Assignment 1
Medical Image Analysis

Azalea Alothmani

January 19, 2022

1 Kernel Estimation

1.1 Introduction

Determining the probability density function of a dataset is essential in order to estimate the probability distribution for specific outcomes of a random variable. However, it is uncommon that the probability density function for a random variable is known. Therefore, the probability density is estimated; so called probability density estimation is considered instead.

1.1.1 Kernel Density Estimation

Kernel density estimation (KDE) is a non-parametric method to estimate the probability density function of the observation of a random sample that does not resemble a common probability distribution. Given a random sample with an unknown density function f for a new given observation x, the kernel density estimator of the function f is

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}K\left(\frac{x-x_i}{h}\right) \tag{1}$$

Here the scaled kernel, $f_h x$ is given, where n is the number of samples, h is the smoothing parameter and represents the kernel width and K is the kernel, most commonly used kernel is Gaussian kernel that controls the contribution of the samples within the window of observations:

$$K(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2} \tag{2}$$

The kernel function smooths the probabilities across a range of observation for a random variable. The kernel density estimates for each data point x_i are summed to achieve the kernel density estimation of the given data. The contribution of observation from a data sample is weighted based on their distance to the given sample. The scope, or Parzen window of observations, that will contribute to the probability estimation for a new observation is controlled by the smoothing parameter h. A large smoothing parameter, or window, may result in oversmoothing hiding important details of the structure. While a small h contains a large amount of details and may not be generalized enough for the new unseen observation. There is a clear trade-off.

In order to find the optimal kernel width, cross validation method is used. The training data is then divided in training part and testing part. An example is so called $leave_one_out$ cross validation where one example is each cross-validation step is taken away and be used as a testing data. This algorithm is implemented as following:

- For the dataset T with n samples, for each width h, n runs are made
  - For each run, take away one example x_i of training data, so that the remaining data has (n-1) samples
  - Estimate distribution based on the remaining data for one type of width h.
  - Compute the log-likelihood on the example x_i
- Choose the width h that maximizes the sum of the log-likelihood

$$E(h) = \sum_{i=1}^{n} \hat{f}_h(x_i; T \backslash x_i)$$

1.1.2 Gaussian Distribution

According to Bays rule the probability that a class is $y = j$ given a measurement x is called posterior probability and it is defined accordingly

$$P(y = j \mid x) = \frac{P(x \mid y = j)P(y = j)}{P(x)}$$

where $P(y = j)$ is the prior and $P(x|y = j)$ is the likelihood. If we assume that x and y are continuous stochastic variables then the likelihood can be calculated using normal probability distribution function (PDF) for a given mean μ and standard deviation σ.

The likelihood, $P(x|y = j)$ is given by the normal distribution. Therefore, PDF value can be used to estimate the likelihood of a random variable to belong to a specific range of values. PDF is a density of probability and its unit is probability per unit length.

Function *normpdf* in MATLAB is used for calculation of probability distribution for each class for a given mean and standard deviation in order to label each measurement x with the class that gives maximized normal probability distribution. The general formula for the probability density function is

$$f(x) = \frac{e^{-(x-\mu)^2/\left(2\sigma^2\right)}}{\sigma\sqrt{2\pi}} \tag{3}$$

where μ is mean value, location parameter, and σ is the standard derivation, scale parameter. Standard normal distribution is when $\mu = 0$ and $\sigma = 1$.

1.1.3 Cumulative Distribution Function

Cumulative distribution function (CDF) F_X represents the probability that a random variable X takes a value in the interval $[a, b]$ that is equal or less than a cutoff value x. For continuous variable the probability can be expressed as

$$P(a \leq X \leq b) = \int_a^b f_X(x)dx = P(a \leq X \leq b) = F_X(b) - F_X(a) \tag{4}$$

CDF is a monotonically increasing function from zero to one and it gives the area underneath the probability density function (PDF) in the interval $[a, b]$.

1.2 Method

In this task we are constructing a two-class problem with only one feature, which is modeled as a Gaussian with equal standard deviation $s_1 = s_2 = s_{model}$ for each class with the means $m_1 = m_2 = 1$. Two cases, i.e $s_{model} = 0.4$ and 4 are considered in this assignment. For each of these two models we generate a training data with 10 points for each class, and testing data with 1000 point for each class. The probability density is estimated using the (non-parametric) parzen window estimation; with two different widths, i.e $h = 0.02$ and 1. For each case plot the likelihoods $p(x|y = 1)$, $p(x|y = 2)$ and $p(x)$ both using the true model, Gaussian distribution, and from the estimation. Next, for each case we plot the conditional probabilities $p(y = 1|x)$ and $p(y = 2|x)$, both using the true model and from the estimation. The estimated probability $p(y = 1|x)$ and $p(y = 2|x)$ is then used to classify the data, and the error rate for the test data is calculated. From the true model with the likelihoods $p(x|y = 1)$, $p(x|y = 2)$ represented as having Gaussian distribution with know mean and standard derivation, the optimal threshold can be calculated by setting eq.**??** for two different standard derivations, $s_{model} = 0.4$ and 4, equals to each other to obtain the threshold x. This threshold is then used in the cumulative distribution function, eq.4 to calculate the theoretical optimal error rate.

Cross-validation method is then used to estimate the optimal kernel width that maximizes the sum of log-likelihood.

1.3 Results

1.3.1 Estimated probability Density functions

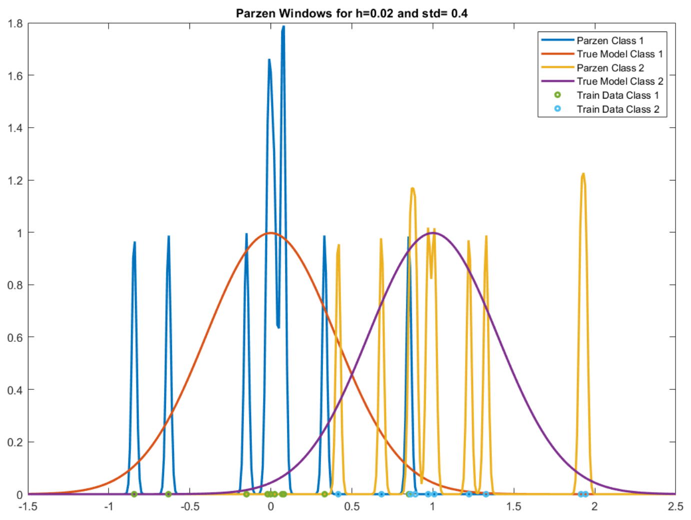


Figure 1: Parzen window estimation with width $h = 0.02$ and $std = 0.4$. The figure shows the estimated $p(x|y = 1)$, $p(x|y = 2)$; for the two classes. True model Gaussian distributions. $p(x)$ for the two classes and the training data for the two classes are also shown. $p(x)$ is calculated using normal probability distribution function (PDF), *normpdf()* function in Matlab, for a given mean μ and standard deviation σ.

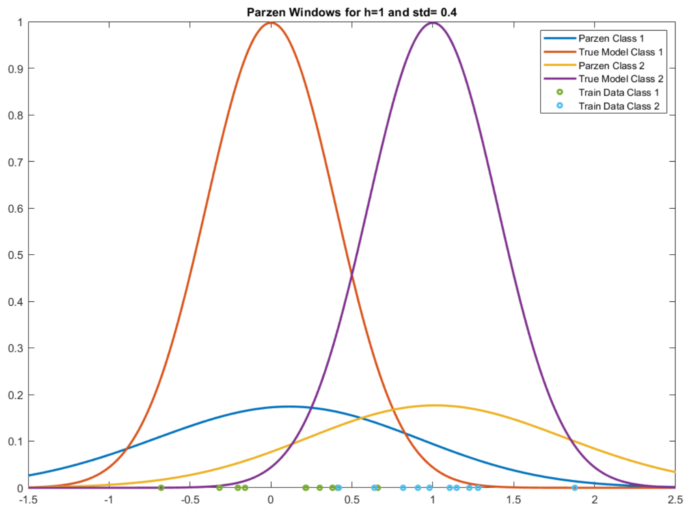


Figure 2: Parzen window estimation with width $h = 1$ and $std = 0.4$. The figure shows the estimated $p(x|y = 1)$, $p(x|y = 2)$; for the two classes. True model Gaussian distributions. $p(x)$ for the two classes and the training data for the two classes are also shown. $p(x)$ is calculated using normal probability distribution function (PDF), *normpdf()* function in Matlab, for a given mean μ and standard deviation σ.

