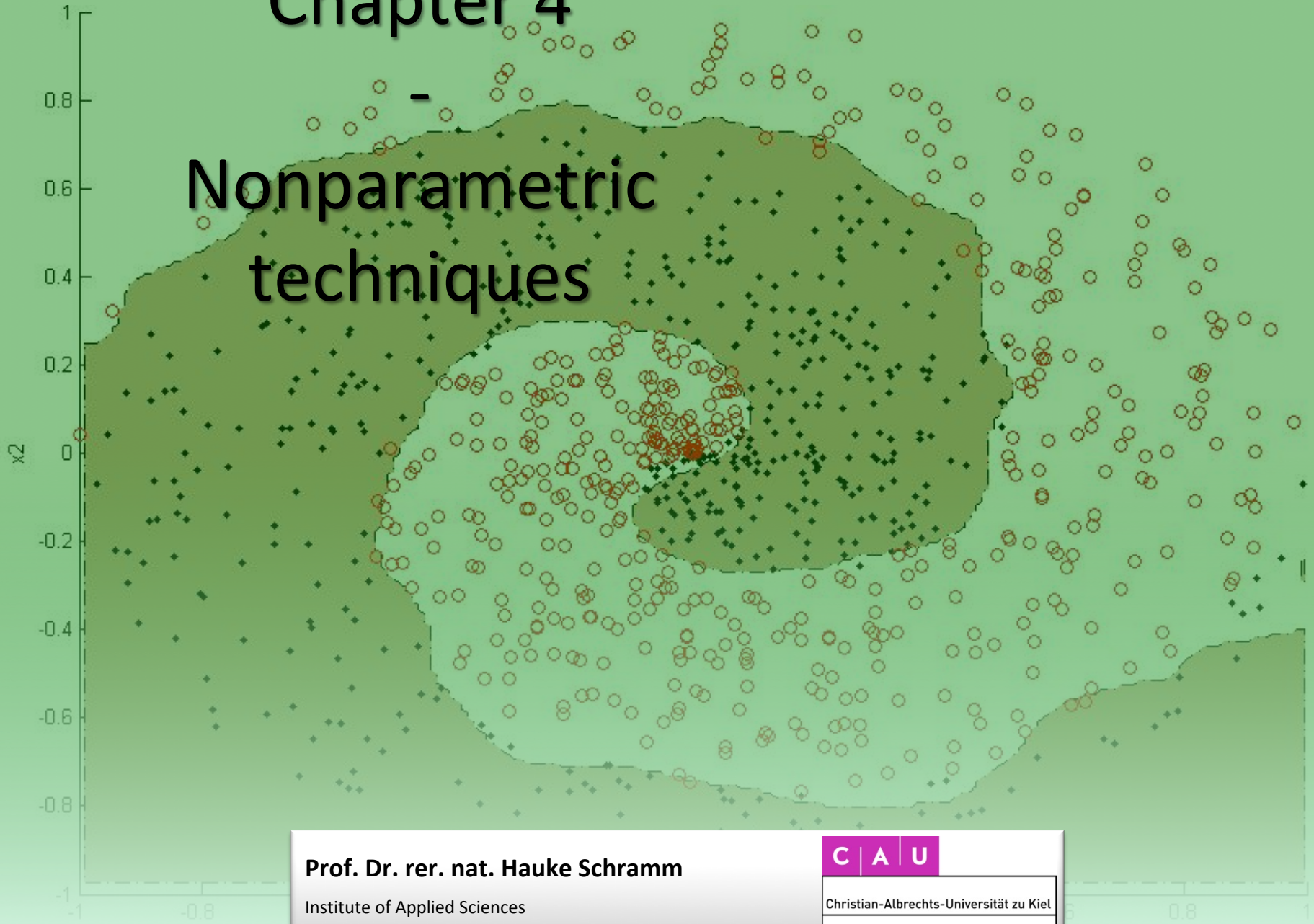
Chapter 4 - Nonparametric techniques



Prof. Dr. rer. nat. Hauke Schramm

Institute of Applied Sciences

University of Applied Sciences Kiel

C | A | U

Christian-Albrechts-Universität zu Kiel

Institut für Informatik

Outline of Chapter 4

4. Nonparametric techniques
    1. Introduction
    2. Parzen windows
    3. Nearest neighbor classification

4. Nonparametric techniques

4.1 Introduction

Main features of **parametric techniques** (see Chapter 3) are:

- Known parametric form (e.g. Gaussian)
- Strong reduction of training data to few parameters (e.g. mean, variance)

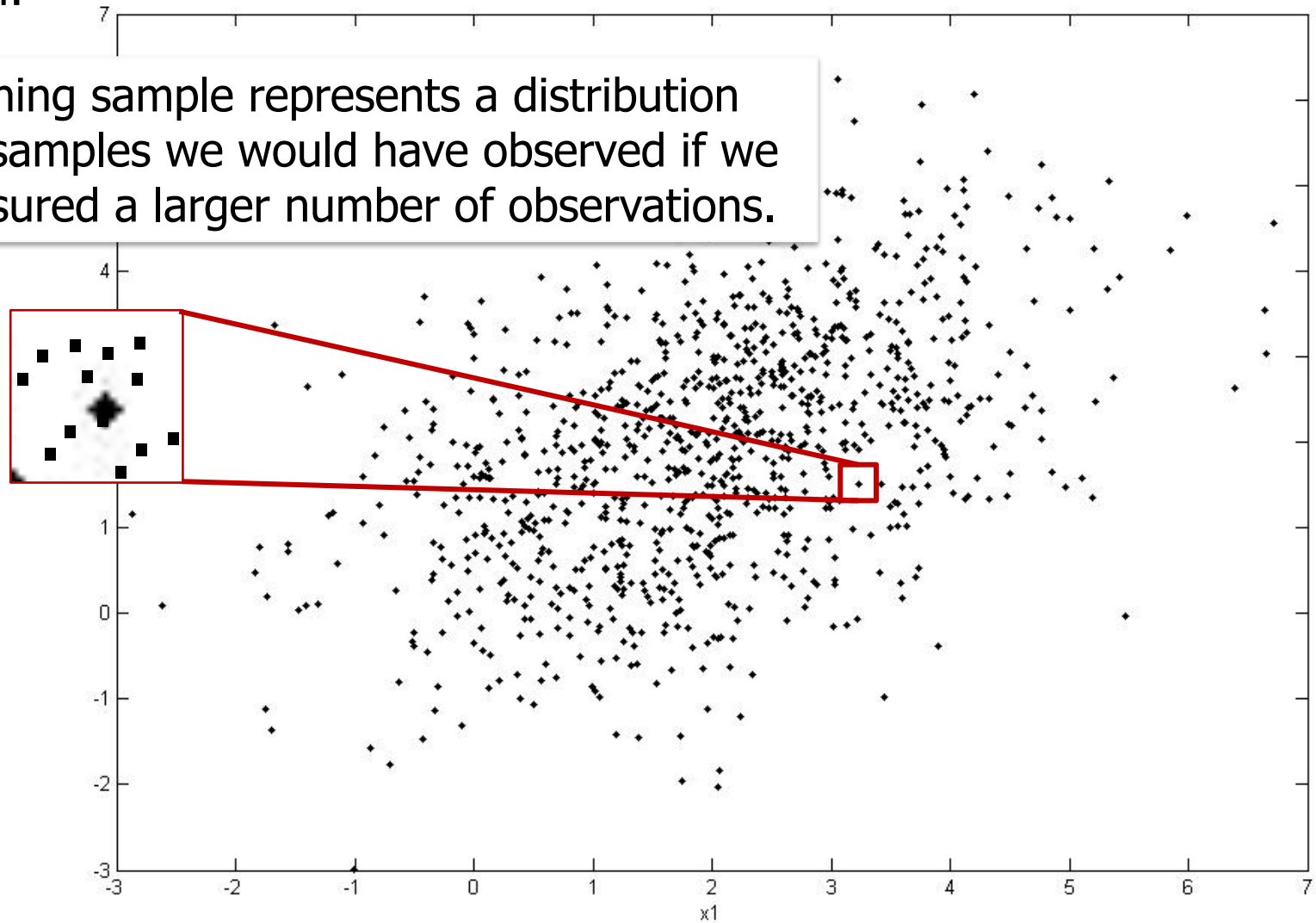
Non-parametric or **model-free** methods

- do not make any assumptions about the form of the distribution
- are flexible enough to approximate the shape of the true probability distributions with arbitrary accuracy.
- require, however, lots of computational resources (memory, runtime).

4.2 Parzen windows

Assumption:

Each training sample represents a distribution of other samples we would have observed if we had measured a larger number of observations.

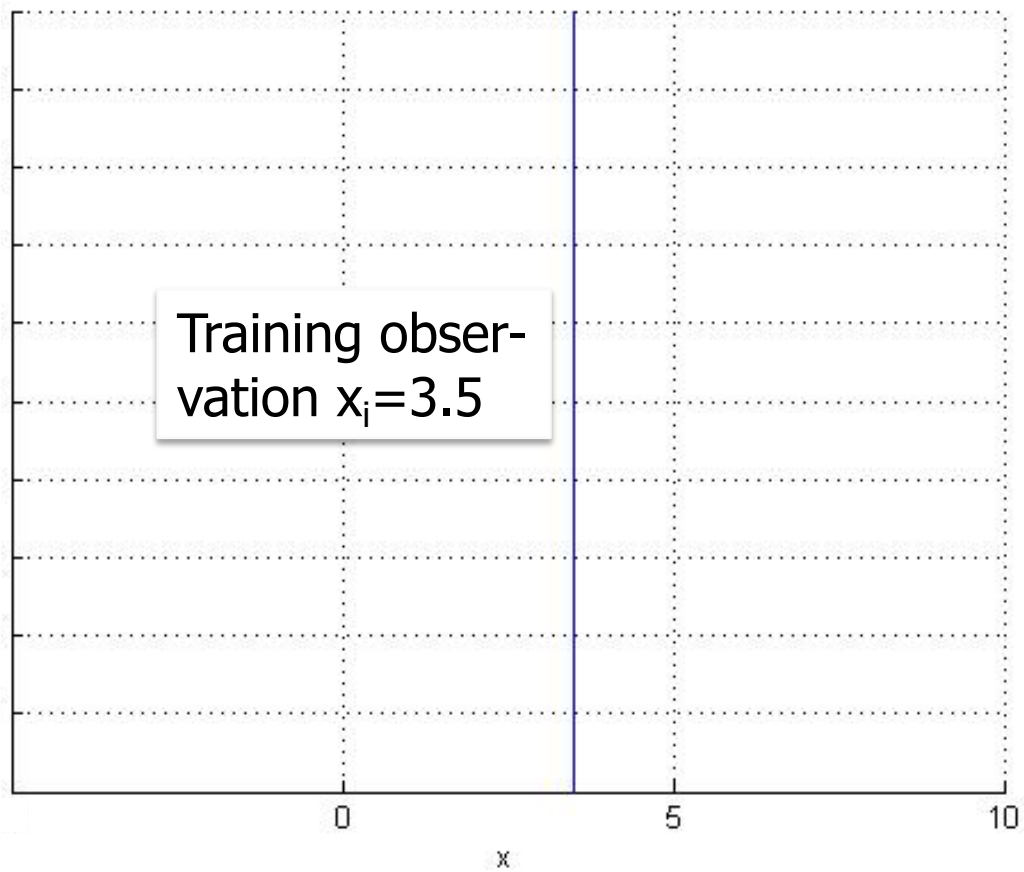


4.2 Parzen windows

Idea of Parzen windows:

- Each true training sample is replaced by histogram of artificial observations

Example in 1-D:

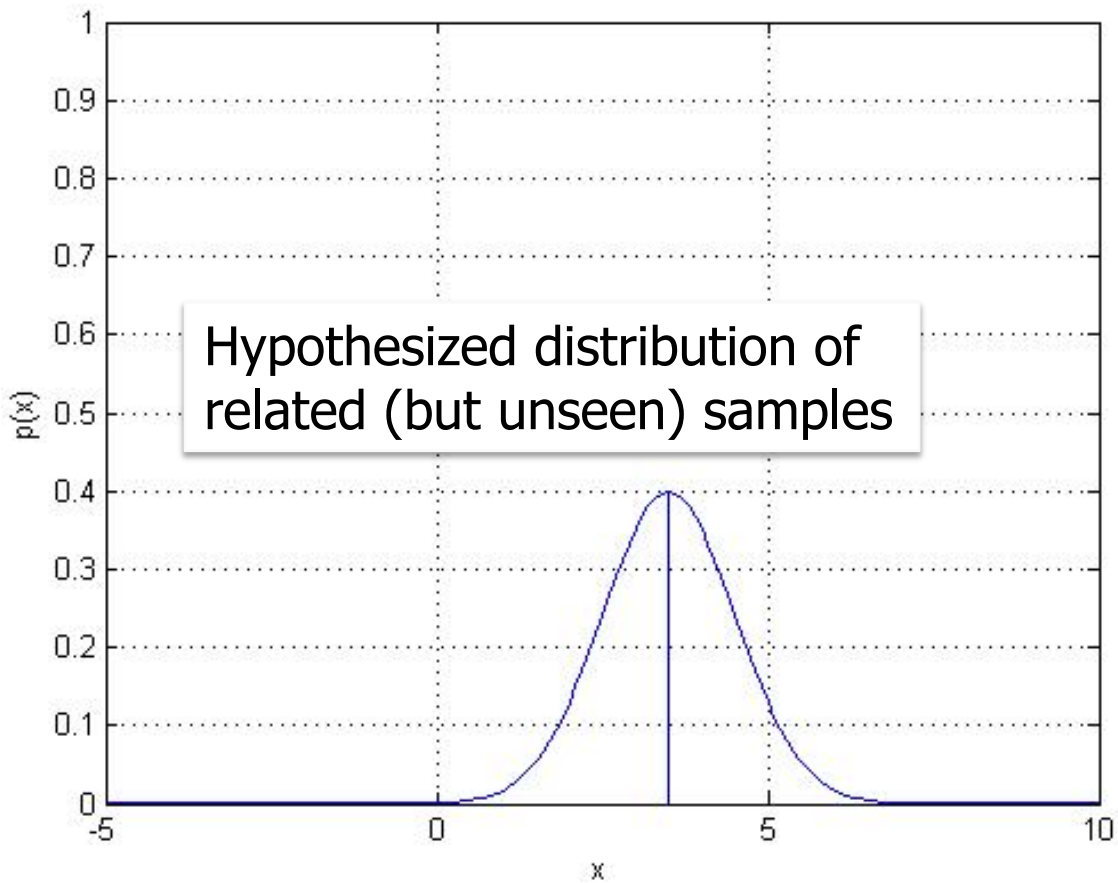


4.2 Parzen windows

Idea of Parzen windows:

- Each true training sample is replaced by histogram of artificial observations

Example in 1-D:

