Lecture 8

Nonparametric Density Estimation

Dept. of Electrical and Computer Engineering
0909.402.02 / 0909.504.04

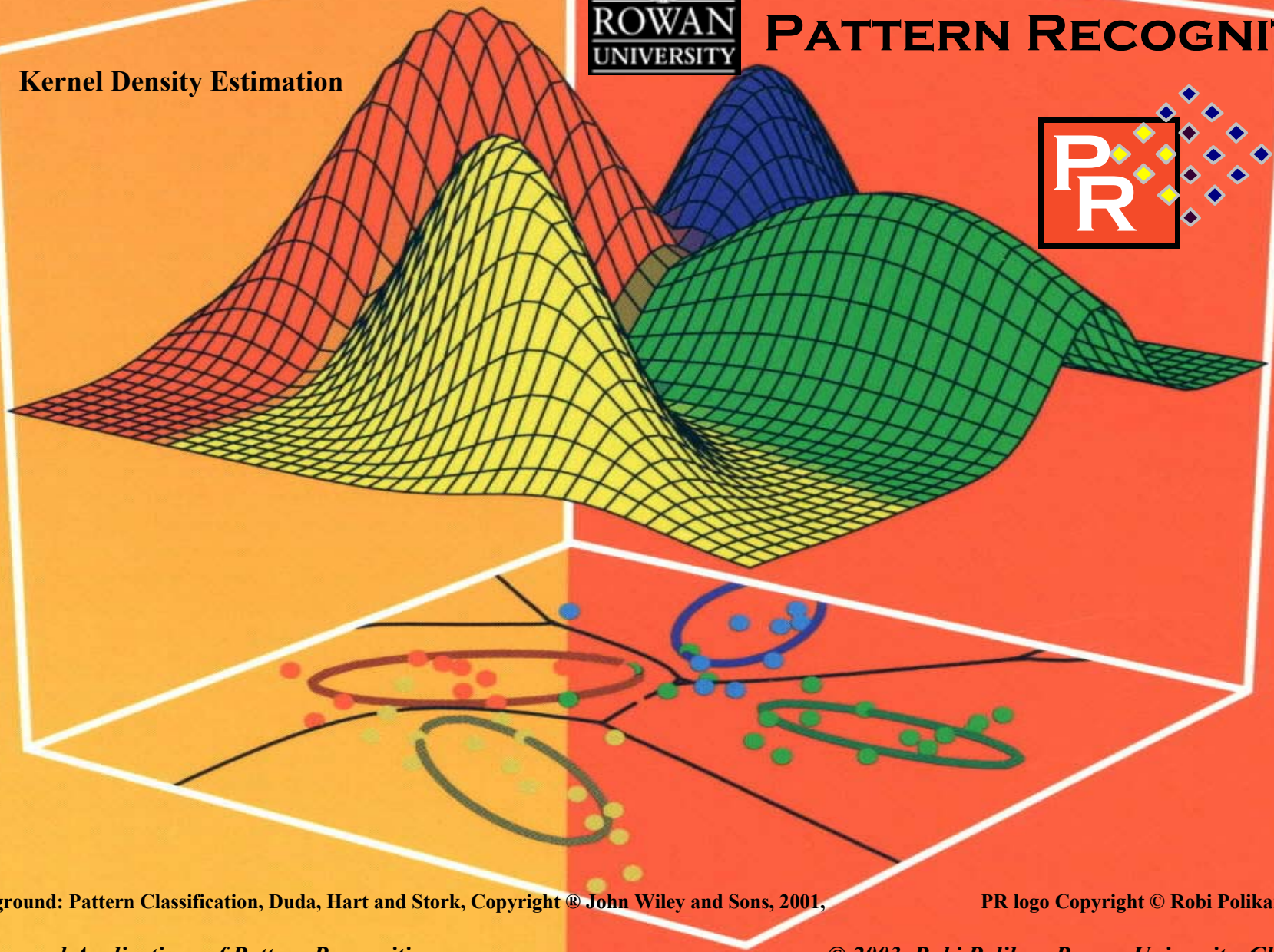
ROWAN UNIVERSITY

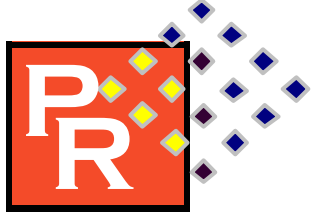
Theory & Applications of

Pattern Recognition

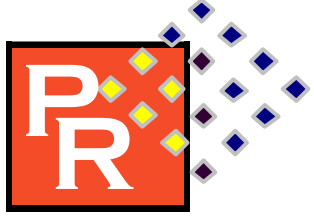
PR

Kernel Density Estimation





- Nonparameteric Density Estimation
  - Density estimation based on probability
  - Two approaches
    - Kernel density estimation - Parzen windows
    - K- nearest neighbors (coming soon to this classroom!)
  - Parzen windows



Non-Parametric Techniques

- Problems with parametric techniques:
  - Assume forms of distributions functions are known. nOt tRUe !
    - We normally do not know the density form for most practical applications.
  - Most known forms of distributions are unimodal (single peak). nOt eVeN cLoSE !
    - In most practical applications, distributions are multimodal
  - In most cases individual features are assumed to be independent. wRonnnG !
    - Approximating a multivariate distribution as a product of univariate distributions do not work well in practice
- Non-parametric techniques: Estimate the distribution function from scratch !
  - But…but would that not be difficult…? fuNNy yOu asKeD !

Density Estimation

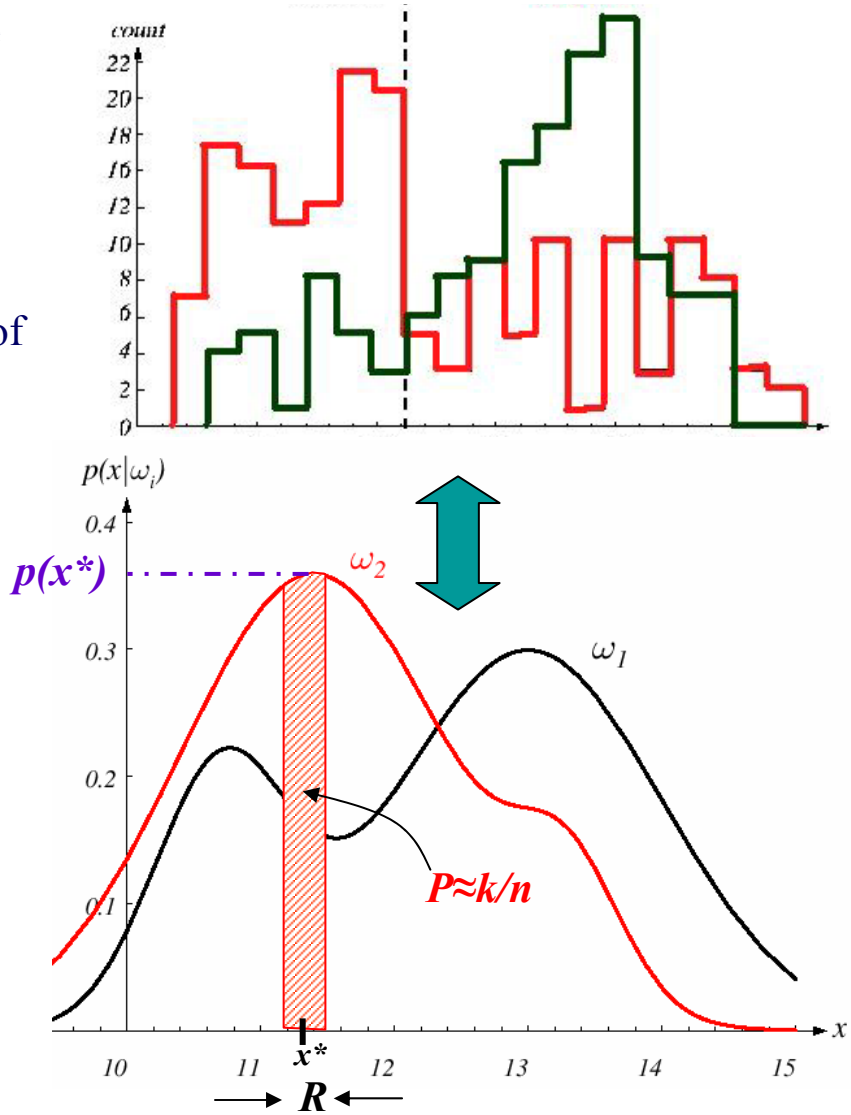
- Based on the fact that the probability that a given sample **x** falls within a region (range) of *R* is given by

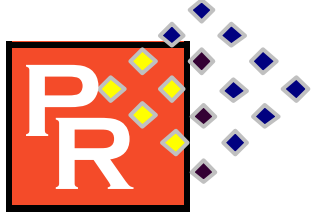
$$P = \int_{\mathbf{x} \in R} p(\mathbf{x}) d\mathbf{x}$$

- This integral can be approximated either by the product of the value of p(**x**) with the area / volume of the region (assuming that this area is small), or by the number of samples fall within the region:
  - if we observe a large number, *n*, of fish and count those whose length fall within the range defined by *R*, then *k*/*n* can be used as n estimate of *P* as n→∞

$$P = \int_{\mathbf{x} \in R} p(\mathbf{x}) d\mathbf{x} \approx p(\mathbf{x}^*).V \approx k/n$$

V: Volume enclosed by R, surrounding x*, in 1-D, V=R
k: Number of samples falling into region R
n: Total number of samples





