Pattern Recognition Practical 3

Group 24: Maikel Withagen (s1867733) Steven Bosch (s1861948)

September 30, 2015

Assignment 1 Classification error, hit/false alarm rates, ROC curve, discriminability

1

Figure 1 shows the ROC-curves we acquired using the code given in the listing for assignment 1.1 in the appendix. The figure shows that the higher the difference between the means of the two distributions is (i.e. the further away the distributions are from each other), the higher the number of hits is per number of fals alarms. This means that classification will go better when distributions are farther away from each other, which is of intuitively comprehensable as well.

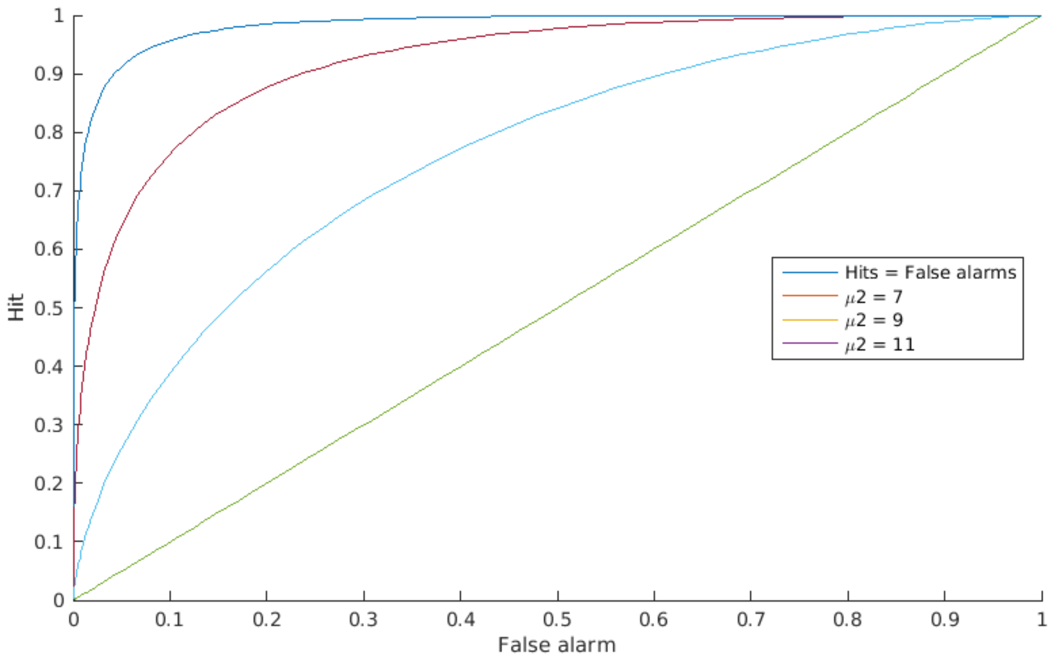


Figure 1: ROC-curves for $\mu_2 = 7, 9, 11$ and the hits = false alarms marker line.

2

Figure 2 shows the point (fa, h) of the two given binary vectors plotted in the plot computed in assignment 1.1. The listing for assignment 1.2 in the appendix gives the code used to compute this point and to compute the ROC-curve with the associated discriminability value d'. Trial and error yielded a ROC-curve with $d' \approx 1.5$ (this is an approximation, the exact value is a decimal value that is time-consuming to find by trial and error). We computed this using $\sigma_{1,2} = 1$, which yields $\mu_2 - \mu_1 = 1.5$ for the found discriminability value. Note that the code in the listing gives $\sigma = 2$, which means $\mu_2 - \mu_1 = 3$, since $d' = \frac{\mu_2 - \mu_1}{\sigma}$. So as long as the discriminability valueb between two distributions is the same, the sigma does not matter for classification.

