MSc Human and Biological Robotics

Author:
Edward McLaughlin

Lecturer:
Dr. Aldo *Faisal*

Machine Learning and Neural Computation: Human Activity Recognition

Department of Bioengineering

December 3, 2017

1 Question 1: MLP Classification

Part a: Figure 1 shows the learning curve of the Neural Network (NN) classifier with one hidden layer of five neurons for various learning rates. The dashed lines correspond to simulations with the decreasing learning rate parameter disabled. Throughout the report the data is plotted as smoothed data over 50 data points to make the figures clearer. This smoothing is validated in figure 2(b) where the blue dashed line is the raw data and the blue solid line is the smoothed data. For simulations with decreased learning rate switched on, the decrease value was calculated as 0.1% of the initial learning rate so that the decrease was not absolute for all simulations. The learning rate can be interpreted as the step size in a gradient decent. By thinking of the learning rate in this manner, a learning rate of 0.0005 is not large enough to ensure the classifier tends to the global minima. Increasing the learning rate to 0.01 has better results but the step size might be too large to reach the bottom of the global minima (this could be rectified by having a more aggressive decreasing learning rate after each epoch). Thus a learning rate in the middle (0.001) gives the best results. In general, we can see that implementing a decreasing learning rate improves the accuracy of the NN classifier. This is because as the classifier tends to the global minima, the reduced step size allows the classifier to reach the minima with much higher precision. The exception to this is where the learning rate is already too small (0.0005) to reach the global minima, thus reducing the learning rate merely amplifies this problem.

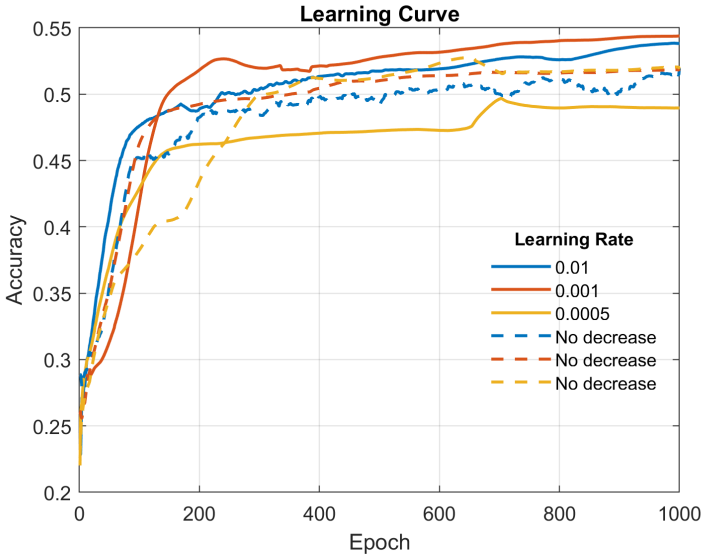
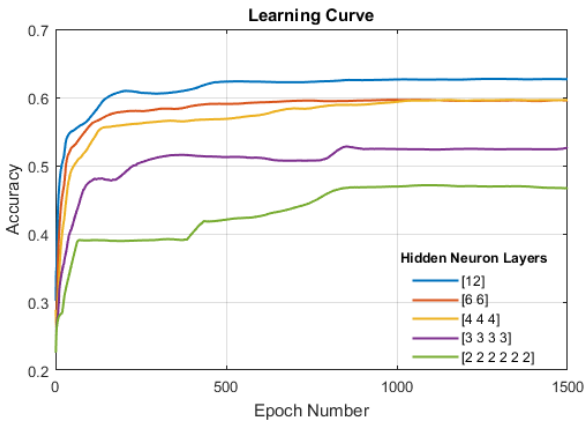
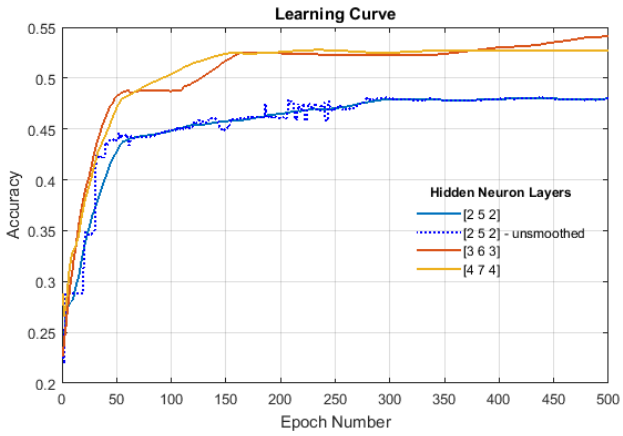


Figure 1: Learning curve for various learning rates. Hidden Neuron Layers: [5]. Dashed lines show the learning curve with no decrease in learning rate.