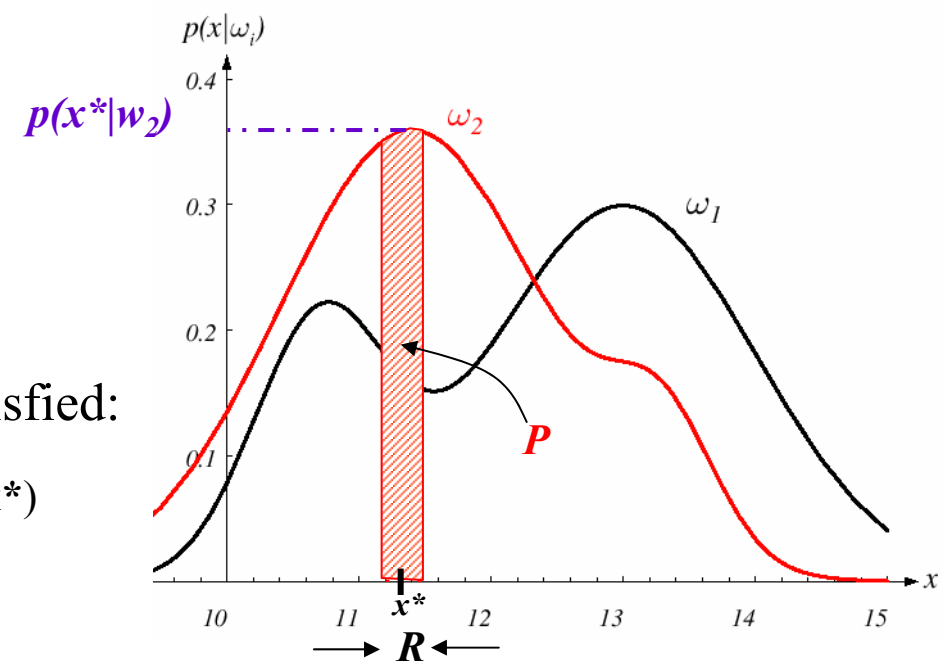
However…

- In order to make sure that we get a good estimate of p(**x**) at each point, we have to have lots of data points (instances) for any given *R* (or volume V). This can be done in two ways
  1. We can fix V and take more and more samples in this volume. Then $k/n \rightarrow P$, however, we then estimate only an average of of p(**x**), not the p(**x**) itself, because *P* can change in any region of nonzero volume
  2. Alternatively, we can fix *n* and make $V \rightarrow 0$, so that p(x) is constant in that region. However, in practice we have a finite number of training data, so as $V \rightarrow 0$, V will be so small that it will eventually contain no samples: $k=0 \rightarrow p(\mathbf{x})=0$, a useless result!
- Therefore, $V \rightarrow 0$ is not feasible and one has to live with the fact that there will always be some variance in k/n and hence some averaging in p(x) within the finite non-zero volume *V*.
  - A compromise need to be found for V so that
    - It will be large enough to contain sufficient number of samples
    - It will be small enough to justify our assumption of p**(x)** be constant within the chosen volume/region.

Density Estimation

To estimate the density at **x***, choose a sequence of regions $R_1, R_2, \ldots, R_n$, containing **x***, where R_i includes i instances. Let V_n be the volume of R_n, k_n be the number of samples falling into the n^{th} region, and $p_n(\mathbf{x}^*)$ be the n^{th} estimate of $p(\mathbf{x}^*)$. Then

$$p_n(\mathbf{x}^*) = \frac{k_n/n}{V_n} \approx p(\mathbf{x}^*)$$

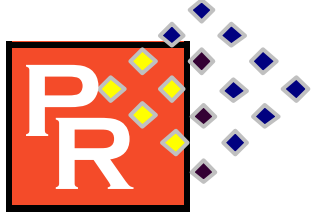


To make sure that k/n is a good estimate of P, and consequently, $p_n(x^*)$ is a good estimate $p(x^*)$, the following need to be satisfied:

$\lim_{n\to\infty} V_n = 0$ → space averaged P/V→p(**x***)

$\lim_{n\to\infty} k_n = \infty$ → $k_n/n \to P$

$\lim_{n\to\infty} k_n/n = 0$ → $p_n(\mathbf{x}^*)$ to converge p(**x***)

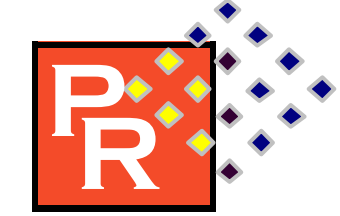


Density Estimation

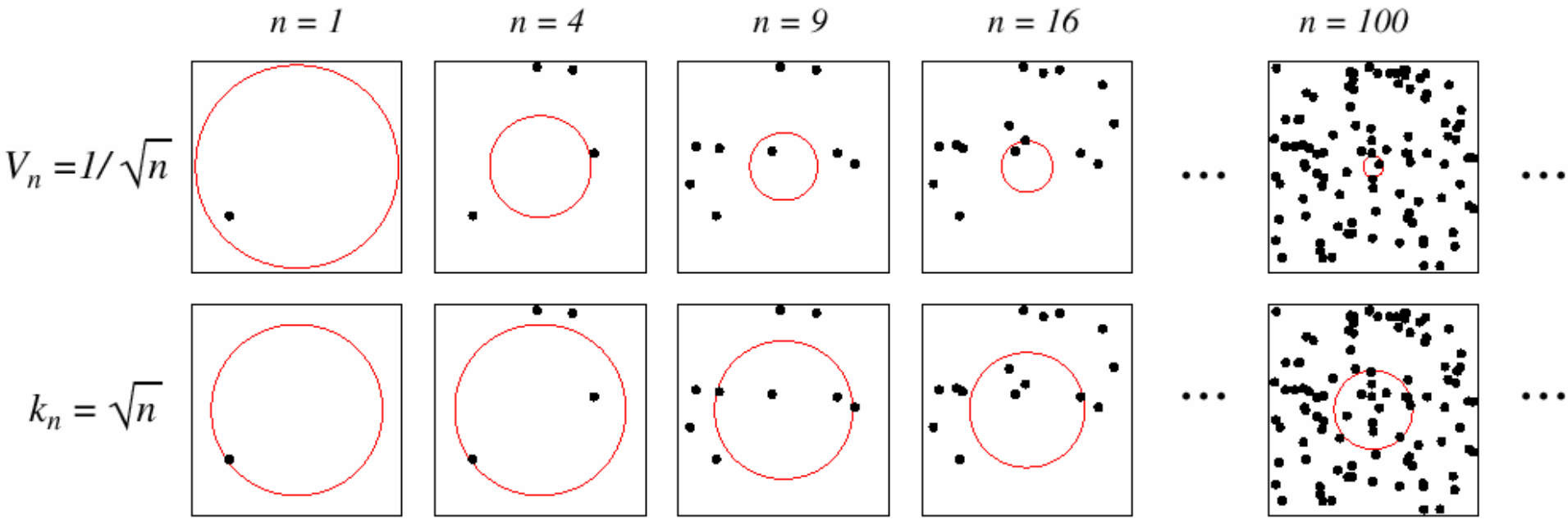
- There are two ways to ensure these conditions:
  1. Shrink an initial volume V_n as a function of n, e.g., $V_n = V_1 / \sqrt{n}$. Then, as n increases so does k, which can be determined from the training data
     - **Kernel Density Estimation**, aka, **Parzen Windows (PW)** density estimation

  2. Specify k_n as a function of n, $k_n = \sqrt{n}$ V_n → grows until it encloses k_n samples. Then V_n can be determined from the training data
     - **K- Nearest Neighbor (KNN)**

- It can be shown that as $n \to \infty$, both KNN and PW approach the true density p(**x**), provided that V_n shrinks and k_n grows proportionately with n.



Two Approaches

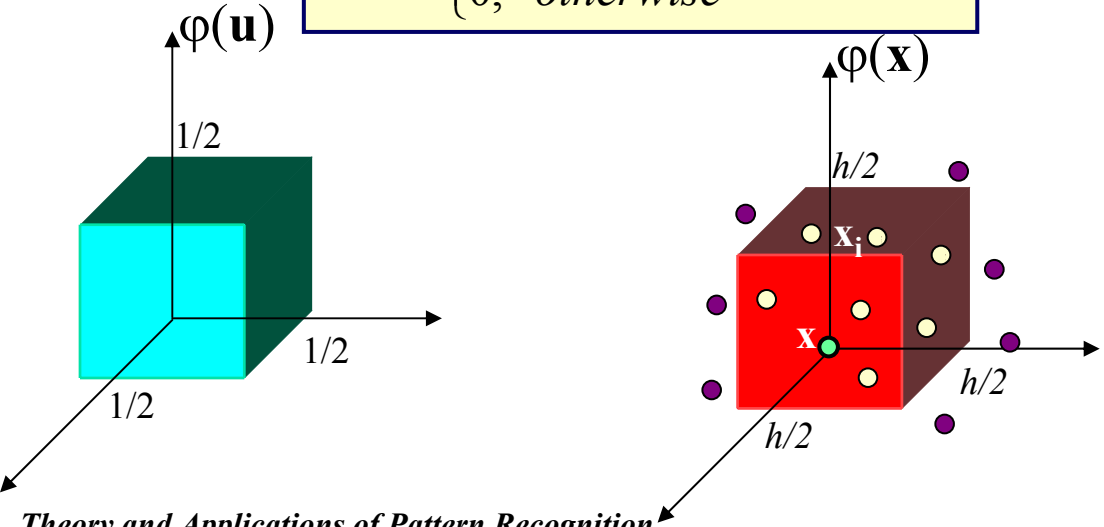


Parzen Windows

- Based on counting the number of samples within a given region, with the region reduced as the number of samples are increased.
- The number of samples falling into the a specified region is obtained by the help of a windowing function, hence the name Parzen windows.
  - We first assume that R is a d-dimensional hypercube of each side h, whose volume is then $V=(h)^d$
  - Then define a window function $\varphi(\mathbf{u})$, called a ***kernel function***, to count the number of samples k that fall into R

$$\varphi(\mathbf{u}) = \begin{cases} 1, & |\mathbf{u}_j| \le 1/2 \quad j = 1,2,\cdots,d \\ 0, & otherwise \end{cases} \Rightarrow \varphi\left(\frac{\mathbf{x}-\mathbf{x_i}}{h}\right) = 1$$

if the point $\mathbf{x_i}$ falls inside the hypercube centered at $\mathbf{x}$ with each side of h_n, 0 otherwise.



