Lab 3

Albert Segarra Roca (S3255050), Carlos Humberto Paz Rodríguez (S3040577)

Group 16

Pattern Recognition

FMNS • RUG

November 23, 2017

1 Assignment 1

1.1 Exercise 1

The purpose of this exercise is to understand the ROC curves and the discriminability index in order to understand how to choose the best threshold when classifying between two or more classes, considering how much we are willing to sacrifice as a false alarm rate.

Since data is already normalized we can use the matlab function "normcdf", to calculate the integral of the two curves from or up to the threshold we are analyzing.

Therefore we use the following code (listing 1)to obtain a point in the false alarm vs hit plot, for a specific threshold (x value), since the whole area under the curve is 1, we can substract 1 minus the result, to obtain the integral from the threshold point onwards.

Listing 1: False acceptance, hit point

```
function [fa,h]=ROC(mean1, mean2,x)
%False alarm is 1 minus the cumulative integral up to the x* point of the
%first curve
fa=1-normcdf(x,mean1,2);

%Hit is 1 minus the cumulative integral up to the x* point of the second
%curve
h=1-normcdf(x,mean2,2);
end
```

Once we know how to obtain a (false alarm, hit) value, we can generate the whole space of values for the curve, by taking points between $[\mu_1 - 3\sigma, \mu_2 + 3\sigma]$ and plotting them, that is how we obtain the blue ROC curve (Figure 1).

We repeat the same procedure for two different scenarios, one with $\mu_1 = 5$ and $\mu_2 = 9$ and the other with $\mu_1 = 5$ and $\mu_2 = 11$, and also plot them on the same figure (figure 1) (lines red and yellow respectively). (The code for this plot is on the appendix (script 4.2))

As we can observe from the plot, the bigger the separation between the μ_1 and μ_2, the curve approaches the upper left corner of the plot; in the real world this means that the two classes ω_1 and ω_2 have less and less overlaping results, which in a normal distribution plot means their curves have very little overlaping, making it easier to find the optimal threshold.

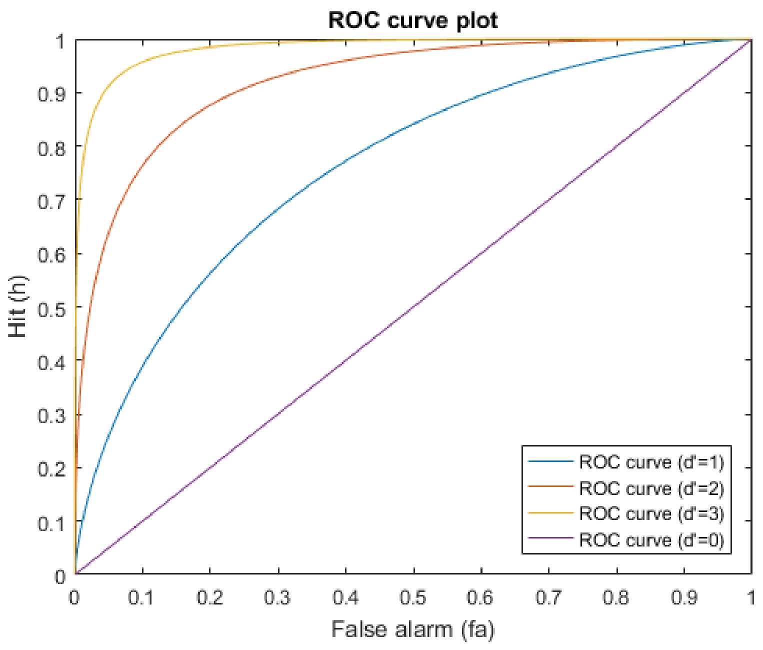

Figure 1: ROC Curves with different discriminability index

This "bending" of the curves, is called the discriminability index and its formula is:

$$d' = \frac{\mu_1 - \mu_2}{\sigma} \tag{1}$$

The discriminability indexes are:

- For $\mu_2 = 7$, $d' = 1$
- For $\mu_2 = 9$, $d' = 2$
- For $\mu_2 = 11$, $d' = 3$

1.2 Exercise 2

For this section we have the results of people saying they detected a signal given that te signal was or not present, so the false alarm rate is when the values correspond to [0 1] and the hit rate to [1 1].

To obtain the false alarm rate, we count how many samples had [0 1] value and divided them by the total population size in order to have a proper rate, we did the same with for the hit rate when finding the [1 1] values (view listing 2).