(a) Comparison of 12 neurons split over various numbers of layers.



(b) Comparison of varying numbers of neurons over three layers.

Figure 2: Plots showing the effects of changing the structures of the hidden neuron layers.

Part b: Figure 2 shows the effects of changing the hidden neuron layers. For this problem, splitting the same number of nodes over multiple layers reduces the accuracy of the solution. This might be because having more nodes in a single layer increases the complexity of that layer more than adding layers increases the complexity of the entire network. This higher complexity allows the network to be more flexible in finding the optimum weights to increase the classification accuracy. Furthermore, the number of bias nodes in the network increases as you increase the number of layers with the same total number of neurons. Figure 2(b) shows the effects of increasing the number of neurons over a given number of hidden neuron layers. It can be seen that increasing the number of nodes increases the accuracy of the model up to a certain point, at which point the increase in

nodes (and computational cost) has diminishing returns on the accuracy of the NN.

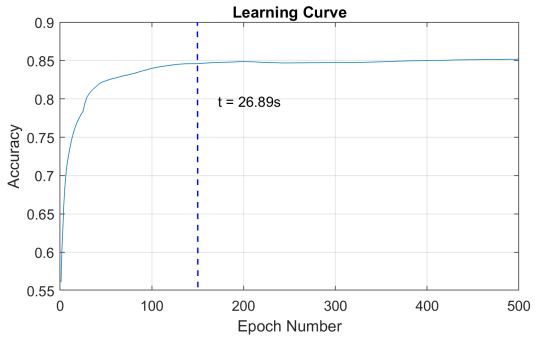
Part c: Based on the analysis above, the parameters given in table 1 were chosen as optimum. A single layer of 12 nodes with a learning rate steadily decreasing from 0.001 was deemed an appropriate compromise to ensure good accuracy and acceptably computing time. While computing time depends on computing power, the number of epochs to reach maximum accuracy is a good measure speed to final solution. Twelve nodes gives a steady accuracy after approximately 150 epochs. Table 1 also shows the confusion matrix for the classifier. In general, it appears that for this dataset distinguishing walking or running is much easier than walking upstairs or walking downstairs.

The number of weights for a single hidden neuron layer with 12 nodes (and one bias node in the input and hidden layers) is 100 (4x12 + 13x4).

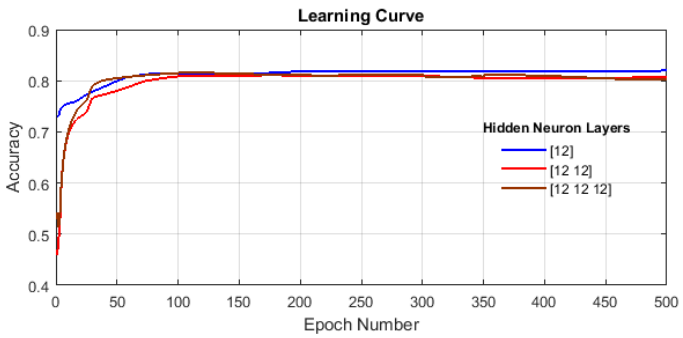
Table 1: Selected parameters for NN classifier

	Learning Rate	Decrease in Learning Rate	Hidden Neuron Layers	Layer Structure	Confusion Matrix
NN Parameters	0.001	On	1	[12]	$\begin{bmatrix} 0.73 & 0.14 & 0.19 & 0.19 \\ 0.09 & 0.71 & 0.09 & 0.07 \\ 0.09 & 0.05 & 0.55 & 0.30 \\ 0.09 & 0.09 & 0.17 & 0.44 \end{bmatrix}$

2 Question 2: Bimodal Classification



(a) Learning curve for one layer of 12 nodes and the time taken to reach 150 epochs



(b) Camparison of multiple layers with 12 nodes per layer.

