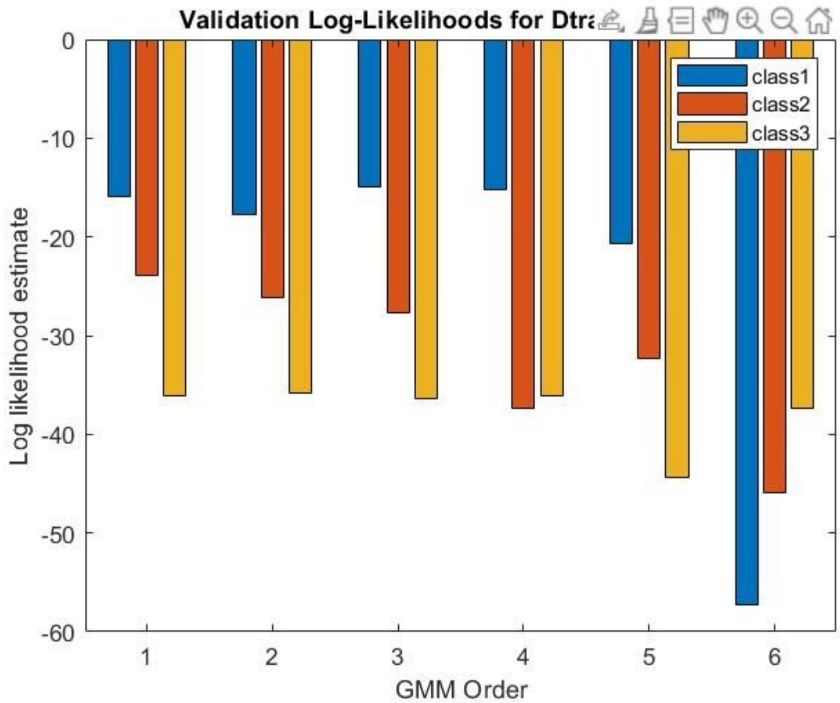

Figure 10. Percentage accuracy of a model trained by 100 training samples with different model orders

Best GMM order for Class 1: 3 Class 2: 3 Class3: 6

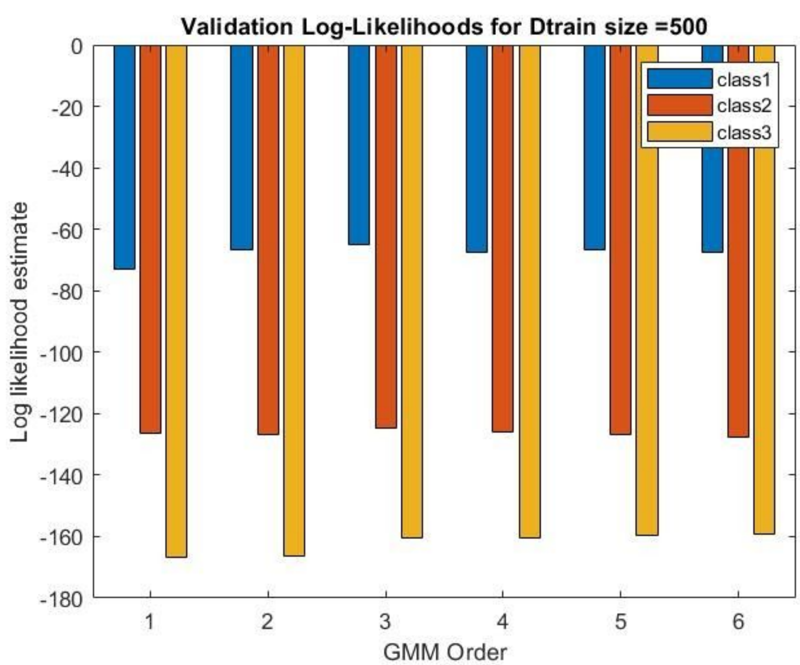

Figure 11. Percentage accuracy of a model trained by 500 training samples with different model orders