Listing 2: False acceptance rate, hit acceptance rate

```matlab
function assig31()

load lab3_1.mat

hits = 0;
false_alarm = 0;

%Count how many correct hits and false alarms were generated
for index = 1:200
    outcome = outcomes(index, :);

    hits = hits + isequal([1,1], outcome);
    false_alarm = false_alarm + isequal([0,1], outcome);
end

%We divide the count by the total amount of observations to have the actual
%probability of ocurrence
hit_rate = hits/200;
false_alarm_rate = false_alarm/200;

% We used the same mean1 and sigma from exercise one
mean1=5;
mean2=8.09;
sigma=2;

index=1;
for x=(mean1-3*sigma):0.1:(mean2+3*sigma)
    [array(index,1),array(index,2)]=ROC(mean1,mean2,x);
    index=index+1;
end

plot(array(:,1),array(:,2),'Displayname','ROC curve (d''=3)');
hold on
plot(false_alarm_rate, hit_rate,'*');


hold off;
end
```

Since we don't have the μ or the σ values, we plot the point (false acceptance rate vs hit acceptance rate) and we use the code from the previous exercise to aproximate by trial and error the ROC curve that fits to this point by adjusting the μ_2 value. We found that with $\mu_1 = 5$, $\mu_2 = 8.09$ and $\sigma = 2$ we are able to find a fitting ROC curve, and using the same formula for the discriminability index, we found a that $d' = 1.545$.

The figure 2 shows the false alarm rate vs hit rate (0.045,0.44) found for the sample data and the discriminability index asociated to it. (The script 4.4 was used to generate the figure 2).

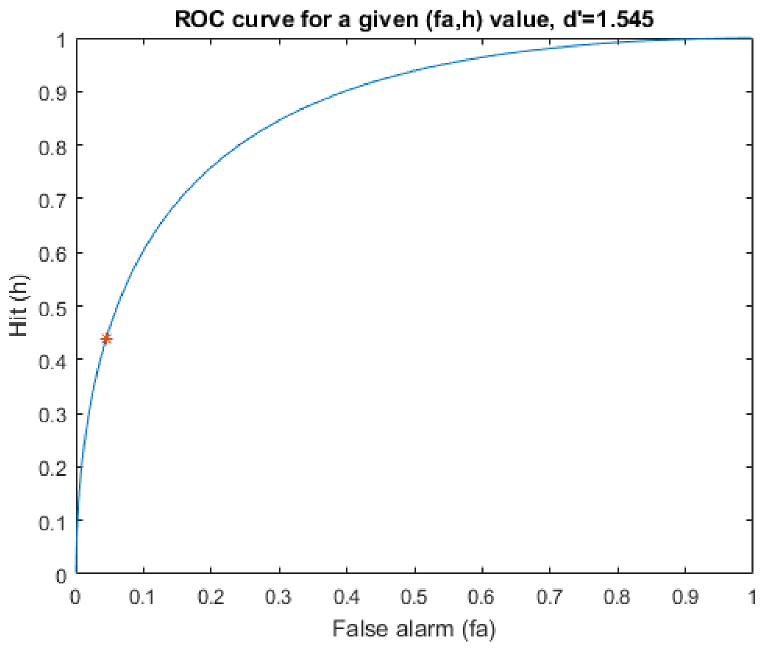

Figure 2: Adjusting the ROC curve for a particular (fa,h) point

2 Assignment 2

2.1 Exercise 1

We are required to generate a code for obtaining the predominant class using the K nearest neighbors method for a given point, K-NN value, and the data and class labels as input in order to be used with the provided code for this exercise, and to finally obtain a classification map, between two different classes. (The code is provided below on the listing 3 and explained in the comments inside the code).

Listing 3: K-nearest neighbor class

```
function class_label = KNN(newPoint, K, data, class_labels)
    %We initialize the control variables
    data_size = size(data, 1);
    euclidean_distance = zeros(data_size, 1);

    %For any given dimension size of the data matrix, we generate a loop
    %and obtain the euclidean distance of the point of interest compared to
    %all the points given on the data matrix
    for i = 1:data_size
        euclidean_distance(i) = pdist([newPoint; data(i,:)]);
    end

    %We then sort the distances in order to obtain the K smallest distances
    [~, I] = sort(euclidean_distance);

    %We look for the highest class occurence in the K nearest points
    k_nearest = I(1:K);
    most_frequent_index = mode(k_nearest);

    %Return the name of the predominant class, using the class_label vector
    class_label = class_labels(most_frequent_index);
```

2.2 Exercise 2

In order to obtain the K-nn for the sample data using different K value, we used the code provided below (code 4), we manually modified the K value before each execution, to generate the K-NN map.