Figure 3

The performance for NN classification by nature varies with each simulation. With this in mind, the performance figures given for NN classification were taken as averages over five simulations. The two classes taken to be considered were 'running' and 'walking upstairs' since, from the confusion matrix in table 1, these two classes show two have the highest relative accuracy. The accuracy after 500 epochs, run time to 150 epochs and confusion matrix were 0.8522, 26.89s and $\begin{bmatrix} 0.86 & 0.15 \\ 0.14 & 0.85 \end{bmatrix}$ respectively. The accuracy after 150 epochs is similar to that after 500 epochs. Once more the impact of adding extra layers was assess. As seen in figure 3(b), this change reduces the 'settling time' to maximum accuracy, however, the final accuracy is marginally worse. These results are appreciably better than for four class classification. This is expected as the two class classification is a simpler problem to solve.

3 Question 3-5: K-NN and Gaussian Mixed Model (GMM)

Question 3 The training and classification was implemented as in thecode given in the appendix.

Question 4 Two classifiers, K-nearestneighbours (K-NN) and Gaussian Mixed Model (GMM), were implemented. K-means was discarded as it is a simplified version of GMM where the variance matrix tends to 0 in all directions, giving point clusters. The K-NN method consists of assigning each test data point the mode

label of the K closest training points. In this case, the number of neighbours was chosen as five as it was the number at which accuracy began to plateau. Using this method demands no explicit training of the model as the classification is made directly using training data. The GMM method trains the classifier using Estimation Maximisation (EM) for multiple GDs per class. In this case, the number of GDs used per class was two as this gave sufficiently accurate results for acceptable computation time. Furthermore, it can be seen in figure 4 that to produce maximum likelihood, the two walking upstairs GDs are in a similar spatial region. Hence, adding an extra is unlikely to increase accuracy remarkably as this region is already well mapped. The probability that each testing data point belongs to each GD is then calculated and summed over each class. The test data point is deemed to have the label corresponding to the GDs with the highest combined probability of producing it.

The results of the two classifiers are compared in table 2 for each combination of classes. The average to computing time across all pairs of movement type for the K-NN classifier was 18.1s, while the GMM algorithm faster at 12.3s. The accuracy for the chosen two classes, running and walking upstairs, for KNN and GMM classifiers were 0.8569 and 0.8375 respectively. The K-NN classifier results were marginally more accurate at the cost of considerably longer running time than GMM, which also achieved acceptable levels of accuracy. As a result, the GMM classifier was submitted for the coursework.

Table 2: Comparison of K-NN and GMM classification

Classifier	Walking/ Running	Walking/ Upstairs	Walking/ Downstairs	**Running/ Upstairs**	Running/ Downstairs	Upstairs/ Downstairs
K-NN	$\begin{bmatrix} 0.92 & 0.17 \\ 0.08 & 0.83 \end{bmatrix}$	$\begin{bmatrix} 0.85 & 0.22 \\ 0.15 & 0.78 \end{bmatrix}$	$\begin{bmatrix} 0.85 & 0.22 \\ 0.15 & 0.78 \end{bmatrix}$	$\begin{bmatrix} \mathbf{0.85} & \mathbf{0.14} \\ \mathbf{0.15} & \mathbf{0.86} \end{bmatrix}$	$\begin{bmatrix} 0.82 & 0.12 \\ 0.18 & 0.88 \end{bmatrix}$	$\begin{bmatrix} 0.66 & 0.32 \\ 0.34 & 0.67 \end{bmatrix}$
GMM	$\begin{bmatrix} 0.80 & 0.22 \\ 0.20 & 0.78 \end{bmatrix}$	$\begin{bmatrix} 0.69 & 0.17 \\ 0.31 & 0.83 \end{bmatrix}$	$\begin{bmatrix} 0.79 & 0.21 \\ 0.21 & 0.79 \end{bmatrix}$	$\begin{bmatrix} \mathbf{0.81} & \mathbf{0.13} \\ \mathbf{0.19} & \mathbf{0.87} \end{bmatrix}$	$\begin{bmatrix} 0.81 & 0.11 \\ 0.19 & 0.89 \end{bmatrix}$	$\begin{bmatrix} 0.71 & 0.44 \\ 0.29 & 0.56 \end{bmatrix}$

The GMM classifier uses two GDs per class - four GDs in total. Each GD is defined by its position (μ - 1x3 position vector) and covariance matrix (Σ - 3x3 symmetric matrix). From this it can be deduced that each matrix is defined by 10 parameters (three for μ, six for Σ and one to define its class), thus the total number of parameters needed for this classifier is 40.

