**Homework 3 Solutions**

Zezhou Li

10405265

**Problem 1:**

As given that a Gaussian Mixture Model (GMM) with two mixtures is expressed as:

To use the Expectation-Maximization method, we need to initialize all the parameters, here we assume , from the Maximum Likelihood method we can calculate the initial values for and .

Given the data samples, based on the above equations, we can get and using Matlab:

|  |
| --- |
| dimension = size(hw3\_1, 2) / 2;    sum1\_x = sum(hw3\_1(:,1:50), 2);  u1\_hat = sum1\_x / dimension;  temp = zeros(size(hw3\_1, 1));  for i = 1:dimension  temp = temp + (hw3\_1(:,i)-u1\_hat)\*(hw3\_1(:,i)-u1\_hat)';  end  delta21\_hat = temp / dimension;    sum2\_x = sum(hw3\_1(:,51:end), 2);  u2\_hat = sum2\_x / dimension;  temp = zeros(size(hw3\_1, 1));  for i = (dimension+1):(dimension\*2)  temp = temp + (hw3\_1(:,i)-u2\_hat)\*(hw3\_1(:,i)-u2\_hat)';  end  delta22\_hat = temp / dimension; |

We will get that the initial and are:

Using the above results as the initial values to do EM calculation.

The aim is to estimate the unknown parameters representing the "mixing" value between the Gaussians and the means and covariances of each:

Using Matlab to implement the EM Algorithm, there is a Matlab scripts of EM Algorithm done by others already, I simply transform them into my codes which will meet my own need.

First is the Expectation step script:

|  |
| --- |
| function [R, llh] = expectation(X, model)  mu = model.mu;  Sigma = model.Sigma;  w = model.w;    n = size(X,2);  k = size(mu,2);  R = zeros(n,k);  for i = 1:k  R(:,i) = loggausspdf(X,mu(:,i),Sigma(:,:,i));  end  R = bsxfun(@plus,R,log(w));  T = logsumexp(R,2);  llh = sum(T)/n; % loglikelihood  R = exp(bsxfun(@minus,R,T));  end |

In this step, the function calculates the log Gaussian pdf of the data with current mean and covariance for each class. Then it calculates the log likelihood of the result ‘membership probability’. This will be used later in Maximization step and the Termination step.

Second is the Maximization step script:

|  |
| --- |
| function model = maximization(X, R)  [d,n] = size(X);  k = size(R,2);  nk = sum(R,1);  w = nk/n;  mu = bsxfun(@times, X\*R, 1./nk);    Sigma = zeros(d,d,k);  r = sqrt(R);  for i = 1:k  Xo = bsxfun(@minus,X,mu(:,i));  Xo = bsxfun(@times,Xo,r(:,i)');  Sigma(:,:,i) = Xo\*Xo'/nk(i)+eye(d)\*(1e-6);  end    model.mu = mu;  model.Sigma = Sigma;  model.w = w;  end |

In this step, the function uses the result coming from Expectation step to recalculate new parameters of the model.

Finally in the main script, write a iterative block to implement these two functions and also add a termination to check in every iteration if it is enough to terminate the iteration.

|  |
| --- |
| x = hw3\_1;  n = size(x, 2);  tol = 1e-6;  maxiter = 500;  llh = -inf(1,maxiter);  model.mu = [u1\_hat,u2\_hat];  model.Sigma(:,:,1) = delta21\_hat;  model.Sigma(:,:,2) = delta22\_hat;  model.w = [0.5 0.5];    for iter = 2:maxiter  disp(['iteration step: ',num2str(iter)]);  [R, llh(iter)] = expectation(x,model);  [~,label(1,:)] = max(R,[],2);  R = R(:,unique(label)); % remove empty clusters  model = maximization(x,R);  if abs(llh(iter)-llh(iter-1)) < tol\*abs(llh(iter)); break; end;  end  llh = llh(2:iter); |

After doing these, we can run the main script to see the results.