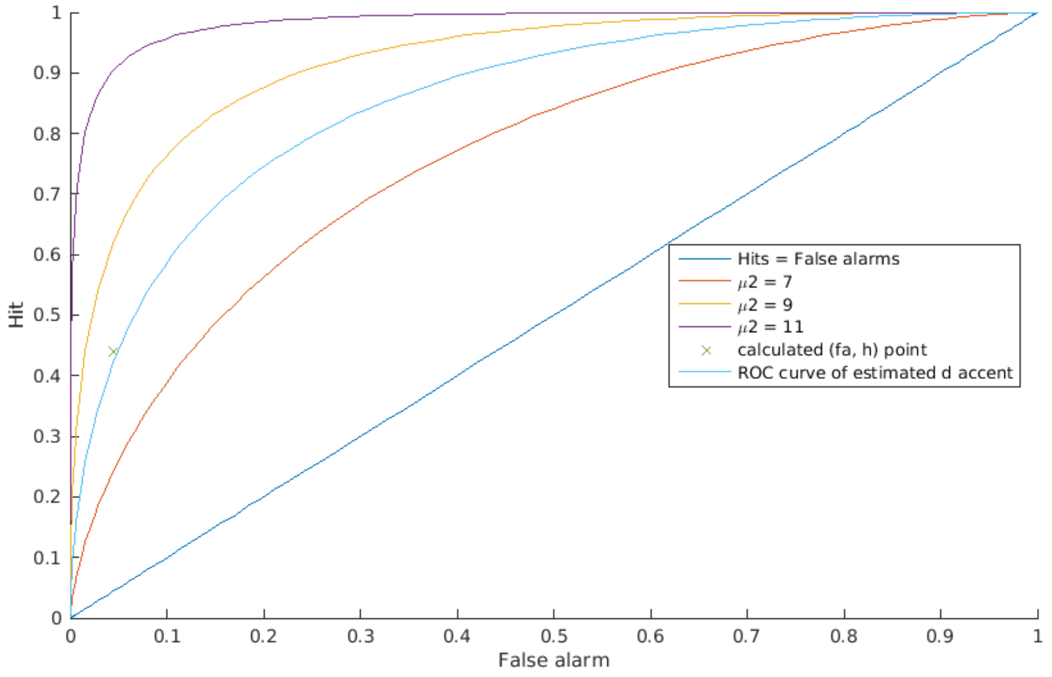


Figure 2: Plot of the (fa, h) point, the ROC-curves for $\mu_2 = 7, 9, 11$, the hits = false alarms marker line, and the ROC-curve with $d' = 1.5$.

Assignment 2 K-nearest neighbor classification

1

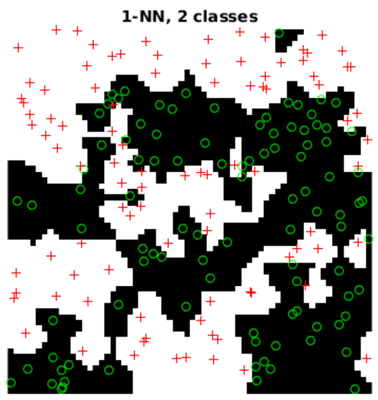
Our implementation of the KNN-function is the following:

../Code/KNN.m

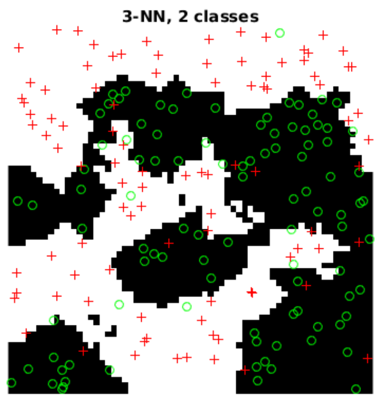
```
function [class] = KNN( X, K, data, class_labels)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

% Calculate distance to each point ([distance class])
distances = zeros(length(data), ndims(data));
for row = 1:length(data)
    dist = 0;
    for dim = 1:ndims(data)
        dist = dist + abs(data(row,dim)-X(1,dim));
    end
    distances(row,:) = [dist class_labels(row)];
end
distances = sortrows(distances);

class = mode(distances(1:K,2));
end
```

2

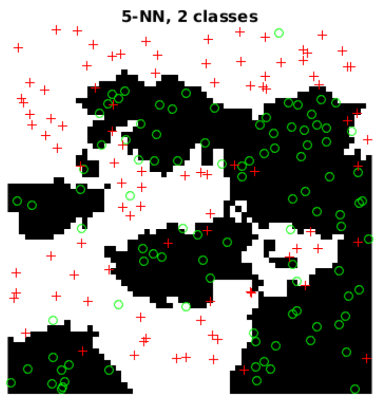
For $k = 1, 3, 5, 7$ this implementation yields the classification grid shown in figure 3:



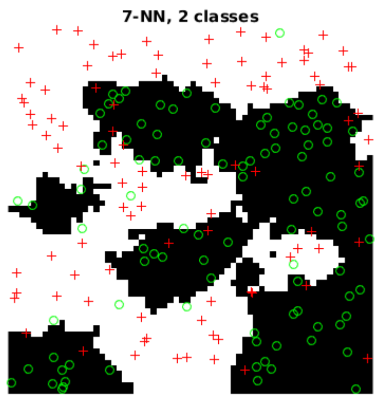
(a) Classification grid of the data set using KNN (for $k = 1$)



(b) Classification grid of the data set using KNN (for $k = 3$)



(c) Classification grid of the data set using KNN (for $k = 5$)



(d) Classification grid of the data set using KNN (for $k = 7$)

Figure 3: Classification grids for different ks

3

Our implementation of the leave-one-out cross validation is given in the appendix. Figure 4 gives the error rates we acquired for the diffferent values for k using our implementation. The figure shows that a k of 3 or

5 yields the best performance with an error rate of around 0.23. It is not illogical that the optimal k is such a value.