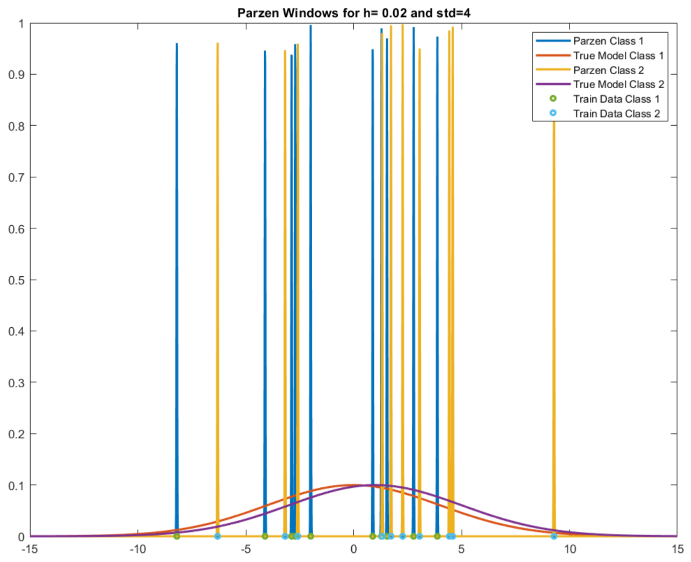


Figure 3: Parzen window estimation with width $h = 0.02$ and $std = 4$. The figure shows the estimated $p(x|y = 1)$, $p(x|y = 2)$; for the two classes. True model Gaussian distributions. p(x) for the two classes and the training data for the two classes are also shown. $p(x)$ is calculated using normal probability distribution function (PDF), *normpdf()* function in Matlab, for a given mean μ and standard deviation σ.

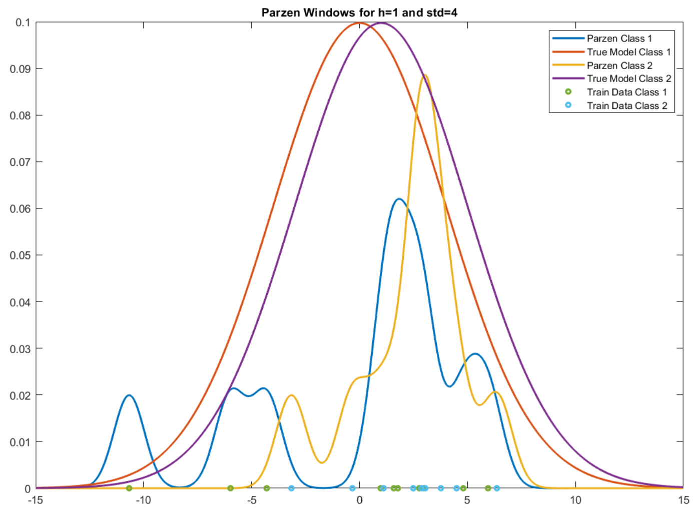


Figure 4: Parzen window estimation with width $h = 1$ and $std = 4$. The figure shows the estimated $p(x|y = 1)$, $p(x|y = 2)$; for the two classes. True model Gaussian distributions. $p(x)$ for the two classes and the training data for the two classes are also shown. $p(x)$ is calculated using normal probability distribution function (PDF), *normpdf()* function in Matlab, for a given mean μ and standard deviation σ.

1.3.2 Conditional Probabilities

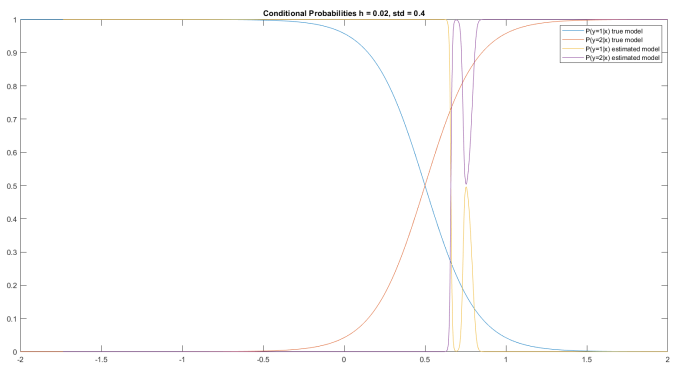


Figure 5: Conditional probabilities for parzen window estimation with width $h = 0.02$ and $std = 0.4$. The figure shows the estimated $p(y = 1|x)$, $p(y = 2|x)$; for the two classes.

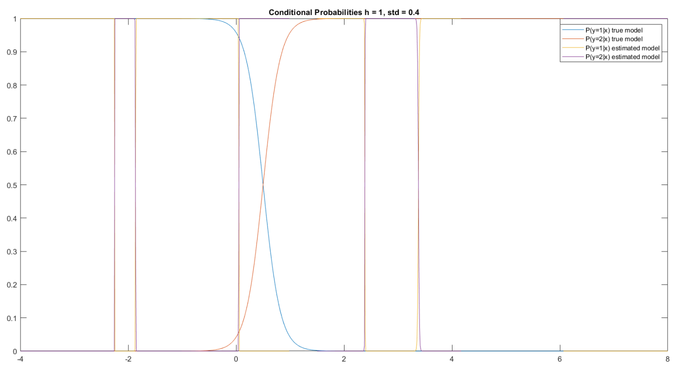


Figure 6: Conditional probabilities for parzen window estimation with width $h = 1$ and $std = 0.4$. The figure shows the estimated $p(y = 1|x)$, $p(y = 2|x)$; for the two classes.

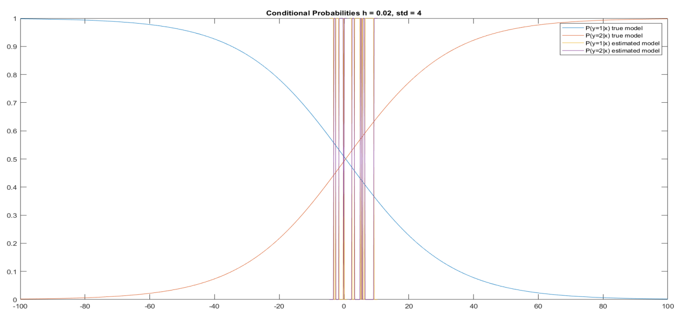


Figure 7: (a) Conditional probabilities for parzen window estimation with width $h = 0.02$ and $std = 4$. The figure shows the estimated $p(y = 1|x)$, $p(y = 2|x)$; for the two classes. (b)

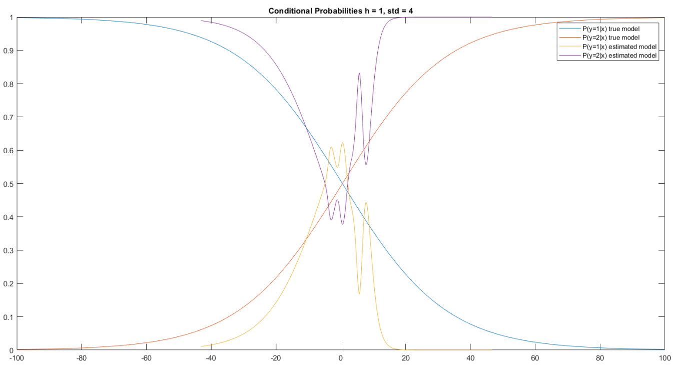


Figure 8: Conditional probabilities for parzen window estimation with width $h = 1$ and $std = 4$. The figure shows the estimated $p(y = 1|x)$, $p(y = 2|x)$; for the two classes.

1.3.3 Classification Error Rates

True Model	**Error rates**
class 1, std = 0.4	10.90%
class 2, std = 0.4	11.50 %
class 1, std = 4	51.2%
class 2, std = 4	48.8%

Table 1: Error rates of the true model for two different standard derivations

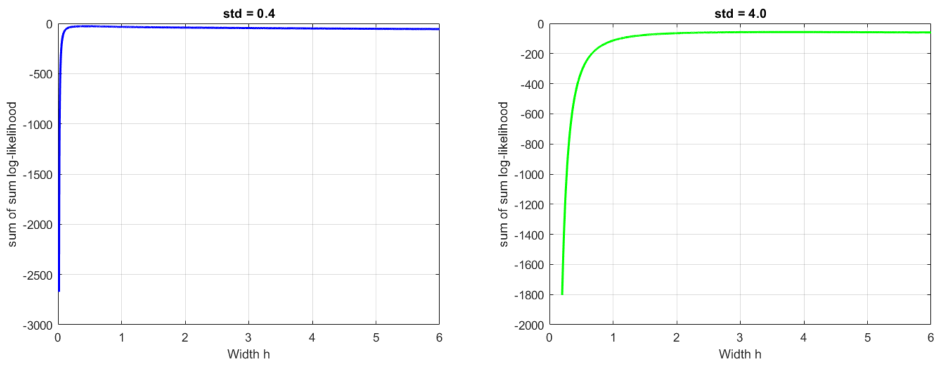
Estimated Model	**Error rates**
class 1, std = 0.4, h = 0.02	25.5%
class 2, std = 0.4, h = 0.02	14.3 %
class 1, std = 0.4, h = 1	11.0%
class 2, std = 0.4, h = 1	10.3 %
class 1, std = 4, h = 0.02	61.2 %
class 2, std = 4, h = 0.02	59.7%
class 1, std = 4, h = 1	58.3 %
class 2, std = 4, h = 1	38.6 %

Table 2: Error rates for kernel density estimation model for two different standard derivations and kernel width.

Optimal Threshold	**Error rates**
class 1, std = 0.4	10.56%
class 2, std = 0.4	10.56 %
class 1, std = 4	45.03 %
class 2, std = 4	45.03 %

Table 3: Error rates using cumulative distribution function *normcdf* with the optimal threshold obtained by setting eq.4 for two different s_{model}, 0.4 and 4, to calculate x. The optimal x is equals to 0.5 for both cases $s_{model} = 0.4$ and $s_{model} = 4$.

