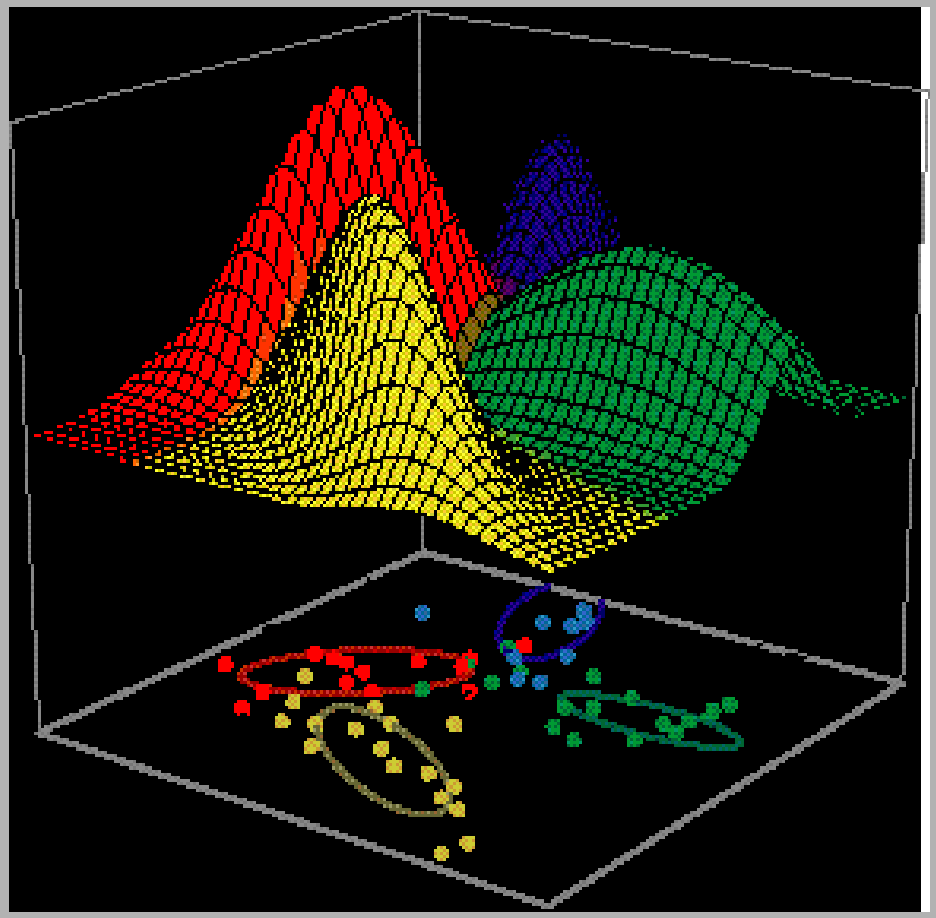


Pattern Recognition

(Αναγνώριση Προτύπων)

Nonparametric Techniques

(Μη-Παραμετρικές Τεχνικές)

Panos Trahanias

UNIVERSITY OF CRETE

DEPARTMENT of COMPUTER SCIENCE

Non-Parametric Techniques

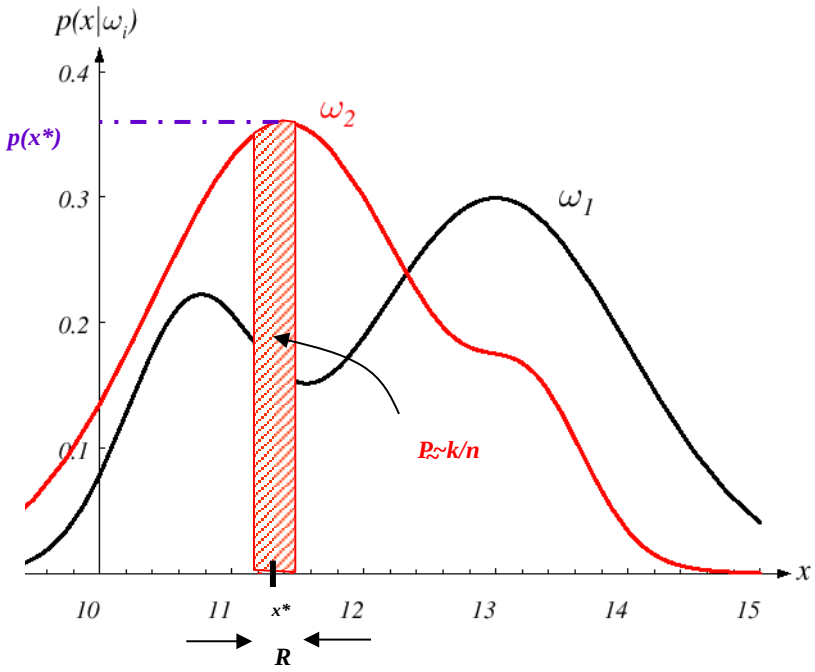
- Issues in Parametric Techniques:
  - Often the shape of the distribution is not known.
  - In practice, many distributions are multimodal (contain more than one modes), whereas most parametric models are unimodal.
  - Approximation of multimodal distributions as a product of unimodal ones is not appropriate.
- Non-parametric techniques: Estimation of distribution function from scratch.
  - Estimation of pdf's $p(x/\omega_i)$ from the data via generalization of the *multi-dimensional histogram*.
  - Direct Estimation of the posterior probabilities $P(\omega_i /x)$ and the discriminant functions.