As we can see when we increase the K value the black area dominates the map, this is because there are more green points dominating the area, we get more certainty on the result but we loose precision. (We can see from figure 3 to 6 the evolving behaviour as we increase the K value)

Listing 4: K-nearest neighbor class

```
clear all;
load lab3_2.mat;

K=1;
samples=64;
data = lab3_2;
nr_of_classes = 4;

% Class labels
class_labels = floor( (0:length(data)-1) * nr_of_classes / length(data) );

% Sample the parameter space
result=zeros(samples);
for i=1:samples
  X=(i-1/2)/samples;
  for j=1:samples
    Y=(j-1/2)/samples;
    result(j,i) = KNN([X Y],K,data,class_labels);
  end;
end;

% Show the results in a figure
imshow(result,[0 nr_of_classes-1],'InitialMagnification','fit')
hold on;
title([int2str(K) '-NN, ' int2str(nr_of_classes) ' classes']);

% Correction to visualize 4 classes
scaled_data=samples*data;
plot(scaled_data(  1:50,1),scaled_data(  1:50,2),'go');
plot(scaled_data(51:100,1),scaled_data(51:100,2),'r+');
plot(scaled_data(  101:150,1),scaled_data(  101:150,2),'y*');
plot(scaled_data(151:200,1),scaled_data(151:200,2),'bx');
```