1.3.4 Optimal kernel width



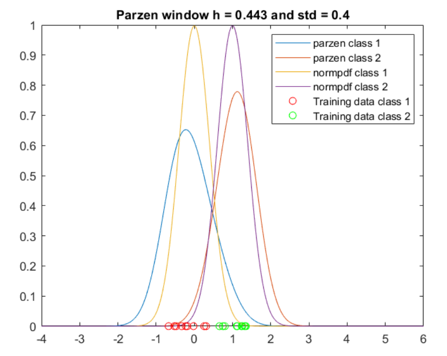
(a) (b)

Figure 9: (a) The kernel width as function sum of sum log-likelihood for $s_{model} = 0.4$, the optimal kernel width is $h = 0.443$. (b) The kernel width as function sum of sum log-likelihood for $s_{model} = 4$, the optimal kernel width is $h = 3.918$.

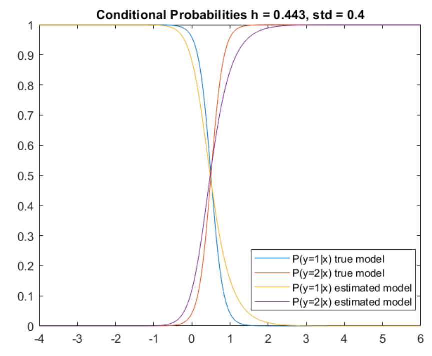
1.4 Conclusion and Discussion

By comparing the obtained error rates from the estimation model in table.2 and the theoretical optimal error rates in table.3 for $s_model = 0.4$, it can be noticed that the error rate for small kernel width, $h = 0.02$ is larger than the optimal error rates for both classes; class 1 and 2. The estimated error rates for the large kernel width, $h = 1$, are closer to the optimal error rate values with 0.44 different for class 1, where for class 2 the estimated error rate seems be a slightly better the theoretical error values. This is due to the fact that for kernel density estimation with the small width, $h = 0.02$, contains many spurious data or local spikes resulting in a undersmooth curve, as it also can be observed in Fig.1. The kernel estimation function in Fig.2 for the large window, $h = 1$, represents a coarse density with little details and it averages over the whole distribution.

By using the cross-validation method, the optimal kernel width could be estimated. For $s_{model} = 0.4$, the optimal width is $h = 0.443$. The resulting probability density estimation and the conditional probability for the estimate model and the true model with the estimated optimal model is shown in Fig.10(a) and (b) respectively. As it could be noticed, the estimated kernel probability better aligned with the true model, having slightly the same distribution. The conditional probabilities in Fig.10(b) are in approximation identical to the true model probabilities. Reflecting back to the conditional probabilities shown in Fig.5, $h = 0.2$ and Fig.7, $h = 1$, it can be observed that the kernel width has an affect on these probabilities. By using too large window, an average probability is achieved without taking into account all the new observation. This results in higher error rate than the true model in Tab.1, 25.5% for class 1 and 14.3% for class 2 Tab-2.



(a)



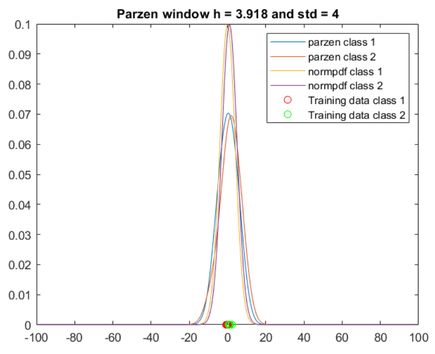
(b)

Figure 10: (a) The kernel estimation probability for $s_{model} = 0.4$ and the optimal width $h = 0.443$ for the estimated and the true model. (b) The conditional probability for $s_{model} = 0.4$ and the optimal width $h = 0.443$ for the estimated and the true model

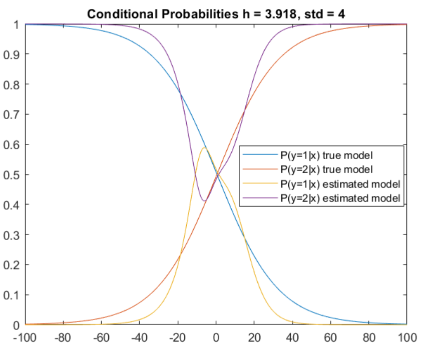
For the larger standard derivation $s_{model} = 4$, the error rates shown in Tab.2 are in general larger for both the true model and estimation in comparison with the error rates for $s_{model} = 0.4$. Thus for larger standard derivation parameter we are getting increasingly difficult classification problems, specially for the small kernel width. The estimated density shown in Fig.3 has a lot of discontinuities and does not have the density distribution. By comparing the error rates of the estimated model in Tab.2 with the optimal threshold error in Tab.3, we can observe that The error rates are also higher for small kernel window than the larger window. Fig.7 shows that with we are getting increasingly difficult classification problem with larger s_{model} and small h. The result achieved here for the kernel density estimation can compared to the nearest-neighbour (KNN) classifier. With small k parameter, the number of observation within the window, the density distribution will have a narrow width/bandwidth makes it more sensitive to abrupt changes and noise leading to unstable decision boundary. Nearest neighbour classifier with small k does not generalize well to the new unseen observations and tends to have a high variance. The same reasoning is applied for kernel estimation with small window.

Added in the second version: The k-nearest neighbour kernel density estimation method is a special type of the kernel density estimation method with the local choice of the bandwidth. The two approaches (kernel and knn) are both based on the idea of using the sample information in a neighborhood of x to form an estimator. The window width in kernel method is fixed regardless of the point x of interest, while that of the knn method varies with x by design. An advantage of this estimator is that smoothing varies according to the number of observations in a particular region. The Kernel estimation fixes the volume V, while KNN fixes K. Either way, it can be shown that both methods converge to the true probability density as N increases providing that V shrinks with N and that K grows with N.
The optimal kernel width for $s_{model} = 4$ is achieved used same procedure as described above, resulting in $h = 3.918$. The resulting probability density estimation and the conditional probability for the estimate model and the true model with the estimated optimal model is shown in Fig.11(a) and (b) respectively. As it could be noticed, the estimated kernel probability better aligned with the true model, having slightly the same distribution. However, the conditional probabilities in Fig.11(b) have large discrepancies in comparison to the true model, meaning that large standard derivation

parameter gives increasing difficulty to classify the data points correctly.



(a)



(b)

Figure 11: (a) The kernel estimation probability for $s_{model} = 4$ and the optimal width $h = 3.918$ for the estimated and the true model. (b) The conditional probability for $s_{model} = 4$ and the optimal width $h = 3.918$ for the estimated and the true model

1.5 Code

1.5.1 Implementing of kernel density function

```
function f = my_parzen(x,T,h)
% The function takes the training data T as input as well as
    width h
% and points x at which the function is evaluated

%K = (1/sqrt(2*pi)).* exp(-T.^2./2);
f = 1/(length(T))*sum((1/h)*(1/sqrt(2*pi))*exp(-((x-T)/h)
    .^2/2));
end
```

1.5.2 Generating of the data, training and test data with two different standard derivations. Kernel estimation probability plots

```
close all; clear all;

%% Generating the data; training and test data with two
    different standard
% derivations

% Dataset for std = 0.4
train1 = normrnd(0,0.4,[10,1]);
test1 = normrnd(0,0.4,[1000,1]);
train2 = normrnd(1,0.4,[10,1]);
test2 = normrnd(1,0.4,[1000,1]);

% Dataset for std = 4.0
train3 = normrnd(0,4,[10,1]);
test3 = normrnd(0,4,[1000,1]);
```

```
train4 = normrnd(1,4,[10,1]);
test4 = normrnd(1,4,[1000,1]);

% Gaussian Kernel width; two alternatives h = 0.02 and h = 1
h = 0.02;

% TO evalulate the kernel density estimation
x_plot = linspace(-20,20, 1000);
% To store kernel estimation results for class 1
y_plot1 = zeros(size(x_plot));
% To store kernel estimation results for class 2
y_plot2 = zeros(size(x_plot));

% Kernel density estimation on training data for the two
    classes
for i = 1:length(x_plot)
    y_plot1(i) = my_parzen(x_plot(i), train3, h);
    y_plot2(i) = my_parzen(x_plot(i), train4, h);
end

% plot of the kernal density estimations for the two classes
figure;
plot(x_plot, y_plot1)
hold on
plot(x_plot,y_plot2)
hold on

% The likelihoods for the true model using normpdf
plot(x_plot, normpdf(x_plot, 0, 4));
hold on
plot(x_plot, normpdf(x_plot, 1, 4));
hold on
plot(train1,zeros(length(train3), 1), 'ro')
hold on
plot(train2, zeros(length(train4), 1), 'go')
legend('parzen class 1', 'parzen class 2', 'normpdf class 1',
     'normpdf class 2', 'Training data class 1', 'Training
    data class 2')
hold off
title('Parzen window h = 0.02 and std = 4')
```