Figure 4 shows a scatter plot of the test data classified by the GMM classifier. The accuracy of the solution can be seen by the hard visual boundary between the colours (few purple hues).

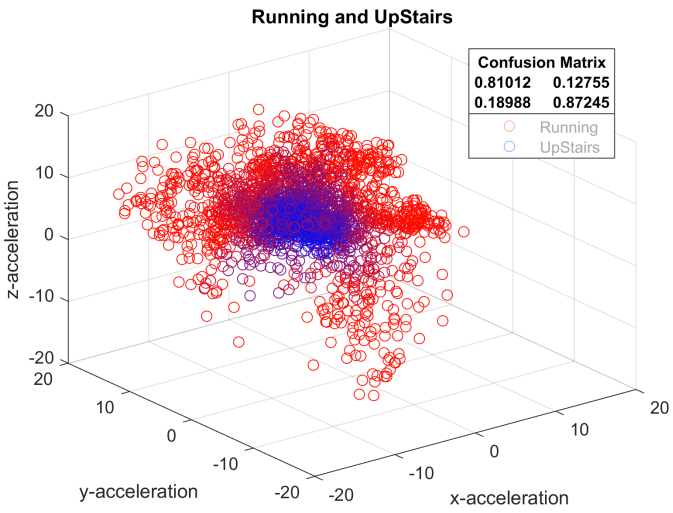


Figure 4: figure
Scatter plot showing the GMM classification of the test data for Running and Upstairs.

Question5 The confusion matrix of the GMM classifier is shown in figure 4. Comparing this to the NN confusion matrix (shown in Question 2), it can be seen that their performance is similar. However, the accuracy for NN is slightly higher than for GMM, 0.8522 and 0.8375 respectively, and the spread of the diagonal terms in the confusion matrix about the overall accuracy for NN classification is much lower - 0.005 compared with 0.030. That being said, the GMM classifier is considerably faster at reaching a solution for the running and walking upstairs classification - 5.89s compared with 26.89s for NN. One further advantage of GMM is that its results indicate mixed membership of labels i.e. for this data, if a data point is 40% running and 60% walking upstairs, the activity may have been walking fast upstairs. This makes the output data from GMM much richer and a better classifier when the desired classification labels are continuous as oppose to discrete. NN and K-NN handle data in a discrete manner and thus are more relevant to discrete classification problems. Hence, since the performance for both was acceptable, but the GMM ran considerably quicker and gives much richer information, the GMM trainer and classifier is preferable over NN in the instance.