Best GMM order for Class 1: 3 Class 2: 4 Class3: 6

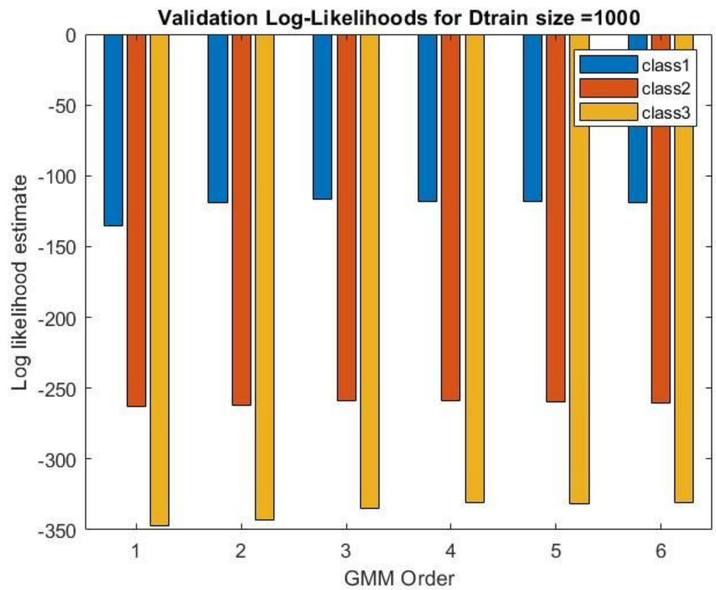

Figure 12. Percentage accuracy of a model trained by 1000 training samples with different model orders

Percent Accuracy = 0.9208 0.9670 0.9717

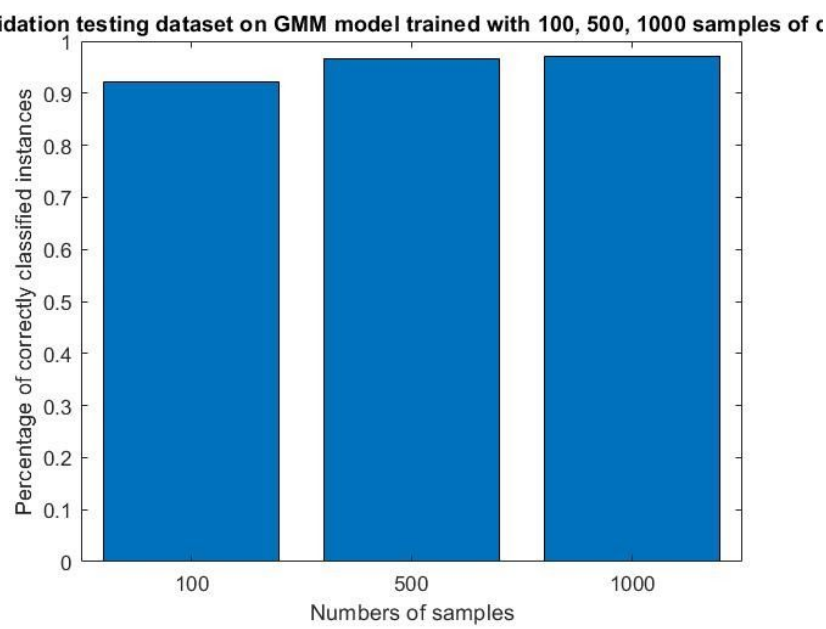

Figure 13. Percentage accuracy of models trained by 100, 500, and 1000 training samples on testing samples

*Prior for each class: 0.3327 0.3297 0.3376

Conclusion:

In figure 10, 11, 12, we can see that the best GMM order goes higher as the number of samples increased. On the other hand, the difference in average maximum likelihood among GMM orders is less obvious as the number of samples increased. Finally, in figure 13, the trained model performs the best when it is trained with a larger dataset.