1.5.3 Conditional probabilities

```
close all; clear all;

%% Generating the data; training and test data with two
    different standard
% derivations

% Dataset for std = 0.4
train1 = normrnd(0,0.4,[10,1]);
test1 = normrnd(0,0.4,[1000,1]);
```

```
train2 = normrnd(1,0.4,[10,1]);
test2 = normrnd(1,0.4,[1000,1]);

% Dataset for std = 4.0
train3 = normrnd(0,4,[10,1]);
test3 = normrnd(0,4,[1000,1]);
train4 = normrnd(1,4,[10,1]);
test4 = normrnd(1,4,[1000,1]);

% Gaussian Kernel width; two alternatives h = 0.02 and h = 1
h = 0.02;

% TO evalulate the kernel density estimation
x_plot = linspace(-20,20, 1000);
% To store kernel estimation results for class 1
y_plot1 = zeros(size(x_plot));
% To store kernel estimation results for class 2
y_plot2 = zeros(size(x_plot));

% Kernel density estimation on training data for the two
    classes
for i = 1:length(x_plot)
    y_plot1(i) = my_parzen(x_plot(i), train3, h);
    y_plot2(i) = my_parzen(x_plot(i), train4, h);
end

% Conditional propabilities using the true model
p12_sum = normpdf(x_plot, 0, 4) + normpdf(x_plot, 1, 4);
P34_sum = normpdf(x_plot, 0, 4) + normpdf(x_plot, 1, 4);
p1 = normpdf(x_plot, 0, 4)./p12_sum;
p2 = normpdf(x_plot, 1, 4)./p12_sum;

% Conditional probabilities for the estimated model
g1_estimated = y_plot1./(y_plot1 + y_plot2);
g2_estimated = y_plot2./(y_plot1 + y_plot2);

% Plotting the condtional probabilities for the true and the
    estimated
% model
figure;
plot(x_plot, p1)
hold on
plot(x_plot, p2)
hold on
plot(x_plot, g1_estimated)
hold on
plot(x_plot,g2_estimated)
hold off
legend('P(y=1|x) true model','P(y=2|x) true model','P(y=1|x)
    estimated model', 'P(y=2|x) estimated model')
title('Conditional Probabilities h = 0.02, std = 4')
```

1.5.4 Classification error rates

```
close all; clear all;

%% Generating the data; training and test data with two
    different standard
% derivations

% Dataset for std = 0.4
train1 = normrnd(0,0.4,[10,1]);
test1 = normrnd(0,0.4,[1000,1]);
train2 = normrnd(1,0.4,[10,1]);
test2 = normrnd(1,0.4,[1000,1]);

% Dataset for std = 4.0
train3 = normrnd(0,4,[10,1]);
test3 = normrnd(0,4,[1000,1]);
train4 = normrnd(1,4,[10,1]);
test4 = normrnd(1,4,[1000,1]);

% Gaussian Kernel width; two alternatives h = 0.02 and h = 1
h = 0.02;

% TO evaluate the kernel density estimation
x_plot = linspace(-20,20, 1000);
% To store kernel estimation results for class 1
y_plot1 = zeros(size(x_plot));
% To store kernel estimation results for class 2
y_plot2 = zeros(size(x_plot));

% Kernel density estimation on training data for the two
    classes
for i = 1:length(x_plot)
    y_plot1(i) = my_parzen(x_plot(i), train3, h);
    y_plot2(i) = my_parzen(x_plot(i), train4, h);
end


% Plug-in: For data classification for the two classes, and
    for two
% different std and parzen width
class1_small_std = 0;
class2_small_std= 0;
class1_large_std = 0;
class2_large_std = 0;
class1_estimated_small_h_small_std = 0;
class2_estimated_small_h_small_std = 0;
class1_estimated_large_h_small_std = 0;
class2_estimated_large_h_small_std = 0;
class1_estimated_large_h_large_std = 0;
class2_estimated_large_h_large_std = 0;
```

```
class1_estimated_small_h_large_std = 0;
class2_estimated_small_h_large_std = 0;

%% std = 4

for i =1:length(test1)
    % True model: Setting the threshold to 0.5 where if the Gaussian
    % distribution is larger than 0.5 then it belongs to class 1
    % otherwise it classifies as class 2.
    if (normpdf(test3(i), 0, 4)/ (normpdf(test3(i), 0, 4) + normpdf(test3(i), 1, 4))) > 0.5
        class1_large_std = class1_large_std + 1;
    end
    % Trure model: Setting the threshold to 0.5 where if the Gaussian
    % distribution is larger than 0.5 then it belongs to class 2
    % otherwise it classifies as class 1.
    if (normpdf(test4(i), 1, 4) / (normpdf(test4(i), 0, 4) + normpdf(test4(i), 1, 4))) > 0.5
        class2_large_std = class2_large_std + 1;
    end
    % Estimated model for h = 1: Setting the threshold to 0.5 where if the
    % distribution is larger than 0.5 then it belongs to class 1
    % otherwise it classifies as class 2.
    if my_parzen(test3(i), train3, 1)/(my_parzen(test3(i), train3, 1) + my_parzen(test3(i), train4, 1)) > 0.5
        class1_estimated_large_h_large_std = class1_estimated_large_h_large_std + 1;
    end
    % Estimated model for h = 1: Setting the threshold to 0.5 where if the
    % distribution is larger than 0.5 then it belongs to class 2
    % otherwise it classifies as class 1.
    if my_parzen(test4(i), train4, 1)/ (my_parzen(test4(i), train4, 1)  + my_parzen(test4(i), train3, 1)) > 0.5
        class2_estimated_large_h_large_std = class2_estimated_large_h_large_std + 1;
    end
    % Estimated model for h = 0.02
    if my_parzen(test3(i), train3, 0.02)/(my_parzen(test3(i), train3, 0.02) + my_parzen(test3(i), train4, 0.02)) > 0.5
        class1_estimated_small_h_large_std = class1_estimated_small_h_large_std + 1;
    end
```

```matlab

    % Estimated model for h = 0.02
    if my_parzen(test4(i), train4, 0.02)/ (my_parzen(test4(i), train4, 0.02)  + my_parzen(test4(i), train3, 0.02)) > 0.5
        class2_estimated_small_h_large_std = class2_estimated_small_h_large_std + 1;
    end
end


%% std = 0.4
for i =1:length(test1)
    if (normpdf(test1(i), 0,0.4)/ (normpdf(test1(i), 0, 0.4) + normpdf(test1(i), 1, 0.4))) > 0.5
        class1_small_std = class1_small_std + 1;
    end

    if (normpdf(test2(i), 1,0.4) / (normpdf(test2(i), 0, 0.4) + normpdf(test2(i), 1, 0.4))) > 0.5
        class2_small_std = class2_small_std + 1;
    end

    if my_parzen(test1(i), train1, 0.02)/(my_parzen(test1(i), train1, 0.02) + my_parzen(test1(i), train2, 0.02)) > 0.5
        class1_estimated_small_h_small_std = class1_estimated_small_h_small_std + 1;
    end

    if my_parzen(test2(i), train2, 0.02)/ (my_parzen(test2(i), train2, 0.02)  + my_parzen(test2(i), train1, 0.02)) > 0.5
        class2_estimated_small_h_small_std = class2_estimated_small_h_small_std + 1;

    end

    if my_parzen(test1(i), train1, 1)/(my_parzen(test1(i), train1, 1) + my_parzen(test1(i), train2, 1)) > 0.5
        class1_estimated_large_h_small_std = class1_estimated_large_h_small_std + 1;
    end

    if my_parzen(test2(i), train2, 1)/ (my_parzen(test2(i), train1,1)  + my_parzen(test2(i), train2, 1)) > 0.5
        class2_estimated_large_h_small_std = class2_estimated_large_h_small_std + 1;
    end

end
```

```


%%  error rates
er_1_true_small_std = (1000 - class1_small_std)/1000;
er_2_true_small_std = (1000 - class2_small_std)/1000;
er_1_true_large_std = (1000 - class1_large_std)/1000;
er_2_true_large_std = (1000 - class2_large_std)/1000;
er_1_estimated_small_h_small_std = (1000 -
    class1_estimated_small_h_small_std)/1000;
er_2_estimated_small_h_small_std = (1000 -
    class2_estimated_small_h_small_std)/1000;
er_1_estimated_small_h_large_std = (1000 -
    class1_estimated_small_h_large_std)/1000;
er_2_estimated_small_h_large_std = (1000 -
    class2_estimated_small_h_large_std)/1000;
er_1_estimated_large_h_small_std = (1000 -
    class1_estimated_large_h_small_std)/1000;
er_2_estimated_large_h_small_std = (1000 -
    class2_estimated_large_h_small_std)/1000;
er_1_estimated_large_h_large_std = (1000 -
    class1_estimated_large_h_large_std)/1000;
er_2_estimated_large_h_large_std = (1000 -
    class2_estimated_large_h_large_std)/1000;

%% Estimating the threshold by setting two Gaussian equations
 % equal to each other. x = 0.5 for std = 0.4 and std = 4
 cdf_mu_0_small_std = normcdf(0.5, 0, 0.4);
 err_cdf_mu_0_small_std = 1- cdf_mu_0_small_std;
 cdf_mu_0_large_std = normcdf(0.5, 0, 4);
 err_cdf_mu_0_large_std = 1 - cdf_mu_0_large_std;
 cdf_mu_1_small_std = normcdf(0.5, 1, 0.4);
 err_cdf_mu_1_small_std = cdf_mu_1_small_std;
 cdf_mu_1_large_std = normcdf(0.5, 1, 4);
 err_cdf_mu_1_large_std = cdf_mu_1_large_std;
```