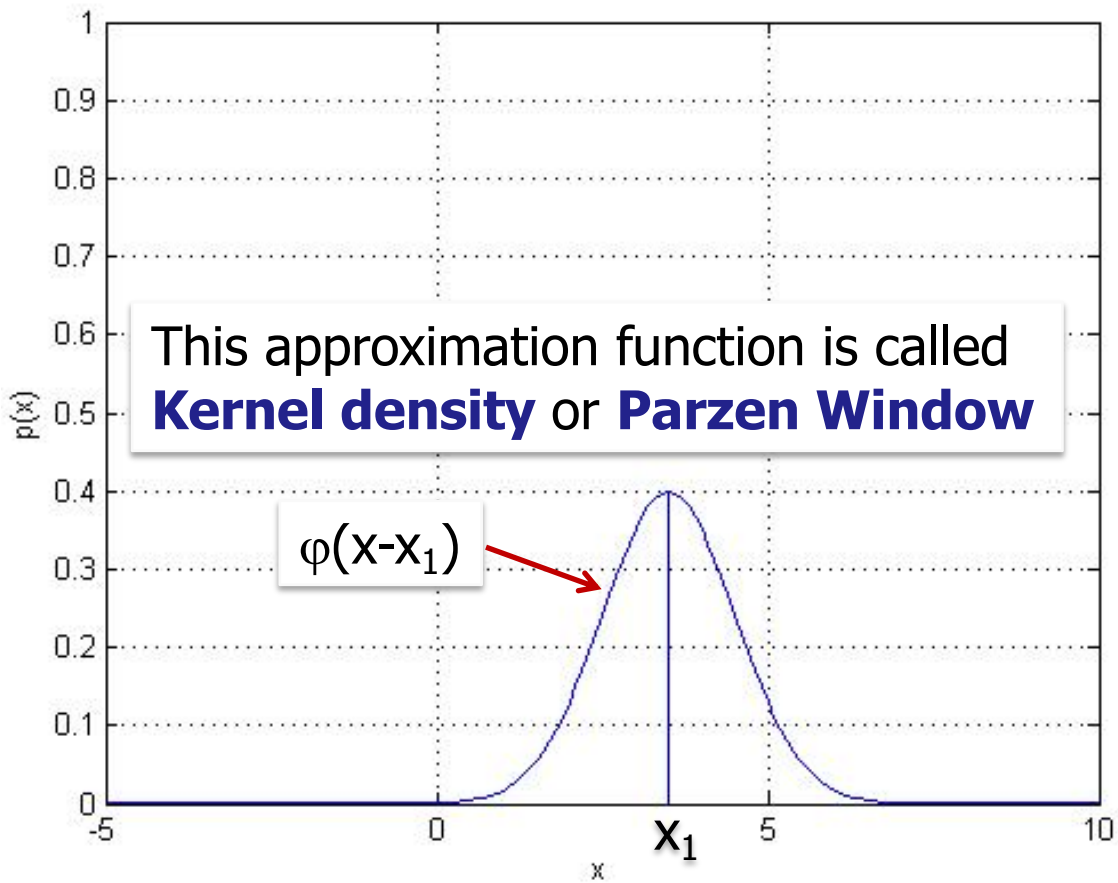


4.2 Parzen windows

Idea of Parzen windows:

- Each true training sample is replaced by histogram of artificial observations

Example in 1-D:



4.2 Parzen windows

Idea of Parzen windows:

- Each true training sample is replaced by histogram of artificial observations
- Parzen windows of all trainings samples are combined to form an estimation of the unknown class-conditional density (likelihood) $p(x \mid \omega)$

4.2 Parzen windows

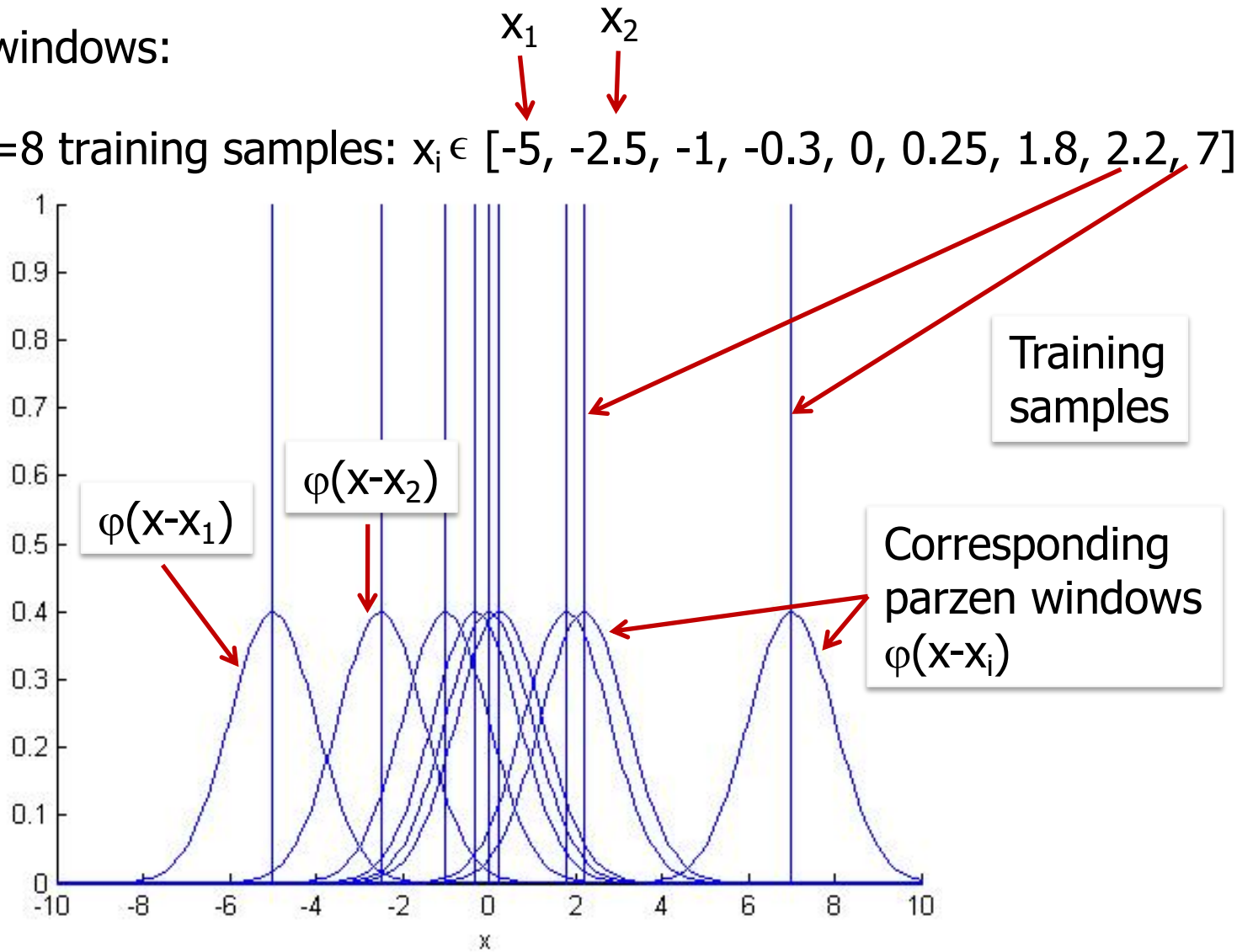
Idea of Parzen windows:

Example with n=8 training samples: $x_i \in$ [-5, -2.5, -1, -0.3, 0, 0.25, 1.8, 2.2, 7]

4.2 Parzen windows

Idea of Parzen windows:

Example with n=8 training samples: $x_i \in$ [-5, -2.5, -1, -0.3, 0, 0.25, 1.8, 2.2, 7]



4.2 Parzen windows

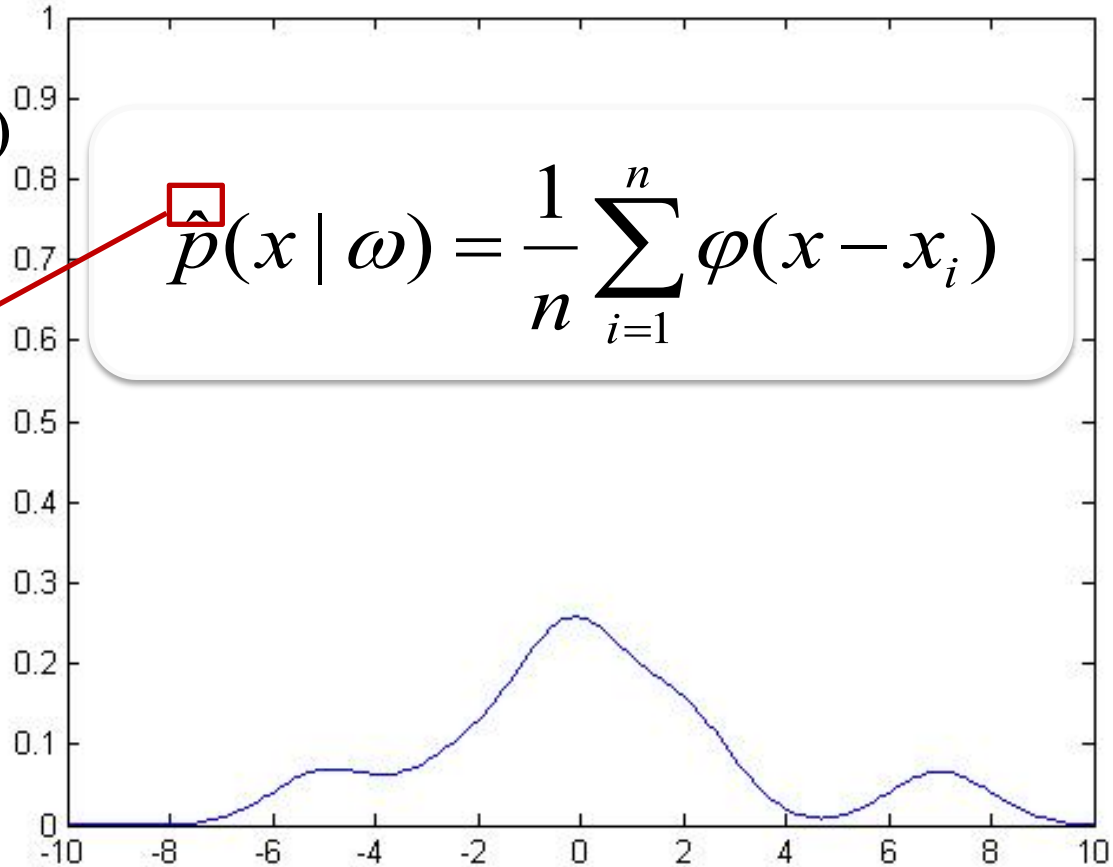
Estimate class-conditional probability of class ω:

Sum over individual Parzen windows and normalize (divide by sample number)

$\hat{p}(x \mid \omega)$

$$\hat{p}(x \mid \omega) = \frac{1}{n} \sum_{i=1}^{n} \varphi(x - x_i)$$

Estimated likelihood (not the true likelihood)

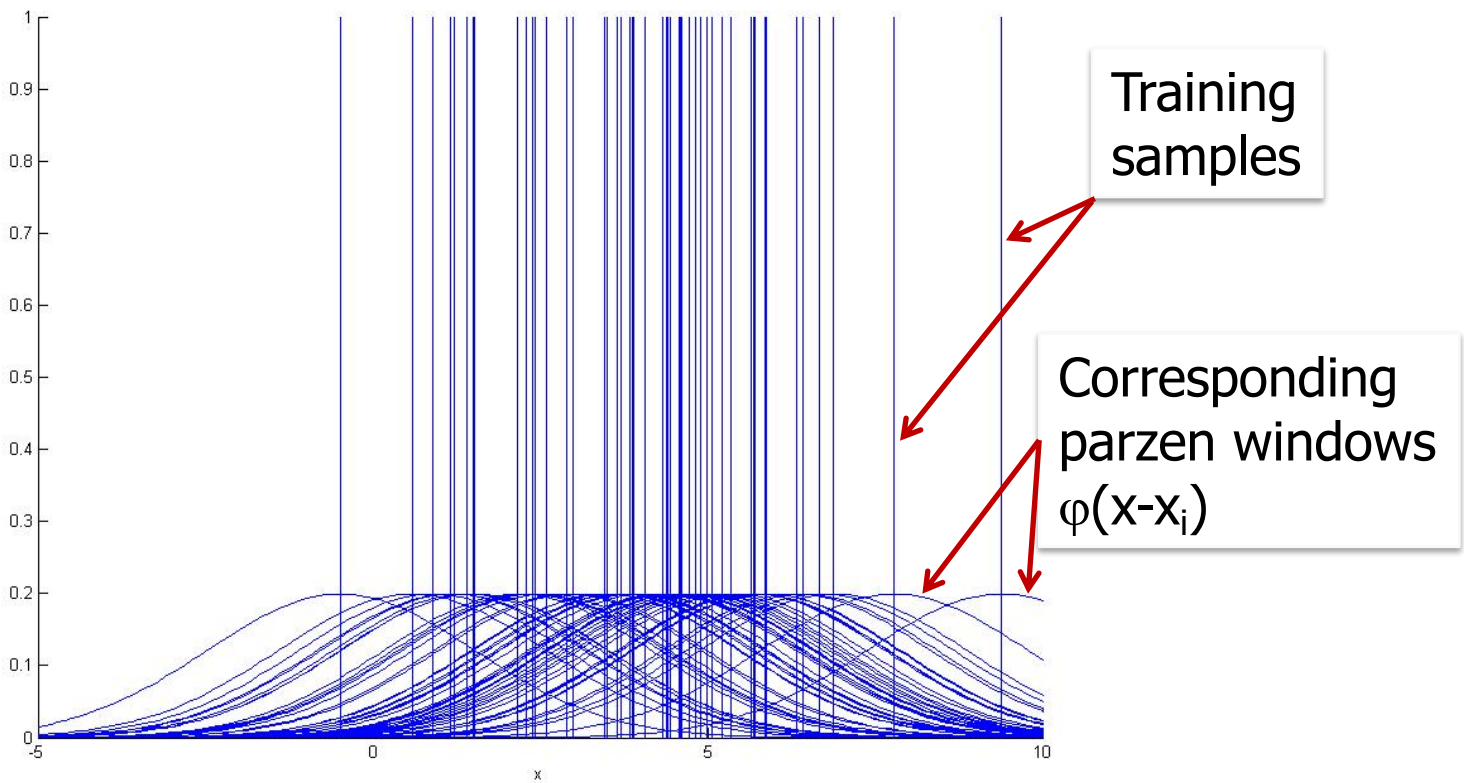


4.2 Parzen windows

$$\hat{p}(x \mid \omega) = \frac{1}{n} \sum_{i=1}^{n} \varphi(x - x_i)$$

Matlab example:

- Generate n=50 normally distributed 1-D training samples
- Estimate underlying true Gaussian density using Parzen windows



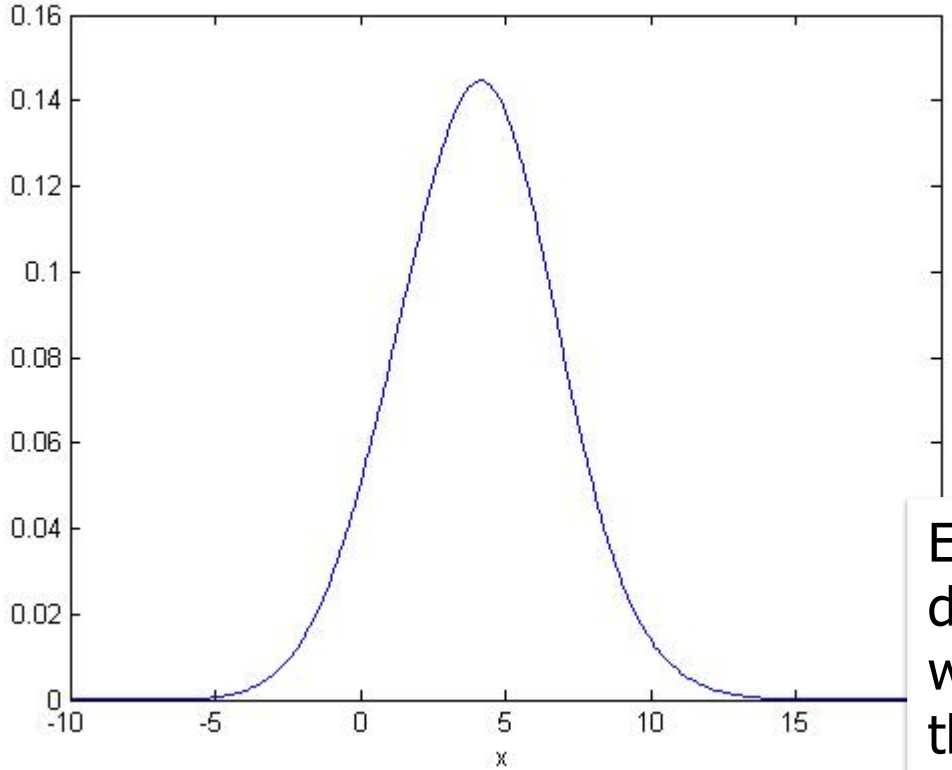
4.2 Parzen windows

$$\hat{p}(x \mid \omega) = \frac{1}{n} \sum_{i=1}^{n} \varphi(x - x_i)$$

Matlab example:

- Generate n=50 normally distributed 1-D training samples
- Estimate underlying true Gaussian density using Parzen windows

$\hat{p}(x \mid \omega)$



Sum over the Parzen windows and normalization

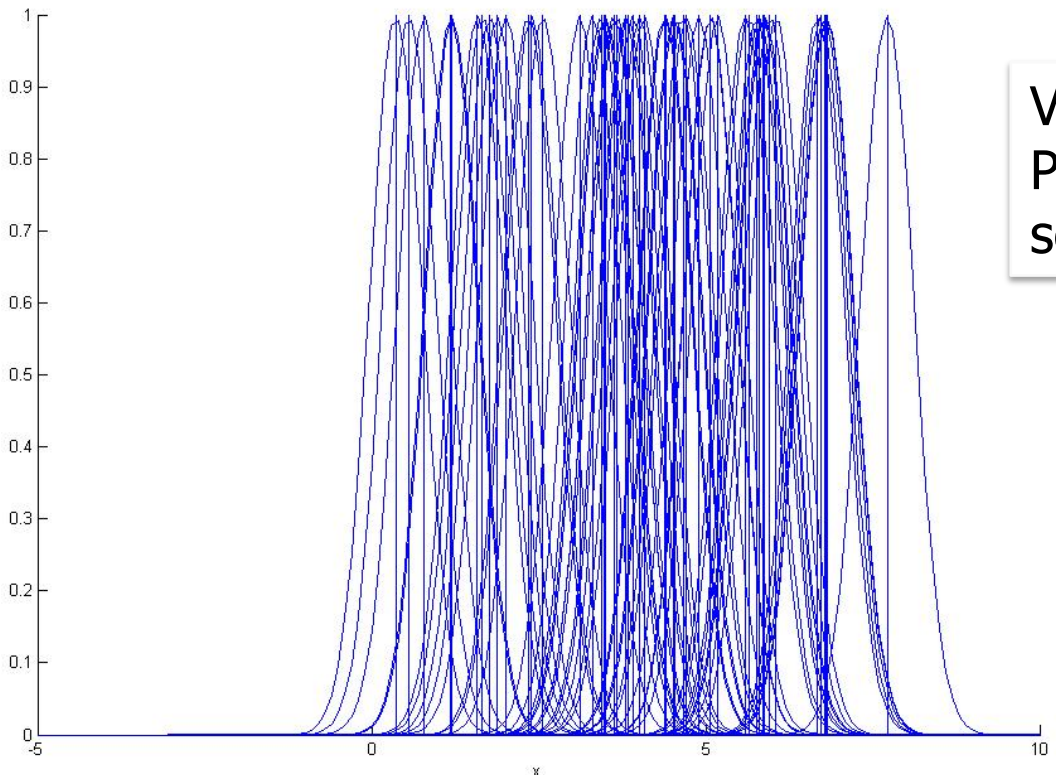
Estimate of the underlying density which produced the training samples.