Appendix

Main Script

```
1
2  load activities.mat
3
4  % labels - ID of the activity
5  % 1 - walking
6  % 2 - running
7  % 3 - walking upstairs
8  % 4 - walking downstairs
9
10 class1 = input('Class 1');
11 class2 = input('Class 2');
12
13 if class1 == 1
14     CLASS1 = 'Walking';
15 elseif class1 == 2
16     CLASS1 = 'Running';
17 elseif class1 == 3
18     CLASS1 = 'UpStairs';
19 elseif class1 == 4
20     CLASS1 = 'DownStairs';
21 end
22
23 if class2 == 1
24     CLASS2 = 'Walking';
25 elseif class2 == 2
26     CLASS2 = 'Running';
27 elseif class2 == 3
28     CLASS2 = 'UpStairs';
29 elseif class2 == 4
30     CLASS2 = 'DownStairs';
31 end
32
33 %% Create Data Split for Two Labels.
34 [train_bimodal, train_bimodal_labels, test_bimodal, test_bimodal_labels]
       = ...
35     bimodalData(train_data, train_labels, test_data, test_labels, class1,
           class2);
36
37 %% K-NN Classifier
38 % parameters.train_data = train_bimodal;
39 % parameters.train_labels = train_bimodal_labels;
40 % parameters.class1 = CLASS1;
41 % parameters.class2 = CLASS2;
42 %for i = 1:length(test_bimodal)
43 %     class(i,1) = ClassifyX1(test_bimodal(i,:), parameters);
44 %end
45
46 %% Gaussian Mixture Model
47 parameters = TrainClassifierX(train_bimodal, train_bimodal_labels);
```

```
parameters.class1 = CLASS1;
parameters.class2 = CLASS2;
for i = 1:length(test_bimodal)
    class(i,1) = ClassifyX(test_bimodal(i,:), parameters);
end

%% Accuracy Calculator & Plotter

accuracy = sum(sum(class == test_bimodal_labels, 2) == 1)/length(
    test_bimodal_labels);

count11 = 0;
count12 = 0;
count21 = 0;
count22 = 0;
num1 = length(test_bimodal_labels(test_bimodal_labels == 1));
num2 = length(test_bimodal_labels(test_bimodal_labels == 2));
for i = 1:length(test_bimodal)
    if class(i,1) == 1 && test_bimodal_labels(i,1) == 1
        count11 = count11 + 1;
        CM(1,1) = count11 / num1;
    elseif class(i,1) == 1 && test_bimodal_labels(i,1) == 2
        count12 = count12 + 1;
        CM(1,2) = count12 / num2;
    elseif class(i,1) == 2 && test_bimodal_labels(i,1) == 1
        count21 = count21 + 1;
        CM(2,1) = count21 / num1;
    else
        count22 = count22 + 1;
        CM(2,2) = count22 / num2;
    end
end
descr = {'Confusion Matrix';
    num2str(CM)};
title(legend, descr)
sprintf('Class %s and class %s, give an accuracy of...', CLASS1, CLASS2)
CM
accuracy
```

3.1 Organising Data in to two classes

```
function [train_bimodal, train_bimodal_labels, test_bimodal,
    test_bimodal_labels] = bimodalData(train_data, train_labels, test_data
    , test_labels, group1, group2)

%extrac all the points and labels of the given classes from test and
    train
%datasets
bimodal_train_index = find(train_labels == group1 | train_labels ==
    group2);
train_bimodal = train_data(bimodal_train_index, :);
train_bimodal_labels = train_labels(bimodal_train_index);
bimodal_test_index = find(test_labels == group1 | test_labels == group2);
test_bimodal = test_data(bimodal_test_index, :);
```

```
10 test_bimodal_labels = test_labels(bimodal_test_index);
11
12 %reallcoate the class number 1 and 2 to the chosen classes for train
13 %dataset
14 minTB = min(train_bimodal_labels);
15 index_minTB = find(train_bimodal_labels == minTB);
16 train_bimodal_labels(index_minTB) = train_bimodal_labels(index_minTB) -
       minTB + 1;
17 maxTB = max(train_bimodal_labels);
18 index_maxTB = find(train_bimodal_labels == maxTB);
19 train_bimodal_labels(index_maxTB) = train_bimodal_labels(index_maxTB) -
       maxTB + 2;
20
21 %reallcoate the class number 1 and 2 to the chosen classes for test
22 %dataset
23 minTestB = min(test_bimodal_labels);
24 index_minTestB = find(test_bimodal_labels == minTestB);
25 test_bimodal_labels(index_minTestB) = test_bimodal_labels(index_minTestB)
       - minTestB + 1;
26 maxTestB = max(test_bimodal_labels);
27 index_maxTestB = find(test_bimodal_labels == maxTestB);
28 test_bimodal_labels(index_maxTestB) = test_bimodal_labels(index_maxTestB)
       - maxTestB + 2;
29
30 end
```