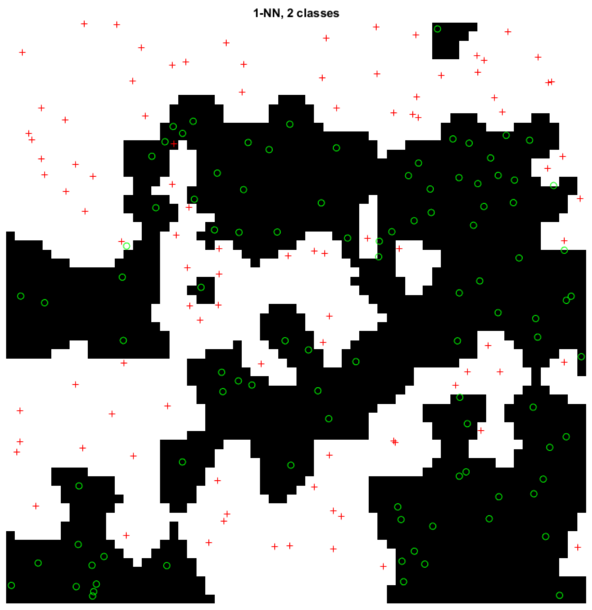

Figure 3: K=1 map

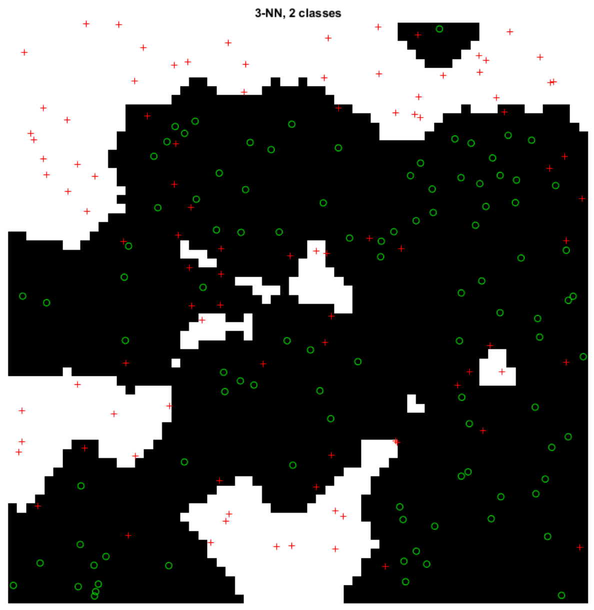

Figure 4: K=3 map

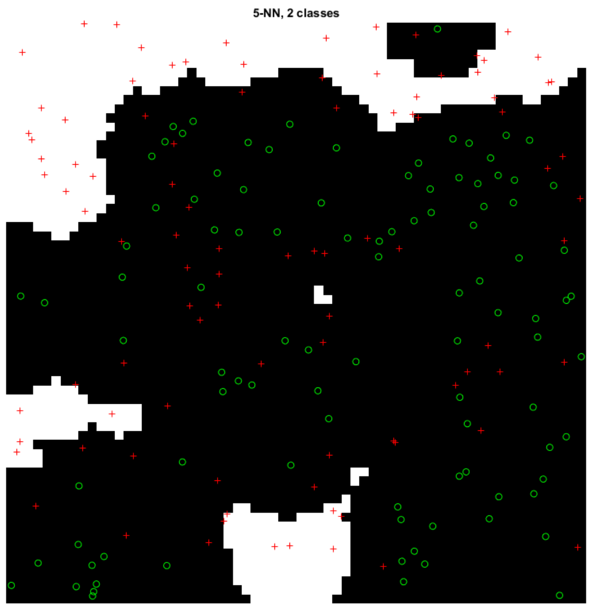

Figure 5: K=5 map

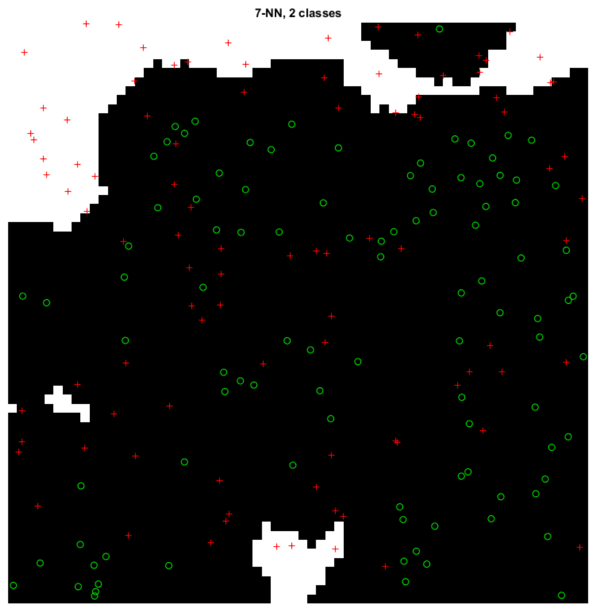

Figure 6: K=7 map

2.3 Exercise 3

In order to determine the best K parameter we use the leave one out method in order to compare each single value with the rest of the data as a training data and then comparing the trained model with the single value, we then count the number of errors and plot them in order to visualize the optimal K value, which is the one with the lowest errors after the whole leave one out cycle. (The code used for finding the optimal result is available on the appendix script 4.5).

We can see in the figure 7 that the optimal K value is 3, that is because after implementing the leave one out method gets the less number of errors. The optimal K value depends a lot on how mixed are the initial data, we should find the best balance between certainty (high K value) and precision (low K value).

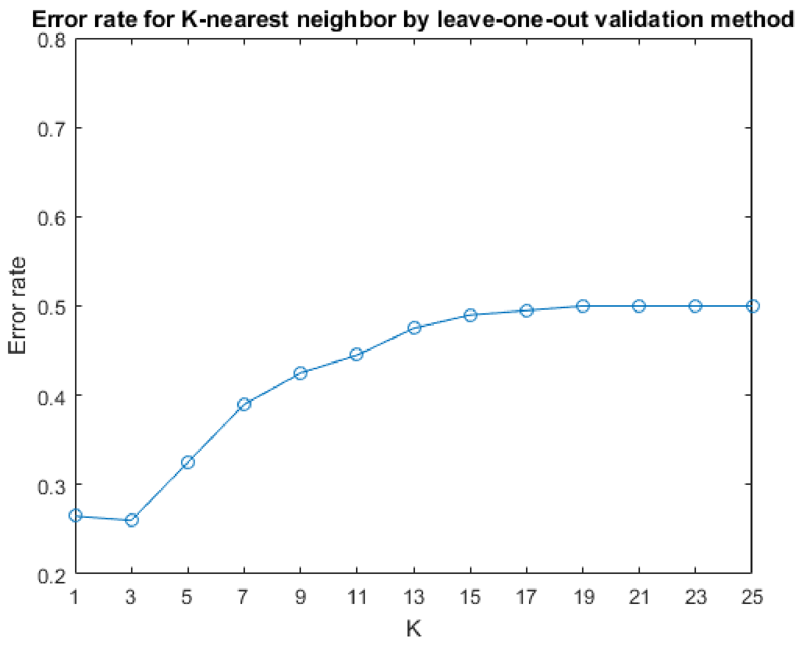

Figure 7: Optimal K value after using the leave one out method

2.4 Exercise 4

In order to repeat the same exercise but with more classes, we modified the $knn_{wrapper}$ and the $optimal_k$ functions to receive as a parameter the number of classes (code 5).

Listing 5: Modification 1

```
nr_of_classes = 4;
```

And also modified the lines 28-32 of the $knn_{wrapper}$ file to be able to print more than two class points (code 6).

Listing 6: Modification 2

```
plot(scaled_data(   1:50,1),scaled_data(   1:50,2),'go');
plot(scaled_data(51:100,1),scaled_data(51:100,2),'r+');
plot(scaled_data(   101:150,1),scaled_data(   101:150,2),'y*');
plot(scaled_data(151:200,1),scaled_data(151:200,2),'bx');
```

NOTE: In the case of tie we used the *class_label* value with the lowest index. We manually modified the K value in the script to generate all the K scenarios. (script 4.5).

We then generated the K-nn map for K values (1,3,5,7) as we can see from figure 8 to figure 11.

In this case we can see how now we have 4 different colors and points representing the 4 different classes.