https://github.com/Alanbition/MachineLearning

Code:

```
% Maximum likelihood training of a 2-layer MLP
% assuming additive (white) Gaussian noise
close all,
clear
dummyOut = 0;
% Input N specifies number of training samples

F = 10;
Perceptrons = 10;
numberOfClasses = 3;
numberOfSamples = 3;
numberOfDtrain_100 = 100;
numberOfDtrain_500 = 500;
numberOfDtrain_1000 = 1000;
numberOfDtest = 10000;


%Generate Dtrain and Dtest
[Dtrain_100,DtrainLabels_100] = generateMultiringDataset(numberOfClasses,numberOfDtrain_100);
[Dtrain_500,DtrainLabels_500] = generateMultiringDataset(numberOfClasses,numberOfDtrain_500);
[Dtrain_1000,DtrainLabels_1000] = generateMultiringDataset(numberOfClasses,numberOfDtrain_1000);
[Dtest,DtestLabels] = generateMultiringDataset(numberOfClasses,numberOfDtest);
fig = 1
figure(fig), clf,
colors = rand(numberOfDtest,3);
for l = 1:numberOfClasses
   ind_l = find(DtestLabels==l);
   plot(Dtest(1,ind_l),Dtest(2,ind_l),'.','MarkerFaceColor',colors(l,:)), axis equal, hold on,
end
xlabel('x1'); ylabel('x2');
title(strcat('Sample data from Generated Dtest = ', num2str(numberOfDtest)));
drawnow()

%Use a for loop to iterate three datasets
for s=1:numberOfSamples
   disp(strcat('Question1 with sample',num2str(s)));
   if s == 1
      numberOfDtrain = numberOfDtrain_100;
      Dtrain = Dtrain_100;
      DtrainLabels = DtrainLabels_100;
   elseif s == 2
      numberOfDtrain = numberOfDtrain_500;
      Dtrain = Dtrain_500;
```

```
        DtrainLabels = DtrainLabels_500;
    else
        numberOfDtrain = numberOfDtrain_1000;
        Dtrain = Dtrain_1000;
        DtrainLabels = DtrainLabels_1000;
    end

    colors = rand(numberOfClasses,3);
    fig = fig+1;
    figure(fig), clf,
    for l = 1:numberOfClasses
        ind_l = find(DtrainLabels==l);
        plot(Dtrain(1,ind_l),Dtrain(2,ind_l),'.','MarkerFaceColor',colors(l,:)), axis equal, hold on,
    end
    xlabel('x1'); ylabel('x2');
    title(strcat('Sample data from Generated Dtrain size = ', num2str(numberOfDtrain)));
    drawnow()

%Seperate Dtrain and Dtest data to a new array with the format:
%(Class number, 1 or 0)

for c=1:numberOfClasses
    Ylabels(c,:)=DtrainLabels==c;
    YtestLabels(c,:)=DtestLabels==c;
end

%Split the data set to 10 block for 10 fold
block = ceil(linspace(0, numberOfDtrain, F+1));

for k = 1:F
    datasetBlock(k,:) = [block(k)+1,block(k+1)];
end
% Intialize likelihood
%
%
% ---------------------------------Start of Question 1-----------------------------------
numberOfaf = 3;

%Initialize array to store errors
errorTrain = zeros(F, Perceptrons);
errorValidate = zeros(F, Perceptrons);
errorAverage = zeros(3,Perceptrons);

%Initialize a struct to store params result at the last 10fold operations
ParamsStore = struct();
FoldParamsStore = struct();
counter = 1;
 for nPerceptrons = 1:Perceptrons
```

```
for af = 1:numberOfaf
    %Initializae params here so that next fold can use the previous
    %result(which result a better accuracy than put inside 10fold loop from my testing)

    counter = counter + 1;
    for k = 1:F

    %Assign validation and train set for each iteration
    validateIndex = [datasetBlock(k,1):datasetBlock(k,2)];
    if k == 1
        trainIndex = [datasetBlock(k, 2)+1:numberOfDtrain];
    elseif k == F
        trainIndex = [1:datasetBlock(k, 1)-1];
    else
        trainIndex = [1:datasetBlock(k-1, 2), datasetBlock(k+1, 2):numberOfDtrain];
    end

    %Select active function from sigmod, ISRU and SOFTPLUS
    if af == 1
        type = "sigmod";
    elseif af == 2
        type = "ISRU";
    else
        type = "SOFTPLUS";
    end

    yValidate = [Ylabels(1,validateIndex);Ylabels(2,validateIndex);Ylabels(3,validateIndex)];
    xValidate = [Dtrain(1,validateIndex);Dtrain(2, validateIndex)];