1.5.5 Optimal kernel width

```
close all; clear all;

%% Generating the data; training and test data with two
    different standard
% derivations

% Dataset for std = 0.4
train1 = normrnd(0,0.4,[10,1]);
test1 = normrnd(0,0.4,[1000,1]);
train2 = normrnd(1,0.4,[10,1]);
test2 = normrnd(1,0.4,[1000,1]);

% Dataset for std = 4.0
train3 = normrnd(0,4,[10,1]);
test3 = normrnd(0,4,[1000,1]);
```

```
train4 = normrnd(1,4,[10,1]);
test4 = normrnd(1,4,[1000,1]);


%% cv partition
% Generation of different kernel widths
 h_val = [0.01:0.001:6];
 % kernel density function for the two std
 fx_small_std = zeros(length(h_val));
 fx_large_std = zeros(length(h_val));



% class 1
for i = 1:length(h_val)
    h = h_val(i);
    % sum of log-likelihood
    sum_p0_p1_small = 0;
    sum_p0_p1_large = 0;

    for  j = 1:length(train1)
    % training data
    train_0_small_std = train1;
    train_1_small_std = train3;
    train_0_large_std = train2;
    train_1_large_std = train4;

    % testing data
    train_0_small_std(j) = [];
    train_1_small_std(j) = [];
    train_0_large_std(j) = [];
    train_1_large_std(j) = [];

    % parzen estimation for 10 observation
    p_0_small_std = my_parzen(train1(j), train_0_small_std, h
        );
    p_1_small_std = my_parzen(train2(j), train_1_small_std, h
        );
    p_0_large_std = my_parzen(train3(j), train_0_large_std, h
        );
    p_1_large_std = my_parzen(train4(j), train_1_large_std, h
        );

    sum_p0_p1_small = sum_p0_p1_small + log(p_0_small_std) +
        log(p_1_small_std);
    sum_p0_p1_large = sum_p0_p1_large + log(p_0_large_std) +
        log(p_1_large_std);

    end

      fx_small_std(i) = sum_p0_p1_small;
```

```
        fx_large_std(i) = sum_p0_p1_large;
end

% optimal width for std = 0.4
xIndex_small_std = find(fx_small_std == max(fx_small_std), 1,
      'first');
maxH_small_std = h_val(xIndex_small_std);
% optimal width for std = 4
xIndex_large_std = find(fx_large_std == max(fx_large_std), 1,
      'first');
maxH_large_std = h_val(xIndex_large_std);

figure;
plot(h_val, fx_small_std , 'b-')
xlabel('Width h')
ylabel('sum of sum log-likelihood')
title('std = 0.4')
grid


figure;
plot(h_val, fx_large_std , 'g-')
xlabel('Width h')
ylabel('sum of sum log-likelihood')
title('std = 4.0')
grid
```

2 Machine Learning for Cell Type Classification using hand-coded features

2.1 Introduction

In this assignment we are studying dataset HEP2 which contains cell images. There are 430 training images and 433 test images of 6 different classes. The cell images are classified using a couple of different machine learning methodologies, such as random forest, k-nearest neighbour KNN, support vector machine and decision tree.

2.2 Method

A set of features have manually been designed as a feature vector. Each trained classifier is evaluated on both the training data and test data to determine training and testing accuracy. The extracted features is used to for the matrices $X1f$ for the training data and $X2f$ for the testing data. The the examples in the training set is then visualized to show the which regions each class occupies in the feature space.

The approach was to try all the four aforementioned classifiers with different combination of features such as mean value of the cell, the standard derivation of cell pixel histogram distribution, median, Some other more advanced features such as circularity, color range (achieved by calculating the difference between the maximum and minimum pixel intensity), perimeter the orientation of the cell, its extent and the eccentricity. All the features are normalized in range [0,1].

2.3 Result

The goodness of the features evaluated with the help of matrix scatter plots, to show the correlation of the features against each other. Based on the how the correlation looked like, the decision on which feature combination to keep was made.

After trying some features; mean intensity, standard deviation of intensities, median intensity, circularity, color range of cell, orientation, extent, perimeter and eccentricity, I found out that the best performance was achieved for the combination of the first five features only. Normalizing the features improved the performance significantly. Tab.4-6 show systems performance after adding some or eliminating some features in order to decide which feature combination to keep.

Tab.7 shows the performance for the five chosen features; mean intensity, standard deviation of intensities, median intensity, circularity, color range of cell. kNN seemed to work best for the hand-coded features and this specific classification features, therefore the approach was to optimize the features for this model. Highest accuracy achieved was 48.882% for the testing data with kNN.

Classification method	**Train accuracy**	**Test accuracy**
Decision tree	90.93%	31.178%
Random forest	100 %	32.794%
Support vector machine	50.465%	27.252%
k-nearest neighbour	100%	41.109%

Table 4: **Intial Results**: Performance of the system based on 2 features; mean intensity and standard derivation. In general, bad performance on test dataset for all four classifiers, lower than 50%. Best performance on test dataset is given by k-nearest neighbour with 41.109% accuracy. Over-fitting occurs on training dataset for random forest and k-nearest neighbour. Decision tree classifier performed better on training dataset than the other classifiers.

Classification method	**Train accuracy**	**Test accuracy**
Decision tree	91.163%	36.259%
Random forest	100 %	38.106%
Support vector machine	57.209%	32.794%
k-nearest neighbour	100%	45.497%

Table 5: **Second try results**: Performance of the system based on 3 features; mean intensity, standard derivation and circularity. In general, bad performance on test dataset for all four classifiers, lower than 50%. Best performance on test dataset is given by k-nearest neighbour with 45.497% accuracy. Over-fitting occurs on training dataset for random forest and k-nearest neighbour. Decision tree classifier performed better on training dataset than the other classifiers. Improved performance on test dataset for all the classifiers in comparison to having only mean intensity and standard derivation as features. Decision tree classifier performed better on training dataset than the other classifiers.

Classification method	**Train accuracy**	**Test accuracy**
Decision tree	94.186%	35.797%
Random forest	100 %	38.337%
Support vector machine	61.628%	33.256%
k-nearest neighbour	100%	47.575%

Table 6: **Third try results**: Performance of the system based on 4 features; mean intensity, standard derivation, circularity and median. In general, bad performance on test dataset for all four classifiers, lower than 50%. Best performance on test dataset is given by k-nearest neighbour with 47.575% accuracy. Improved performance on test dataset in comparison to having only mean intensity, standard derivation and circularity as features. Over-fitting occurs on training dataset for random forest and k-nearest neighbour. Decision tree classifier performed better on training dataset than the other classifiers.

Classification method	**Train accuracy**	**Test accuracy**
Decision tree	94.884%	32.564%
Random forest	100 %	39.492%
Support vector machine	65.581%	36.49%
k-nearest neighbour	100%	48.037%

Table 7: **Final Results**: Performance of the system based on 5 features; mean intensity, standard derivation, circularity, median and color range. In general, bad performance on test dataset for all four classifiers, lower than 50%. Best performance on test dataset is given by k-nearest neighbour with 48.037% accuracy. Improved performance on test dataset in comparison to having only mean intensity and standard derivation as features. Over-fitting occurs on training dataset for random forest and k-nearest neighbour. Decision tree classifier performed better on training dataset than the other classifiers.

Fig.12 shows the matrix of scatter plots for each combination of variables/features in the data set. The goodness of the features was investigated by looking at their correlation and how they take up the feature space. PCB is used for visualization and to determining the impact of the different features.