4.2 Parzen windows

$$\hat{p}(x \mid \omega) = \frac{1}{n} \sum_{i=1}^{n} \varphi(x - x_i)$$

Matlab example:

Role of the Parzen window variance:



Variance of each Parzen window set to 0.4.

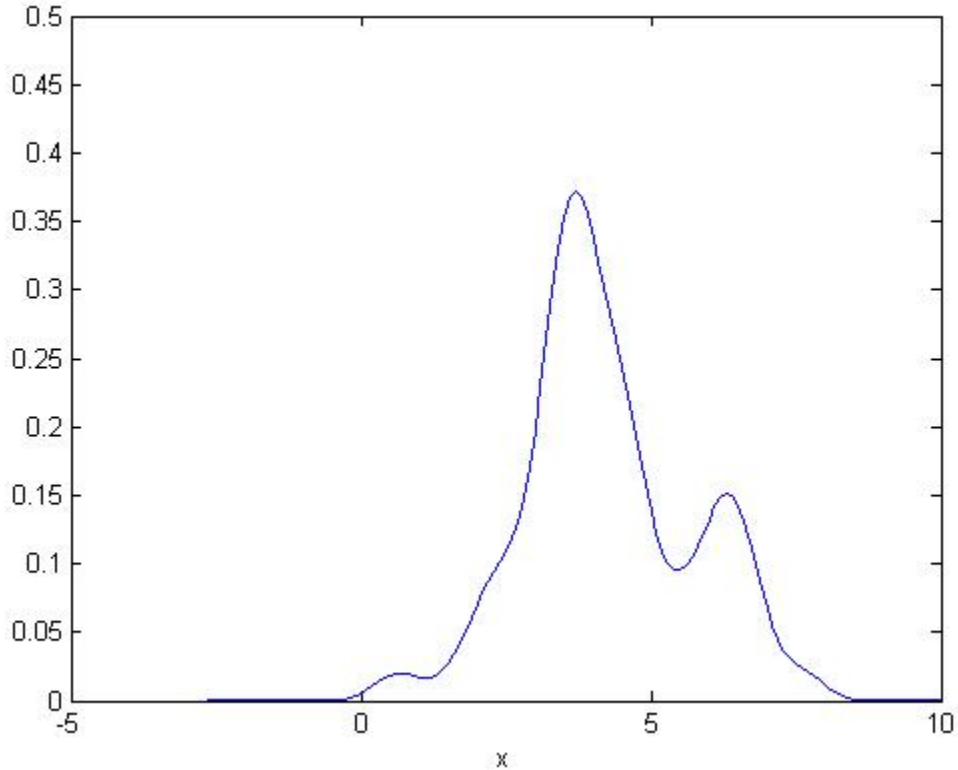
4.2 Parzen windows

$$\hat{p}(x \mid \omega) = \frac{1}{n} \sum_{i=1}^{n} \varphi(x - x_i)$$

Matlab example:

Role of the Parzen window variance:

$\hat{p}(x \mid \omega)$



Bad estimate of the underlying density.

4.2 Parzen windows

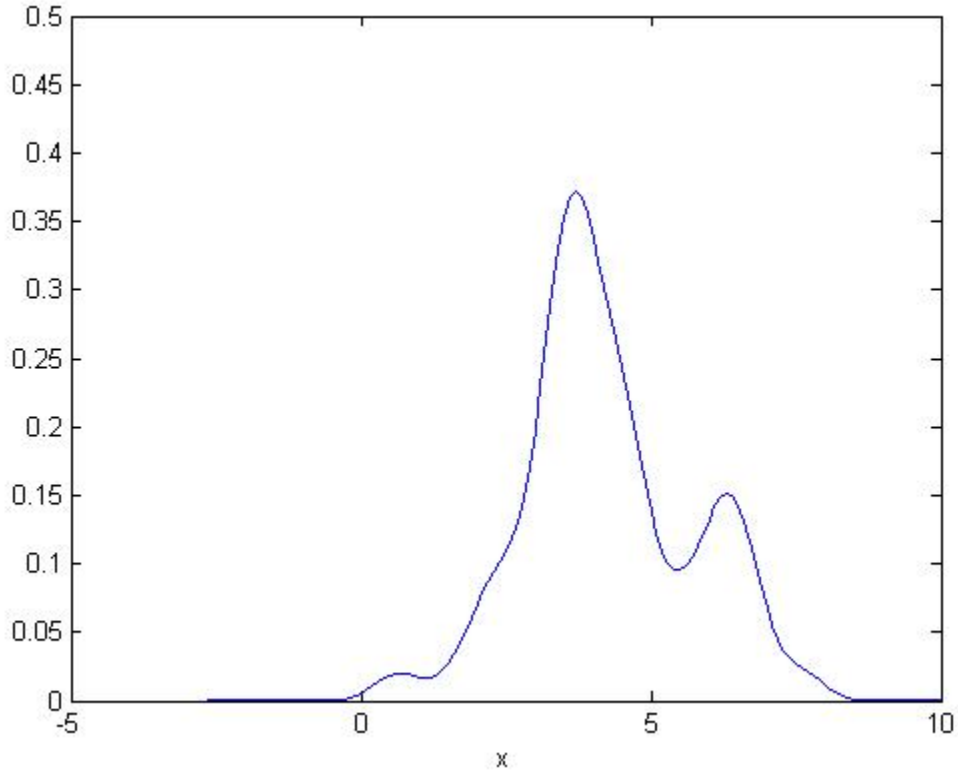
$$\hat{p}(x \mid \omega) = \frac{1}{n} \sum_{i=1}^{n} \varphi(x - x_i)$$

Matlab example:

Role of the number of training samples.

So far: n = 50 → Now increase n keeping the Parzen window variance of 0.4

$\hat{p}(x \mid \omega)$



n = 50

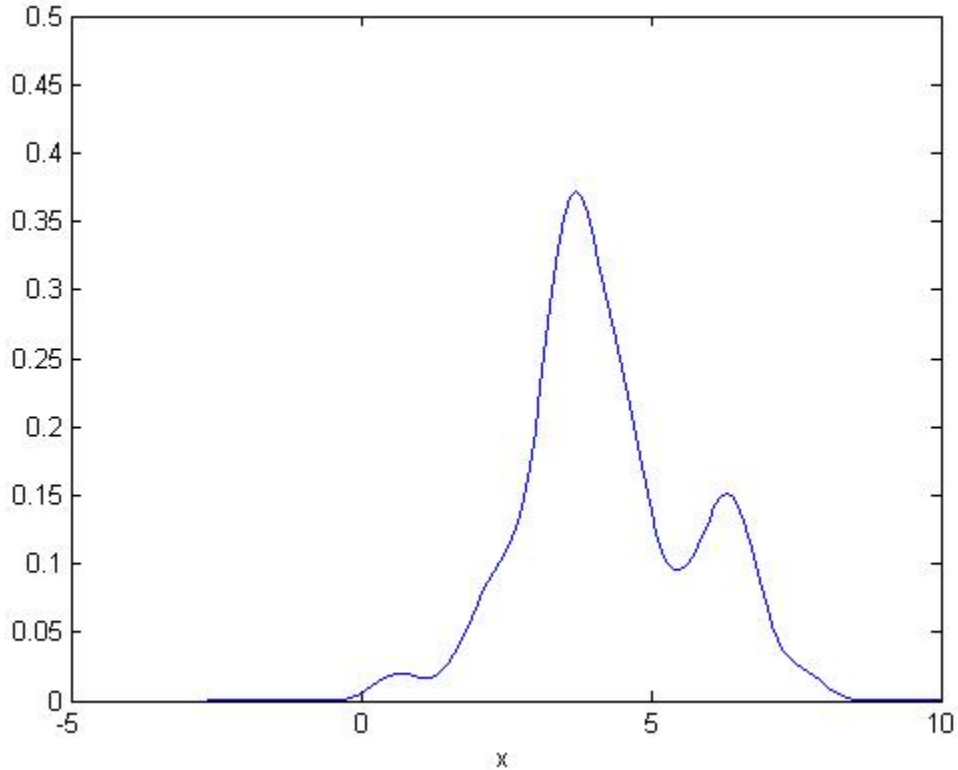
4.2 Parzen windows

$$\hat{p}(x \mid \omega) = \frac{1}{n} \sum_{i=1}^{n} \varphi(x - x_i)$$

Matlab example:

Role of the number of training samples.

$\hat{p}(x \mid \omega)$



n = 50

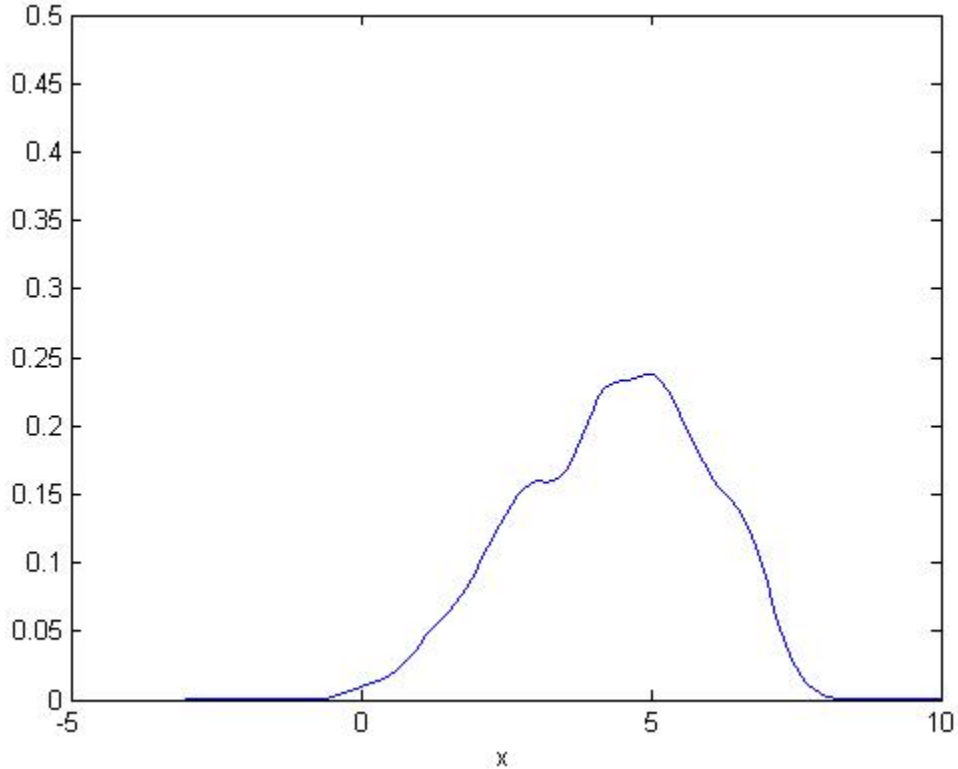
4.2 Parzen windows

$$\hat{p}(x \mid \omega) = \frac{1}{n} \sum_{i=1}^{n} \varphi(x - x_i)$$

Matlab example:

Role of the number of training samples.

$\hat{p}(x \mid \omega)$



n = 100

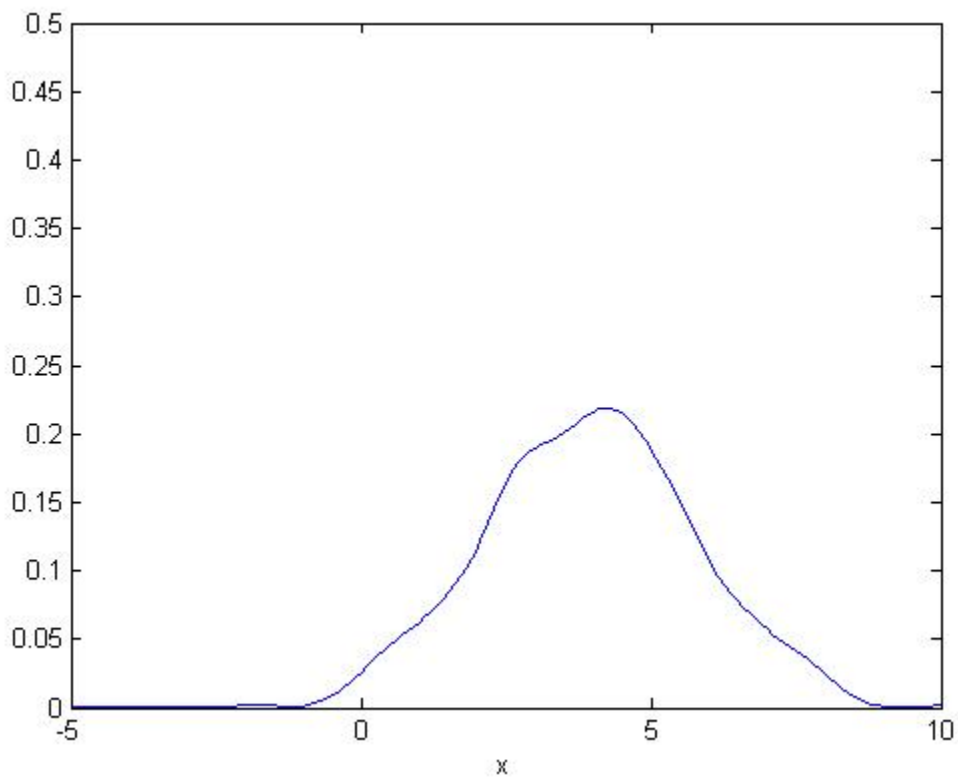
4.2 Parzen windows

$$\hat{p}(x \mid \omega) = \frac{1}{n} \sum_{i=1}^{n} \varphi(x - x_i)$$

Matlab example:

Role of the number of training samples.