    validateLen = length(validateIndex);

    yTrain = [Ylabels(1,trainIndex);Ylabels(2,trainIndex);Ylabels(3,trainIndex)];
    xTrain = [Dtrain(1,trainIndex);Dtrain(2, trainIndex)];
    trainLen = length(trainIndex);

    X = xTrain;
    Y = yTrain;

    params.A = randn(nPerceptrons,2);
    params.b = randn(nPerceptrons,1);
    params.C = randn(3,nPerceptrons);
    params.d = mean(Y,2);

    %disp(["xTrain", size(xTrain)])
    %Determine/specify sizes of parameter matrices/vectors
    nX = size(X,1);
    nY = size(Y,1);
    sizeParams = [nX;nPerceptrons;nY];
```

```
        %Initialize model parameters
        %zeros(nY,1); % initialize to mean of y

        %params = paramsTrue;

        vecParamsInit = [params.A(:);params.b;params.C(:);params.d];

        %Optimize model
        options = optimset('MaxFunEvals',200000, 'MaxIter',200000); %Increase MaxFunEvals and MaxIter
        vecParams = fminsearch(@(vecParams)(objectiveFunction(type,
X,Y,sizeParams,vecParams)),vecParamsInit, options);


        params.A = reshape(vecParams(1:nX*nPerceptrons),nPerceptrons,nX);
        params.b = vecParams(nX*nPerceptrons+1:(nX+1)*nPerceptrons);
        params.C =
reshape(vecParams((nX+1)*nPerceptrons+1:(nX+1+nY)*nPerceptrons),nY,nPerceptrons);
        params.d = vecParams((nX+1+nY)*nPerceptrons+1:(nX+1+nY)*nPerceptrons+nY);

        H = mlpModel(type, xValidate,params);

        %Calculate error percentage
        [val, testIdx] = max(H);
        [val, labelIdx] = max(yValidate);
        error = find(testIdx~=labelIdx);
        errorP = size(error)/size(testIdx);
        errorTrain(k, nPerceptrons) = errorP;
        FoldParamsStore(k, nPerceptrons).A = params.A;
        FoldParamsStore(k, nPerceptrons).b = params.b;
        FoldParamsStore(k, nPerceptrons).C = params.C;
        FoldParamsStore(k, nPerceptrons).d = params.d;

    end
    disp([counter,"/31"])
    %Calculate error Average
    errorAverage(af, nPerceptrons) = mean(errorTrain(:, nPerceptrons));
    errorTrain(:, nPerceptrons);
    [val, minK] = min(errorTrain(:, nPerceptrons));
    minK = min(minK(:))
    %Store best parms for fulture usage
    ParamsStore(af,nPerceptrons).A = FoldParamsStore(minK, nPerceptrons).A;
    ParamsStore(af,nPerceptrons).b = FoldParamsStore(minK, nPerceptrons).b;
    ParamsStore(af,nPerceptrons).C = FoldParamsStore(minK, nPerceptrons).C;
    ParamsStore(af,nPerceptrons).d = FoldParamsStore(minK, nPerceptrons).d;
    clear FoldParamsStore
    end
 end
```

```
 errorAverage;

%  for nPerceptrons = 1:Perceptrons
%     for af = 1:numberOfaf
%         bar(1:Perceptrons, errorAverage(af, nPerceptrons))
%     end
%  end
 fig = fig + 1;
 figure(fig), clf,
 b1 = bar(1:nPerceptrons, errorAverage),
 title(strcat('Probability of error for model trained with each activation functions and perceptrons with
Dtrain size = ', num2str(numberOfDtrain))),
 ylabel('Probability of error'),
 xlabel('Numbers of perceptrons'),
 legend('Sigmod','ISRU', 'SOFTPLUS'),
 drawnow()

 [val, idx] = min(errorAverage(:));
 [af, nPerceptrons] =find(errorAverage==val);


 af = min(af(:));% In case select same af performance
 nPerceptrons = max(nPerceptrons(:));% In case same perceptrons performance

 BestMLP(s,1) = af;
 BestMLP(s,2) = nPerceptrons;

if af == 1
    type = "sigmod";
elseif af == 2
    type = "ISRU";
else
    type = "SOFTPLUS";
end

X = Dtrain;%Y
Y = Ylabels;%training label
%Determine/specify sizes of parameter matrices/vectors
nX = size(X,1);
nY = size(Y,1);
sizeParams = [nX;nPerceptrons;nY];
%Initialize model parameters

params.A = randn(nPerceptrons,nX);
params.b = randn(nPerceptrons,1);
params.C = randn(nY,nPerceptrons);
params.d = mean(Y,2);
```

```
%Init with pervious best params