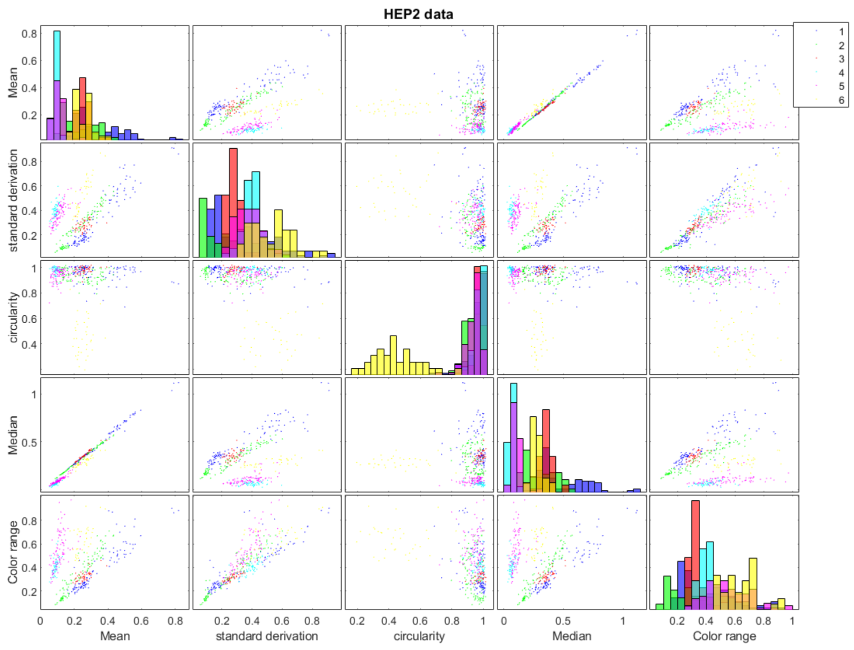
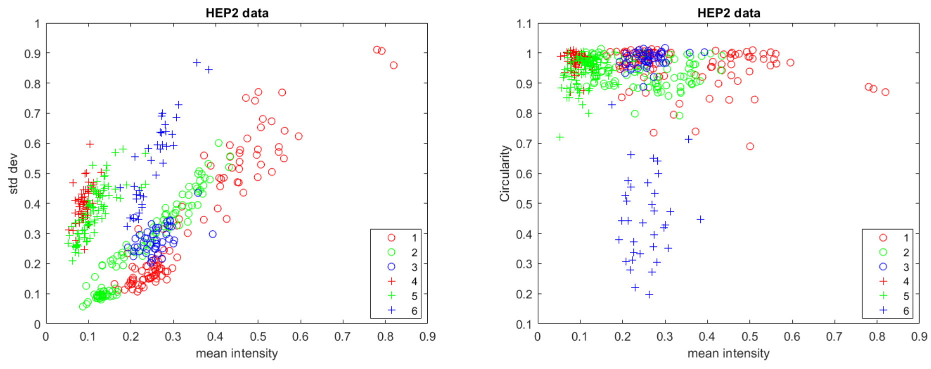


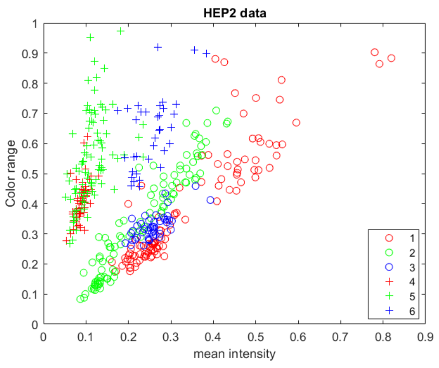
Figure 12: Matrix of scatter plots for each combination of variables in the data set. The different features that have be tried out are mean, standard derivation, median, circularity and color range.

For better visualisation, the figures below show the correlation between the five chosen features, two at a time.

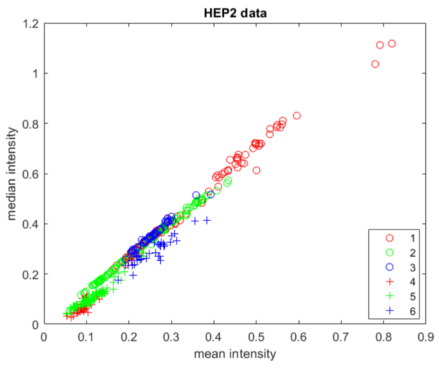


(a) (b)

Figure 13: (a) Correlation between mean intensity and standard derivation. (b) Correlation between mean intensity and circularity.

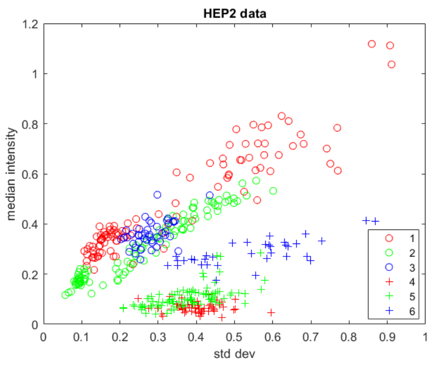


(a)

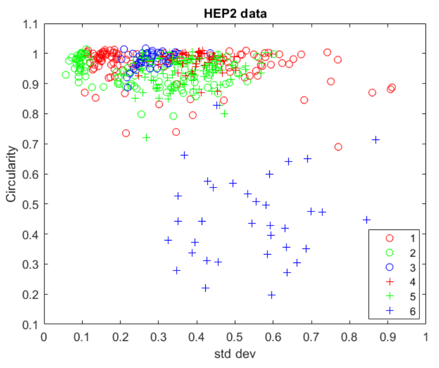


(b)

Figure 14: (a) Correlation between mean intensity and color range (b) Correlation between mean intensity and median intensity.

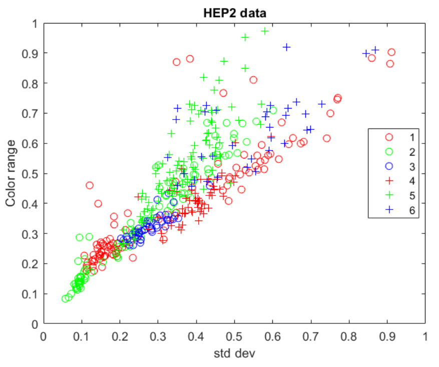


(a)

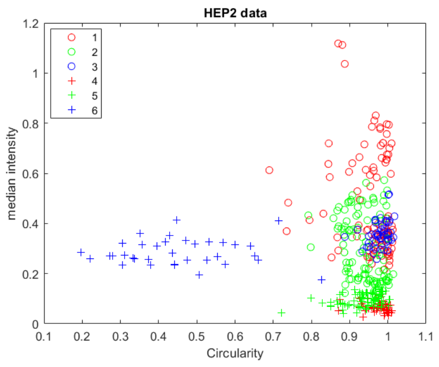


(b)

Figure 15: (a) Correlation between standard derivation and median intensity. (b) Correlation between standard derivation and circularity.

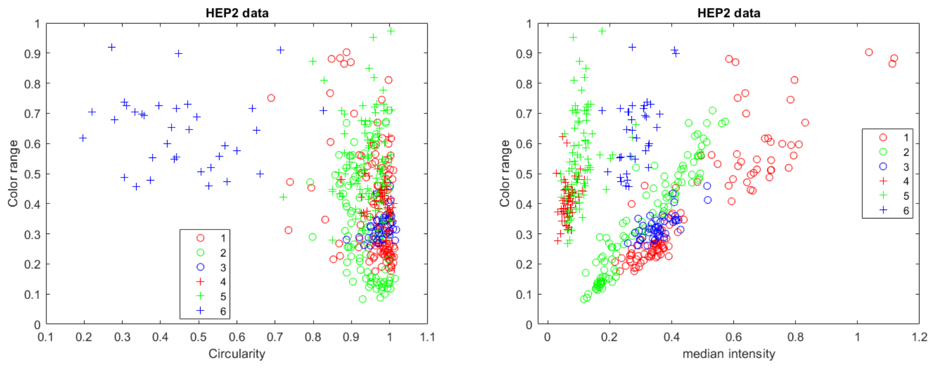


(a)



(b)

Figure 16: (a) Correlation between standard derivation and color range (b) Correlation between circularity and median intensity.



(a) (b)

Figure 17: (a) Correlation between circularity and color range (b) Correlation between median intensity and color range.

2.4 Conclusion and Discussion

According to Tab.4,5, 6 and 7, k-nearest neighbour classifier has the highest testing accuracy in comparison to the other three classifiers. The performance of KNN was increased by adding more features, specially circularity, color range and median. The performance of random forest and support vector machine could also be increased by the additional features. However, the testing accuracy of the decision tree classifier slightly decreased by adding the 'color range' feature.

KNN does not build any classification model, it learns from the training instead and start to build data first after the unseen test observation. For training dataset the training accuracy is 100%, since the nearest neighbour for $K = 1$, to any data point is itself. This means that the model is over-fitting the boundaries and the error rate reaches a minimum, and it is increasing with the number of neighbours. Same argument applies to random forest classifier.

According to Fig.12, best combination of features is mean, standard derivation, median, circularity and color range. Using these feature showed a slightly good performance for kNN model. Therefore, these features and kNN model are recommended to be used for the classification task of the dataset provided. However, hand-coding meaningful features is difficult. For the ML methods, it is important to normalize the features. Having few data samples makes the classification task difficult and achieving a good performance can be challenging. The classification result achieved can be better when using for example CNN, since this model is performing a more complex classification and can perform slightly better. The classification task was challenging since it was difficult to find good features and method to use in order to get good performance. Therefore, for future data and in order to achieve good performance (at least more than 60 %) it might be good to try other method such as CNN since it automatically detects the important features without any human supervision.