On the one hand a k of 1 performs worse, because it does not account for outliers. With a k of 1 outliers would classify a number of data points incorrectly, while increasing k to 3 would already correct for that, because it is very unlikely that two outliers would be that close to each other that they would together misclassify a data point (you could even argue whether they would actually be outliers if they are close to each other).

On the other hand when k would become too high, data points are prone to be misclassified when they are closer to the decision boundary (for example when there are multiple points of the other class on the other side of the decision boundary which would be taken into account).

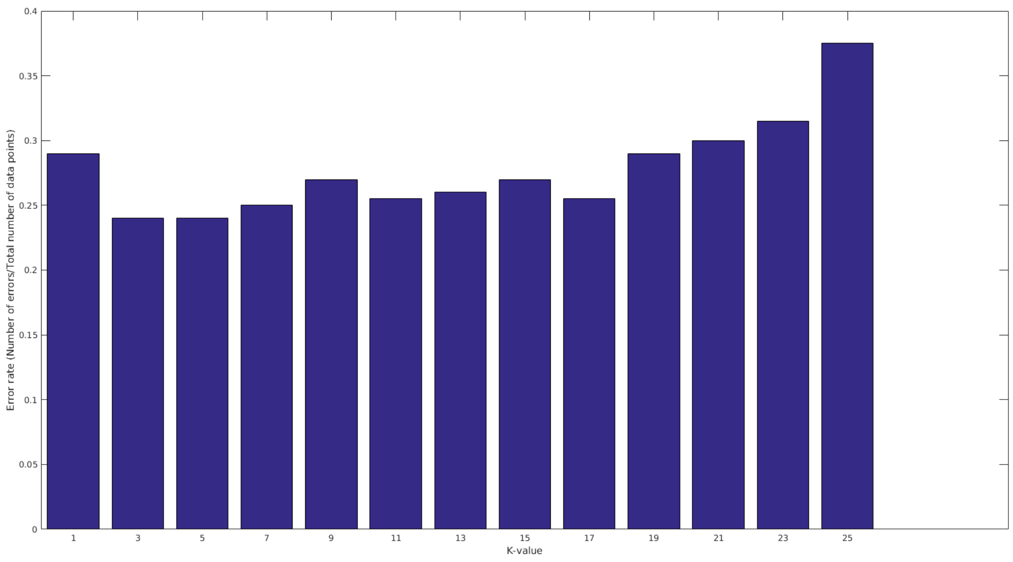
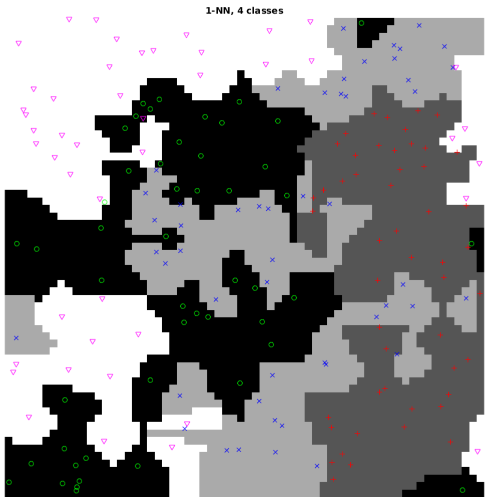


Figure 4: Error rate for different values of k using leave-one-out cross validation.