It rans 18 iterations before termination. Then we plot the finally classification using the model we get.

|  |
| --- |
| figure(1);  plot(llh);  figure(2);  plotClass(x,label); |

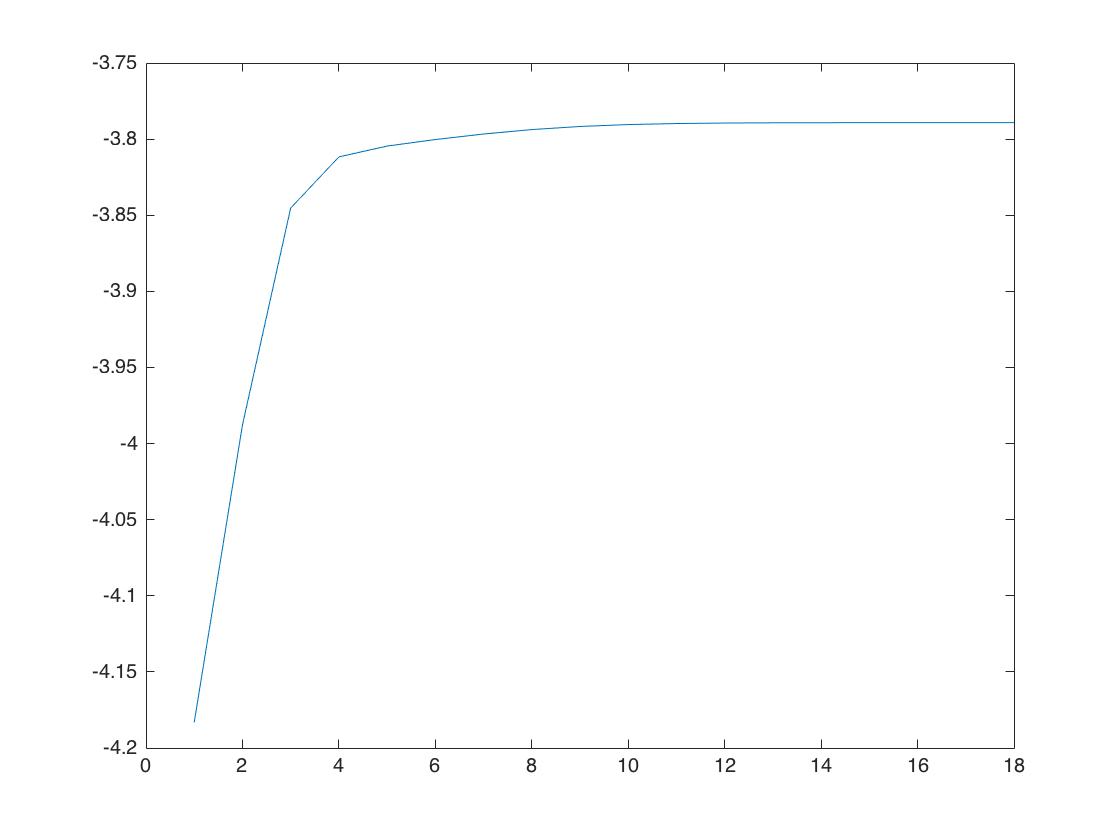


Figure 1. Log Likelihood of every iteration.