$\hat{p}(x \mid \omega)$



n = 500

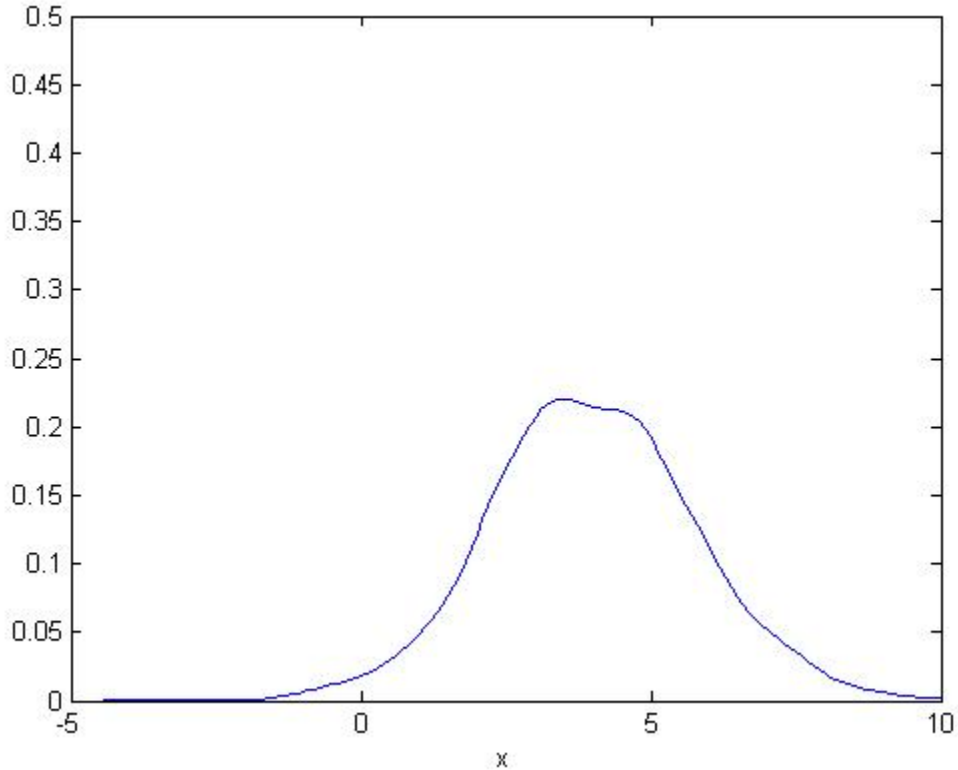
4.2 Parzen windows

$$\hat{p}(x \mid \omega) = \frac{1}{n} \sum_{i=1}^{n} \varphi(x - x_i)$$

Matlab example:

Role of the number of training samples.

$\hat{p}(x \mid \omega)$



n = 1000

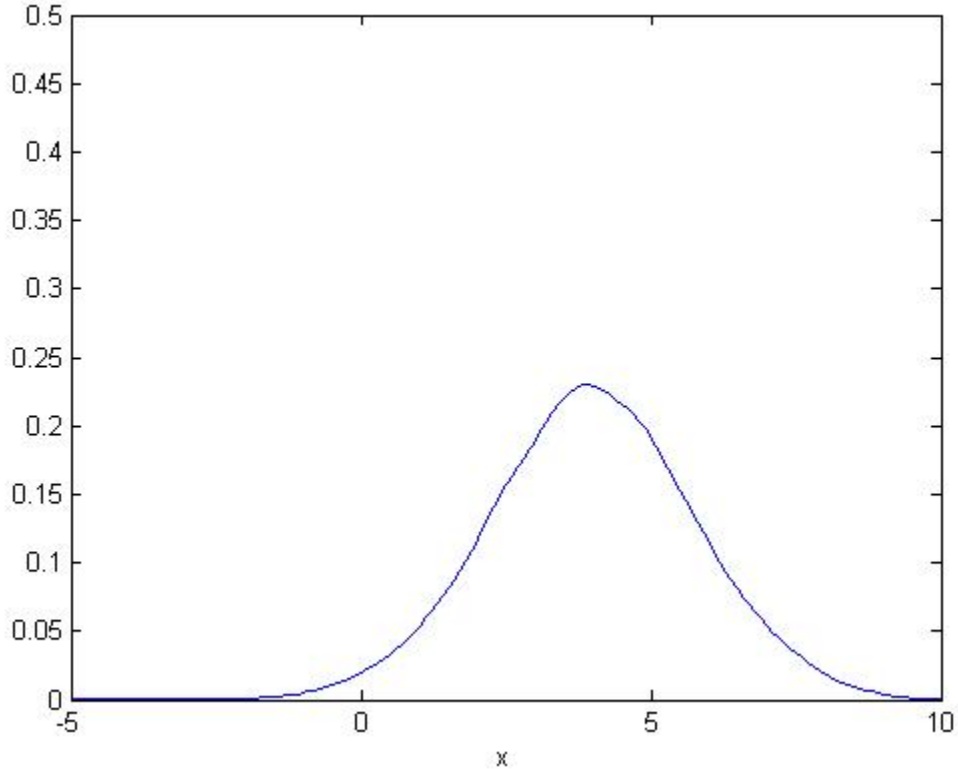
4.2 Parzen windows

$$\hat{p}(x \mid \omega) = \frac{1}{n} \sum_{i=1}^{n} \varphi(x - x_i)$$

Matlab example:

Role of the number of training samples.

$\hat{p}(x \mid \omega)$



n = 5000

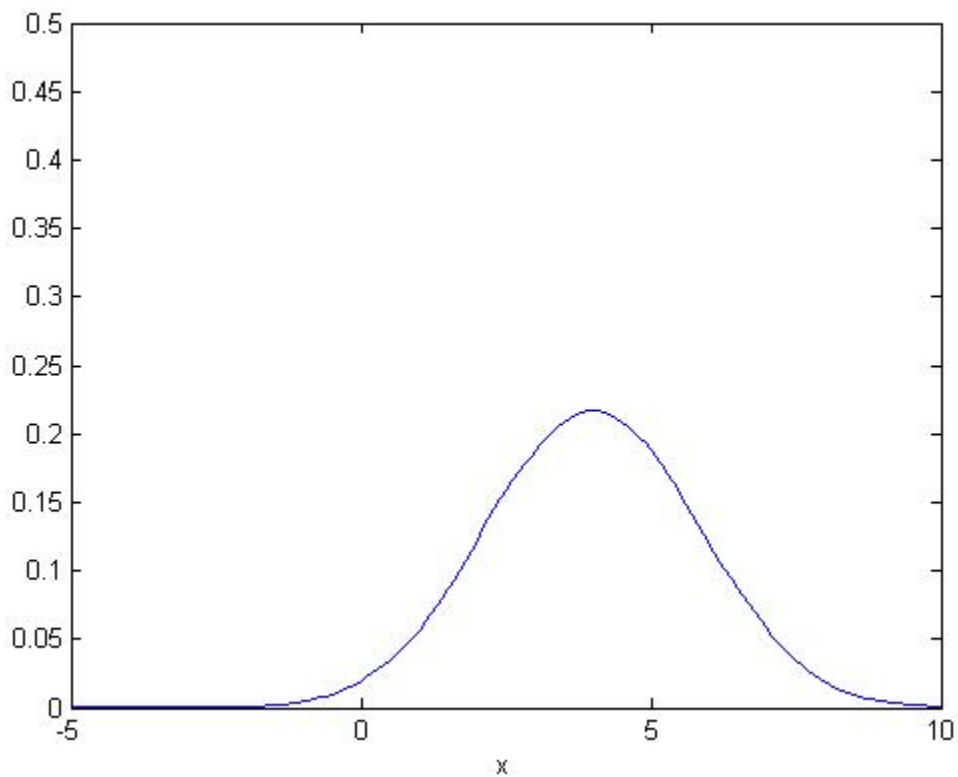
4.2 Parzen windows

$$\hat{p}(x \mid \omega) = \frac{1}{n} \sum_{i=1}^{n} \varphi(x - x_i)$$

Matlab example:

Role of the number of training samples.

$\hat{p}(x \mid \omega)$

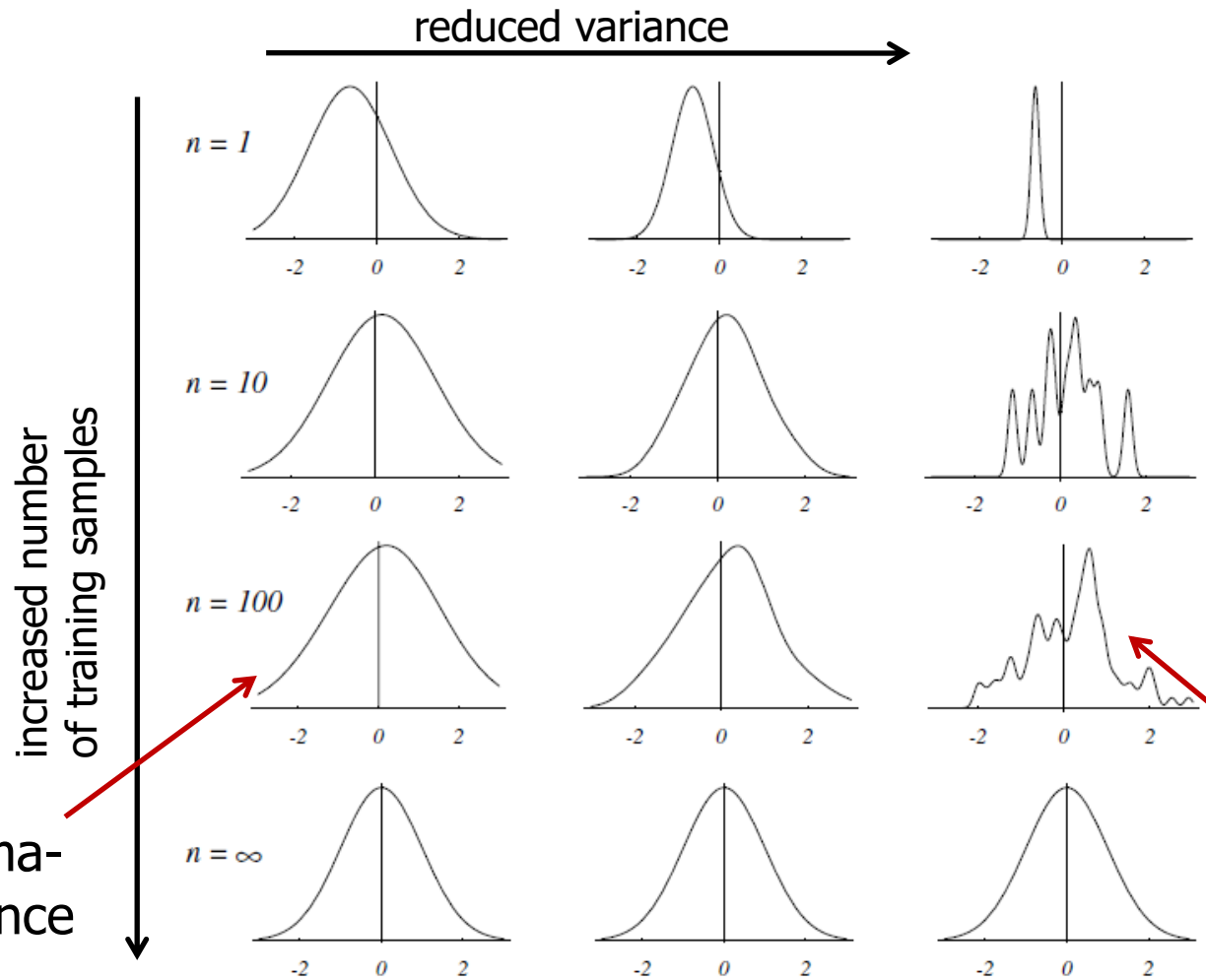


n = 10000

4.2 Parzen windows

Overview: Variance and training sample number

1-D Example



overestimated variance

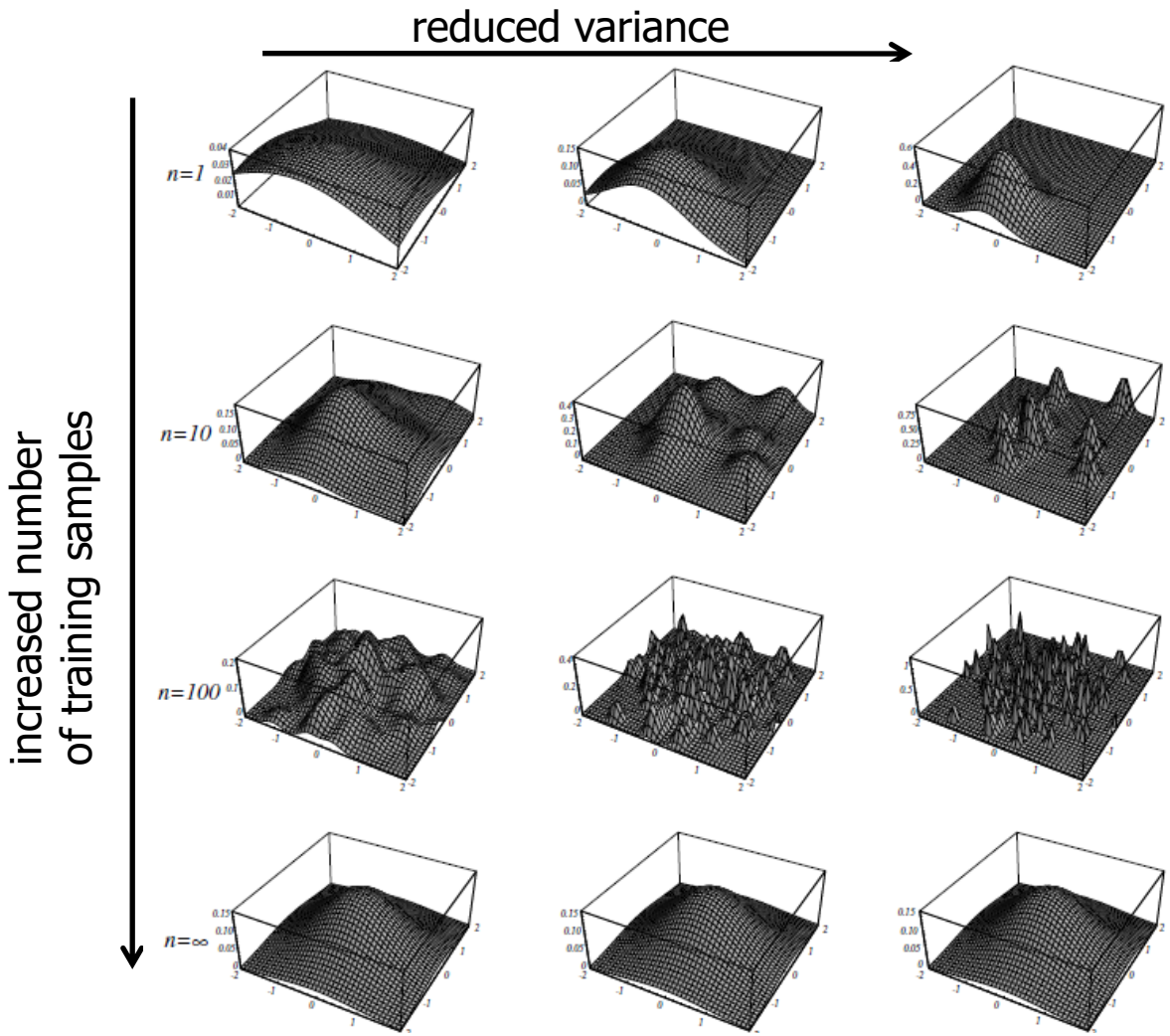
underestimated variance

Copyright © 2001 by John Wiley & Sons, Inc.

4.2 Parzen windows

Overview: Variance and training sample number

2-D Example



© 2001 by John Wiley & Sons, Inc.