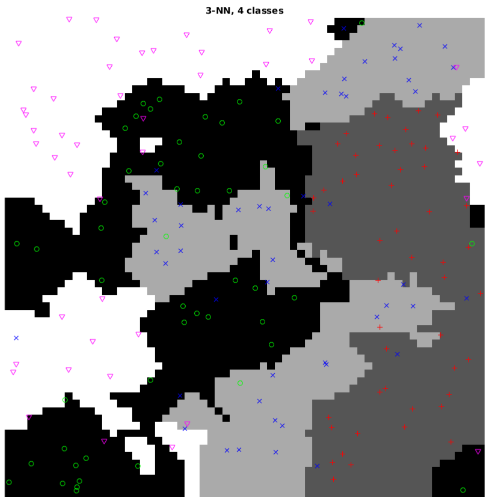
4

To do classification for four classes we changed the parameter of classes to 4 and added the new classes to the plots. This resulted in the classification grids shown in figure 5. Cross validation with four classes yields the error rates given in figure 5b. The figures show that again most of the points get classified correctly. This time we find an optimum at only a k of 5 (not of 3 as well, as was the case with two classes). The average error rate does seem to be higher than with two classes though. This seems logical, because with random data you would expect that with more classes there is a higher probability of a data point being classified into the wrong class. This is however, totally dependent on the data. If the actual data represent four well divided classes, the classification would of course go better when you would actually use four classes, because when you would just use two, the data might be distributed over the two classes in such a way that there is not such a clear separation boundary as with four classes.

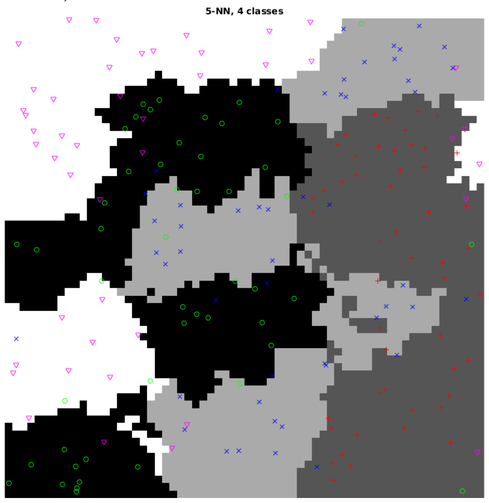
In normal circumstances however, the classifier data would be seperated into logical groups instead of random groups like we do now. Then a difference in classifying performance would be logically explainable, because the data would actually represent groups already.



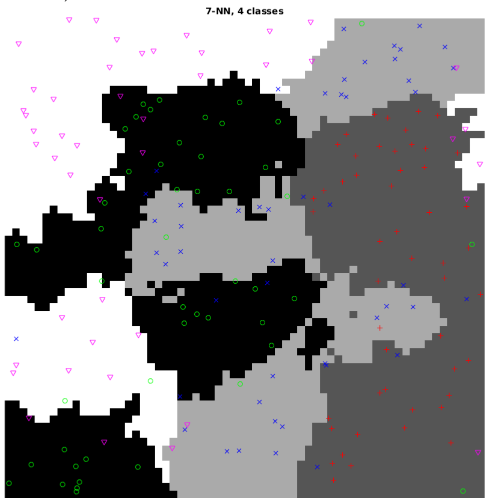
(a) Classification grid of the data set using KNN (for $k = 1$)



(b) Classification grid of the data set using KNN (for $k = 3$)



(c) Classification grid of the data set using KNN (for $k = 5$)



(d) Classification grid of the data set using KNN (for $k = 7$)

Figure 5: Classification grids for different ks

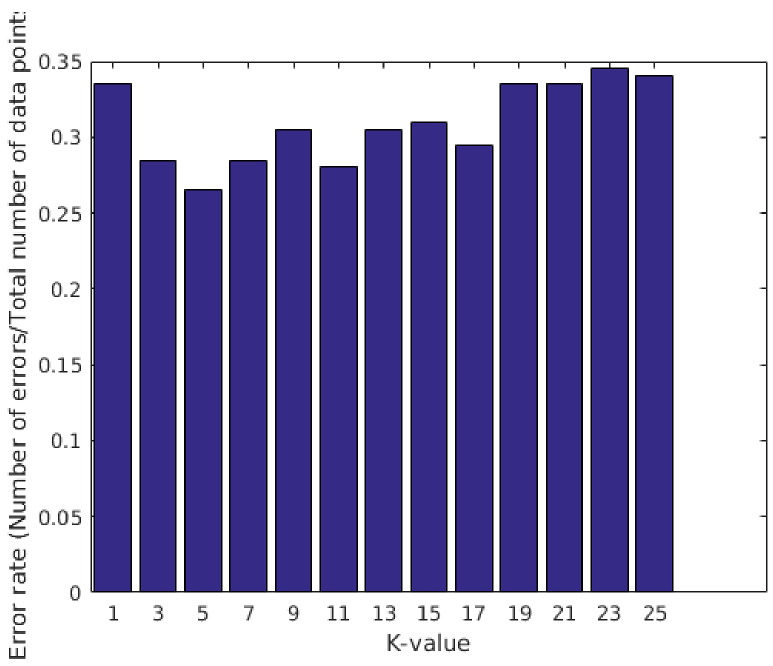


Figure 6: Error rate for different values of k using leave-one-out cross validation.