2.5 Code

2.5.1 Script for testing different classifier and feature combinations

```
%% Load data (training data X1 - labels Y1)
```

```
% test data X2 - labels Y2

close all
clear all
load(fullfile('databases','hep_proper_mask'));
X1_masks = Y1;
X2_masks = Y2;
load(fullfile('databases','hep_proper'));

%% Use hand made features

nr_of_training_images = size(X1,4)
for i = 1:nr_of_training_images,
    [fv,str]=get_features(X1(:,:,1,i),X1_masks(:,:,1,i));
    X1f(i,:)=fv;
end

nr_of_test_images = size(X2,4)
for i = 1:nr_of_test_images,
    [fv,str]=get_features(X2(:,:,1,i),X2_masks(:,:,1,i));
    X2f(i,:)=fv;
end


figure;
xnames = {'Mean', 'standard derivation' , 'circularity', '
    Median', 'Color range'};
gplotmatrix(X1f,[],Y1, [], [], [],[], 'grpbars', xnames)
title('HEP2 data')

% %% DIMENSIONALITY REDUCTION
% [u,s,v]=svd(allX);
% allX_reduced = u(:,1:2)'*allX;
% figure;
% for kk = 1  :size(x1f,2)
%     %kk
%     %text(allX_reduced(1,kk),allX_reduced(2,kk),alfabet(
    allY(kk)))
%     hold on;
%     %pause;
% end
% axis([min(allX_reduced(1,:))-0.5 max(allX_reduced(1,:))+0.5
     min(allX_reduced(2,:))-0.5 max(allX_reduced(2,:))+0.5])
%% Visualize the data

% figure(1);
% gscatter(X1f(:,1), X1f(:,2), Y1 ,'rgbrgb','ooo+++');
% xlabel(str{1});
% ylabel(str{2});


```

```matlab
%% Train machine learning models to data

%% Fit a decision tree model
disp(' ');
disp('Decision tree');
model1 = fitctree(X1f(:,1:length(fv)),Y1,'PredictorNames',str
    );
% Test the classifier on the training set
[Y_result1,node_1] = predict(model1,X1f);
accuracy1 = sum(Y_result1 == Y1)/numel(Y_result1);
disp(['The accuracy on the training set: ' num2str(accuracy1)
    ]);
% Test the classifier on the test set
[Y_result2,node_2] = predict(model1,X2f);
accuracy2 = sum(Y_result2 == Y2)/numel(Y_result2);
disp(['The accuracy on the test set: ' num2str(accuracy2)]);

%% Fit a random forest model
disp(' ');
disp('Random forest');
model2 = TreeBagger(50,X1f,Y1,'OOBPrediction','On',...
    'Method','classification')
% Test the classifier on the training set
[Y_result1,node_1] = predict(model2,X1f);
accuracy1 = sum(Y_result1 == Y1)/numel(Y_result1);
disp(['The accuracy on the training set: ' num2str(accuracy1)
    ]);
% Test the classifier on the test set
[Y_result2,node_2] = predict(model2,X2f);
accuracy2 = sum(Y_result2 == Y2)/numel(Y_result2);
disp(['The accuracy on the test set: ' num2str(accuracy2)]);

%% Fit a support vector machine model
disp(' ');
disp('Suppport vector machine');
model3 = fitcecoc(X1f,Y1);
% Test the classifier on the training set
[Y_result1,node_1] = predict(model3,X1f);
accuracy1 = sum(Y_result1 == Y1)/numel(Y_result1);
disp(['The accuracy on the training set: ' num2str(accuracy1)
    ]);
% Test the classifier on the test set
[Y_result2,node_2] = predict(model3,X2f);
accuracy2 = sum(Y_result2 == Y2)/numel(Y_result2);
disp(['The accuracy on the test set: ' num2str(accuracy2)]);

%% Fit a k-nearest neighbour model
disp(' ');
disp('k-nearest neighbour');
model4 = fitcknn(X1f,Y1);
% Test the classifier on the training set
```

```
[Y_result1,node_1] = predict(model4,X1f);
accuracy1 = sum(Y_result1 == Y1)/numel(Y_result1);
disp(['The accuracy on the training set: ' num2str(accuracy1)
    ]);
% Test the classifier on the test set
[Y_result2,node_2] = predict(model4,X2f);
accuracy2 = sum(Y_result2 == Y2)/numel(Y_result2);
disp(['The accuracy on the test set: ' num2str(accuracy2)]);
```

2.5.2 Features

```
function [F, STR] = get_features(image, mask)
%keyboard;
F(1) = mean(image(find(mask)))/200;
STR{1} = 'mean intensity';

F(2) = std(image(find(mask)))/50;
STR{2} = 'std dev';

% STR{4} = 'min intensity';
[m,n] = size(mask);
s = m*n;
props = regionprops(mask, 'all');

%sum of the intensities smaller than a certain value
% F(5) = sum(image(find(mask)<0.20));
% STR{5} = 'intensity < 0.40';

% Circularity
F(3) = props.Circularity;
STR{3} = 'Circularity';

% Perimeter
% F(4) = (props.Perimeter/s - 0.43)*35;
% STR{4} = 'Perimeter';
% Median
F(4) = median(image(find(mask)))/150;
STR{4} = 'median intensity';

% Orientation
% F(5) = abs(props.Orientation);
% STR{5} = 'Orientation';

%maximum of the intensities
% F(4) = max(image(find(mask)));
% STR{4} = 'max intensity';
% %minimum of the intensities
% F(5) = min(image(find(mask)));
% STR{5} = 'min intensity';

% Color range
min_F= min(image(find(image)));
```

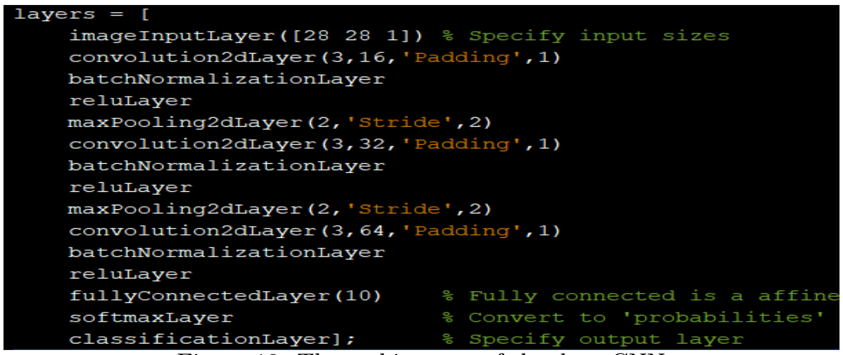
```
max_F = max(image(find(image)));
F(5) = (max_F - min_F)/255;
STR{5} = 'Color range';
% Need to name all features.
 assert(numel(F) == numel(STR));
```

3 Classification using Deep Learning (CNN)

3.1 Digit Datasets

The training and testing accuracy for not so deep CNN network is shown in Fig.8 together with the performance result for the deeper network. The training accuracy is the accuracy of a model on examples it was constructed on. While the test accuracy is the accuracy of a model on examples it has not seen before. Training accuracy means that identical samples are used both for training and testing, while test accuracy represents that the trained model identifies independent samples that were not used in training.

In order to make the CNN network deeper, I added 3 conventional layers of size $3x3$ with a padding of 1. 'minibatchsize' has been included in the training options since adding this parameter helped in improving the testing accuracy. The deep CNN architecture is shown in Fig.18



```
layers = [
    imageInputLayer([28 28 1]) % Specify input sizes
    convolution2dLayer(3,16,'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,32,'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,64,'Padding',1)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(10)     % Fully connected is a affine
    softmaxLayer                % Convert to 'probabilities'
    classificationLayer];       % Specify output layer
```

Figure 18: The architecture of the deep CNN

Configuration	**Training accuracy**	**Testing accuracy**
Notsodeep CNN	84.44%	72.7%
Deep CNN	91.98%	88.7%

Table 8: Training and testing accuracy for the not-so-deep architecture and for the deeper architecture.

The following learning options were used to achieve the performance shown in tab.8:

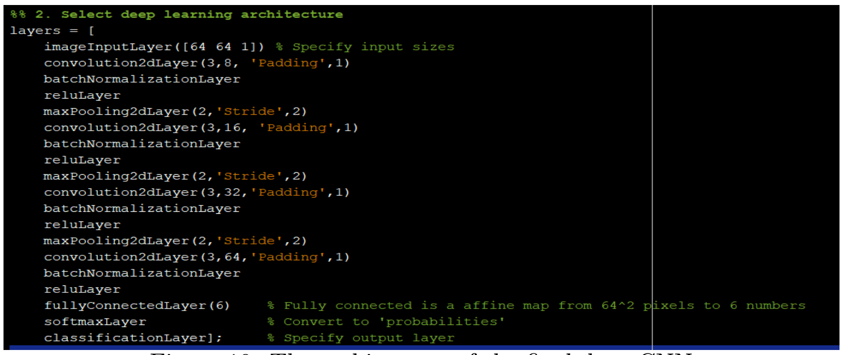
- miniBatchSize = 512
- max_epochs = 30
- learning_rate = 1
- optimizer = SGDM

As shown in tab.8, the training and testing accuracy improved by adding more convolution layers to the network.

3.2 HEP2 dataset

The CNN classifier used on the digits dataset is tested on another dataset called HEP2. The initial step was to test the deep CNN network with the learning options tested previously on the digit dataset, with an initial learning rate of 1, 30 epochs and mini-batch size of 512. The resulting training accuracy was 23.953% and testing accuracy 27.021%. The performance needed therefore to be improved since the trained classifier did not generalize well to the test set. To improve the performance, the approach was to experimenting with the learning options; minimum batch size, maximum number of epochs and learning rate. Besides, the data was augmented by adding a random rotation in range $[0, 360]$. 'Adam' optimizer, has also been tested besides to SGDM.