% params.A = ParamsStore(af,nPerceptrons).A;
% params.b = ParamsStore(af,nPerceptrons).b;
% params.C = ParamsStore(af,nPerceptrons).C;
% params.d = mean(Y,2);%ParamsStore(af,nPerceptrons).d;


vecParamsInit = [params.A(:);params.b;params.C(:);params.d];

%Optimize mode
options = optimset('MaxFunEvals',200000, 'MaxIter',200000);
vecParams = fminsearch(@(vecParams)(objectiveFunction(type,
X,Y,sizeParams,vecParams)),vecParamsInit, options);


%Visualize model output for training data
params.A = reshape(vecParams(1:nX*nPerceptrons),nPerceptrons,nX);
params.b = vecParams(nX*nPerceptrons+1:(nX+1)*nPerceptrons);
params.C = reshape(vecParams((nX+1)*nPerceptrons+1:(nX+1+nY)*nPerceptrons),nY,nPerceptrons);
params.d = vecParams((nX+1+nY)*nPerceptrons+1:(nX+1+nY)*nPerceptrons+nY);
H = mlpModel(type, Dtest,params);
[val, testIdx] = max(H);
[val, labelIdx] = max(YtestLabels);
error = find(testIdx~=labelIdx);
errorP = size(error)/size(testIdx);
errorStore(s) = 1- size(error)/size(testIdx);
disp(["Final Accuracy:", 1-errorP])
clear Ylabels
clear YtestLabels
clear block
clear datasetBlock

end

 fig = fig +1;
 figure(fig), clf,
 X = categorical({'100','500','1000'});
 X = reordercats(X,{'100','500','1000'});
 b1 = bar(X, errorStore),
 title('Validation testing dataset on MLP model trained with 100, 500, 1000 samples of data'),
 ylabel('Percentage of correctly classified instances'),
 xlabel('Numbers of samples'),
 drawnow()

% %----------------------------------End of Question 1------------------------------------


%----------------------------------Start of Question 2------------------------------------
```

```
delta = 1e-5; % tolerance for EM stopping criterion
regWeight = 1e-11; % regularization parameter for covariance estimates
%Get seperate data from 3 classes

%We have 3 class, the the number of row of mu should be 2
mu_true = [-8 -8 8;-8 8 8];%This is just a dummy for row = 2
[d,M] = size(mu_true);




for s=1:numberOfSamples

    disp(strcat('Question2 with sample',num2str(s)));
    if s == 1
        numberOfDtrain = numberOfDtrain_100;
        Dtrain = Dtrain_100;
        DtrainLabels = DtrainLabels_100;
    elseif s == 2
        numberOfDtrain = numberOfDtrain_500;
        Dtrain = Dtrain_500;
        DtrainLabels = DtrainLabels_500;
    else
        numberOfDtrain = numberOfDtrain_1000;
        Dtrain = Dtrain_1000;
        DtrainLabels = DtrainLabels_1000;
    end

    for c=1:numberOfClasses
        Ylabels(c,:)=DtrainLabels==c;
        YtestLabels(c,:)=DtestLabels==c;
    end

    %Split the data set to 10 block for 10 fold
    block = ceil(linspace(0, numberOfDtrain, F+1));

    for k = 1:F
        datasetBlock(k,:) = [block(k)+1,block(k+1)];
    end


    %Splite the data to 3 class, along with the 10 blocks for each class
    for c=1:numberOfClasses
        if c== 1
            index1 = find(Ylabels(c,:));
            class1Train = Dtrain(:,index1);
```

```
        [row, col] = size(class1Train);
        block1 = ceil(linspace(0, col, F+1));
        for k = 1:F
            datasetBlock1(k,:) = [block1(k)+1,block1(k+1)];
        end
    elseif c == 2
        index2 = find(Ylabels(c,:));
        class2Train = Dtrain(:,index2);
        [row, col] = size(class2Train);
        block2 = ceil(linspace(0, col, F+1));
        for k = 1:F
            datasetBlock2(k,:) = [block2(k)+1,block2(k+1)];
        end
    else
        index3 = find(Ylabels(c,:));
        class3Train = Dtrain(:,index3);
        [row, col] = size(class3Train);
        block3 = ceil(linspace(0, col, F+1));
        for k = 1:F
            datasetBlock3(k,:) = [block3(k)+1,block3(k+1)];
        end
    end
end


%Intialize likelihood for eval
likelihoodTrain = zeros(F, 6);
likelihoodValidate = zeros(F, 6);
Averagelltrain = zeros(numberOfClasses,6);