We also generate the plot showing the error rate for every K value, in Figure 12. In this case we see how the optimal value for K has changed from 3 to 1, and the error rate is always increasing, rising to higher values as compared to 2 classes.

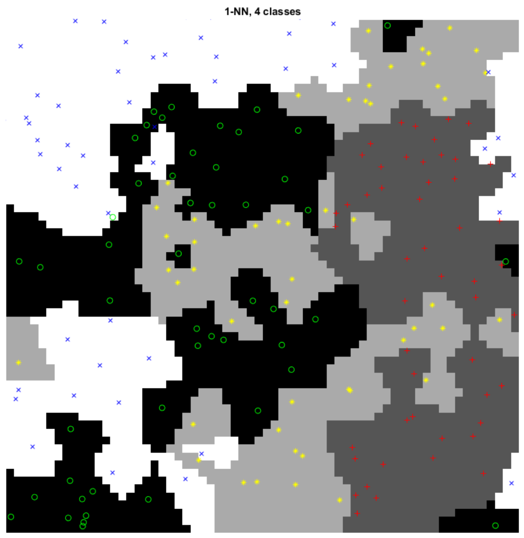

Figure 8: K=1 for 4 classes

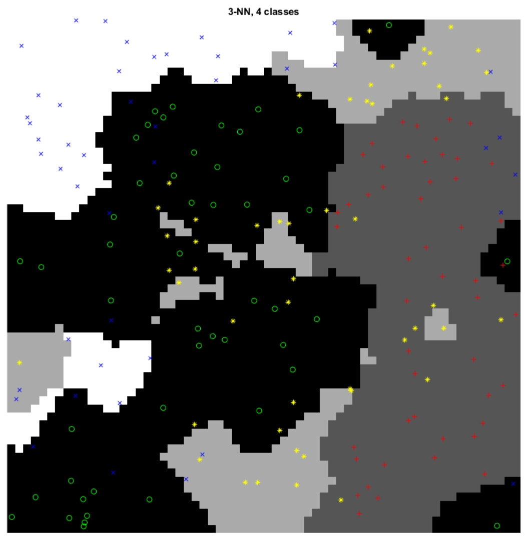

Figure 9: K=3 for 4 classes

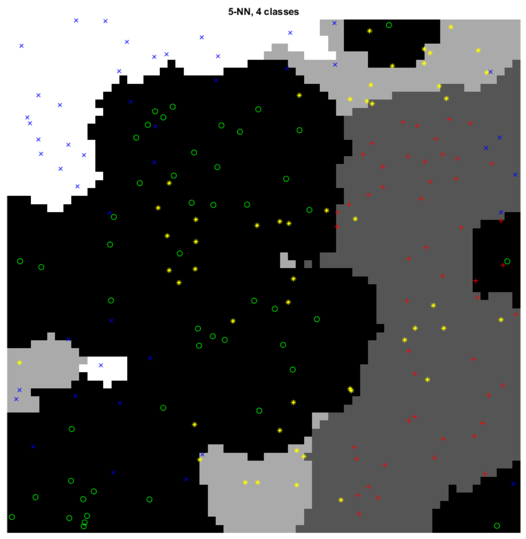

Figure 10: K=5 for 4 classes

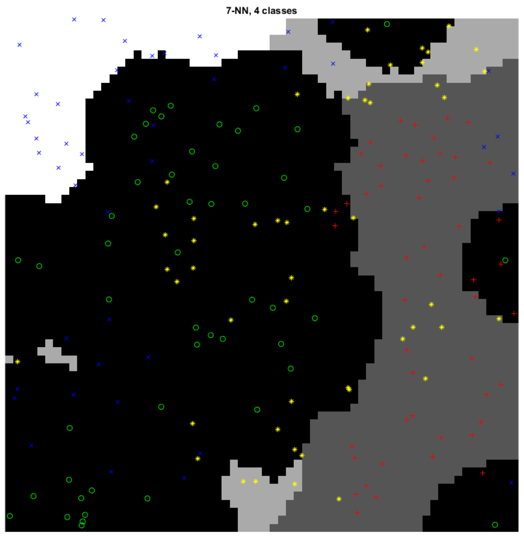

Figure 11: K=7 for 4 classes

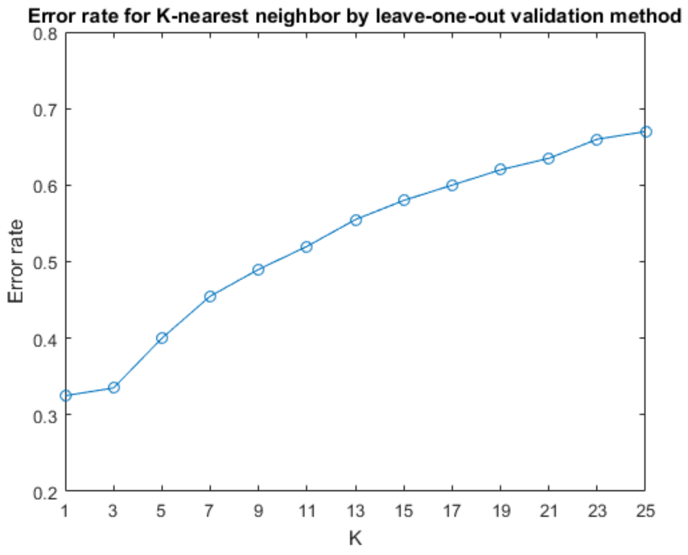

Figure 12: K=7 for 4 classes

3 Assignment 3

3.1 Exercise 1

With an initial $h = 1$ value, we use the Parzen window function

$$theta = exp(\frac{-((x_1 - x_1^j)^2 + (x_2 - x_2^j)^2 + (x_3 - x_3^j)^2)}{2h^2}) \tag{2}$$

to compute all the densities for all points and classes. We do it using the function `parzen4` (Section 4.6), obtaining the following results, which are printed to the console:

Densities			
Category	$\vec{u}$	$\vec{v}$	$\vec{w}$
1	0.0080	0.0097	0.0001
2	0.0299	0.0307	0.0001
3	0.0253	0.0144	0.0000

Table 1: Densities

3.2 Exercise 2

As we do not know what is the likelihood of being presented points of a certain category, we can assume that the points have an equal distribution and so each category has the same likelihood to appear, which leads to the following prior probabilities:

$P_1 = 1/3$
$P_2 = 1/3$
$P_3 = 1/3$

3.3 Exercise 3

Using the previously obtained prior probabilities and densities, we can compute the posterior probabilities:

Posterior probabilities			
Category	$\vec{u}$	$\vec{v}$	$\vec{w}$
1	0.0080*1/3=0.0027	0.0097*1/3=0.0032	0.0001*1/3=0.00004586
2	0.0299*1/3=0.0100	0.0307*1/3=0.0102	0.0001*1/3=0.00002257
3	0.0253*1/3=0.0084	0.0144*1/3=0.0048	0.0000*1/3=0.00000160

Table 2: Posterior probabilities

3.4 Exercise 4