Assignment 3 Parzen windows, posterior probabilities

Appendix

../Code/assign1_1.m

```
% 1. Choose a value of x     and compute the probabilities of hit (h) and false alarm (fa).
% x* = 6
% hit = integral 6 to inf for dist 2 = 1 - integral -inf to 6 =
1 - normcdf(6, 7, 2)
% fa = integral 6 to inf for dist 1 = 1 - integral -inf to 6 =
1 - normcdf(6, 5, 2)

% Plot the point (fa, h) in a graph with horizontal axis fa and vertical axis h.
% Choose a few other values of x     in the interval [   1     3  ;    2 + 3   ] (-1 ; 13)
% and plot the corresponding (fa, h) points too. Repeat the computation of a ROC
% curve for the cases    2 = 9 and    2 = 11 and plot all three ROC curves in the same
% diagram.
x = []; y1 = []; y2 = []; y3 = [];
for xStar = -1:0.1:13
    x(end+1) = (1 - normcdf(xStar, 5, 2));
    y1(end+1) = (1 - normcdf(xStar, 7, 2));
    y2(end+1) = (1 - normcdf(xStar, 9, 2));
    y3(end+1) = (1 - normcdf(xStar, 11, 2));
end
hold on
```

```
plot(x,x)
plot(x,y1)
plot(x,y2)
plot(x,y3)
xlabel('False alarm')
ylabel('Hit')
legend('Hits = False alarms', '\mu2 = 7','\mu2 = 9','\mu2 = 11', 'Location', 'east')

% What is the value of the discriminability d for
% each of these cases?
% D(9) = (9-5)/2 = 2,
% D(11) = (11-5)/2 = 3
```

../Code/assign1_2.m

```
% Compute the values of the hit rate h and the false alarm rate fa and plot the
    point (fa, h) in the plot computed in assignment 1.1 above. This point lies on a
    ROC curve
hitrate= sum(outcomes(:,1) == 1 & outcomes(:,2) == 1)/length(outcomes)
false =  sum(outcomes(:,1) == 0 & outcomes(:,2) == 1)/length(outcomes)
plot(false,hitrate, 'x')
% with a given disciminability value d . Determine this value by trial and error, i.
    e. taking different values d and drawing the corresponding ROC curves until you
    find a value of d for which the corresponding ROC curve passes through the
    experimentally determined point.

x = [];y4 = [];
for xStar = -100:.1:100
    x(end+1) = (1 - normcdf(xStar, 5, 2));
    y4(end+1) = (1 - normcdf(xStar, 8, 2));
end
plot(x,y4)
legend('Hits = False alarms', '\mu2 = 7','\mu2 = 9','\mu2 = 11', 'calculated (fa, h)
    point', 'ROC curve of estimated d accent', 'Location','east')
% So d' is approximately 1.5.
```

../Code/KNN.m

```
function [class] = KNN( X, K, data, class_labels)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

% Calculate distance to each point ([distance class])
distances = zeros(length(data), ndims(data));
for row = 1:length(data)
    dist = 0;
    for dim = 1:ndims(data)
        dist = dist + abs(data(row,dim)-X(1,dim));
    end
    distances(row,:) = [dist class_labels(row)];
end
distances = sortrows(distances);

class = mode(distances(1:K,2));
end
```

../Code/assign2_3.m

```
clear all;
load lab3_2.mat;

K=1;
samples=64;
data = lab3_2;
nr_of_classes = 2;

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

%Determine the optimal choice of the parameter K in the range 1, 3, . . . , 25 using
    leave-
%one-out cross validation:
error = zeros (25, 1);
for K = 1:2:25
    for i = 1:length(data)
        point = data(i,:);
        % For each point in the data set, make a copy of the dataset without that
            point.
        temp_data = data;
        temp_class_labels = class_labels;
        temp_data(i,:) = [];
        temp_class_labels(i) = [];

        % Classify the point using the reduced dataset.
        if class_labels(i) ~= KNN(point, K, temp_data, temp_class_labels)
            error(K) = error(K) + 1;
        end
    end
end

% Plot the resulting errors
bar(1:2:25, error(1:2:25)/200)
xlabel('K-value')
ylabel('Error rate (Number of errors/Total number of data points)')
```