Averagellvalidate = zeros(numberOfClasses,6);

%1-6 Gaussian components
for c = 1:numberOfClasses
    if c == 1
        datasetBlock = datasetBlock1;
        x = class1Train;
    elseif c == 2
        datasetBlock = datasetBlock2;
        x = class2Train;
    else
        datasetBlock = datasetBlock3;
        x = class3Train;
    end


    for M = 1:6
        for k = 1:F
```

```
    [row, col] = size(x);
    N = col;
    % Assign validation and train set for each iteration
    validateIndex = [datasetBlock(k,1):datasetBlock(k,2)];

    if k == 1
      trainIndex = [datasetBlock(k, 2)+1:N];
    elseif k == F
      trainIndex = [1:datasetBlock(k, 1)-1];
    else
      trainIndex = [1:datasetBlock(k-1, 2), datasetBlock(k+1, 2):N];
    end

    xValidate = [x(1,validateIndex);x(2, validateIndex)];
    validateLen = length(validateIndex);

    xTrain = [x(1,trainIndex);x(2, trainIndex)];
    trainLen = length(trainIndex);


  % Initialize the GMM to randomly selected samples
  alpha = ones(1,M)/M;
  shuffledIndices = randperm(trainLen);
  mu = xTrain(:,shuffledIndices(1:M)); % pick M random samples as initial mean estimates
  [~,assignedCentroidLabels] = min(pdist2(mu',xTrain'),[],1); % assign each sample to the nearest
mean
  for m = 1:M % use sample covariances of initial assignments as initial covariance estimates
    Sigma(:,:,m) = cov(xTrain(:,find(assignedCentroidLabels==m))') + regWeight*eye(d,d);
  end
  t = 0; %displayProgress(t,x,alpha,mu,Sigma);

  Converged = 0; % Not converged at the beginning
  for i = 1:5000 %At least 100
    for l = 1:M
      temp(l,:) = repmat(alpha(l),1,trainLen).*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,trainLen);
    muNew = xTrain*w';
    for l = 1:M
      v = xTrain-repmat(muNew(:,l),1,trainLen);
      u = repmat(w(l,:),d,1).*v;
      SigmaNew(:,:,l) = u*v' + regWeight*eye(d,d); % adding a small regularization term
    end
    Dalpha = sum(abs(alphaNew-alpha'));
    Dmu = sum(sum(abs(muNew-mu)));
```

```
            DSigma = sum(sum(abs(abs(SigmaNew-Sigma))));
            Converged = ((Dalpha+Dmu+DSigma)<delta); % Check if converged
            if Converged
                break
            end
            alpha = alphaNew; mu = muNew; Sigma = SigmaNew;
            t = t+1;
            %displayProgress(t,xTrain,alpha,mu,Sigma);
        end
        likelihoodTrain(k,M) = sum(log(evalGMM(xTrain,alpha,mu,Sigma)));
        likelihoodValidate(k,M) = sum(log(evalGMM(xValidate,alpha,mu,Sigma)));
        end
        %Store the average likelihood in an matrix
        AverageTrain(c,M) = mean(likelihoodTrain(:,M));
        AverageValidate(c,M) = mean(likelihoodValidate(:,M));
        %Clear the values after each 10 fold
        clear Sigma
        clear mu
        clear alpha
        clear SigmaNew
        clear muNew
        clear alphaNew
    % If there is any inf number in likelihood, replace it with the
    % minimum likelihood in validation set
        if isinf(AverageValidate(c,M))
            AverageValidate(c,M) = (min(AverageValidate(c,(find(isfinite(AverageValidate(c,:)))))));
        end
    end
end

%Disp result
AverageValidate;


 fig = fig + 1;
 figure(fig), clf,
 b1 = bar(1:6, AverageValidate),
 title(strcat('Validation Log-Likelihoods for Dtrain size = ',num2str(numberOfDtrain))),
 xlabel('GMM Order'),
 ylabel(strcat('Log likelihood estimate')),
 legend('class1','class2', 'class3'),
 drawnow()

[val, testIdx] = max(AverageValidate,[], 2);

%Disp the best GMM model order for each class

BestGMM(s,:) = testIdx;
```

```
%create a struct to store the mean, covariance and prior
GMMStore = struct();


%Train with the whole Dtrain dataset and store the component for each classes
for c = 1:numberOfClasses
   if c == 1