The number of samples in this hypercube is then

$$k = \sum_{i=1}^{n} \varphi\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)$$

Parzen Windows

$$p(\mathbf{x}) \approx \frac{k_n / n}{V_n} \qquad \& \quad k = \sum_{i=1}^{n} \varphi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)$$

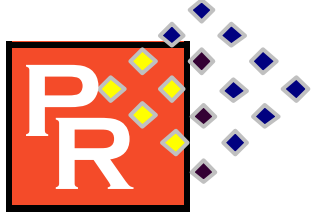
$$\tilde{p}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{V} \varphi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)$$

Estimate of the density: number of samples falling into R, where R is centered at $\mathbf{x}$ with a width of h

The volume covered by the window function

Window function (kernel)

The width of the window function



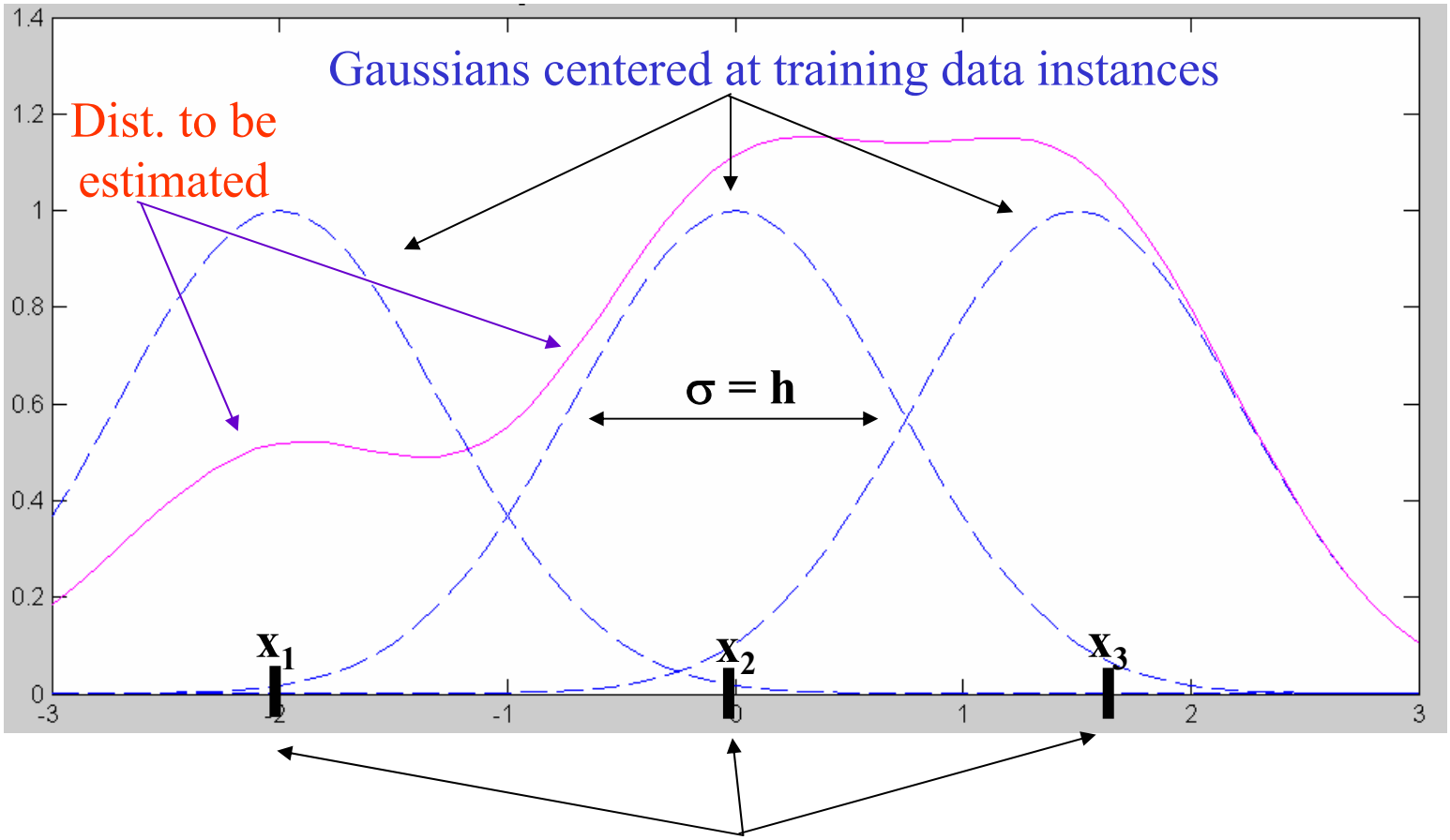
PARZEN WINDOWS

- Now consider φ(.) as a general function, typically a smooth and continuous function, instead of a hypercube. The general expression of p(**x**) remains unchanged

$$\tilde{p}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{V}\varphi\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) = \frac{1}{nh^d}\sum_{i=1}^{n}\varphi\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)$$

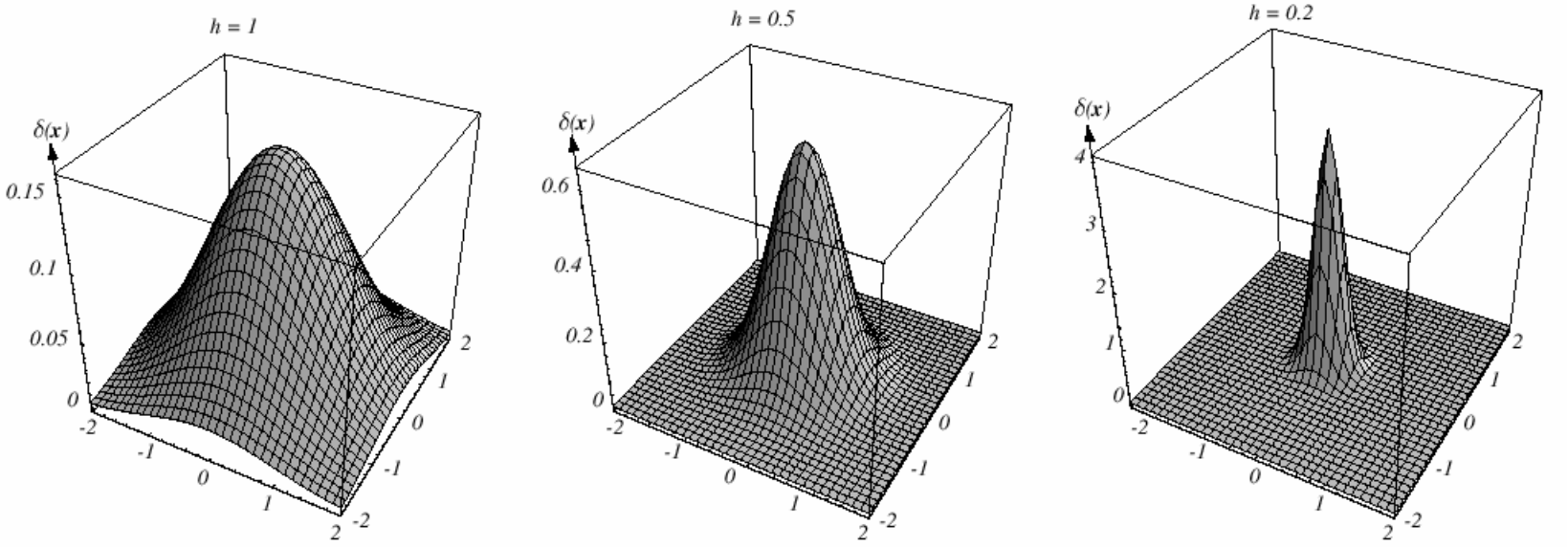
- Then $\tilde{p}(\mathbf{x})$ is a superposition (interpolation) of φ(.)s, where each φ(.) measures how far a given $\mathbf{x}_i$ is from **x** !!!
  - In practice, $\mathbf{x}_i$ are the training data points and we estimate $\tilde{p}(\mathbf{x})$ by interpolating the contributions of each sample data point $\mathbf{x}_i$ based on its distance from **x**, the point at which we want to estimate the density. The kernel function φ(.) provides the numerical value of this distance!
  - If φ(.) is itself a distribution, then $\tilde{p}(\mathbf{x})$ will converge to p(**x**) as *n* increases. A typical choice for φ(.) is – you guessed it right… – the Gaussian !

- The density p(**x**) is then estimated simply by a superposition of Gaussians, where each Gaussian is centered at the training data instances. The parameter *h* is then the variance of the Gaussian !!!



Choosing the Kernel Width

- The parameter h, also called the ***smoothing parameter*** or ***kernel bandwidth***, the width of the kernel (window) function, plays a crucial rule. Why? Consider the two extreme cases
  - If h is too large $\mathbf{x}$ must be too far away from $\mathbf{x_i}$ before the it can fall outside the window of width h. In this case, $p(\mathbf{x})$ is the summation of many broad slowly changing out-of-focus windows
  - If h is too small, then the windows are very narrow (say dirac deltas) that are very focused at $\mathbf{x_i}$. Any $\mathbf{x}$ that is just a little bit away from $\mathbf{x_i}$ will fall outside of the window. In this case, $p(\mathbf{x})$ is the sum of many sharp pulses centered at $\mathbf{x_i}$, giving an erratic noisy estimate.



Effect of Kernel Width

- A large bandwidth will over-smooth the density and mask the structure in the data
- A small bandwidth will yield a density estimate that is spiky and very hard to interpret