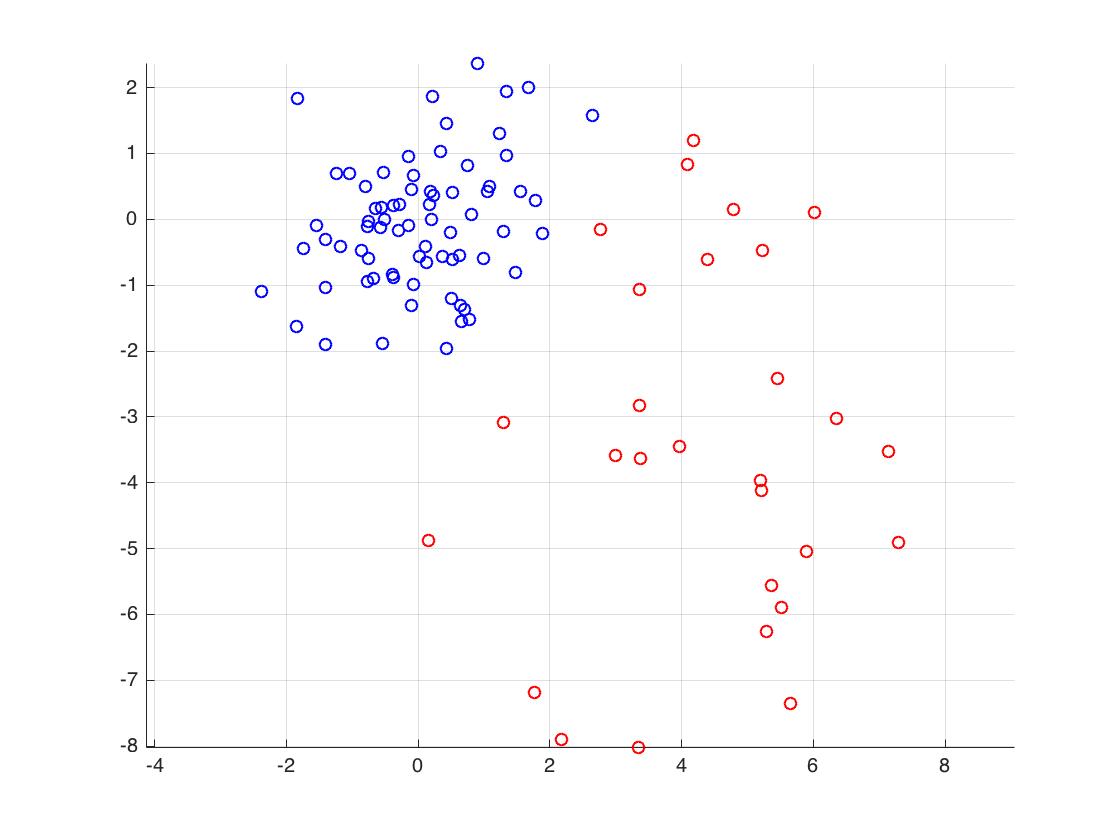


Figure 2. Final classification of EM algorithm.

From the figure we can see that the final classification is good enough after 18 iterations. Thus the EM algorithm is successful.

Then we can get the final parameters from Matlab:

|  |
| --- |
| mu1 = model.mu(:,1)  mu2 = model.mu(:,2)  Sigma1 = model.Sigma(:,:,1)  Sigma2 = model.Sigma(:,:,2)  w = model.w |

|  |
| --- |
| mu1 =  0.0173  -0.0764  mu2 =  4.2363  -3.3269  Sigma1 =  1.0016 0.2617  0.2617 0.9437  Sigma2 =  3.1706 -0.0148  -0.0148 7.3567  w =  0.7111 0.2889 |

Thus the final estimation parameters are:

**Problem 2:**

1. From the lecture, we know that we need a parzen window, which is a normal distribution, from the given data, we also know that we should initialize the parzen window to be a multivariate normal distribution since the dimension of given data is 2.

First step is to calculate p(x|w) for two classes. The equation is given as the following:

Here, , where is chosen to be 2.

Using Matlab to do the calculation:

|  |
| --- |
| clear;  clc;  load('/Users/zezhouli/Documents/2016-2017 Fall Semester/CPE646-Pattern Recognition/Assignments/Assignment 3/hw3.mat');  data1 = hw3\_2\_1;  data2 = hw3\_2\_2;  x1 = -4:0.1:8;  x2 = -4:0.1:8;  h1 = 2;  n = size(data1, 2);  hn = h1 / sqrt(n);  l = 0;  for i = -4:0.1:8      l = l+1;      m=0;      for j = -4:0.1:8          m=m+1;          p1(l,m) = 0;          p2(l,m) = 0;          for k = 1:n              xd1 = [i;j] - data1(:,k);              xd2 = [i;j] - data2(:,k);              xd1 = xd1(1)^2+xd1(2)^2;              xd2 = xd2(1)^2+xd2(2)^2;              p1(l,m) = p1(l,m) + 1/(n\*hn\*sqrt(2\*pi))\*exp(-xd1/(2\*hn^2));              p2(l,m) = p2(l,m) + 1/(n\*hn\*sqrt(2\*pi))\*exp(-xd2/(2\*hn^2));          end          if i==1 && j==-2              p1x = p1(l,m)              p2x = p2(l,m)          end      end  end  figure(1);  mesh(p1);  figure(2);  mesh(p2); |

After calculation, we use ‘mesh’ function to plot the result, as shown below.