Taking the row with the maximum value for each column of Table 4, we obtain that $\vec{u}$ belongs to category 2, $\vec{v}$ belongs to category 2 and $\vec{w}$ belongs to category 1.

3.5 Exercise 5

We need to call the `parzen4` method with parameter h = 2, and we obtain the following results:

Densities			
Category	$\vec{u}$	$\vec{v}$	$\vec{w}$
1	0.002528	0.002695	0.000671
2	0.006048	0.006223	0.001118
3	0.005811	0.005018	0.000515

Table 3: Densities

Posterior probabilities			
Category	$\vec{u}$	$\vec{v}$	$\vec{w}$
1	0.0008427	0.0008985	0.00022399
2	0.0000020	0.0000020	0.00000037
3	0.0000019	0.0000016	0.00000017

Table 4: Posterior probabilities

As we can see, now the results are different, we obtain that all vectors belong to category 1.

3.6 Exercise 6

Using the following script:

```
u=[0.5 1 0];
v=[0.31 1.51 -0.5];
w=[-1.7 -1.7 -1.7];

load lab3_3_cat1.mat;
load lab3_3_cat2.mat;
load lab3_3_cat3.mat;

data = [x_w1; x_w2; x_w3];

cat_u = KNN(u, 3, data, [ones(1, 10), 2*ones(1, 10), 3*ones(1, 10)]);
cat_v = KNN(v, 3, data, [ones(1, 10), 2*ones(1, 10), 3*ones(1, 10)]);
cat_w = KNN(w, 3, data, [ones(1, 10), 2*ones(1, 10), 3*ones(1, 10)]);

disp(cat_u)
disp(cat_v)
disp(cat_w)
```

that uses our already implemented `KNN` function, we obtain again the same results as in exercise 3.4. Note that we used $K = 3$. The results are again different from exercise 3.5.

3.7 Exercise 7

Now, changing the K value to $K = 5$, we obtain that $\vec{u}$ belongs to class 2, $\vec{v}$ belongs to class 1 and $\vec{w}$ belongs to class 1. This is different from all previously obtained results, but again we see the same behaviour as when we increased the h value of the parzen window, because in both cases we see that more vectors start being classified as of class 1 (if we try $K = 7$ we already get that all of them belong to class 1).