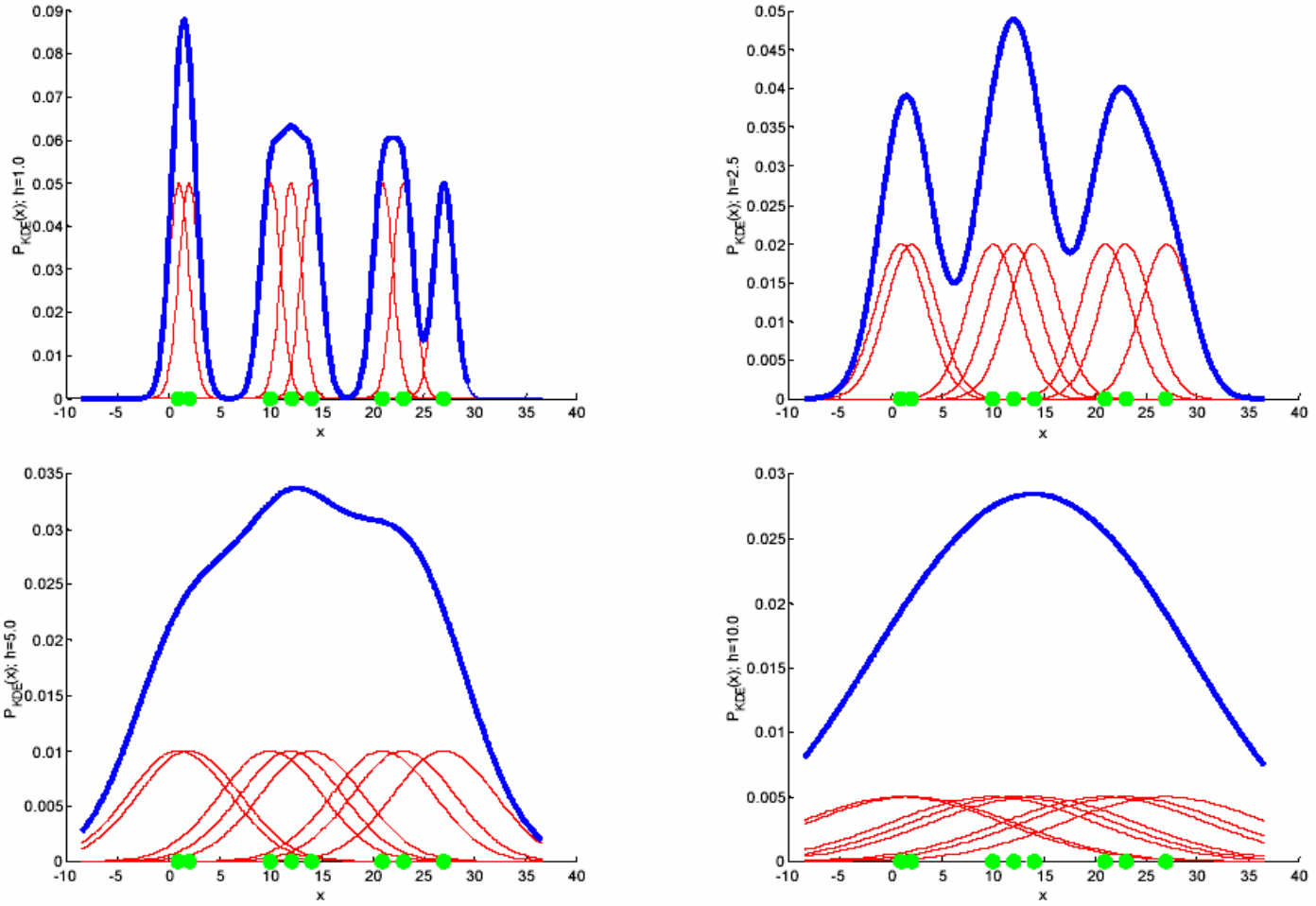


Figure courtesy of R. Gutierrez

Bias – Variance Dilemma Revisited…

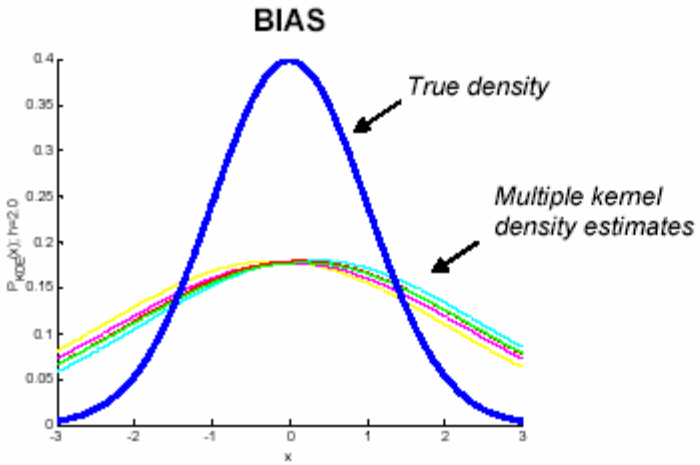
- Clearly we would like to obtain a density estimate that is as close to the original as possible. One way to measure this closeness is the mean square error at the estimation point, say **x** as

$$MSE_{\mathbf{x}}\left(\tilde{p}(\mathbf{x})\right) = E\left[\left(\tilde{p}(\mathbf{x}) - p(\mathbf{x})\right)^2\right] = \left\{E\left[\tilde{p}(\mathbf{x}) - p(\mathbf{x})\right]\right\}^2 + \mathrm{var}\left(\tilde{p}(\mathbf{x})\right)$$

Bias of the estimate | Variance of the estimate

  - Bias: Systematic error in the estimate
  - Variance: Random error in the estimate – variation of the estimate from one dataset to another

- Large bandwidth reduces variance (through superposition of many broad kernels), but adds bias to the estimate, whereas small bandwidth reduces the bias but adds random variance (due to very narrow – spiky looking – kernels).



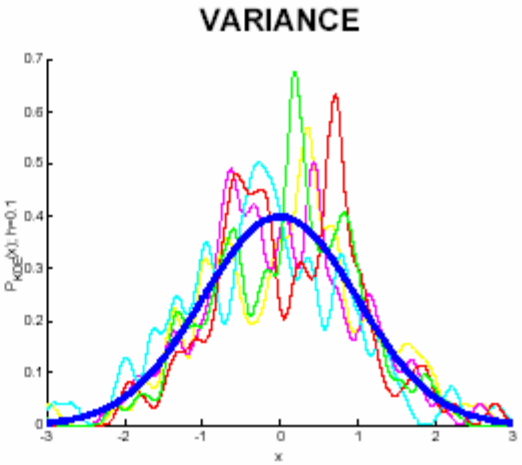


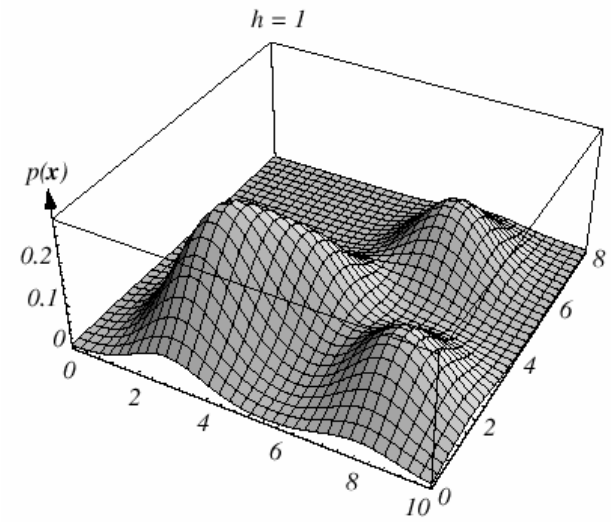
Figure courtesy of R. Gutierrez

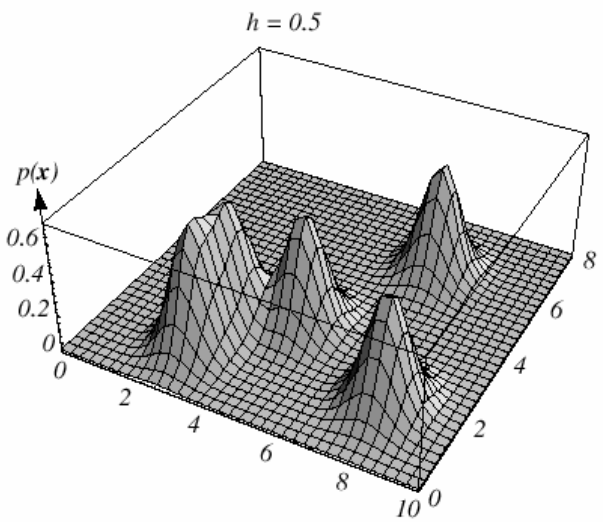
Effect of Kernel Width

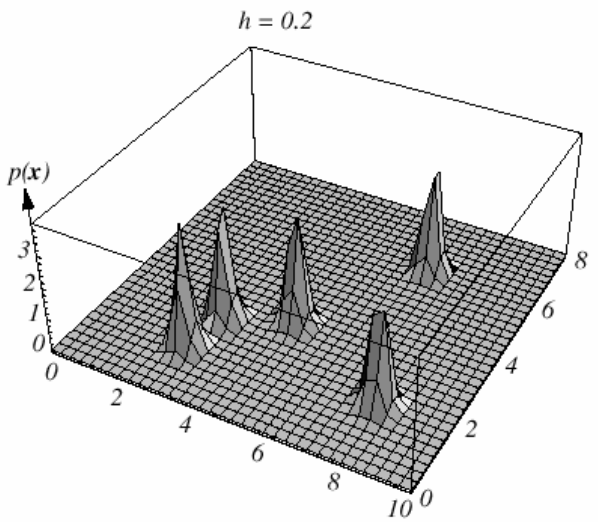
2-D

Large V_n ➔ too little resolution

Small V_n ➔ too much statistical variability







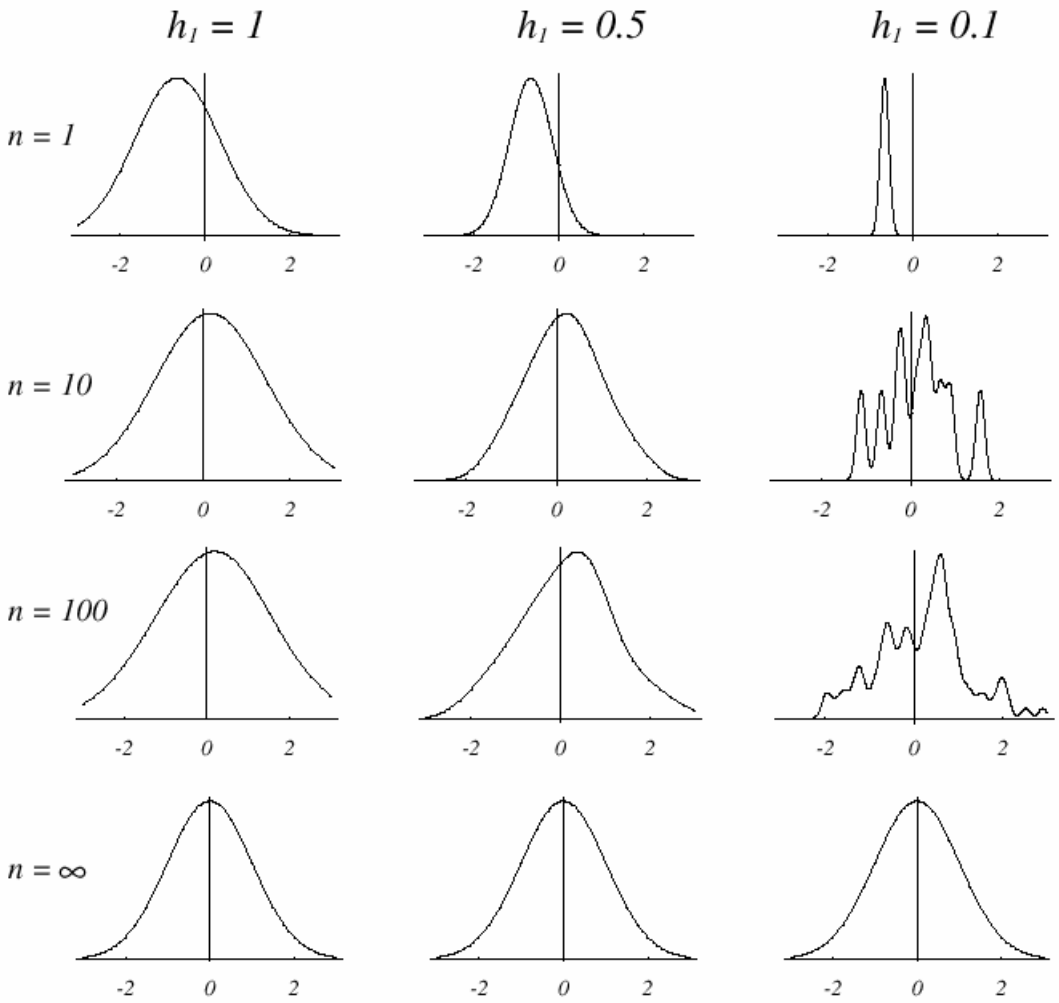
Effect of Kernel Width

Let p(**x**)~N(0,1)