4.2 Parzen windows

Octave code:

```
% Illustration area
dx = 0.1;
x = -10:dx:20;

% Parzen Window variance
s = 0.4;

% Number of training observations
n = 10000;

% Parameters of the density used to generate the training samples
trn_mean = 4; % mean
trn_var  = 3; % variance

% Initialize class-conditional probability
px = zeros(1, size(x,2));

figure(1), hold on;

% Continuation on next page
```

4.2 Parzen windows

Octave code:

```
% continuation
for i=1:n        % n: number of training samples used for estimation
    % get the next training sample mean m from the generating normal density
    xi = mvnrnd(trn_mean, trn_var);

    parzen = exp(-0.5*((x-xi)./s).^2)./(sqrt(2*pi)*s);  % compute current window
    plot(x, parzen);                                    % plot this Parzen wind.
    line([xi xi]', [0 1]');                             % plot a line at sample

    px = px + parzen;                                   % summation
end
axis([-5 10 0 1]);
xlabel('x');

px = px ./ n;                                           % normalization
fprintf('Normalization check: Area below p(x) is %.2f\n',sum(px*dx));
figure(2), plot(x, px);
axis([-5 10 0 0.5]);
xlabel('x');
```

$$\hat{p}(x \mid \omega) = \frac{1}{n} \sum_{i=1}^{n} \varphi(x - x_i)$$

4.2 Parzen windows

General mathematical formulation of a Parzen window:

$$\varphi(\vec{x}): \quad \Re^D \rightarrow [0,\infty[$$

$$\vec{x} \rightarrow \varphi(\vec{x})$$

The Parzen window maps a D-dimensional vector **x** to a scalar

$$\int_{\Re^D} \varphi(\vec{x})dx = 1$$

Normalization constraint

4.2 Parzen windows

General mathematical formulation of a Parzen window:

$$\varphi(\vec{x}): \quad \Re^D \rightarrow [0,\infty[$$

$$\vec{x} \rightarrow \varphi(\vec{x})$$

The Parzen window maps a D-dimensional vector **x** to a scalar

$$\int_{\Re^D} \varphi(\vec{x})dx = 1$$

Normalization constraint

Typical choice for φ(**x**) in case of a higher-dimensional **x**:

Multivariate Gaussian with independent components and mean vector μ = 0.

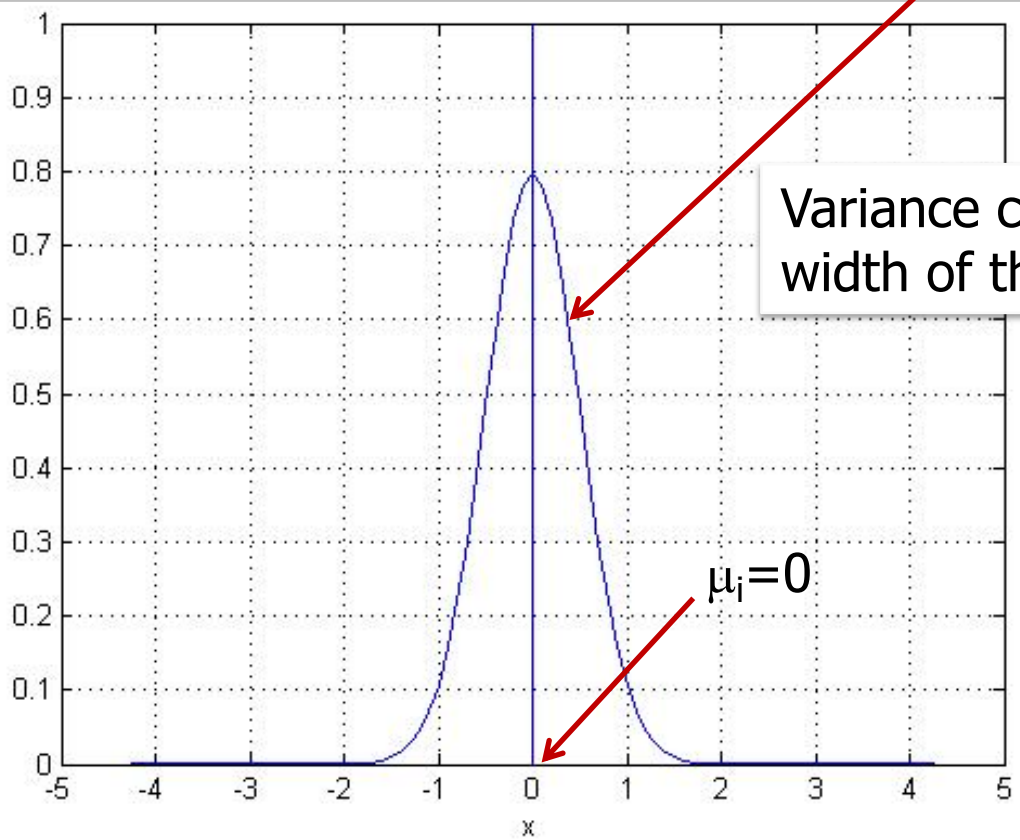
$$\varphi(x_1, x_2, \ldots, x_d) = \prod_{i=1}^{d} p(x_i) = \prod_{i=1}^{d} \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{1}{2}\cdot\left(\frac{x_i - \mu_i}{\sigma_i}\right)^2}$$

μ_i → =0

4.2 Parzen windows

Example:

$$\varphi(x_1, x_2, \ldots, x_d) = \prod_{i=1}^{d} p(x_i) = \prod_{i=1}^{d} \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{x_i^2}{2\cdot\sigma_i^2}}$$



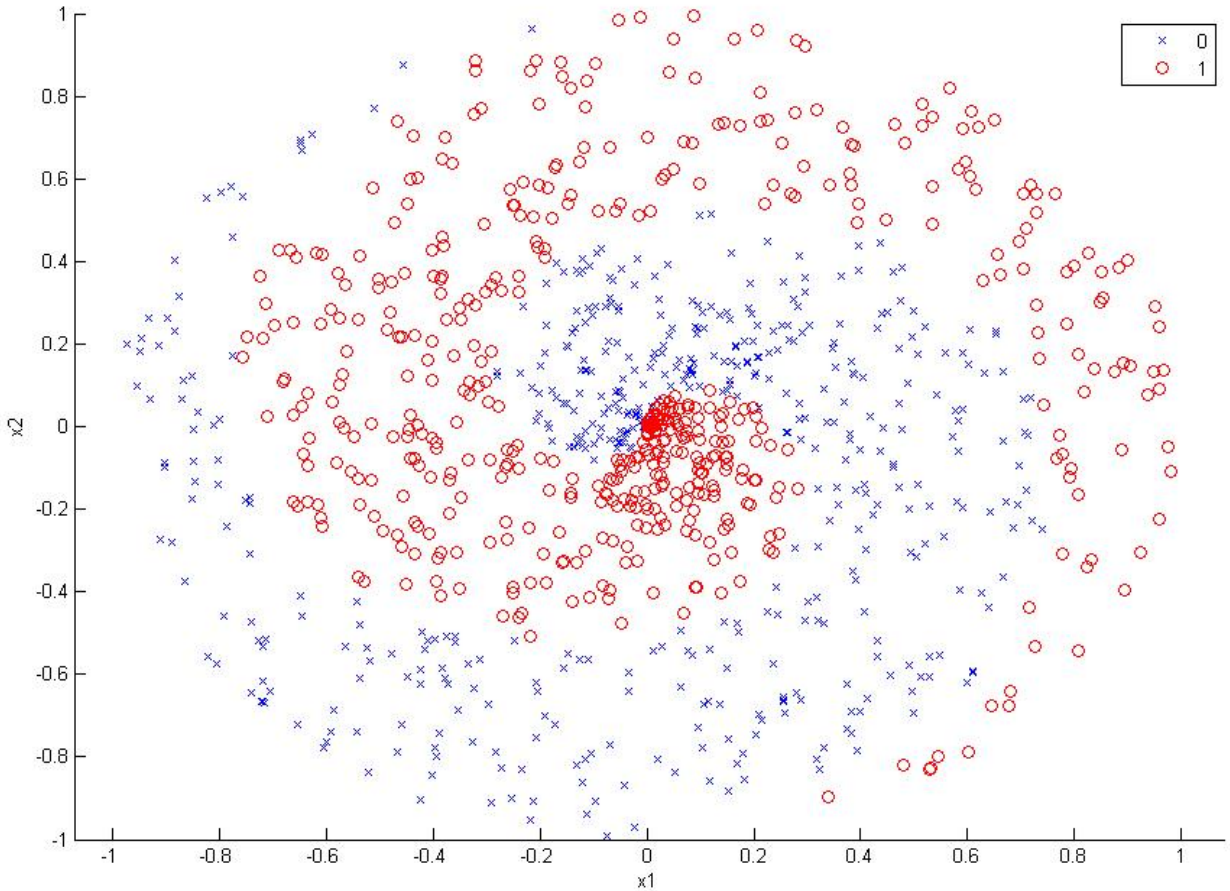
Variance controls the width of the kernel

1-D example

4.2 Parzen windows

2-D Matlab application example:

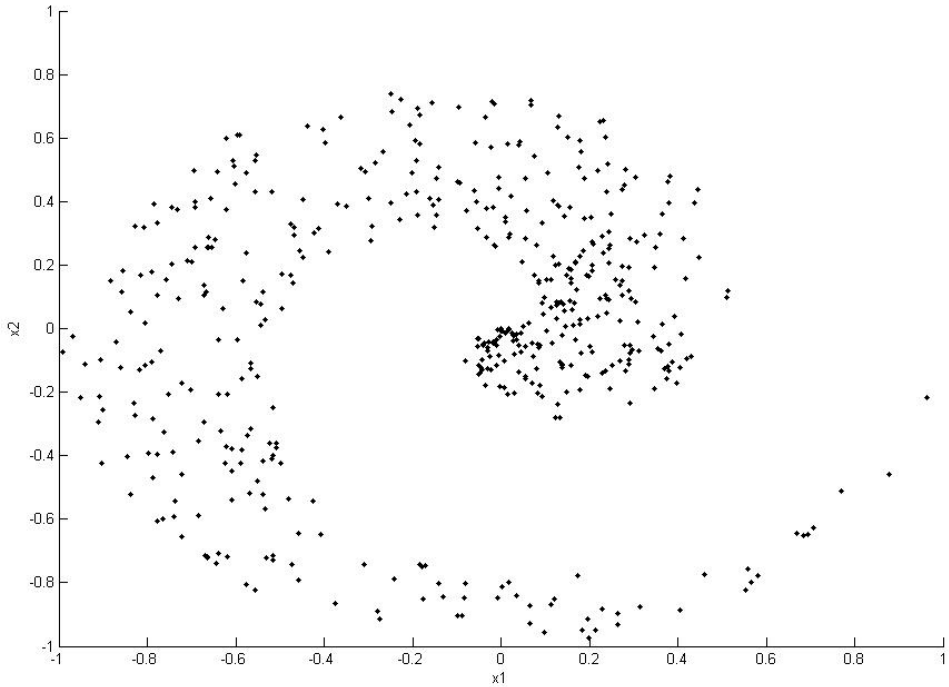
```
load 'D:/.../spiral.mat'  % -> patterns, targets
```

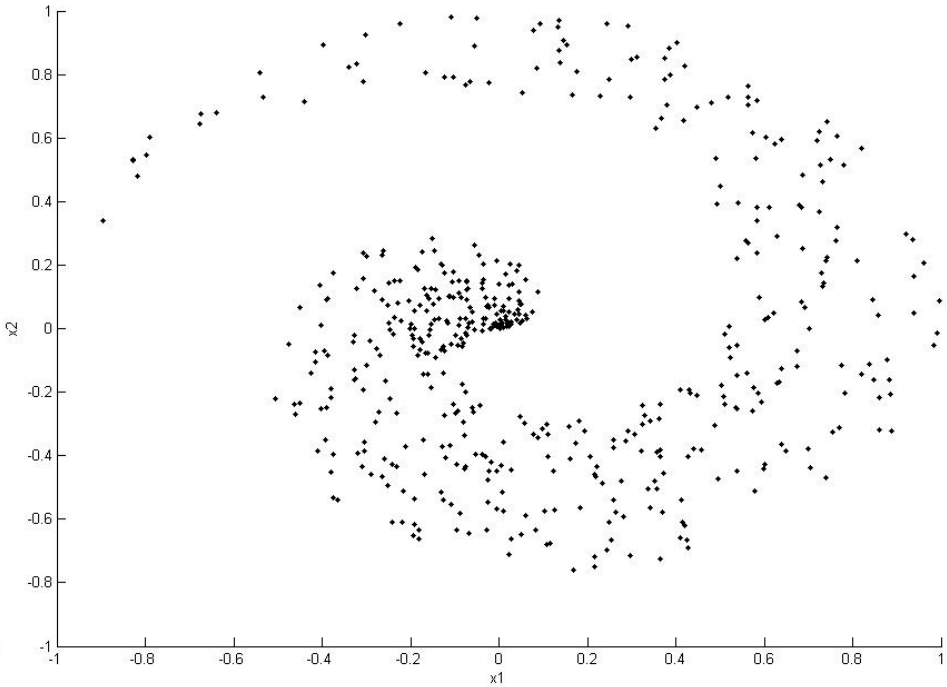


4.2 Parzen windows

2-D Matlab application example:

```
load 'D:/.../spiral.mat'
```

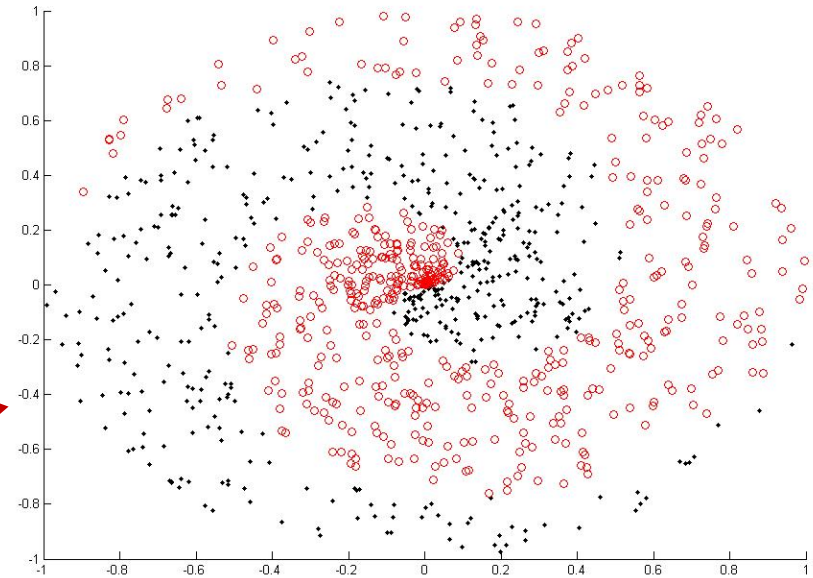


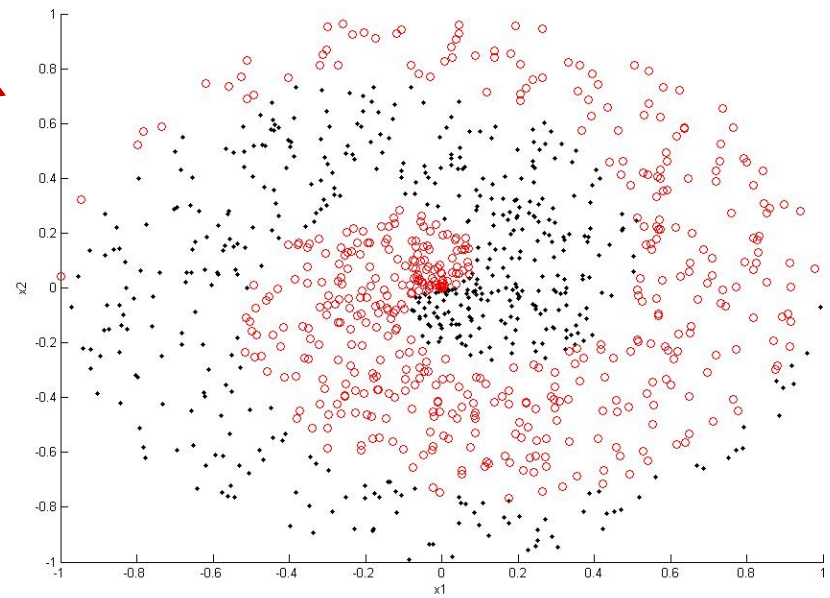


4.2 Parzen windows

2-D Matlab application example:

```
load 'D:/.../spiral.mat'

% split into train and test samples
train_pat = patterns(:,1:1000);
train_tar = targets(1:1000);

test_pat = patterns(:,1001:2000);
test_tar = targets(1001:2000);
```