Fig.19 shown below illustrates the architecture of the deep CNN used. The input images are of size 64x64. Three convolution layers with *padding* of 1, to add layers of zeros to the input images in order to preserved information on the borders of the images. A max-pooling layer with stride of 2 has also been added. The connected layer contains 6 nodes since we have 6 different classes/labels.



```
%% 2. Select deep learning architecture
layers = [
    imageInputLayer([64 64 1]) % Specify input sizes
    convolution2dLayer(3,8, 'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,16, 'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,32,'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,64,'Padding',1)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(6)     % Fully connected is a affine map from 64^2 pixels to 6 numbers
    softmaxLayer               % Convert to 'probabilities'
    classificationLayer];      % Specify output layer
```

Figure 19: The architecture of the final deep CNN

The next step was to experimenting with the learning parameters, first without adding augmentation and then adding augmentation.

As shown in tab.9, the best minimum batch size was found to be 15. In combination with maximum number of epochs of 80 and learning rate of 0.01. The best possible accuracy could be achieved with this combination. It could be noticed that by increasing the epochs the testing accuracy improved. Training the network with larger number of epoch than 80 impacts the accuracy negatively as the model starts to overfit, the training accuracy was close to 100% and the testing accuracy decreases. Besides, I experimented further with the learning rate and found out that decreasing this parameter to lower rate than 0.01 resulted in lower testing accuracy.

The size of minimum batch was reduced from 512 to 14, the number of maximum epochs increased to 80, and the learning rate reduced from 1 to 0.01. Training accuracy 100% and testing accuracy 56.12% could be achieved. The classifier fits perfectly to the training data causing over-fitting since it will not generalize well to the unseen data. Testing accuracy with these training options gave the best performance on the testing data in comparison to the the other configuration that had been tried out as shown in Tab.9.

However, before reaching this result different alternatives of minimum batch size have been tried out, with the conclusion that the increasing this hyperparameter results in lower testing accuracy since the model will slowly converge to the global minimum. Moreover, the number of epochs affects the testing accuracy since it could been shown that increasing this hyperparameter together with having a lower minimum batch size can lead to higher accuracy. There is however a trade-off in the tuning of the these two parameters. Another parameter that has been experimenting with is the learning rate. It has been shown that decreasing this parameter results in better testing accuracy. But, there is a limit to how small the learning rate should be. Decreasing the learning rate further made the performance worse since the testing accuracy reduced. The highest testing accuracy was 56.12%. However, it could be observed that the model started to overfit as the testing accuracy increased. Regularization can be used to add penalty and to avoid overfitting.

Configuration	**Training accuracy**	**Testing accuracy**
3 CONV layers, minibatch size = 15, max_epochs = 70, learning rate = 0.01 no augmentation	100%	55.889%
3 CONV layers, minibatch size = 15, max_epochs = 80, learning rate = 0.01 no augmentation	100%	56.12%
3 CONV layers, minibatch size = 32, max_epochs = 80, learning rate = 0.01 no augmentation	100%	47.344%
3 CONV layers, minibatch size = 15, max_epochs = 80, learning rate = 0.001 no augmentation	100%	46.651%
3 CONV layers, minibatch size = 15, max_epochs = 100, learning rate = 0.01 no augmentation	100%	51.039%
3 CONV layers, minibatch size = 16, max_epochs = 70, learning rate = 0.01 no augmentation	100%	49.192%
3 CONV layers, minibatch size = 16, max_epochs = 60, learning rate = 0.01 no augmentation	100%	50.808%
3 CONV layers, minibatch size = 16, max_epochs = 60, learning rate = 0.1 no augmentation	23.488%	15.704%
3 CONV layers, minibatch size = 16, max_epochs = 60, learning rate = 0.0001 no augmentation	98.372%	40.878%
3 CONV layers, minibatch size = 15, max_epochs = 60, learning rate = 0.0001 no augmentation	98.837%	36.952%

Table 9: Training and testing accuracy for the own CNN with the non augmentation data.

After testing different learning options I could not achieve a better result. It was difficult to avoid overfitting. Having more data could improve the result, but it is not always easy to get large data in real life. My approach on how to choose the number of epochs and learning rate was based on trial and error of different combination while evaluation the performance.

Adding data augmentation makes the model generalizing better to the test data than without it. Data augmentation is an important method which is used to increase the diversity of the training set and that could significant for small dataset. The data has been randomly rotated in the range $[0, 360]$.

For the automated data, the best size of the minimum batch was found to be 15. In combination with maximum number of epochs of 70 and learning rate of 0.01. For this combination, the best possible accuracy could be achieved was 68.36% and the training accuracy was 99.302%. This result was achieved after experimenting with the three aforementioned hyperparameters; learning rate, number of epochs and minimum batch size. The choice of the parameters was initially based on the parameters used before adding data augmentation. Here, by keeping the learning rate, 0.01, and the batch size, 15, as before and only reducing the number of epochs from 80 to 70 showed a large improvement on the testing accuracy, with an increase of more than 10%. However, it was difficult to avoid overfitting even with adding data augmentation. By reducing the learning rate from 0.01 to 0.004 the training accuracy could be reduced slightly from 99.302% to 96.512%. However, the model was still overfitting since it performs better on the training data. Some of the parameter options that have been tried can be seen in 10.

Configuration	Training accuracy	Testing accuracy
3 CONV layers, minibatch size = 15, max_epochs = 70, learning rate = 0.01 augmentation	99.302%	68.36%
3 CONV layers, minibatch size = 15, max_epochs = 80, learning rate = 0.01 augmentation	98.837%	65.358%
3 CONV layers, minibatch size = 15, max_epochs = 60, learning rate = 0.01 augmentation	98.837%	60.739%
3 CONV layers, minibatch size = 16, max_epochs = 70, learning rate = 0.01 augmentation	99.302%	66.975%
3 CONV layers, minibatch size = 15, max_epochs = 70, learning rate = 0.1 augmentation	51.163%	37.182%
3 CONV layers, minibatch size = 15, max_epochs = 70, learning rate = 0.001 augmentation	99.07%	57.968%
3 CONV layers, minibatch size = 15, max_epochs = 70, learning rate = 0.005 augmentation	99.07%	59.122%
3 CONV layers, minibatch size = 16, max_epochs = 80, learning rate = 0.005 augmentation: 'RandRotation', [0 360]	99.302%	65.358%
3 CONV layers, minibatch size = 16, max_epochs = 80, learning rate = 0.004 augmentation: 'RandRotation', [0 360]	98.837%	67.206%
3 CONV layers, minibatch size = 16, max_epochs = 70, learning rate = 0.004 augmentation: 'RandRotation', [0 360]	96.512%	61.894%

Table 10: Training and testing accuracy for the own CNN with non augmentation data.

It has been shown that having a larger learning rate than 0.01 decreases the testing and training accuracy. Smaller learning rate than 0.01 had not show any further improvement in the result. Therefore learning rate of 0.01 was chosen after some trial and error experimenting. Increased testing accuracy was achieved as the number of maximum epochs was reduced from 80 to 70. No further improvement could be achieved with larger or smaller number of epochs than 70. The testing accuracy decreasing as the number of epochs increased to something larger than 70 epochs.

3.3 Code

```
close all
clear all
clc
%% 1. Load mnist data
load(fullfile('databases','hep_proper_mask'));
load(fullfile('databases','hep_proper'));
train_im = X1;
```

```matlab
test_im = X2;
train_classes = Y1;
test_classes = Y2;

% Augumentation of data
augmenter = imageDataAugmenter( ...
    'RandRotation',[0 360]);
imageSize = [64 64 1];
% Generating the new augmentated data
auimds = augmentedImageDatastore(imageSize,train_im,train_classes,'DataAugmentation',augmenter);
%% 2. Select deep learning architecture
layers = [
    imageInputLayer([64 64 1]) % Specify input sizes
    % convolution layer of size 3x3, padding of 1 and max pooling with
    % stride 2
    convolution2dLayer(3,8, 'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,16, 'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,32,'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,64,'Padding',1)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(6)    % Fully connected is a affine map from 64^2 pixels to 6 numbers
    softmaxLayer              % Convert to 'probabilities'
    classificationLayer];     % Specify output layer
%% 3. Train deep learning network
miniBatchSize = 16;
max_epochs = 80;           % Specify how long we should optimize
learning_rate = 0.004;     % Try different learning rates
options = trainingOptions( 'sgdm',...
    'miniBatchSize', miniBatchSize, ...
    'MaxEpochs',max_epochs,...
    'InitialLearnRate',learning_rate, ...
    'Shuffle', 'every-epoch');
    %'Plots', 'training-progress');
%net = trainNetwork(train_im, train_classes, layers, options);
net = trainNetwork(auimds, layers, options);
%% 4. Test the classifier on the test set
```

```
[Y_result2,scores2] = classify(net,test_im);
accuracy2 = sum(Y_result2 == test_classes)/numel(Y_result2);
disp(['The accuracy on the test set: ' num2str(accuracy2)]);
%% Test the classifier on the training set
[Y_result1,scores1] = classify(net,train_im);
accuracy1 = sum(Y_result1 == train_classes)/numel(Y_result1);
disp(['The accuracy on the training set: ' num2str(accuracy1)
    ]);
```