$$\varphi(u) = -\frac{1}{\sqrt{2\pi}} e^{-u^2/2}$$

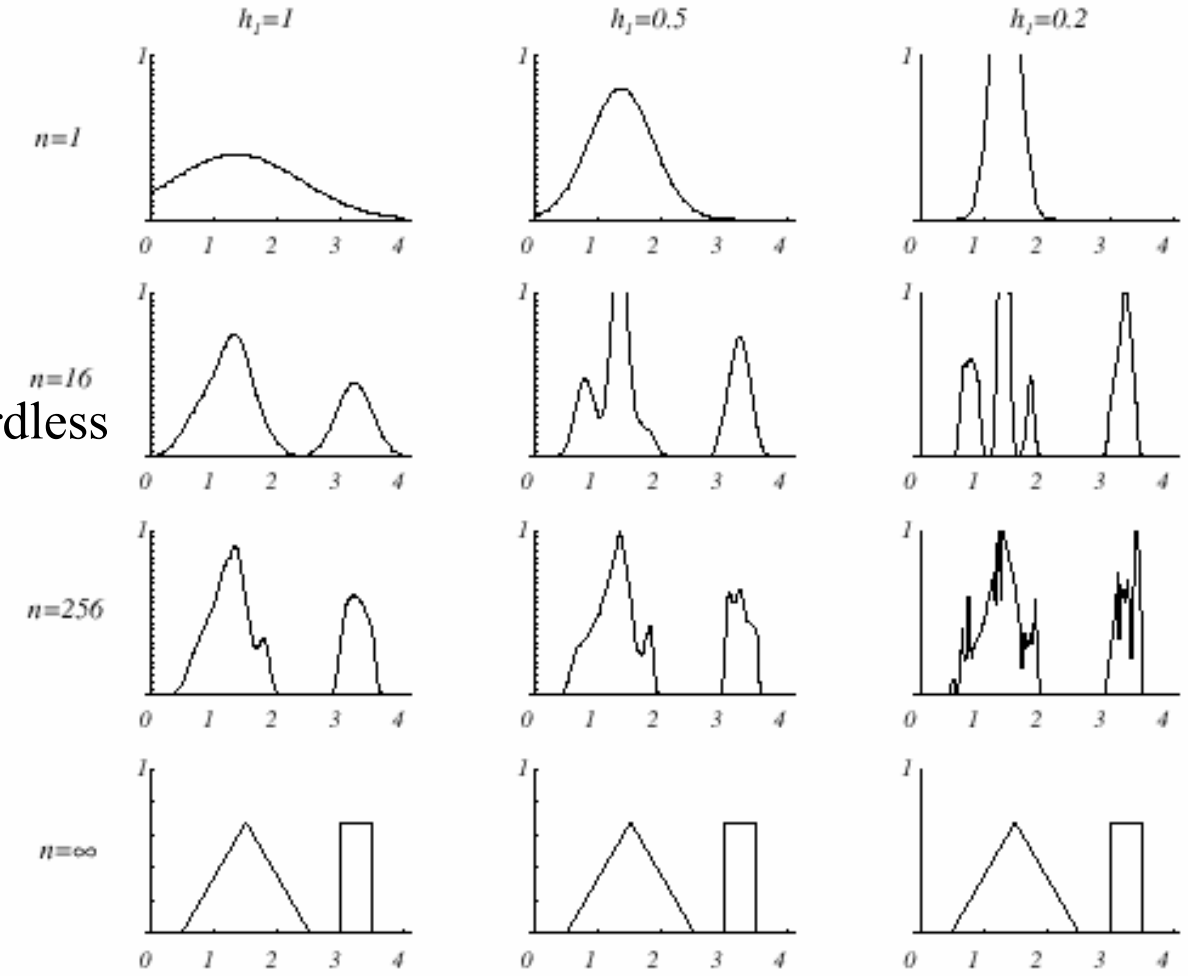
$$h = h_1 / \sqrt{n}$$

$$\tilde{p}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h} \varphi\left(\frac{x - x_i}{h} \right)$$



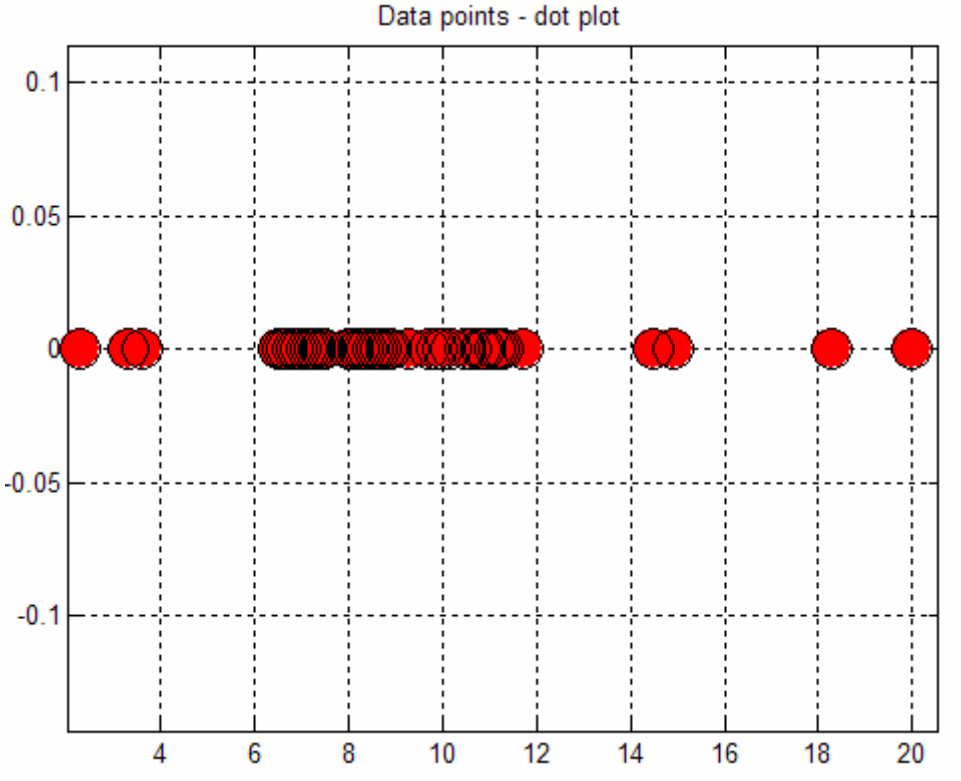
Effect of Sample Size

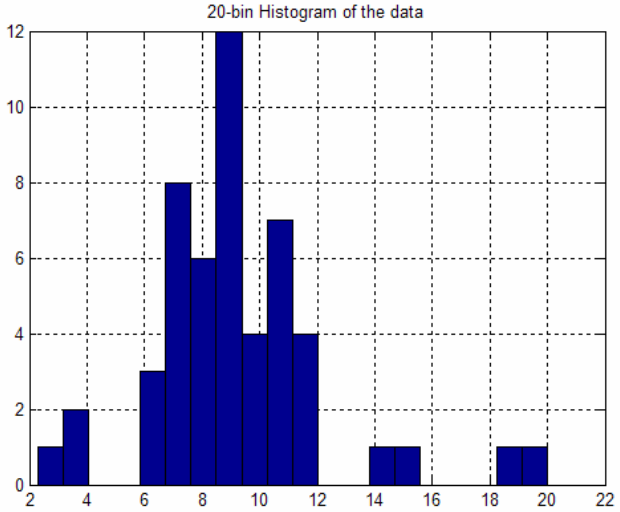
Note: As n approaches infinity, the estimation becomes accurate, regardless of the window length.