4.2 Parzen windows

2-D Matlab application example:

```
load 'D:/.../spiral.mat'

% split into train and test samples
train_pat = patterns(:,1:1000);
train_tar = targets(1:1000);

test_pat = patterns(:,1001:2000);
test_tar = targets(1001:2000);

% Generate a vector X containing all 'interesting' x1-x2-value combinations
% (Estimate likelihoods for these X-values, size(X): 2 rows (x1, x2), N columns)
[X1,X2] = meshgrid(linspace(min(train_pat(1,:)), max(train_pat(1,:)), steps),
                   linspace(min(train_pat(2,:)), max(train_pat(2,:)), steps));
X = [X1(:) X2(:)]';

% Call Parzen window function (see below for details)
[p_x_c, p_c] = parzen_HS(train_pat, train_tar, X, sm);
```

```
function [p_x_c, p_c] = parzen_HS(trn_pat, trn_tar, test_pat, var)

% Parzen window based estimate of the likelihood.

% Classify using the Parzen windows algorithm
% Inputs:
%   trn_pat             - Training patterns (features x_i)
%   trn_tar             - Training targets (class labels)
%   test_pat            - Compute likelihood for these values (e.g. meshgrid)
%   var                 - Variance of Gaussian Parzen window
%
% Outputs:
%   p_x_c               - Estimated likelihood p(x|c) (computed for test_pat)
%   p_c                 - Estimated prior probability

N       = size(test_pat, 2);       % number of test samples
Uc      = unique(trn_tar);         % store the existing class IDs in Uc
p_x_c   = zeros(length(Uc), N);    % build Uc x N zero matrix
x_i     = trn_pat;                 % renaming for consistence with equations



% Continuation on next page
```

```
% ... continuation

for j = 1:length(Uc),                          % for each class
    indices   = find(trn_tar == Uc(j));        % Uc(j) is label of current class
```

Get indices of samples
of current class Uc(j)

Example:

```
>> x = uint8(10*rand (1,30))

x = 3    2    7    8    3    8    7    0    6    4    9    0    5    4    5    8    3

    8    5    0    2    7    5    2    3    6    2    7    2    9

>> indices = find(x == 3)

indices =

     1     5    17    25
```

```
% ... continuation

for j = 1:length(Uc),                          % for each class
    indices   = find(trn_tar == Uc(j));        % Uc(j) is label of current class
    p_c(j)    = length(indices)/size(x_i,2);   % prior probability
    n         = length(indices);               % number of training samples
```

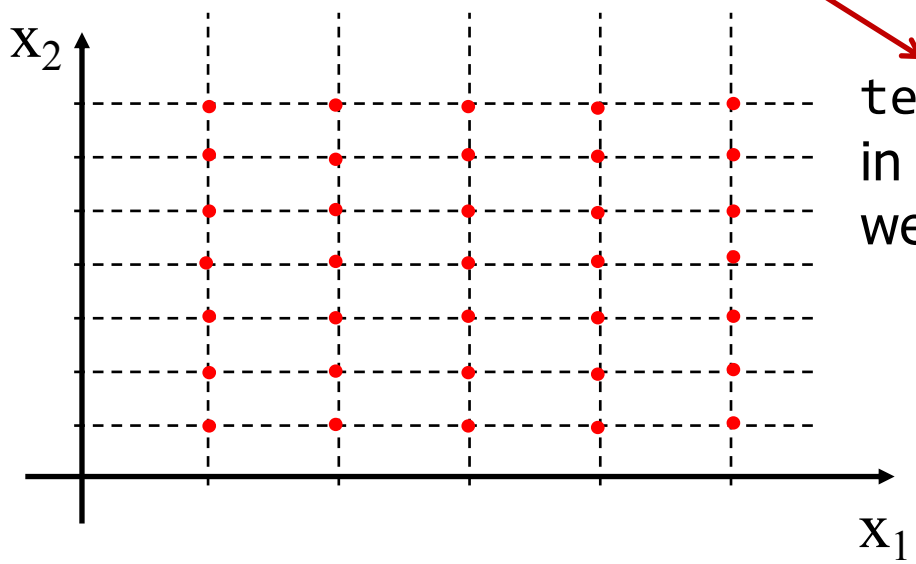
Compute prior probability of current class:
Number of class samples / Total sample number

```
% ... continuation

for j = 1:length(Uc),                              % for each class
    indices   = find(trn_tar == Uc(j));            % Uc(j) is label of current class
    p_c(j)    = length(indices)/size(x_i,2);       % prior probability
    n         = length(indices);                   % number of training samples

    for i = 1:n,                                   % for each training(!) sample
        % Compute Euclidean distance to current sample for each grid point
        temp      = sum((test_pat - x_i(:,indices(i))*ones(1,N)).^2);
```

Euclidean Distance

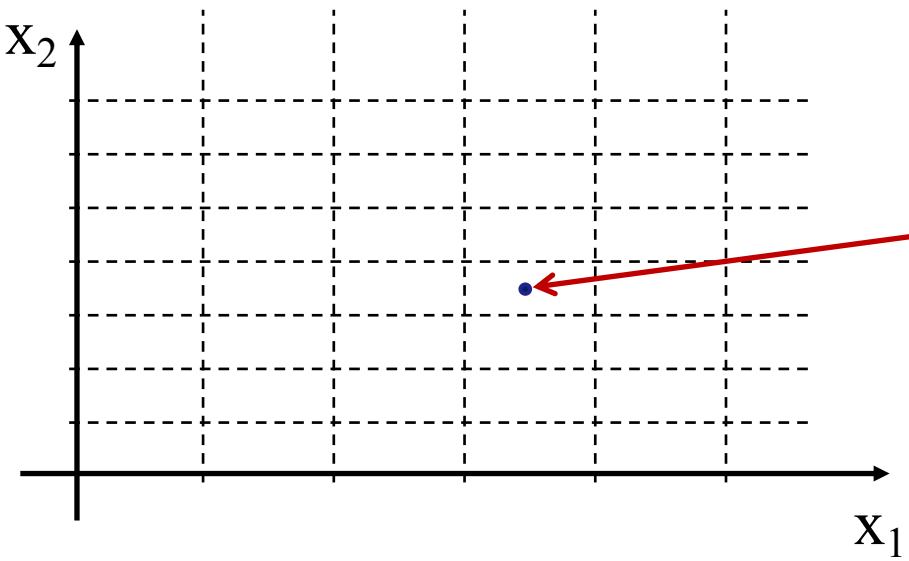
$$d_2(x,y) = \sqrt{\sum_{d=1}^{D} (x_d - y_d)^2}$$

```
% ... continuation

for j = 1:length(Uc),                                   % for each class
    indices   = find(trn_tar == Uc(j));                 % Uc(j) is label of current class
    p_c(j)    = length(indices)/size(x_i,2);            % prior probability
    n         = length(indices);                        % number of training samples

    for i = 1:n,                                        % for each training(!) sample
        % Compute Euclidean distance to current sample for each grid point
        temp      = sum((test_pat - x_i(:,indices(i))*ones(1,N)).^2);
```



`test_pat` contains all pairs (x_1, x_2) in the 'area of interest' (grid) for which we want to compute the likelihood.

```
% ... continuation

for j = 1:length(Uc),                                   % for each class
    indices   = find(trn_tar == Uc(j));                 % Uc(j) is label of current class
    p_c(j)    = length(indices)/size(x_i,2);            % prior probability
    n         = length(indices);                        % number of training samples

    for i = 1:n,                                        % for each training(!) sample
        % Compute Euclidean distance to current sample for each grid point
        temp      = sum((test_pat - x_i(:,indices(i))*ones(1,N)).^2);
```



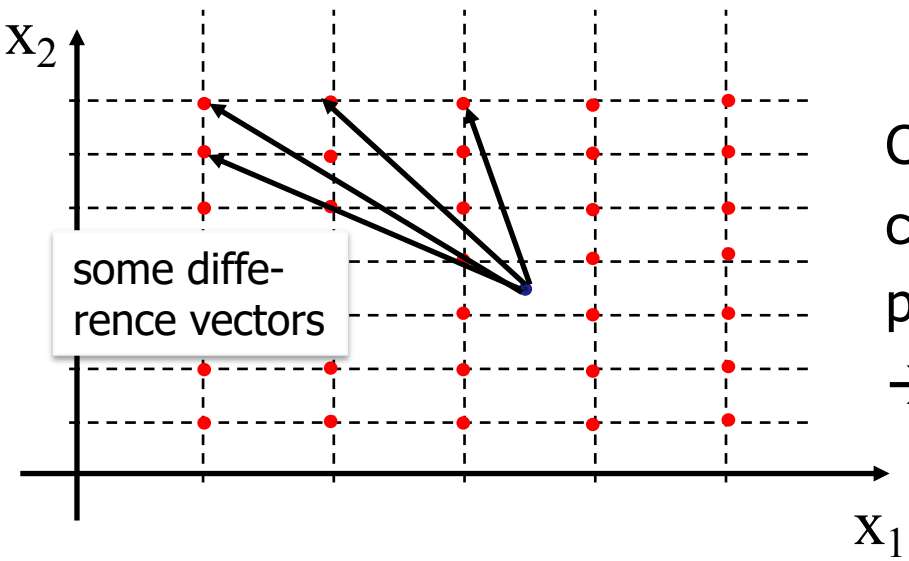
`x_i(:,indices(i))` is the i-th training sample.

```
% ... continuation

for j = 1:length(Uc),                                   % for each class
    indices   = find(trn_tar == Uc(j));                 % Uc(j) is label of current class
    p_c(j)    = length(indices)/size(x_i,2);            % prior probability
    n         = length(indices);                        % number of training samples

    for i = 1:n,                                        % for each training(!) sample
        % Compute Euclidean distance to current sample for each grid point
        temp      = sum((test_pat - x_i(:,indices(i))*ones(1,N)).^2);
```

Repeat the i-th training sample N times to get the same size as `test_pat`.

```
>> x_i = [4; 7]

x_i =       4
            7

>> x_i * ones (1, 10)

     4     4     4     4     4     4     4     4     4     4
     7     7     7     7     7     7     7     7     7     7
```

```
% ... continuation

for j = 1:length(Uc),                                   % for each class
    indices   = find(trn_tar == Uc(j));                 % Uc(j) is label of current class
    p_c(j)    = length(indices)/size(x_i,2);            % prior probability
    n         = length(indices);                        % number of training samples

    for i = 1:n,                                        % for each training(!) sample
        % Compute Euclidean distance to current sample for each grid point
        temp      = sum((test_pat - x_i(:,indices(i))*ones(1,N)).^2);
```



Compute difference vector between current training sample and each point in `test_pat`

→ Matrix with 2-D difference vectors

```
% ... continuation

for j = 1:length(Uc),                              % for each class
    indices   = find(trn_tar == Uc(j));            % Uc(j) is label of current class
    p_c(j)    = length(indices)/size(x_i,2);       % prior probability
    n         = length(indices);                   % number of training samples

    for i = 1:n,                                   % for each training(!) sample
        % Compute Euclidean distance to current sample for each grid point
        temp      = sum((test_pat - x_i(:,indices(i))*ones(1,N)).^2);
```

Squared difference vectors

$$d_2(x,y) = \sqrt{\sum_{d=1}^{D} (x_d - y_d)^2}$$

```
% ... continuation

for j = 1:length(Uc),                          % for each class
    indices   = find(trn_tar == Uc(j));        % Uc(j) is label of current class
    p_c(j)    = length(indices)/size(x_i,2);   % prior probability
    n         = length(indices);               % number of training samples

    for i = 1:n,                               % for each training(!) sample
        % Compute Euclidean distance to current sample for each grid point
        temp      = sum((test_pat - x_i(:,indices(i))*ones(1,N)).^2);
```

Sum over the individual components

$$d_2(x,y) = \sqrt{\sum_{d=1}^{D} (x_d - y_d)^2}$$

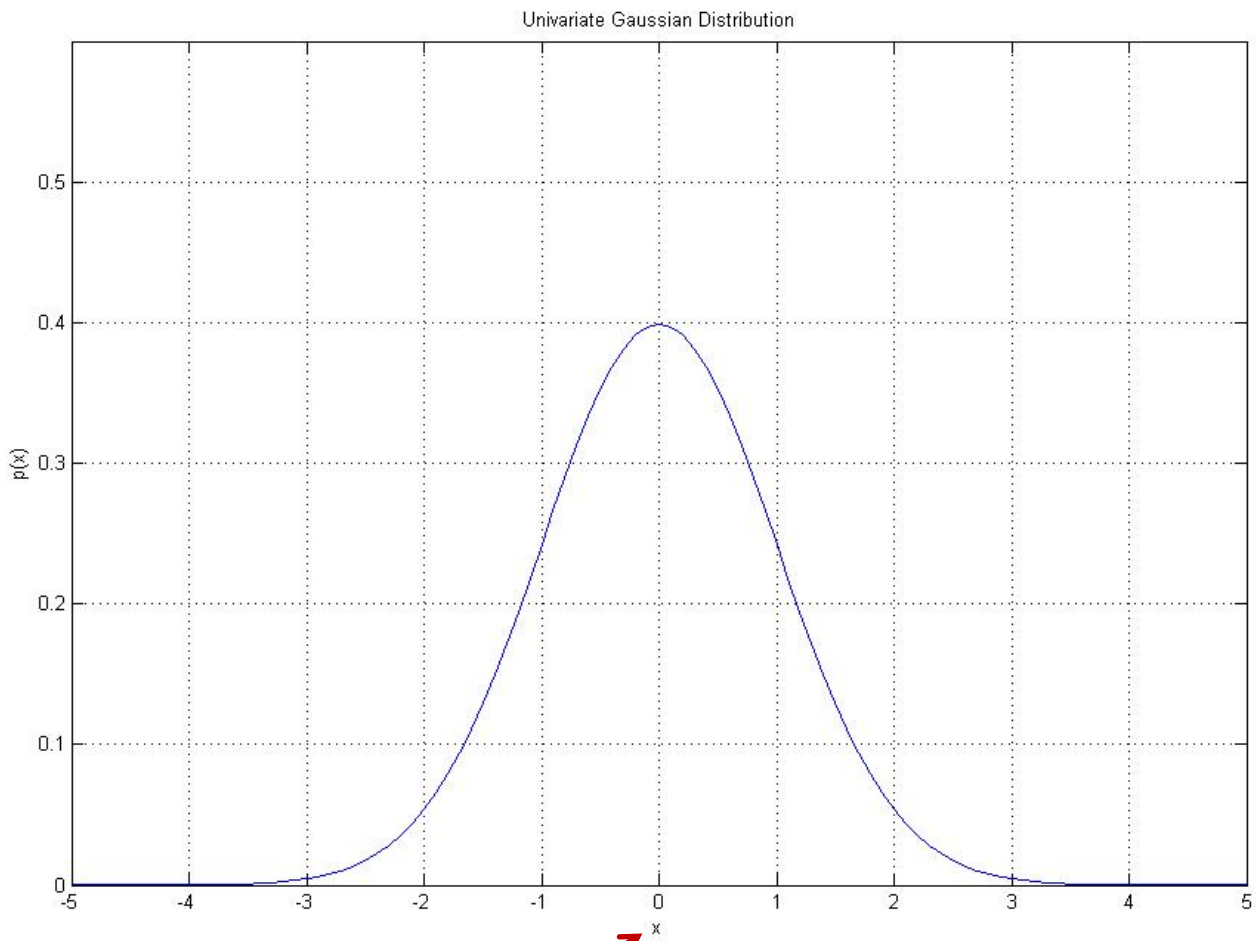
Note: Square root is not considered here. We work with squared $d_2(x,y)$

```
% ... continuation

for j = 1:length(Uc),                                   % for each class
    indices   = find(trn_tar == Uc(j));                 % Uc(j) is label of current class
    p_c(j)    = length(indices)/size(x_i,2);            % prior probability
    n         = length(indices);                        % number of training samples

    for i = 1:n,                                        % for each training(!) sample
        % Compute Euclidean distance to current sample for each grid point
        temp      = sum((test_pat - x_i(:,indices(i))*ones(1,N)).^2);

        % Sum the parzen window over each training sample
        p_x_c(j,:)    = p_x_c(j,:) + phi(temp, var);
    end
```

Computes the contribution of this training sample's Parzen window (which is a Gaussian density here) to each point's likelihood.

```
% Gaussian distribution
function p = phi(val, s)

% Probability of the Euclidean distance
p = exp(-0.5*((val)./s).^2)./(sqrt(2*pi)*s);
```

Support function



Important remark:
The variable x of the Parzen window used here is the squared Euclidean distance!