3.2 Gaussian Mixture Model

3.2.1 TrainClassifierX

```
1
2 function parameters = TrainClassifierX(input,output)
3
4 % Split data into two classes
5 maxclass = max(output(:,1));
6 minclass = min(output(:,1));
7 category{1} = input(output(:,1) == minclass,:);
8 category{2} = input(output(:,1) == maxclass,:);
9
10 % Place down x number of clusters for each category
11 x = 2;
12 mu{1}=[1 1 1;...
13       0 5 10];
14 mu{2}=[1 1 1;...
15       0 5 10];
16
17 % For each class, use k-means to appropriately cluster the 2 gaussians.
18 for k = 1:2
19 closestCluster = ones(length(category{k}),1);
20 muChange{k}(1:x) = ones(1,x);
21     while sum(muChange{k}(1:end)) > 0.001 %convergence createria for k-
         means
22         for i = 1:length(category{k})
23             for ii = 1:x
```

```
24                  dist2cluster(ii) = sqrt(sum((category{k}(i,:) - mu{k}(ii
                        ,:)).^2));
25              end
26              [~, mindistindex] = min(dist2cluster(:));
27              closestCluster(i) = mindistindex;
28          end
29          for i = 1:x
30              index_data = [];
31              index_data = find(closestCluster == i);
32              cluster_data{k}{i} = zeros(length(index_data),3);
33              cluster_data{k}{i}(:,1:3) = category{k}(index_data, 1:3);
34              muOLD{k}(:,1:3) = mu{k};
35              mu{k}(i,1:3) = [mean(cluster_data{k}{i}(:,1)), mean(
                    cluster_data{k}{i}(:,2)), mean(cluster_data{k}{i}(:,3))];
36              muChange{k}(i) = sqrt(sum((mu{k}(i,:) - muOLD{k}(i,:)).^2));
37              parameters.mu{k}(:,:) = mu{k}(:,:);
38              sigmaGauss{k}(:,:,i) = cov(cluster_data{k}{i}(:,:)); %
                    calulate gaussian covar
39          end
40      end
41  end
42
43  for k = 1:2 %consider each class one at a time
44      muChange{k}(1:x) = ones(1,x); %reset the initial muChange for the new
            class
45      while sum(muChange{k}(1:end)) > 0.001 %convergence criteria
46          muOLD{k} = parameters.mu{k};
47          %taking each guassian one at a time
48          %calculate the probability each that of the data points belongs
49          %to the given guassian
50          for ii = 1:x
51              for i = 1:length(category{k})
52                  P{k}(i,ii) = 1./((2*pi)^(3/2)*det(sigmaGauss{k}(:,:,ii))
                        ^(1/2)) * ...
53                      exp(-0.5*(category{k}(i,:) - parameters.mu{k}(ii,:))*
                            inv(sigmaGauss{k}(:,:,ii))*(category{k}(i,:) -
                            parameters.mu{k}(ii,:))');
54              end
55          end
56          %normalise the probability
57          for ii = 1:x
58              probdata2Gauss{k}(:,ii) = [P{k}(:,ii)./sum(P{k}(:,:),2)];
59          end
60          %find the gaussian the data point most likely belongs to
61          [~, maxprobindex] = max(probdata2Gauss{k}(:,:),[],2);
62          %calculate all the data which belong to each gaussian
63          for ii = 1:x
64              thisGauss = find(maxprobindex == ii);
65              %calculate the new gaussian positions and covar matrices
66              for j = 1:3
67                  parameters.mu{k}(ii,j) = probdata2Gauss{k}(thisGauss,ii)
                        '*category{k}(thisGauss,j)/sum(probdata2Gauss{k}(
                        thisGauss,ii));
```

```
                weightedPos{k}(:,j) = probdata2Gauss{k}(thisGauss,ii).*
                    category{k}(thisGauss,j);
            end
            sigmaGauss{k}(:,:,ii) = cov(weightedPos{k}(:,:));
            muChange{k}(ii) = sqrt(sum((parameters.mu{k}(ii,:) - muOLD{k
                }(ii,:)).^2));
            weightedPos{k} = [];
        end
    end
end
parameters.sigma = sigmaGauss;
end
```