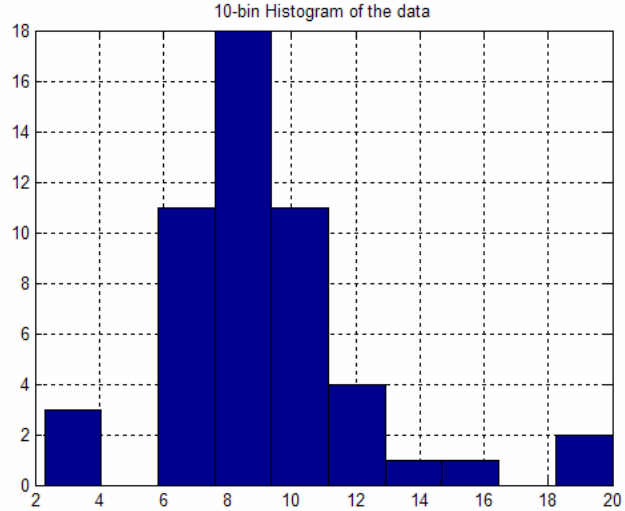


An Example

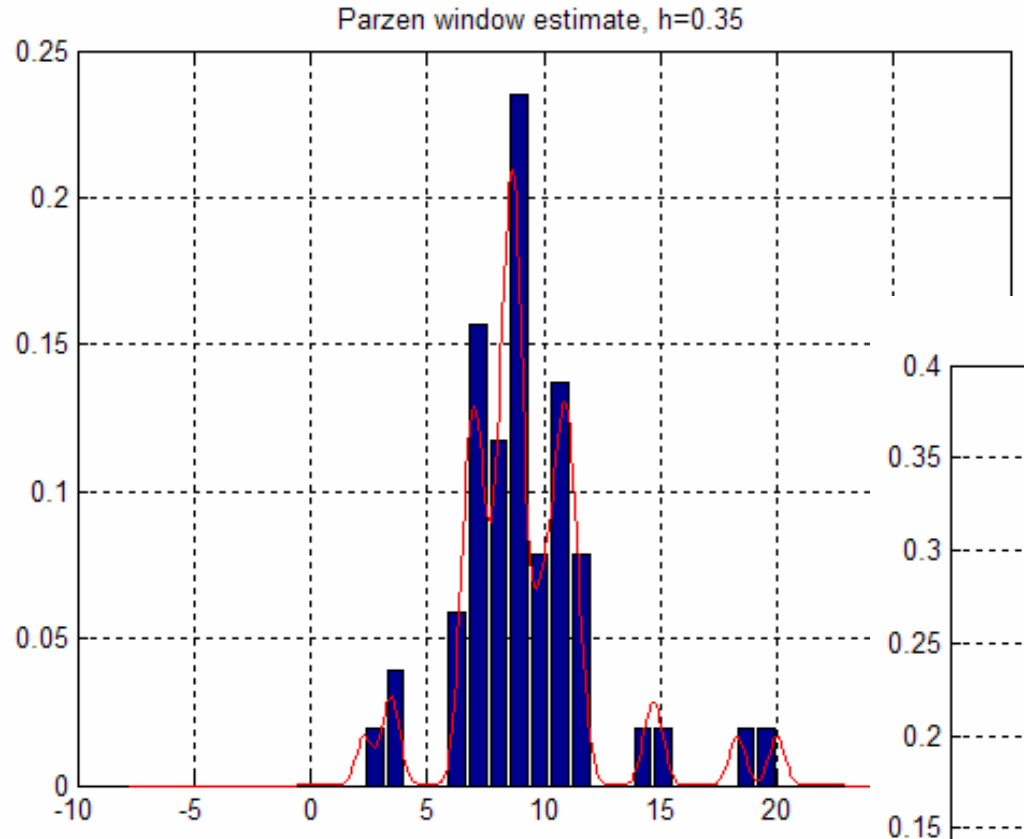
- D= [2.3 3.3 3.6 6.5 6.6 6.7 6.8 6.9 7 7.1 7.2 7.3 7.4 7.5 8.0 8.1 8.2 8.2 8.3 8.4 8.5 8.6 8.6 8.7 8.8 8.8 8.8 8.9 8.9 8.9 8.9 9.3 9.7 9.9 10 10.2 10.5 10.6 10.6 10.8 10.9 10.9 11.1 11.2 11.2 11.3 11.7 14.5 14.9 18.3 20];





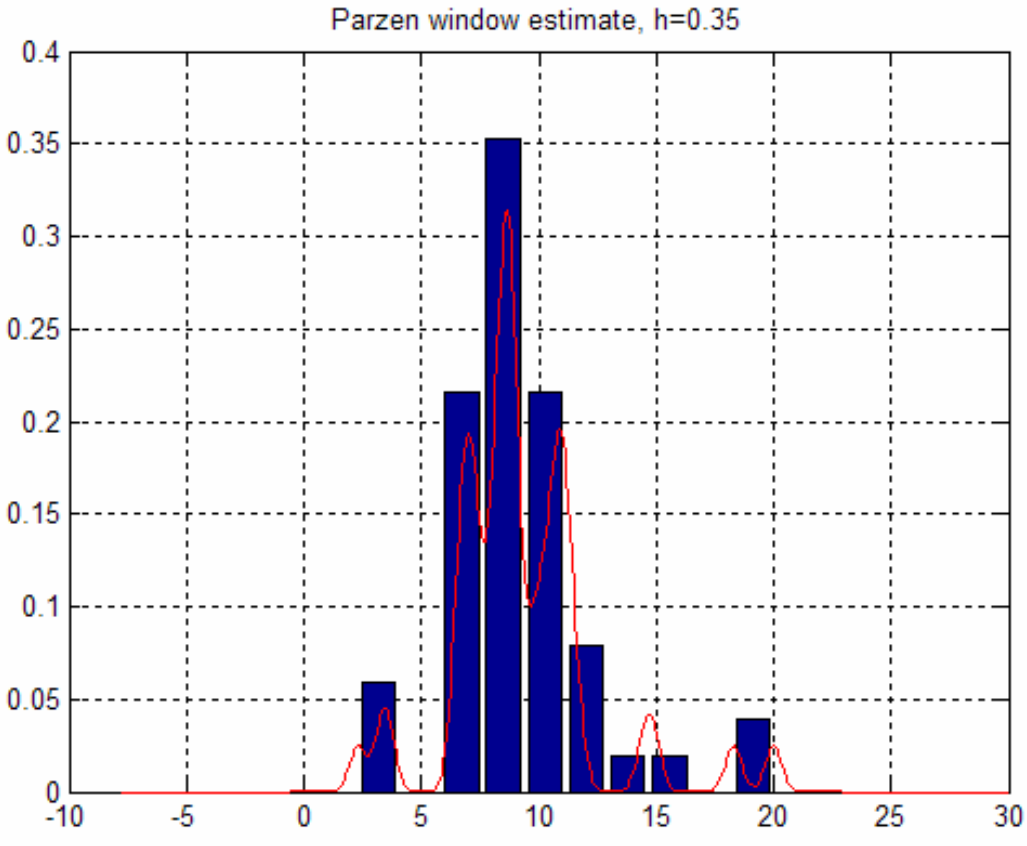


Parzen Window Estimates

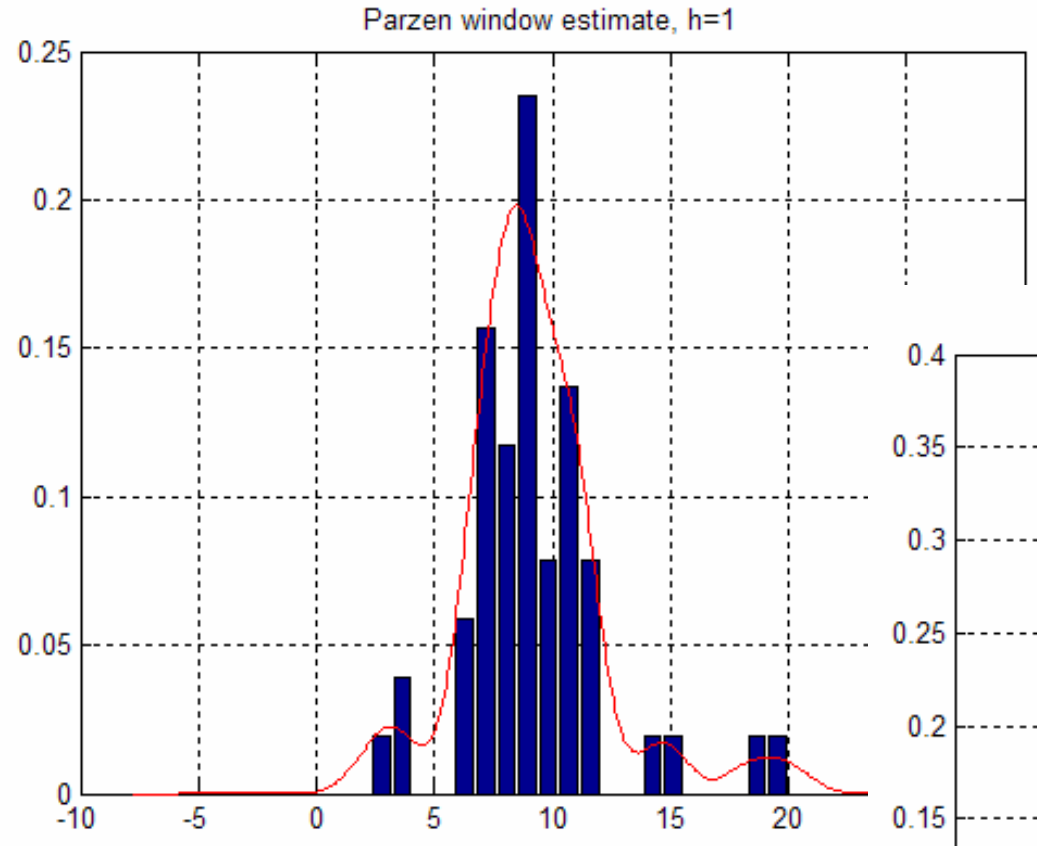


(Histogram bin size 20)

(Histogram bin size 10)