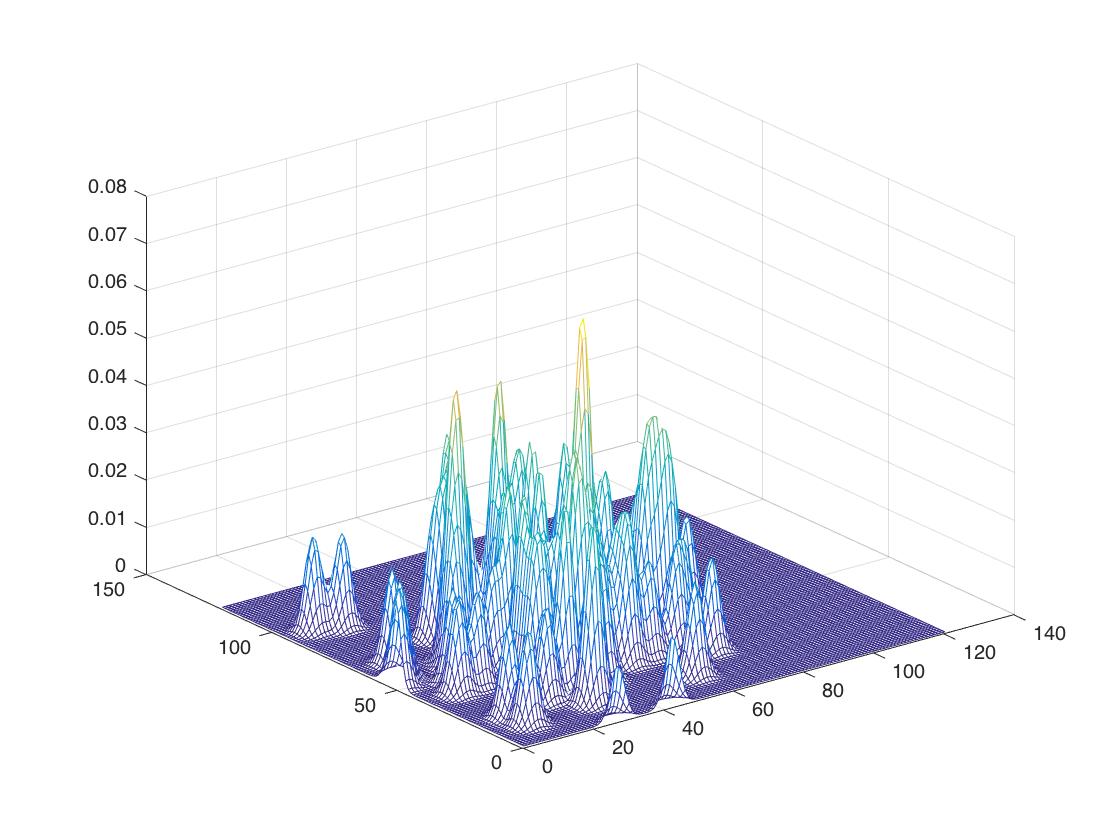


Figure 3. p(x|w1) figure

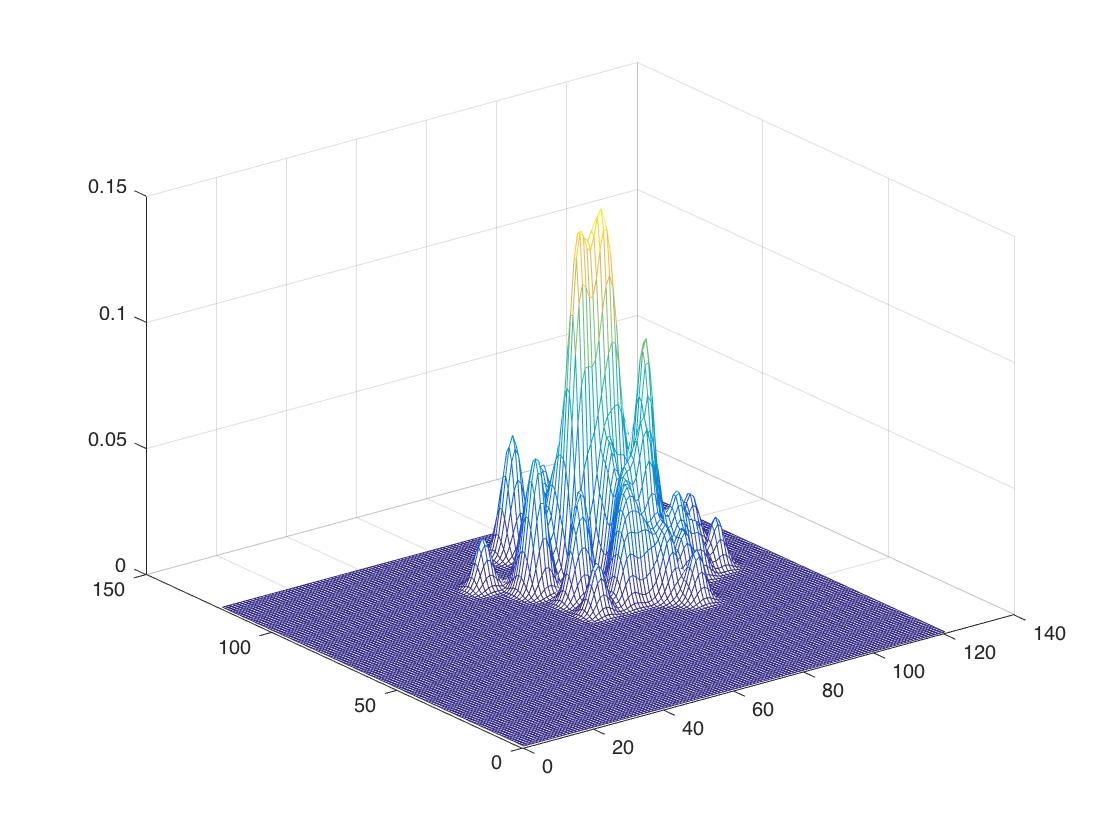


Figure 4. p(x|w2) figure

Then given a point ***x*=[1,-2]*t***, we can get the class conditional probability from the results we got:

As we know from Bayes formula that

Here , thus we classify the test data to the class with bigger .

In this case, , so we classify data to .

2. To implement the algorithm for PNN, I simply follow the steps in the textbook.

First is to train the PNN, which is like the algorithm 1 shown in the book. Two main steps are normalization and training.

Each pattern x of the training set is normalized to have unit length. Then modifiable weights linking the input units and the first pattern unit are set.

|  |
| --- |
| for j = 1:n      data1\_new(1,j) = data1(1,j) / sqrt(data1(1,j)^2 + data1(2,j)^2);      data1\_new(2,j) = data1(2,j) / sqrt(data1(1,j)^2 + data1(2,j)^2);      data2\_new(1,j) = data2(1,j) / sqrt(data2(1,j)^2 + data2(2,j)^2);      data2\_new(2,j) = data2(2,j) / sqrt(data2(1,j)^2 + data2(2,j)^2);  end  wk1 = data1\_new;  wk2 = data2\_new; |

After getting the weights, we then give the new input x=[1;-2]; Using equation (29) and calculate the summation of g(x), finally compare two g(x) of each class and classify the new input to the class with bigger g(x).

|  |
| --- |
| x = [1;-2];  x = [x(1)/sqrt(x(1)^2+x(2)^2);x(2)/sqrt(x(1)^2+x(2)^2)];    sigma = 0.2;  net1 = x' \* wk1;  net2 = x' \* wk2;    g1 = sum(exp((net1-1)/sigma^2))  g2 = sum(exp((net2-1)/sigma^2)) |

The most important thing in this last step is to normalize the new input similar to that in the training step.

Finally, we get the values of g(x):

Thus, we classify x=[1;-2] to class 1.