Density (Distribution) Estimation

- Based on the fact that the probability of sample $\boldsymbol{x}$ is found within area R results from

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

- The above integral can be approximated either by the product of p(**x**) times the area R, or by the number of samples that are found within R



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

V: Area R. In one-dimensional case, V= length of R

k: Number of samples within R

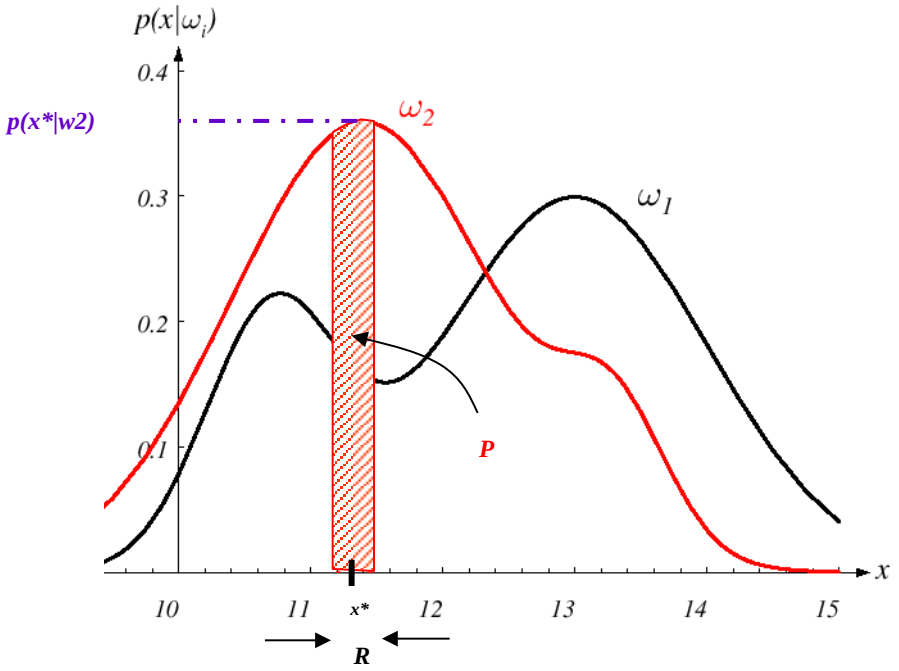
n: Total number of samples

Density (Distribution) Estimation

In order to estimate the density at ***x***, we choose a series of areas *R1, R2,…,Rn,*

that contain ***x***, whereby *Ri* is used for *i* samples. Let *Vn* the volume of *Rn*, *kn* the number of samples within the n-th area, and *pn(**x**)* the n-th estimation of *p(**x**)*. Then,

$$p_n(\mathbf{x}) = \frac{k_n/n}{V_n} \xrightarrow[n\to\infty]{} p(\mathbf{x})$$



For *k/n* to be a good estimate of *P*,

and hence *pn(x)* a good estimate of *p(x)*,

the following should hold:

$$\lim_{n\to\infty} V_n = 0$$

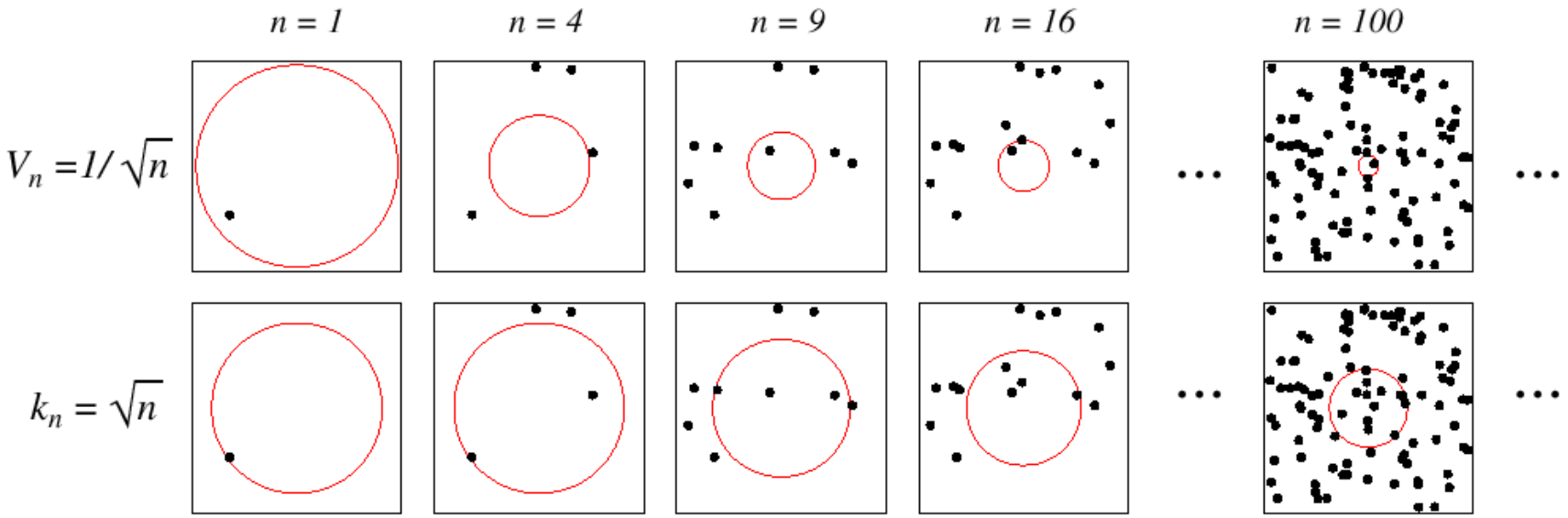
$$\lim_{n\to\infty} k_n = \infty$$

$$\lim_{n\to\infty} k_n/n = 0$$

Density (Distribution) Estimation

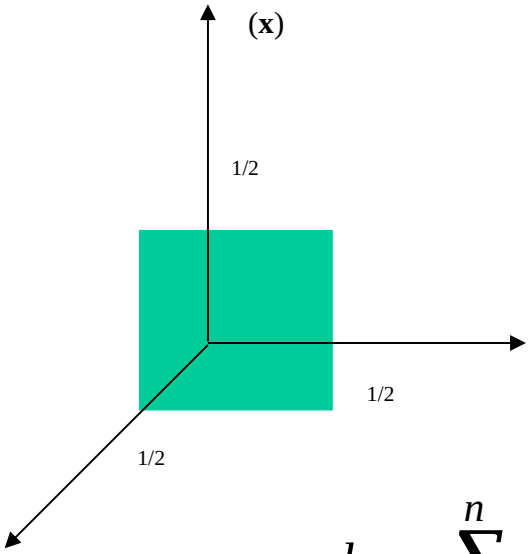
- There are two ways to create series of areas R_i for $p_n(\boldsymbol{x})$ to converge to $p(\boldsymbol{x})$:
  - The volume of an initial area is reduced by defining a sequence of volumes V_n as functions of n, e.g. $V_n = V_1 / \sqrt{n}$
    - Density estimation via the Parzen Windows method
  - We define k_n as a function of n, $k_n = \sqrt{n}$ hence V_n increases until it contains k_n samples. e.g.
    - Density estimation via the k_n-Nearest Neighbor method

Two Approaches



Parzen Windows

- Method based on finding the number of samples within a given area, where the area shrinks as the number of samples grows larger.
- The number of samples within the area is computed via the use of a "window function", the Parzen Window.