Both `h` and `K` parameters are related because they both indicate how big will be the region around the point that will be considered for the classification, or similarly, how many points will be considered. When `h` increases, we consider a bigger volume, and when `K` increases we consider more surrounding points. That is why when we consider bigger `K` and `h` values the behaviour or the results that we get may be very similar with both methods.

4 Appendix

4.1 Script for calculating the false acceptance and hit between two normal distributions with given means, and a given threshold value

```
function [fa,h]=ROC(mean1, mean2,x)
%False alarm is 1 minus the cumulative integral up to the x* point of the
%first curve
fa=1-normcdf(x,mean1,2);

%Hit is 1 minus the cumulative integral up to the x* point of the second
%curve
h=1-normcdf(x,mean2,2);
end
```

4.2 Script generating the ROC curve map with a given mean a sigma values, compared to mean values of 7, 9 and 11

```
function ROCCurve(mean1, sigma)

    mean2=7;

    index=1;
    for x=(mean1-3*sigma):0.1:(mean2+3*sigma)
        [array(index,1),array(index,2)]=ROC(mean1,mean2,x);
        index=index+1;
    end

    plot1=plot(array(:,1),array(:,2),'Displayname','ROC curve (d''=1)');
    hold on
    %==========================================

    mean2=9;

    index=1;
    for x=(mean1-3*sigma):0.1:(mean2+3*sigma)
        [array(index,1),array(index,2)]=ROC(mean1,mean2,x);
        index=index+1;
    end

    plot2=plot(array(:,1),array(:,2),'Displayname','ROC curve (d''=2)');
    %========================================

    mean2=11;

    index=1;
    for x=(mean1-3*sigma):0.1:(mean2+3*sigma)
        [array(index,1),array(index,2)]=ROC(mean1,mean2,x);
        index=index+1;
    end

    plot2=plot(array(:,1),array(:,2),'Displayname','ROC curve (d''=3)') ;
    %========================================

    %d'=0
    xx=0:0.1:1;
    title('ROC curve plot');
    xlabel('False alarm (fa)'), ylabel('Hit (h)');
    axis on, axis normal, hold on;

    yy=xx;
    plot0=plot(xx,yy,'Displayname','ROC curve (d''=0)');

    legend('show','Location','SouthEast')

end
```

4.3 Script for generating the ROC curve for a particular sample data, and adjusting the discriminability index manually

```
function assig31()

load lab3_1.mat

hits = 0;
false_alarm = 0;

%Count how many correct hits and false alarms were generated
for index = 1:200
    outcome = outcomes(index, :);

    hits = hits + isequal([1,1], outcome);
    false_alarm = false_alarm + isequal([0,1], outcome);
end

%We divide the count by the total amount of observations to have the actual
%probability of ocurrence
hit_rate = hits/200
false_alarm_rate = false_alarm/200

% We used the same mean1 and sigma from exercise one
mean1=5;
```

```
mean2=8.09;
sigma=2;

index=1;
for x=(mean1-3*sigma):0.1:(mean2+3*sigma)
    [array(index,1),array(index,2)]=ROC(mean1,mean2,x);
    index=index+1;
end

plot(array(:,1),array(:,2),'Displayname','ROC curve (d''=3)');
title('ROC curve for a given (fa,h) value, d''=1.545')
xlabel('False alarm (fa)'), ylabel('Hit (h)');
    axis on, axis normal
hold on
plot(false_alarm_rate, hit_rate,'*');


hold off;
end
```

4.4 Script for finding the optimal K value of the K-nn method utilizing the leave one out technique

```
function optimal_k()

load lab3_2.mat;
data = lab3_2;
data_size = size(data, 1);
nr_of_classes = 2;

% Class labels
class_labels = floor( (0:length(data)-1) * nr_of_classes / length(data) );

error_rates = zeros(13, 1);

for K = 1:2:25
    errors = 0;
    for leave_out_index = 1:data_size
        data_copy = data;
        point = data_copy(leave_out_index, :);
        data_copy(leave_out_index, :) = [];

        assigned_class = KNN(point, K, data_copy, class_labels);
        real_class = class_labels(leave_out_index);
        is_error = assigned_class ~= real_class;

        errors = errors + is_error;
    end
    index = floor(K/2) + 1;
    error_rates(index) = errors/data_size;
end

plot([1:2:25], error_rates, '-o');
title('Error rate for K-nearest neighbor by leave-one-out validation method');
set(gca, 'XTick', [1:2:25]);
axis([1 25 0.2 0.6]);
xlabel('K');
ylabel('Error rate');
```