3.2.2 ClassifierX

```
function class = ClassifyX(input, parameters)
%% Gaussian Mixed Model Classification

x = 2;
% For all test points, calculate the likelihood it belongs to each
% Gaussian
for i = 1:size(input,1)
    for k = 1:2
        for ii = 1:length(parameters.mu{k}(:,1))
            prob2Gauss(k,ii,i) = 1./((2*pi)^(3/2)*det(parameters.sigma{k
                }(:,:,ii))^(1/2)) * ...
                    exp(-0.5*(input(i,:) - parameters.mu{k}(ii,:))...
                    *inv(parameters.sigma{k}(:,:,ii))*(input(i,:) -
                        parameters.mu{k}(ii,:))');
        end
    end
end

%normalise the probability
for ii = 1:x
    for i = 1:size(input,1)
        normProb2Gauss(1:2,ii,i) = [prob2Gauss(1,ii,i)./(x*(prob2Gauss(1,
            ii,i)+prob2Gauss(2,ii,i))),...
                                    prob2Gauss(2,ii,i)./(x*(prob2Gauss(1,
                                        ii,i)+prob2Gauss(2,ii,i)))];
    end
end

% For each test point calculate the set of Gaussians (i.e. the class) it
    most likely belongs to and
% designate the label of the Gaussian to it.
for i = 1size(input,1)
    [~, Gaussindex] = max(sum(prob2Gauss(:,:,i),2));
    class(i,1) = Gaussindex(1,1);
end

%plot the test data with relevant colour representing the probability
    they
%belong to each class
```

```
% figure
% h = plot3(1,1,1,'ro');
% hold on
% h1 = plot3(1,1,1,'bo');
% set(h,'Visible','off')
% set(h1,'Visible','off')
% xlabel('x')
% ylabel('y')
% zlabel('z')
% for i = 1:size(input,1)
%     colour = sum(normProb2Gauss(1:2,1:end,i),2);
%     redcolour = colour(1,1);
%     bluecolour = colour(2,1);
%     plot3(input(i,1),input(i,2),input(i,3), 'o', 'MarkerEdgeColor', [
    redcolour 0.05 bluecolour])
%     grid on
% end
% hold off
% title([parameters.class1, ' and ', parameters.class2])
% lgd = legend([h, h1],{[parameters.class1], [parameters.class2]},'
    TextColor','k','location', 'best');

end
```

3.3 K Nearest Neighbours

```
function class = ClassifyX1(input, parameters)
%% K-NearestNeighbours

% Initialise counts for test data labels and total number of each label
    in training data
NN = 5;

for i = 1:length(input) %For all test data
    for ii = 1:length(parameters.train_data) %For all training data
        distance2neighbours(i) = sqrt(sum((input(i,:) - parameters.
            train_data(ii,:)).^2)); %distance from test to training data
            %if the closest neighbours isn't fully populated with NN
                entries, this is one of the NN closest neighbours
            if ii <= NN
                closestNeighbours(ii,i) = distance2neighbours(i);
                closestNeighbours_label(ii,i) = parameters.train_labels(
                    ii);
                test_labels(i) = mode(closestNeighbours_label(:,i),1);
            %elseif the distance is less than the maximum of the current
            %NN, replace the point and its corresponding label
            elseif distance2neighbours(i) < max(closestNeighbours(1:NN,i)
                )
                [replace, replaceIndex] = max(closestNeighbours(1:NN,i));
                closestNeighbours(replaceIndex,i) = distance2neighbours(i
                    );
                closestNeighbours_label(replaceIndex,i) = parameters.
                    train_labels(ii);
                test_labels(i) = mode(closestNeighbours_label(:,i),1);
```

```
22                 end
23             end
24 end
25
26 %Plotting
27 figure
28 title([parameters.class1, ' and ', parameters.class2])
29 h = plot3(1,1,1,'ro');
30 hold on
31 h1 = plot3(1,1,1,'bo');
32 set(h,'Visible','off')
33 set(h1,'Visible','off')
34 xlabel('x')
35 ylabel('y')
36 zlabel('z')
37 for i = 1:length(input)
38     if test_labels(i) == 1
39         plot3(input(i,1),input(i,2),input(i,3), 'o', 'MarkerEdgeColor',
               [1 0 0])
40     else
41         plot3(input(i,1),input(i,2),input(i,3), 'o', 'MarkerEdgeColor',
               [0 0 1])
42     end
43 end
44 grid on
45 hold off
46 title([parameters.class1, ' and ', parameters.class2])
47 lgd = legend([h, h1],{[parameters.class1], [parameters.class2]},'
       TextColor','k','location', 'best');
48
49 class = test_labels';
50 end
```