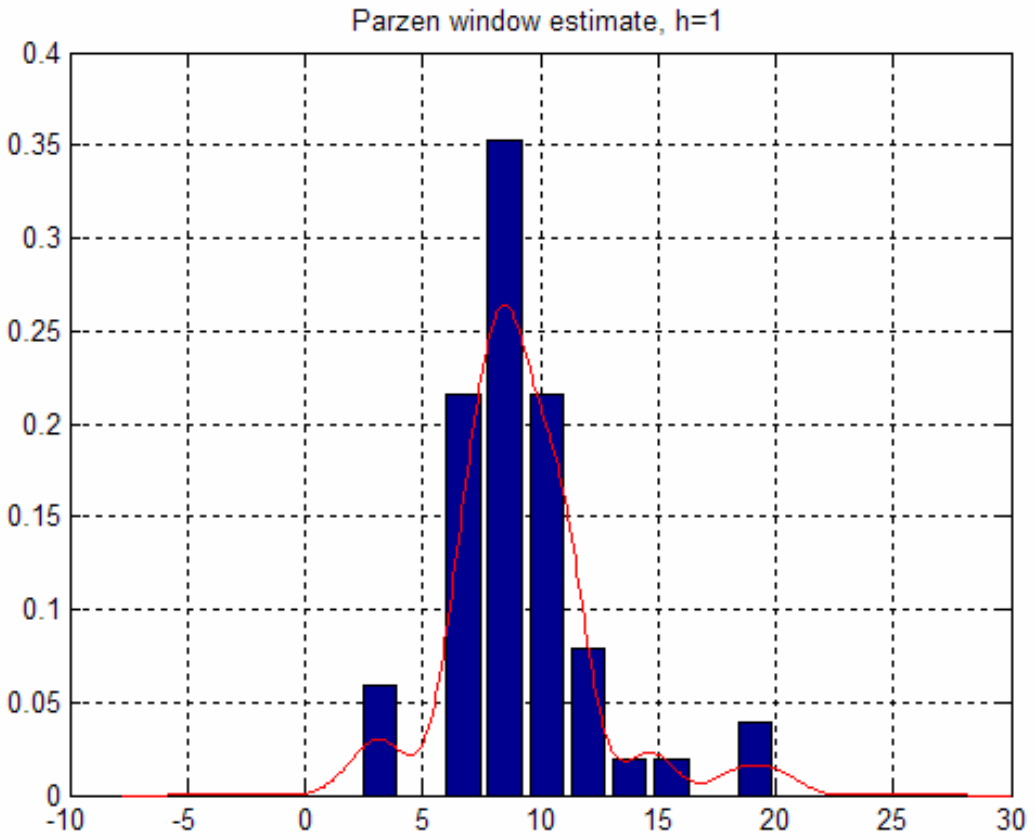


Parzen Window Estimates

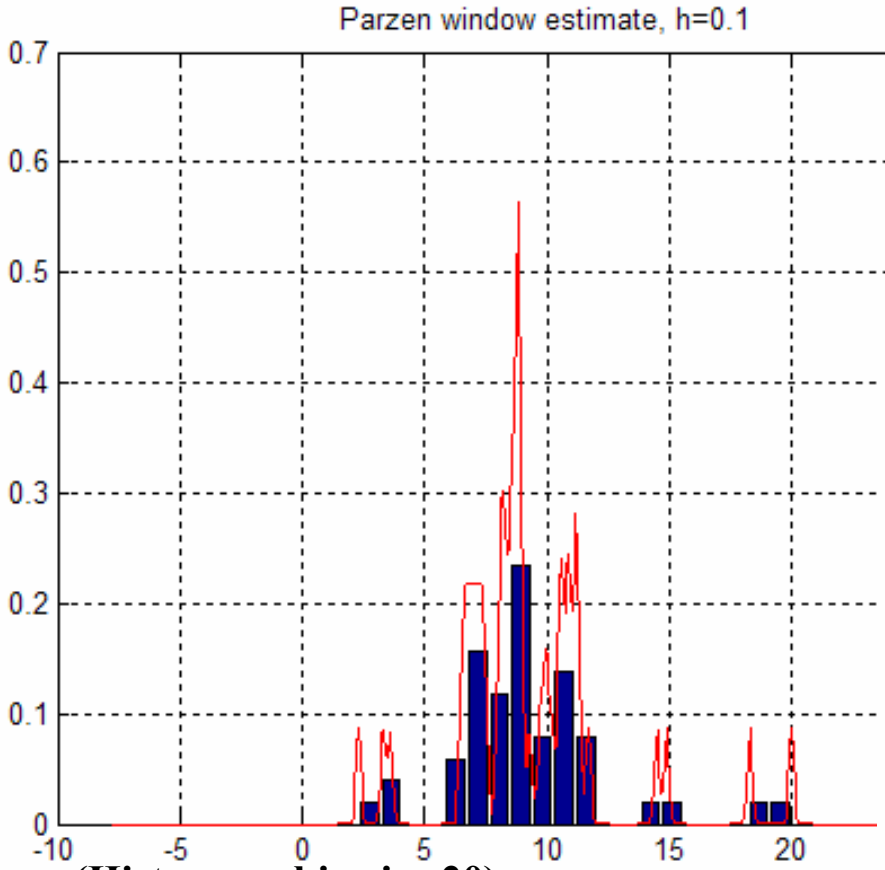


(Histogram bin size 20)

(Histogram bin size 10)

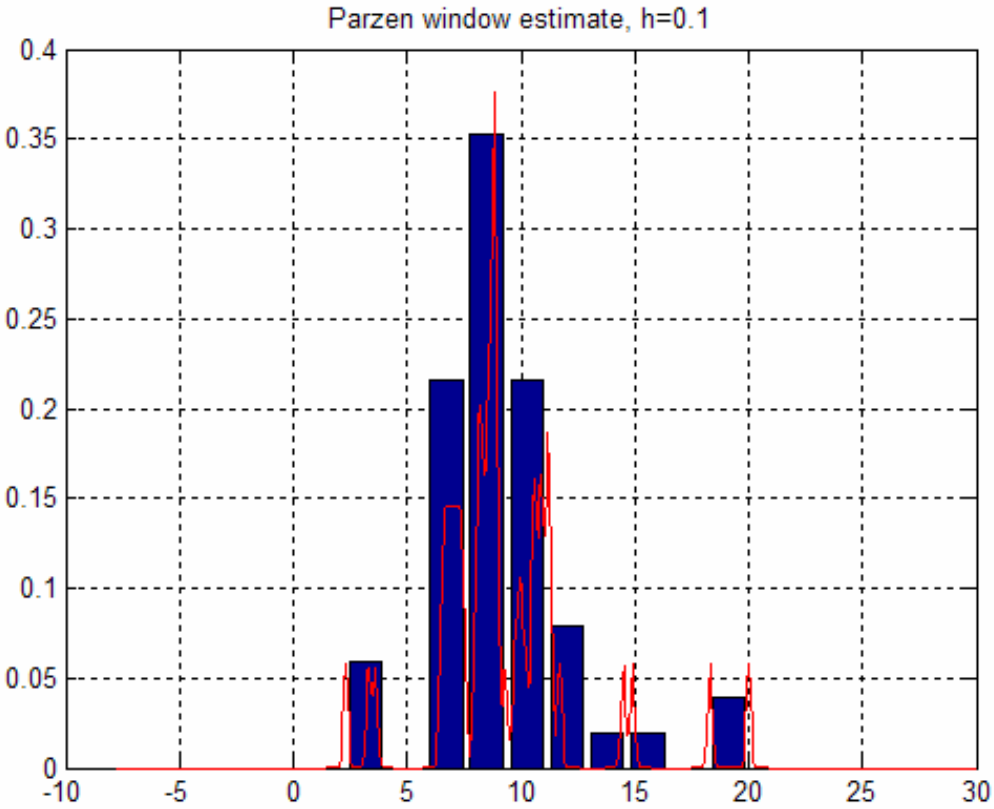


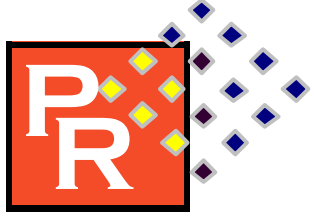
Parzen Window Estimates



(Histogram bin size 20)

(Histogram bin size 10)





Well then …How to Choose?

- A few suggestions:

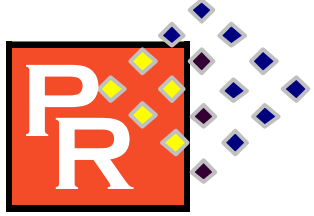
1. Trial and error: Try a number of different h values and decide on the one that appears to give the best looking density ➔ Obviously very subjective and impractical in large dimensional problems
2. For a known Gaussian distribution, the h_{opt} can be obtained as the maximum likelihood estimate of h that minimizes the error between the actual density and the estimated density. If the distribution is in fact non-Gaussian but still unimodal, this estimate may be good enough (Silverman '86):

$$h_{opt} = \left(\frac{4}{d+2}\right)^{\frac{1}{d+4}} \sigma n^{-\frac{1}{d+4}}$$

3. For multi-modal densities in one dimension, the above estimate is empirically updated to give a decent estimate:

$$h_{opt} = 0.9 \cdot n^{-1/5} \cdot \min\left(\sigma, \frac{IQR}{1.34}\right)$$

where IQR is the interquartile range (fourth spread), a robust estimate of the standard deviation.



PRODUCT KERNEL

- One simplification in estimating multi-dimensional densities is by multiplying one-dimensional kernels, called product kernels

$$\tilde{p}_p(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\varphi(\mathbf{x}, \mathbf{x}_i, h_1, \cdots h_d)$$

$$where \quad \varphi(\mathbf{x}, \mathbf{x}_i, h_1, \cdots h_d) = \frac{1}{h_1 \cdots h_d}\prod_{j=1}^{d}\varphi_j\left(\frac{x^{(j)} - x_i^{(j)}}{h_j}\right)$$

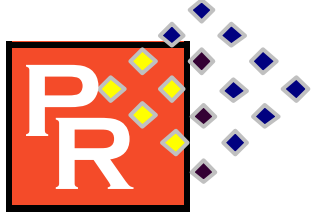
j^{th} dimension of $\mathbf{x}$ or $\mathbf{x_i}$

Different widths are used for each dimension

Kernel function for the j^{th} dimension. Typically $\varphi_j(.) = \varphi(.)$

- Note: Kernel independence is assumed above – which does NOT imply feature independence, which would give

$$\tilde{p}_{feat\_ind}(\mathbf{x}) = \prod_{j=1}^{d}\frac{1}{nh_j}\sum_{i=1}^{n}\varphi_d\left(\frac{x^{(j)} - x_i^{(j)}}{h_j}\right)$$



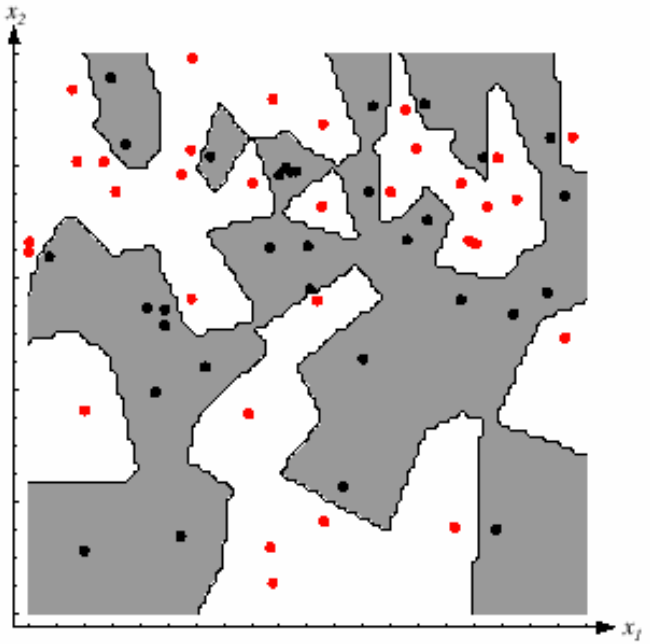
Classification using Parzen Windows

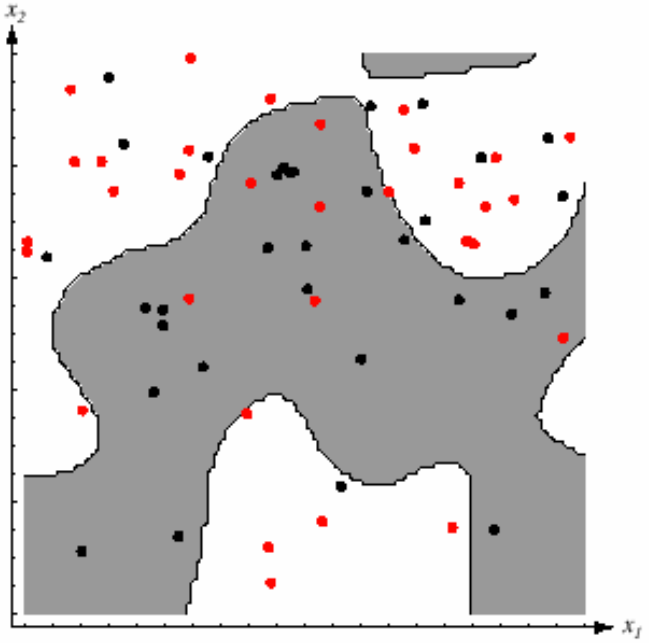
- Estimate the likelihood p(**x|w**), from the data using the Parzen window approach, and use Bayes rule for classification, that is, compute the posterior probabilities, and decide on the class that has the largest posterior probability.
- Advantages: Does not assume any prior knowledge!
- Disadvantages: Need (lots of)^n data to make sure that the estimate converges to the true distribution.
  - Furthermore, as dimensionality increases, the demand for (lots of)^n data becomes ((lots of)$^{\text{lots of (n)}}$)^n !!!! ➔ Curse of dimensionality !
  - The only way to break this curse is to have some "correct" prior information !
- Training error can be made arbitrarily small (or even zero), by choosing small enough windows ! However, this is not desirable, since it is sure to cause overfitting and deteriorated test data performance.