4.5 Script for printing the K-NN map for four classes.

```
clear all;
load lab3_2.mat;

K=5;
samples=64;
data = lab3_2;
nr_of_classes = 2;

% Class labels
class_labels = floor( (0:length(data)-1) * nr_of_classes / length(data) );

% Sample the parameter space
result=zeros(samples);
for i=1:samples
  X=(i-1/2)/samples;
  for j=1:samples
    Y=(j-1/2)/samples;
    result(j,i) = KNN([X Y],K,data,class_labels);
  end;
end;

% Show the results in a figure
imshow(result,[0 nr_of_classes-1],'InitialMagnification','fit')
hold on;
title([int2str(K) '-NN, ' int2str(nr_of_classes) ' classes']);

% Correction to visualize 4 classes
scaled_data=samples*data;
plot(scaled_data(  1:100,1),scaled_data(  1:100,2),'go');
plot(scaled_data(101:200,1),scaled_data(101:200,2),'r+');
```

4.6 Script for calculating the Parzen Window

```
function parzen4(h)

%We load the three categories files
load lab3_3_cat1.mat;
load lab3_3_cat2.mat;
load lab3_3_cat3.mat;

%Initialize the accumulators
sum_theta_u=zeros(3);
sum_theta_v=zeros(3);
sum_theta_w=zeros(3);

%Set the initial points
u=[0.5 1 0];
v=[0.31 1.51 -0.5];
w=[-1.7 -1.7 -1.7];

%For each point we calculate the sum of al the sample values
for i=1:length(x_w1(:,1))
     sum_theta_u(1)=sum_theta_u(1) + theta(h,u, x_w1(i,:));
     sum_theta_u(2)=sum_theta_u(2) + theta(h,u, x_w2(i,:));
     sum_theta_u(3)=sum_theta_u(3) + theta(h,u, x_w3(i,:));

     sum_theta_v(1)=sum_theta_v(1) + theta(h,v, x_w1(i,:));
     sum_theta_v(2)=sum_theta_v(2) + theta(h,v, x_w2(i,:));
```

```
    sum_theta_v(3)=sum_theta_v(3) + theta(h,v, x_w3(i,:));

    sum_theta_w(1)=sum_theta_w(1) + theta(h,w, x_w1(i,:));
    sum_theta_w(2)=sum_theta_w(2) + theta(h,w, x_w2(i,:));
    sum_theta_w(3)=sum_theta_w(3) + theta(h,w, x_w3(i,:));
end

%Normalize theta
norm_theta_u(1)=sum_theta_u(1)/(h*sqrt(2*pi))^3;
norm_theta_u(2)=sum_theta_u(2)/(h*sqrt(2*pi))^3;
norm_theta_u(3)=sum_theta_u(3)/(h*sqrt(2*pi))^3;

norm_theta_v(1)=sum_theta_v(1)/(h*sqrt(2*pi))^3;
norm_theta_v(2)=sum_theta_v(2)/(h*sqrt(2*pi))^3;
norm_theta_v(3)=sum_theta_v(3)/(h*sqrt(2*pi))^3;

norm_theta_w(1)=sum_theta_w(1)/(h*sqrt(2*pi))^3;
norm_theta_w(2)=sum_theta_w(2)/(h*sqrt(2*pi))^3;
norm_theta_w(3)=sum_theta_w(3)/(h*sqrt(2*pi))^3;

%Divide by the number of points
new_theta_u(1)= norm_theta_u(1)/length(x_w1(:,1));
new_theta_u(2)= norm_theta_u(2)/length(x_w2(:,1));
new_theta_u(3)= norm_theta_u(3)/length(x_w3(:,1));

new_theta_v(1)= norm_theta_v(1)/length(x_w1(:,1));
new_theta_v(2)= norm_theta_v(2)/length(x_w2(:,1));
new_theta_v(3)= norm_theta_v(3)/length(x_w3(:,1));

new_theta_w(1)= norm_theta_w(1)/length(x_w1(:,1));
new_theta_w(2)= norm_theta_w(2)/length(x_w2(:,1));
new_theta_w(3)= norm_theta_w(3)/length(x_w3(:,1));

PP_c1=[new_theta_u(1) new_theta_v(1) new_theta_w(1)];
PP_c2=[new_theta_u(2) new_theta_v(2) new_theta_w(2)];
PP_c3=[new_theta_u(3) new_theta_v(3) new_theta_w(3)];

disp('Prior probabilities');
disp(PP_c1)
disp(PP_c2)
disp(PP_c3)

%So far we have the probability function of each class, since we don't know the prior
    probabilities of the data, but we have the same amount of sample data of each class, we
    will use 1/3
%for each one
P1=1/3;
P2=1/3;
P3=1/3;

%We calculate the posterior probabilities of each class
PP_c1=PP_c1*P1;
PP_c2=PP_c2*P2;
PP_c3=PP_c3*P3;

disp('Posterior probabilities');
disp(PP_c1);
```