      x = class1Train;
   elseif c == 2
      x = class2Train;
   else
      x = class3Train;
   end
   M = testIdx(c, 1);
   [row, col] = size(x);
   N = col;

   xTrain = x;
   trainLen = length(x);


% Initialize the GMM to randomly selected samples
   alpha = ones(1,M)/M;
   shuffledIndices = randperm(trainLen);
   mu = xTrain(:,shuffledIndices(1:M)); % pick M random samples as initial mean estimates
   [~,assignedCentroidLabels] = min(pdist2(mu',xTrain'),[],1); % assign each sample to the nearest mean
   for m = 1:M % use sample covariances of initial assignments as initial covariance estimates
      Sigma(:,:,m) = cov(xTrain(:,find(assignedCentroidLabels==m))') + regWeight*eye(d,d);
   end
   t = 0; %displayProgress(t,x,alpha,mu,Sigma);

   Converged = 0; % Not converged at the beginning
      for i = 1:10000 %At least 100
         for l = 1:M
            temp(l,:) = repmat(alpha(l),1,trainLen).*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,trainLen);
         muNew = xTrain*w';
         for l = 1:M
            v = xTrain-repmat(muNew(:,l),1,trainLen);
            u = repmat(w(l,:),d,1).*v;
            SigmaNew(:,:,l) = u*v' + regWeight*eye(d,d); % adding a small regularization term
         end
         Dalpha = sum(abs(alphaNew-alpha'));
```

```
        Dmu = sum(sum(abs(muNew-mu)));
        DSigma = sum(sum(abs(abs(SigmaNew-Sigma))));
        Converged = ((Dalpha+Dmu+DSigma)<delta); % Check if converged
        if Converged
            break
        end
        alpha = alphaNew; mu = muNew; Sigma = SigmaNew;
        t = t+1;
        %displayProgress(t,xTrain,alpha,mu,Sigma);
    end
%likelihoodTrain(k,M) = sum(log(evalGMM(xTrain,alpha,mu,Sigma)));
%likelihoodValidate(k,M) = sum(log(evalGMM(xValidate,alpha,mu,Sigma)));

    %Store GMM component here in GMMStore struct
    GMMStore(c).alpha = alpha;
    GMMStore(c).mu = mu;
    GMMStore(c).Sigma = Sigma;
    clear Sigma
    clear mu
    clear alpha
    clear SigmaNew
    clear muNew
    clear alphaNew
  end




  %Seperate the Dtest data and calculate the priors for each class
  for c=1:numberOfClasses
    if c== 1
        index1 = find(YtestLabels(c,:));
        class1Test = Dtest(:,index1);
        [row1, col1] = size(class1Test);
        p(c) = col1/numberOfDtest;
    elseif c == 2
        index2 = find(YtestLabels(c,:));
        class2Test = Dtest(:,index2);
        [row2, col2] = size(class2Test);
        p(c) = col2/numberOfDtest;
    else
        index3 = find(YtestLabels(c,:));
        class3Test = Dtest(:,index3);
        [row3, col3] = size(class3Test);
        p(c) = col3/numberOfDtest;
    end
```

```
   end


   %Store the GMM eval result for each class into GMMProbability array
   for i = 1:3
       alpha = GMMStore(i).alpha;
       mu = GMMStore(i).mu;
       Sigma = GMMStore(i).Sigma;
       GMMProbability(i, :) = evalGMM(Dtest,alpha,mu,Sigma).*p(i); %P(X|Thelta) * P(Thelta)
       Prior(s,i) = p(i);
   end

%Apply MAP to find which class has the highest probability result and
%compare with the test labels
[val, testIdx] = max(GMMProbability);
[val, labelIdx] = max(YtestLabels);
error = find(testIdx~=labelIdx);
errorP = size(error)/size(testIdx);
AccuracyP(s) = 1-size(error)/size(testIdx);
disp(["Final Accuracy GMM:", 1-errorP])
clear Ylabels
clear YtestLabels
clear block
clear datasetBlock
clear Sigma
clear mu
clear alpha
clear SigmaNew
clear muNew
clear alphaNew
end

 fig = fig +1;
 figure(fig), clf,
 X = categorical({'100','500','1000'});
 X = reordercats(X,{'100','500','1000'});
 b1 = bar(X, AccuracyP),
 title('Validation testing dataset on GMM model trained with 100, 500, 1000 samples of data'),
 ylabel('Percentage of correctly classified instances'),
 xlabel('Numbers of samples'),
 drawnow()
```