Classification using Parzen Windows

Small window width→ Too much dissection of the feature space. No good !

Larger window width→higher training error, but most likely better generalization performance! Better generalization performance : Good





In fact, what we really need to do is to pick windows of narrow width at those areas of high data density, and windows of wide large width where data is sparsely populated! How can we do this…?

Probabilistic Neural Networks (PNN)

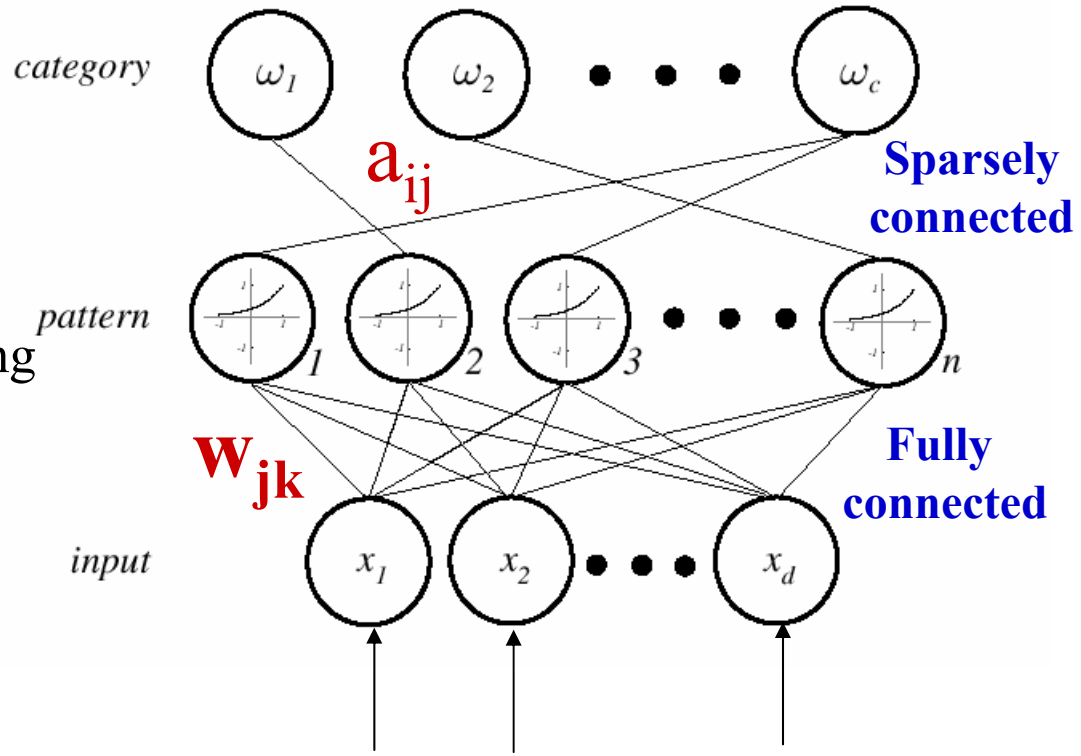
Input $\mathbf{x_k}$: $k=1,2,\ldots,d$ d – nodes, each corresponding to one feature

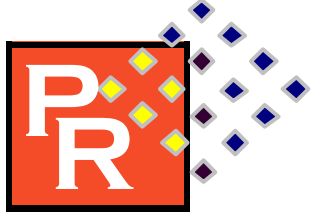
$\mathbf{W_{jk}}$: weights connecting k^{th} input to j^{th} hidden layer node (pattern node)

Hidden layer: n nodes, each corresponding to one of the training data patterns. $j=1,2,\ldots,n$

Output layer: c nodes, each representing one of the classes

a_{ij}: weights connecting j^{th} hidden node to i^{th} output node $i=1,2,\ldots,c$



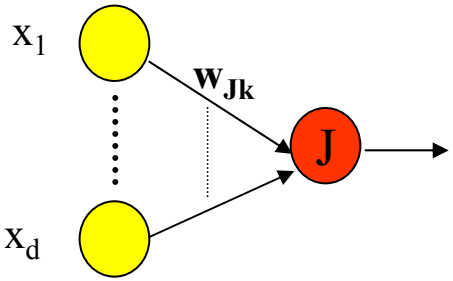


PNN
Training & Classification

- Training: For each pattern $\mathbf{x}_j$ of the training dataset $j=1,\ldots,n$
  - Normalize $\mathbf{x}_j$ to unit length
  - Set the weights for the corresponding hidden node as $\mathbf{w}_j = \mathbf{x}_j$
  - Connect j^{th} pattern node to corresponding i^{th} class node $i=1,\ldots,c$ with a weight of 1

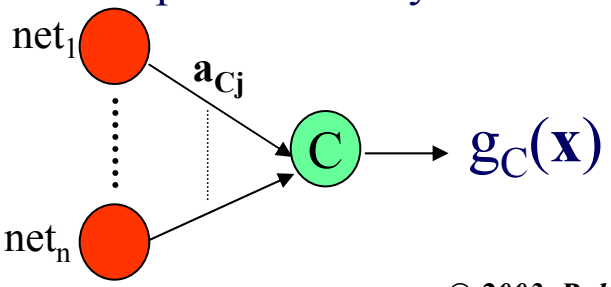
- Classification – averaging n kernels centered at training data points
  - Place a normalized test pattern $\mathbf{x}$ at the PNN input
  - Compute the net activation of the j^{th} node, net_j
  - For each class i add the total contribution from all hidden nodes to obtain $g_i(\mathbf{x})$
  - Pick the class for which $g_i(\mathbf{x})$ is maximum

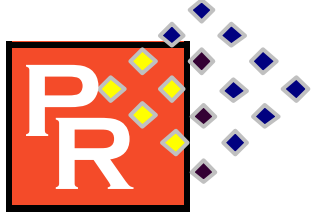
- A probabilistic neural network (PNN) consists of d input units, n pattern units, and c category units. Each pattern unit forms the inner product of its weight vector and the normalized pattern vector **x** to form $net_J = \mathbf{w}^t\mathbf{x}$, and then it emits $e^{[(net_J - 1)/\sigma^2]}$. Note that σ serves as the width of the kernel function.



$$e^{\left((net_J - 1)/\sigma^2\right)} \qquad net_J = \sum_{k=1}^{d} w_{Jk} \cdot x_k$$

- Each category unit sums such contributions from the pattern (hidden) unit connected to it. This ensures that the activity in each of the category units represents the Parzen-window density estimate using a circularly symmetric Gaussian window of co-variance $\sigma^2\mathbf{I}$, where **I** is the d ×d identity matrix. Each output essentially estimates the posterior probability!





Naïve Bayes Classifier

- As mentioned 23,847,232,347 times before, the best classifier we can build is the Bayes classifier, which picks the class that has the largest posterior probability, computed from the product of the likelihood and the prior probability.
- Also as mentioned 23,847,232,347 times before, even a modest increase in the dimensionality increases the complexity of the problem exponentially (curse of dimensionality), which is a major pain in the ***!
- A highly practical solution to this problem is to assume ***class-conditional independence of the features*** which yields the so-called ***Naïve Bayes classifier.***

$$p(\mathbf{x}\,|\,\omega_i)=\prod_{j=1}^{d} p\left(x^{(j)}\,|\,\omega_i\right)$$

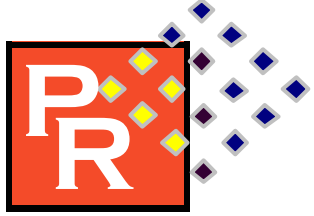
  - Note that this is not as restrictive of a requirement as full feature independence

$$p(\mathbf{x})=\prod_{j=1}^{d} p\left(x^{(j)}\right)$$

  - The discriminant function corresponding to the Naïve Bayes classifier is then

$$g_i^{NB}(\mathbf{x})=p(\omega_i)\prod_{j=1}^{d} p\left(x^{(j)}\,|\,\omega_i\right)$$

the main advantage of which is that we only need to compute univariate densities $p(x(d)\,|\,\omega i)$, which are much easier to compute then the multivariate densities $p(\mathbf{x}\,|\,\omega i)$. In practice, Naïve Bayes has shown to have respectable performance, comparable to that of neural networks.



Homework 2
Due October 16

Problems for Chapter 3:

- Write a program that computes principal components. You may use Matlab's basic matrix operation commands, such as `cov(.)`, `eig(.)`. Test your algorithm on a sample data that you generate. Compare your results to that obtained by Matlab's `prepca(.)` function.
- Computer Exercises 1, 9 and 10.

Problems for Chapter 4

- Problem 4
- Implement the Parzen window density estimation using the Gaussian window function in 1 and 2 dimensions. Take $V_n = h_n = 1/\sqrt{n}$ and test it on a number of distributions. Generate random numbers from different distributions using the data generation commands in the statistics toolbox. Then modify your algorithm for 2-dimensions (modify V_n accordingly).
- Implement Algorithms 1 and 2 in your text book for PNN. Make up several databases of 1 2 and 3 dimension from known non-symmetric Gaussian distributions. Classify these data using the discriminant based method as well as the PNN. Compare your results.
- Classify the data you generated for the above problem using Naïve Bayes. Interpret / compare your results.
- Use the program you wrote for Parzen windows for Computer Exercise 2 of Chapter 4.

Important note: Since you have the choice of creating your databases, make sure that you fully document your work with abundant number of plots showing the data, the decision boundaries generated / classification etc. You must show all work for full credit!