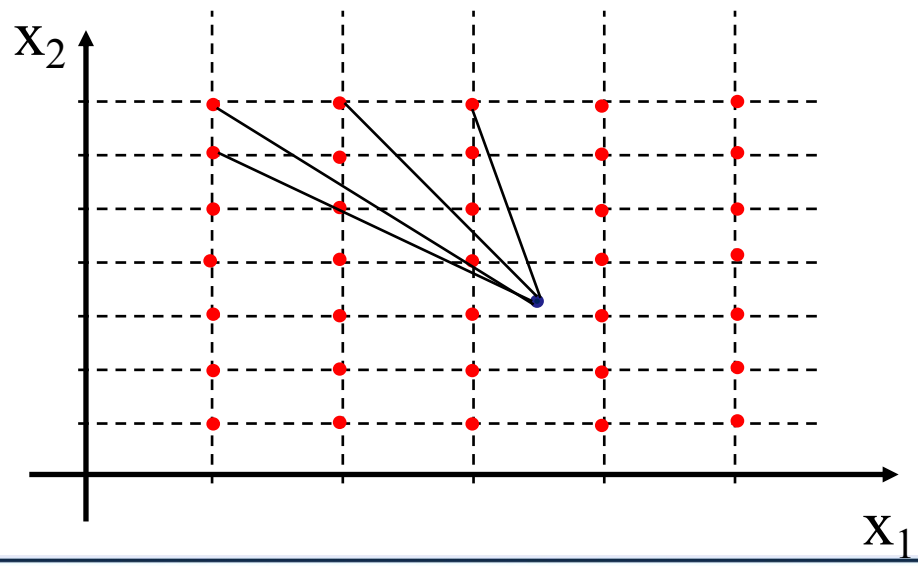
$$\varphi\,(x-x_i) \rightarrow \varphi\,(\,d_2(x,x_i)^2\,)$$

Reason: Prevent to work with 2D Gaussian Parzen Window

```
% Gaussian distribution
function p = phi(val, s)

% Probability of the Euclidean distance
p = exp(-0.5*((val)./s).^2)./(sqrt(2*pi)*s);
```

Support function

```
% ... continuation

for j = 1:length(Uc),                              % for each class
    indices   = find(trn_tar == Uc(j));            % Uc(j) is label of current class
    p_c(j)    = length(indices)/size(x_i,2);       % prior probability
    n         = length(indices);                   % number of training samples

    for i = 1:n,                                   % for each training(!) sample
        % Compute Euclidean distance to current sample for each grid point
        temp      = sum((test_pat - x_i(:,indices(i))*ones(1,N)).^2);

        % Sum the parzen window over each training sample
        p_x_c(j,:)    = p_x_c(j,:) + phi(temp, var);
    end
```



Each point (x_1, x_2) is assigned a likelihood contribution coming from the Parzen window of this training sample's point.

```
% ... continuation

for j = 1:length(Uc),                                % for each class
    indices   = find(trn_tar == Uc(j));              % Uc(j) is label of current class
    p_c(j)    = length(indices)/size(x_i,2);         % prior probability
    n         = length(indices);                     % number of training samples

    for i = 1:n,                                     % for each training(!) sample
        % Compute Euclidean distance to current sample for each grid point
        temp      = sum((test_pat - x_i(:,indices(i))*ones(1,N)).^2);

        % Sum the parzen window over each training sample
        p_x_c(j,:)    = p_x_c(j,:) + phi(temp, var);
    end

    % Normalization to 1
    p_x_c(j,:) = p_x_c(j,:) / sum(p_x_c(j,:));
end
```

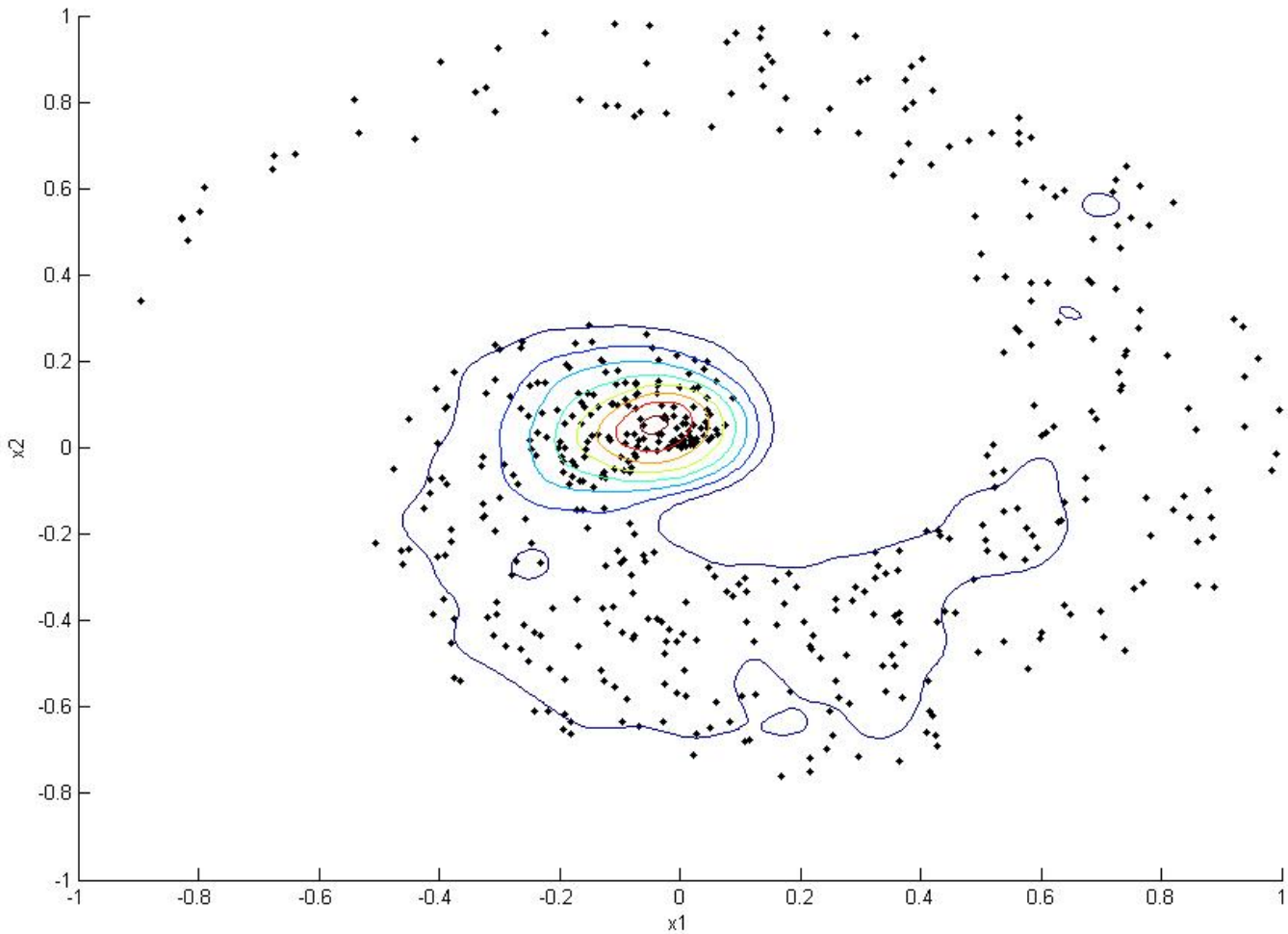
4.2 Parzen windows

... continue with the script which called `parzen_HS`

```
% Call Parzen window function (see below for details)
[p_x_c, p_c] = parzen_HS(train_pat, train_tar, X, sm);

% Reshape and illustrate the obtained likelihoods
p_x1_x2_class1 = reshape(p_x_c(1,:),size(X1,2),size(X2,2));
p_x1_x2_class2 = reshape(p_x_c(2,:),size(X1,2),size(X2,2));
```