BestMLP

BestGMM

```
  Prior

  errorStore

  AccuracyP

% BestMLP =
%
%      1    10
%      3    10
%      3     8
%
%
% BestGMM =
%
%      3     1     2
%      3     3     6
%      3     4     6
%
%
% Prior =
%
%     0.3327    0.3297    0.3376
%     0.3327    0.3297    0.3376
%     0.3327    0.3297    0.3376
%
%
% errorStore =
%
%     0.5091    0.9602    0.9613
%
%
% AccuracyP =
%
%     0.9208    0.9670    0.9717
```

```
function objFncValue = objectiveFunction(type, X,Y,sizeParams,vecParams)
N = size(X,2); % number of samples
nX = sizeParams(1);
nPerceptrons = sizeParams(2);
nY = sizeParams(3);
params.A = reshape(vecParams(1:nX*nPerceptrons),nPerceptrons,nX);
params.b = vecParams(nX*nPerceptrons+1:(nX+1)*nPerceptrons);
params.C = reshape(vecParams((nX+1)*nPerceptrons+1:(nX+1+nY)*nPerceptrons),nY,nPerceptrons);
params.d = vecParams((nX+1+nY)*nPerceptrons+1:(nX+1+nY)*nPerceptrons+nY);
H = mlpModel(type, X,params);
```

```
objFncValue = sum(sum((Y-H).*(Y-H),1),2)/N;
%objFncValue = sum(-sum(Y.*log(H),1),2)/N;
% Change objective function to make this MLE for class posterior modeling
end

%
function H = mlpModel(type, X,params)
N = size(X,2);                          % number of samples
nY = length(params.d);                  % number of outputs
U = params.A*X + repmat(params.b,1,N);  % u = Ax + b, x \in R^nX, b,u \in R^nPerceptrons, A \in
R^{nP-by-nX}
Z = activationFunction(type, U);             % z \in R^nP, using nP instead of nPerceptons
V = params.C*Z + repmat(params.d,1,N);  % v = Cz + d, d,v \in R^nY, C \in R^{nY-by-nP}
%H = V; % linear output layer activations
H = exp(V)./repmat(sum(exp(V),1),nY,1); % softmax nonlinearity for second/last layer
% Add softmax layer to make this a model for class posteriors
%
end

function out = activationFunction(type, in)
if type == "sigmod"
   out = 1./(1+exp(-in)); % logistic function
elseif type == "ISRU"
   out = in./sqrt(1+in.^2); % ISRU
else
   out = log(1+exp(in));% Soft Plus
end
end

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 [x1Grid,x2Grid,zGMM] = contourGMM(alpha,mu,Sigma,rangex1,rangex2)
x1Grid = linspace(floor(rangex1(1)),ceil(rangex1(2)),101);
x2Grid = linspace(floor(rangex2(1)),ceil(rangex2(2)),91);
[h,v] = meshgrid(x1Grid,x2Grid);
GMM = evalGMM([h(:)';v(:)'],alpha, mu, Sigma);
zGMM = reshape(GMM,91,101);
%figure(1),
contour(horizontalGrid,verticalGrid,discriminantScoreGrid,[minDSGV*[0.9,0.6,0.3],0,[0.3,0.6,0.9]*maxDS
GV]); % plot equilevel contours of the discriminant function
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 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
```