**Homework 4 Solutions**

Zezhou Li

10405265

**Problem 1:**

To use k-nearest neighbor method to calculate the class conditional density functions ***p*(*x*|*ω*1)** and ***p*(*x*|*ω*2)** for every ***x*** in {-4:0.1:8, -4:0.1:8}, we need first know what is the estimation function for the density.

In this problem, the dataset was given, the number of samples is n=100, k is given as 10. Then for the above equation, we only need to calculate the value of Vn which is the volume of the cell. In this case, it would be the area of the circle when it contains 10 points.

Here we use Matlab to iteratively calculate the distance between each point in the range of -4:8 and each point in the sample.

|  |
| --- |
| clear,  clc,    %% Load data sets  load('hw3.mat');  x1 = hw3\_2\_1;  x2 = hw3\_2\_2;    %% Find neighbors  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;          [neighbors1, maxDis1] = getNeighbors(x1, [i;j], 10);          [neighbors2, maxDis2] = getNeighbors(x2, [i;j], 10);          Vn1 = pi \* maxDis1^2;          Vn2 = pi \* maxDis2^2;          p1(l,m) = 10/100/Vn1;          p2(l,m) = 10/100/Vn2;      end  end |

In the ‘getNeighbors’ function, we use a function called ‘euclideanDistance’ to calculate the distance between two points:

|  |
| --- |
| function [neighbors, maxDis] = getNeighbors(dataSet, Instance, k)      distances = [];      for i = 1:size(dataSet,2)          dist = euclideanDistance(Instance, dataSet(:,i));          distances = [distances;dist];      end      [B, I] = sort(distances);      neighbors = [];      for i = 1:k          neighbors = [neighbors, dataSet(:,I(i))];      end      maxDis = euclideanDistance(Instance, neighbors(:,end));  end |

Here the ‘euclidean’ function is shown below:

|  |
| --- |
| function y = euclideanDistance(instance1, instance2)      distance = 0;      for i = 1:2          distance = distance + (instance1(i) - instance2(i))^2;      end      y = sqrt(distance);  end |

Finally, plot the 3-D figure using ‘mesh’ function:

|  |
| --- |
| figure(1);  mesh(p1);  figure(2);  mesh(p2); |

As a result, we can get the following density estimation figures from two classes:

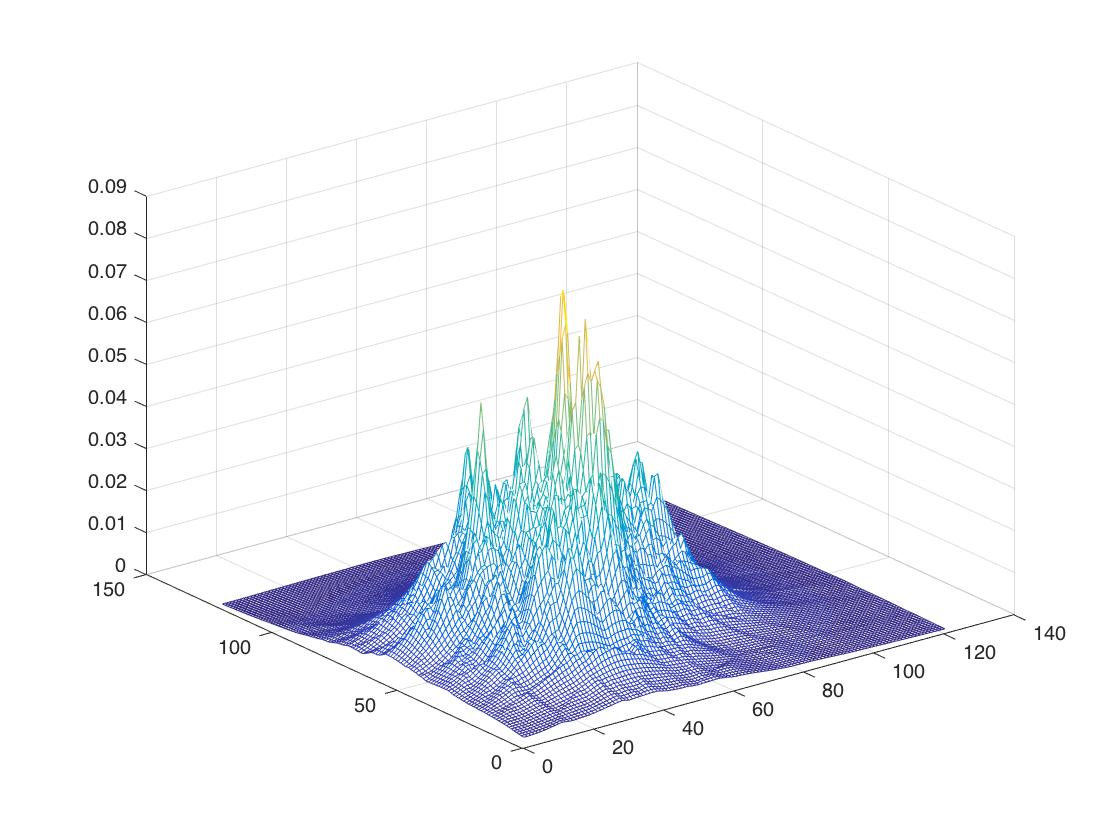


Figure 1. P(x|w1)