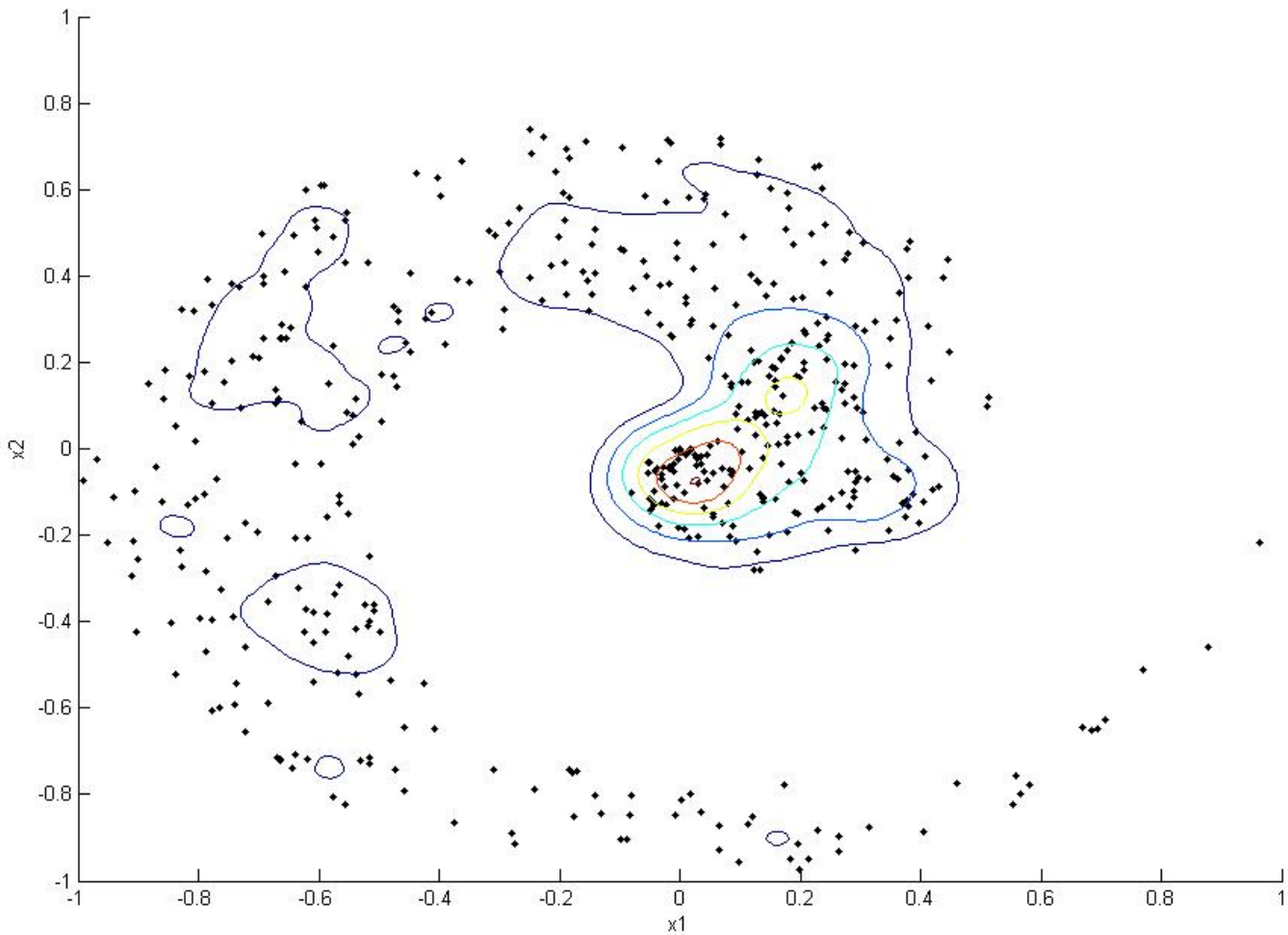
4.2 Parzen windows

Estimated likelihood $p(x \mid \omega_1)$

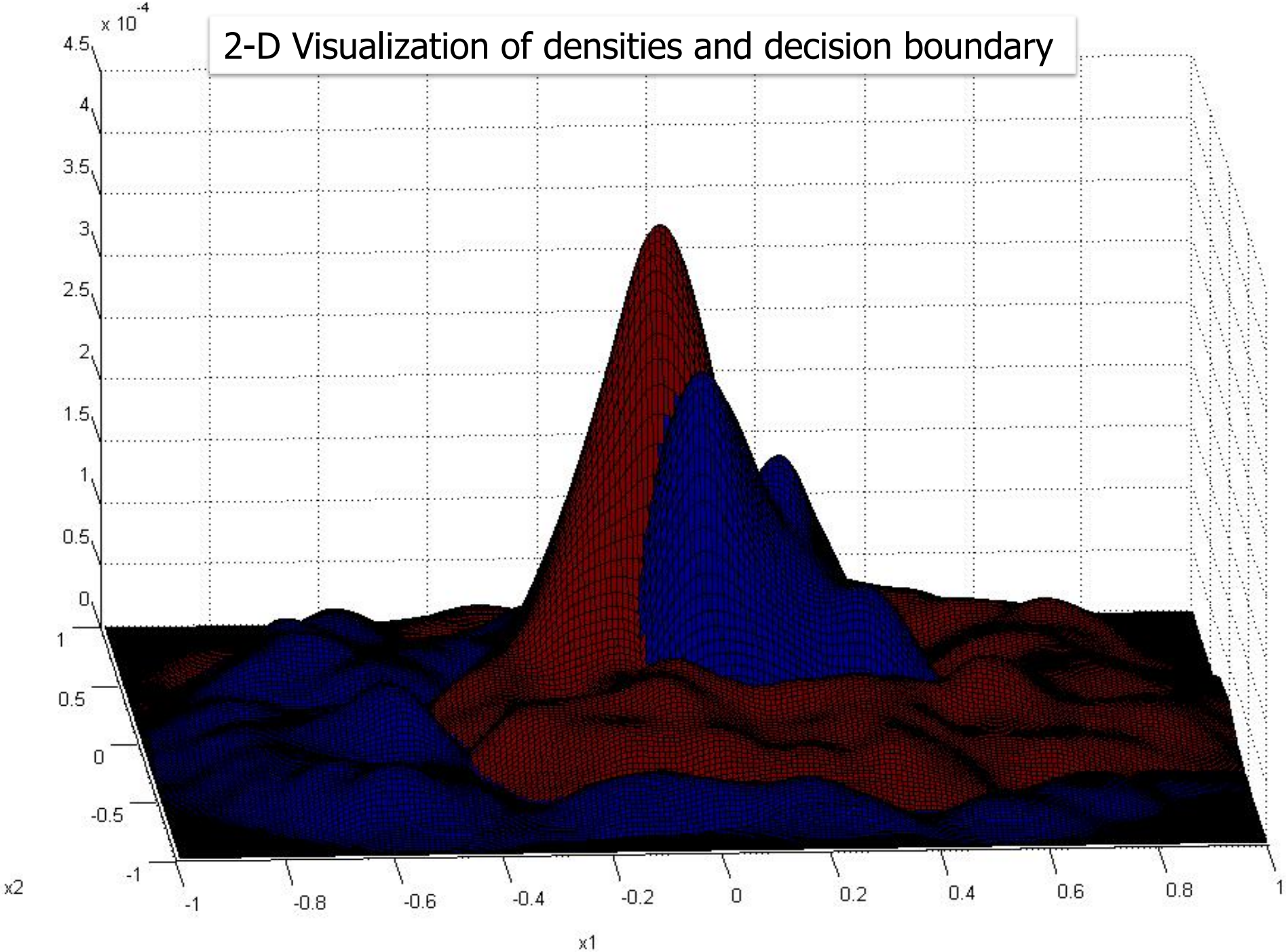


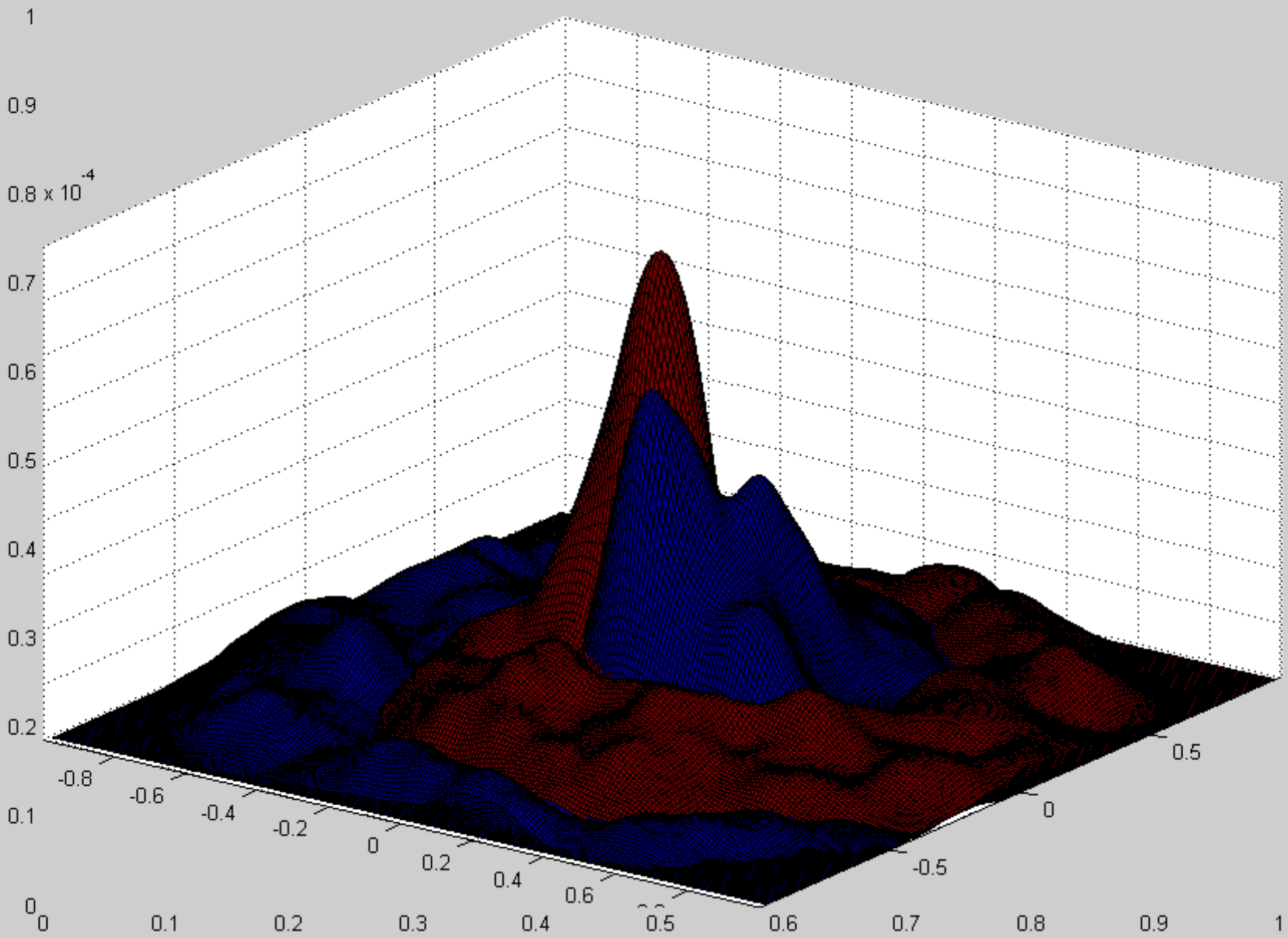
4.2 Parzen windows

Estimated likelihood $p(x \mid \omega_2)$



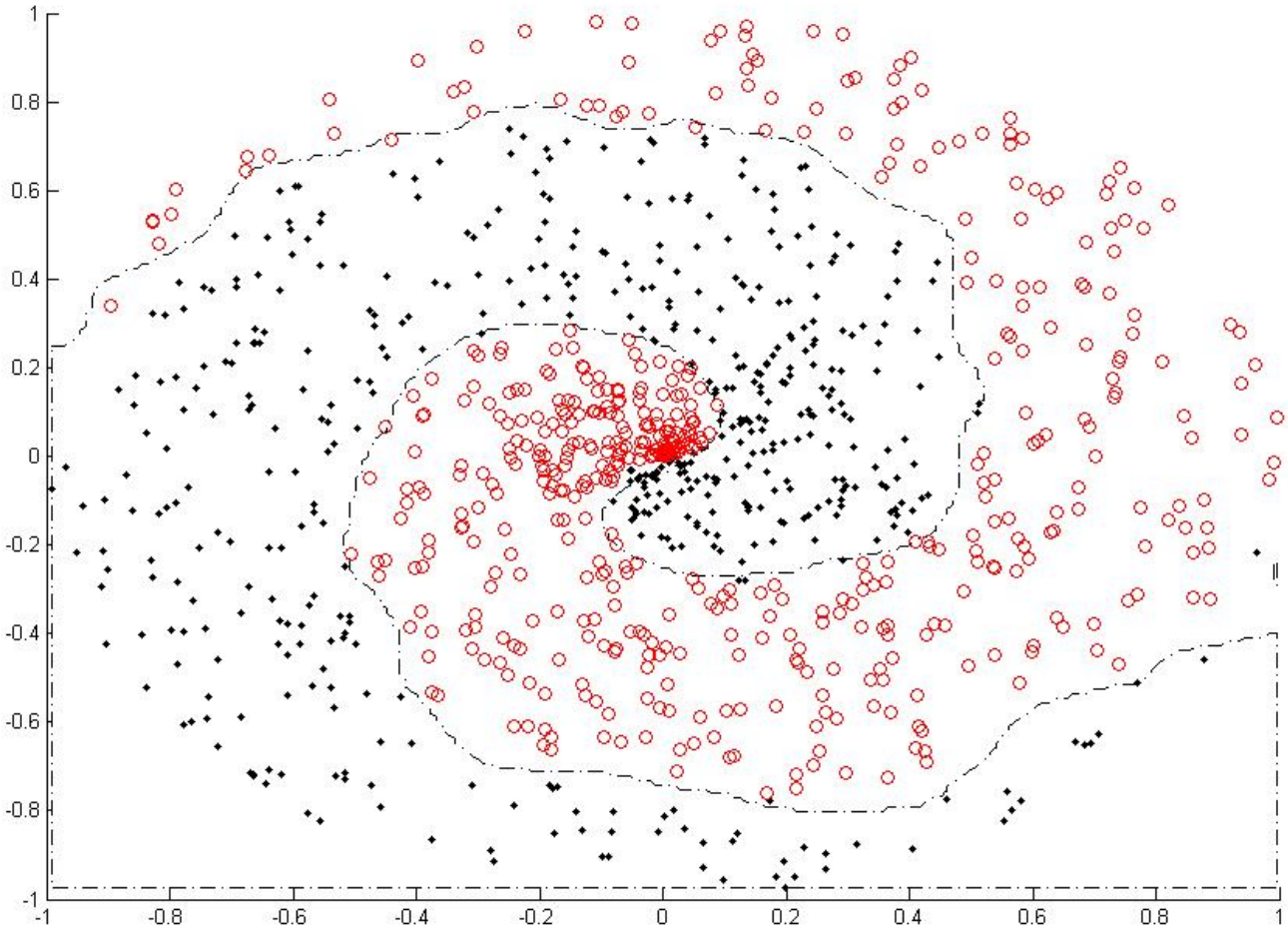
2-D Visualization of densities and decision boundary




1
0.9
0.8 x 10^-4
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0.5
0
-0.5

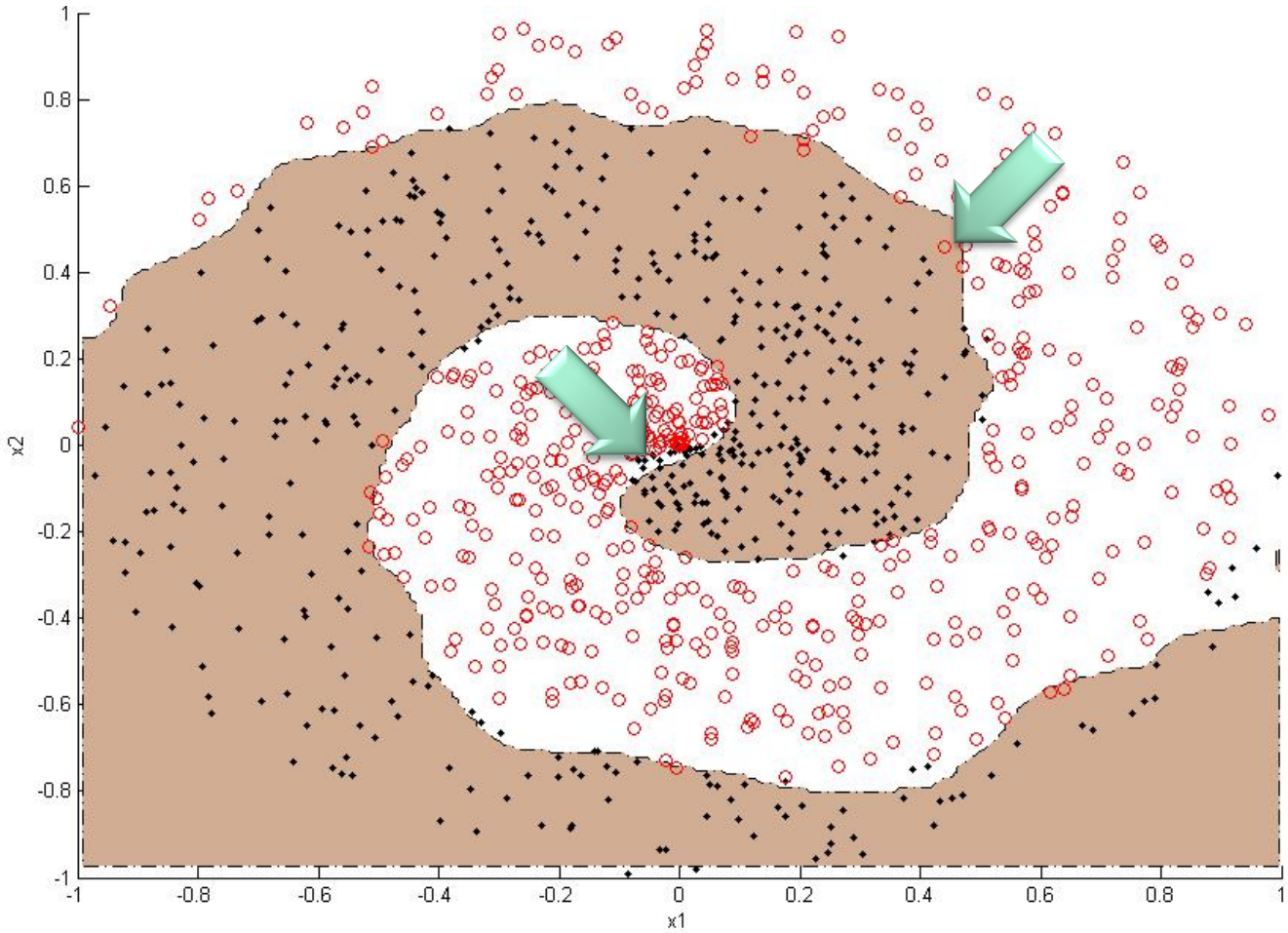
4.2 Parzen windows

Decision boundary (training data samples)



4.2 Parzen windows

Decision boundary (test data samples)



4.2 Parzen windows

$$g_i(x) = p(x|\omega_i)P(\omega_i)$$

Estimated likelihoods and priors can be used to perform classification following Bayes decision rule.

Classification result for the given test data:

```
Total error rate: 6.70%

    Class 1 error: 9.92%

    Class 2 error: 3.43%
```

4.3 Nearest Neighbor

Given: Training vectors $x_1, x_2, \ldots, x_N$ with known class dependency

$$(x_1, k_1), (x_2, k_2), \ldots, (x_N, k_N)$$

Nearest neighbor rule: $r(x) = \arg\min_k \left\{ \min_{n=1,\ldots,N_k} ||x - x_{nk}|| \right\}$

Minimum over all classes

All training samples

Adequate distance measure

→ Assign unknown test vector to the class of the training sample with the smallest distance

4.3 Nearest Neighbor

Derivation from Bayes Decision Rule:

Decision rule: $$r(x) = \arg\max_k \{p(k) \cdot p(x|k)\}$$

With
$$\hat{p}(k) = \frac{N_k}{N}$$

and
$$\hat{p}(x|k) = \frac{1}{N_k} \sum_{n=1}^{N_k} \varphi(||x - x_{nk}||)$$

Use Parzen window method to estimate distribution

we obtain
$$r(x) = \arg\max_k \left\{ \frac{N_k}{N} \cdot \frac{1}{N_k} \cdot \sum_{n=1}^{N_k} \varphi(||x - x_{nk}||) \right\}$$

4.3 Nearest Neighbor

Derivation from Bayes Decision Rule:

independent of k

Thus

$$
\begin{aligned}
r(x) &= \arg\max_k \left\{ \frac{1}{N} \cdot \sum_{n=1}^{N_k} \varphi(||x - x_{nk}||) \right\} \\
&\approx \arg\max_k \left\{ \max_{n=1,\ldots,N_k} \boxed{\varphi(||x - x_{nk}||)} \right\}
\end{aligned}
$$

maximum-approximation

function ist centered at zero → maximum at minimum distance

$$
r(x) \approx \arg\min_k \left\{ \min_{n=1,\ldots,N_k} ||x - x_{nk}|| \right\}
$$

4.3 Nearest Neighbor

Reduction of computational effort:

- High effort for computation of distances between test observation and all training samples
- Possible approximation: Determine the „partial distance"

$$||x - x_{nk}||_p = \left[\sum_{d=1}^{D' < D} (x_d - x_{nkd})^p \right]^{\frac{1}{p}}$$

Determined only for the first components

- Alternative: Approximately determine the class boundary and only consider training observations close to it.

4.3 Nearest Neighbor Classification

Error rate estimation

Definition:

$$x_{NN} := \arg \min_{x_n : x_1, \ldots, x_N} ||x - x_n||$$

nearest neighbor of x

Error probability for given x and x_{NN} when using the nearest neighbor rule?

4.3 Nearest Neighbor Classification

Error cases:

- Training sample x_n has class k_1 AND x_{NN} has class $c \neq k_1$ OR
- Training sample x_n has class k_2 AND x_{NN} has class $c \neq k_2$ OR
- …

$$p(k_1|x_n) \cdot \sum_{c \neq k_1} p(c|x_{NN}) + p(k_2|x_n) \cdot \sum_{c \neq k_2} p(c|x_{NN}) + \ldots$$

$$p(\text{error}|x, x_{NN}) = \sum_k \left[p(k|x) \cdot \sum_{c \neq k} p(c|x_{NN}) \right]$$

4.3 Nearest Neighbor Classification

$$
\begin{aligned}
p(\text{error}|x, x_{NN}) &= \sum_k \left[p(k|x) \cdot \sum_{c \neq k} p(c|x_{NN}) \right] \\
p(\text{error}|x, x_{NN}) &= \sum_k \left[p(k|x) \cdot \left[\sum_c p(c|x_{NN}) - p(k|x_{NN}) \right] \right] \\
p(\text{error}|x, x_{NN}) &= \sum_k \left[p(k|x) \cdot [1 - p(k|x_{NN})] \right] \\
p(\text{error}|x, x_{NN}) &= \sum_k p(k|x) - \sum_k p(k|x) \cdot p(k|x_{NN}) \\
p(\text{error}|x, x_{NN}) &= 1 - \sum_k p(k|x) \cdot p(k|x_{NN})
\end{aligned}
$$

4.3 Nearest Neighbor Classification

With an infinite number of training samples (NN is identical with x):

$$p(\text{error}|x, x_{NN}) = 1 - \sum_k p^2(k|x)$$

4.3 Nearest Neighbor Classification

Comparison with Bayes error:

$$p(\text{error}) = \int_x p(\text{error}, x)dx = \int_x p(\text{error}|x) \cdot p(x)dx$$

Local Bayes error:

$$\begin{aligned}
p(\text{error}|x) &= \sum_{k \neq r(x)} p(k|x) \\
p(\text{error}|x) &= \sum_k p(k|x) - p(r(x)|x) \\
p(\text{error}|x) &= 1 - p(r(x)|x) \\
p(\text{error}|x) &= 1 - p(\hat{k} = \arg\max_k p(k|x)|x) \\
p(\text{error}|x) &= 1 - max_k p(k|x)
\end{aligned}$$

4.3 Nearest Neighbor Classification

Insert

$$p(\text{error}|x) = 1 - max_k p(k|x)$$

into global error

$$p(\text{error}) = \int_x p(x) \cdot [1 - max_k p(k|x)] dx$$

$$p(\text{error}) = 1 - \int_x max_k p(x,k) dx$$

4.3 Nearest Neighbor Classification

Approximation of the error rate

$$p_B(\text{error}) \leq p_{NN}(\text{error}) \leq 2p_B(\text{error})$$

Comparison of local error:

Bayes $$p_B(\text{error}|x) = 1 - \max_k p(k|x)$$

Nearest Neighbor $$p_{NN}(\text{error}|x) = 1 - \sum_{k=1}^{K} p^2(k|x)$$

(inf. training samples)