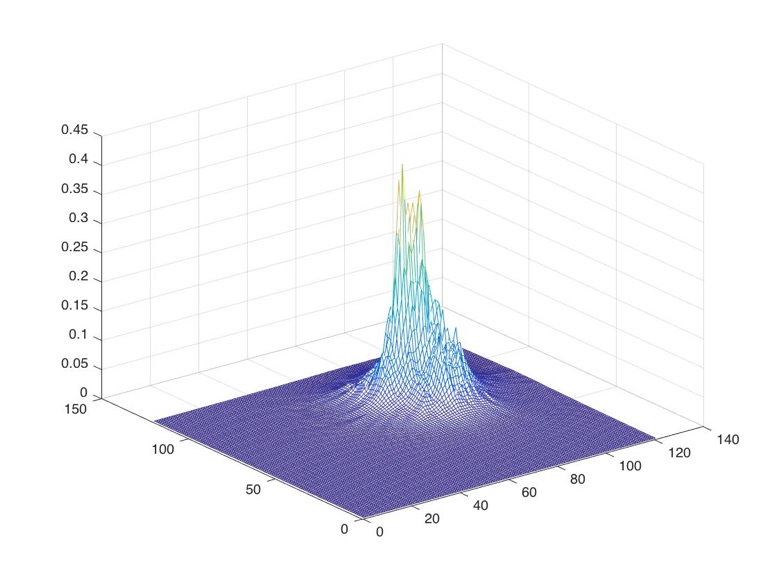


Figure 2. P(x|w2)

Then two classify the point ***x*=[1,-2]*t***, we only need to modify the code and output the possibilities of this point for two classes:

|  |
| --- |
| if i == 1 && j == -2  p1(l,m)  p2(l,m)  end |

The outputs are:

P(x|w1) = 0.0276

P(x|w2) = 0.0010

To classify the point, we use Bayes formula which is:

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

In this case, , so we classify data to .

**Problem 2:**

1. Plot the data set in a 2-D figure:

|  |
| --- |
| %% Plot the data set in 2-D  figure(1);  plot(x1(1,:), x1(2,:), 'ro', x2(1,:), x2(2,:), 'o'); |

Then the figure is shown as the following, red points are data from class 1, blue points are from class 2.

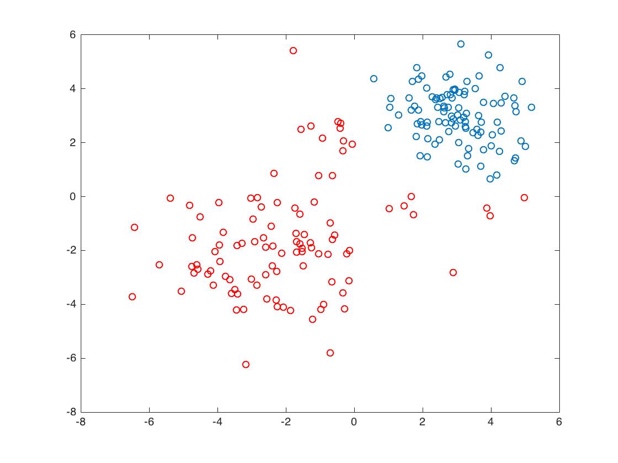


Figure 3. Data set in 2-D for two classes

2. Project the data set , then plot the data set in 3-d space:

|  |
| --- |
| %% Plot the data set in 3-D  figure(2);  plot3(x1(1,:), x1(2,:), x1(1,:).\*x1(2,:), 'ro',...      x2(1,:), x2(2,:), x2(1,:).\*x2(2,:), 'o');  grid on |

Then the figure is shown as the following, red points are data from class 1, blue points are from class 2.

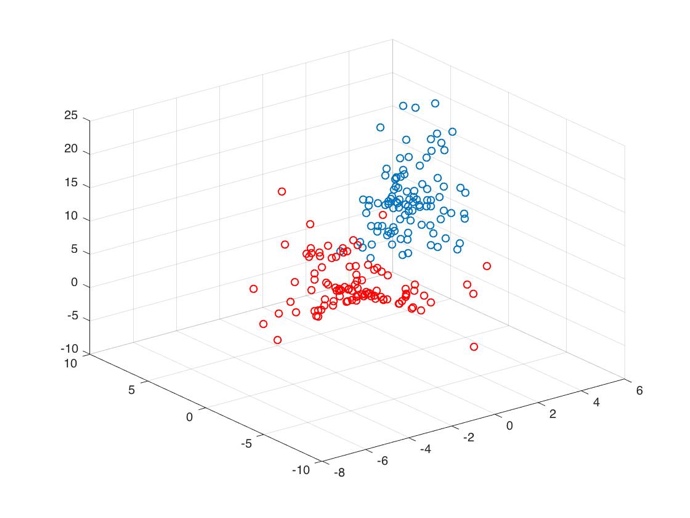


Figure 4. Projected data in 3-d space.

3. To find the weights vector , we need first construct the vector :

|  |
| --- |
| %% Batch Perceptron  y1 = [ones(1,100); x1; x1(1,:).\*x1(2,:)];  y2 = [ones(1,100); x2; x2(1,:).\*x2(2,:)]; |

From the lecture, we know that, we can replace all samples labeled w2 by their negatives, thus we then combine the y1 and –y2 together to make a new y vector:

|  |
| --- |
| y = [y1, -y2]; |

There are some parameters for the Batch Perceptron to use and we can set the initial values for them:

|  |
| --- |
| theta = 1;  ita = 1;  a = sum(y, 2); |

After setting the parameters, we then do the Batch Perceptron algorithm iteration to update the weights vector a.

|  |
| --- |
| while true      missClass = find((a'\*y)<0)      a=a+ita\*sum(y(:,missClass),2);      if abs(ita\*sum(y(:,missClass),2))<theta          break      end  end  a |

When the termination requirement is satisfied, the iteration stopped and return the final a vector.