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

Volume of function, Vn=(hn)^d

Window function

Area/width of function

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

No of samples within Rn,

Rn with center $\mathbf{x}$ *and width hn*

Parzen Windows

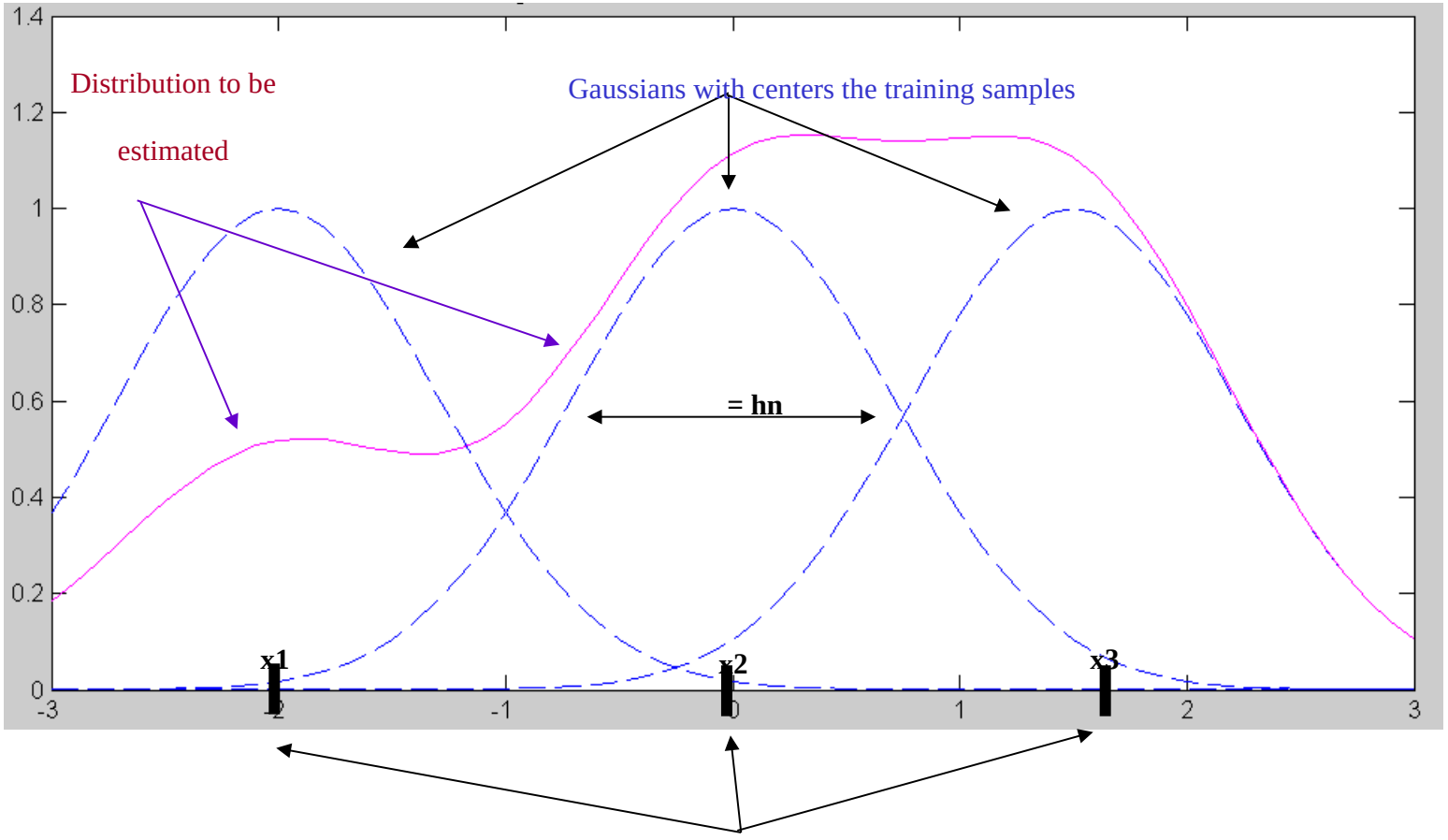
$$p_n(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{V_n}\varphi\left(\frac{\mathbf{x}-\mathbf{x}_i}{h_n}\right) = \frac{1}{n}\sum_{i=1}^{n}\delta_n(\mathbf{x}-\mathbf{x}_i) \quad \text{όπου} \quad \delta_n(\mathbf{x}) = \frac{1}{V_n}\varphi\left(\frac{\mathbf{x}}{h_n}\right)$$

- Window *(.)* can be a general functional, not necessarily hypercube. For $p_n(x)$ to be a valid p.d.f. for every n, it should hold

$$\varphi(\mathbf{u}) \geq 0 \quad \text{και} \quad \int_{\mathbf{u}} \varphi(\mathbf{u})d\mathbf{u} = 1$$

- $p_n(x)$ is a linear combination of *(.)*, where every sample $\boldsymbol{x}_i$ contributes to the estimation of $p(\boldsymbol{x})$ according to its distance from $\boldsymbol{x}$. If *(.)* is a valid p.d.f., then $p_n(x)$ will converge to $p(x)$ as *n increases*. A typical choice for *(.)* is – Wow, you've made a great guess– the Gaussian Function!
- $p(x)$ is computed as a superposition of Gaussians, where each Gaussian is centered in the corresponding training sample. Parameter h_n is the Gaussian standard deviation!!!

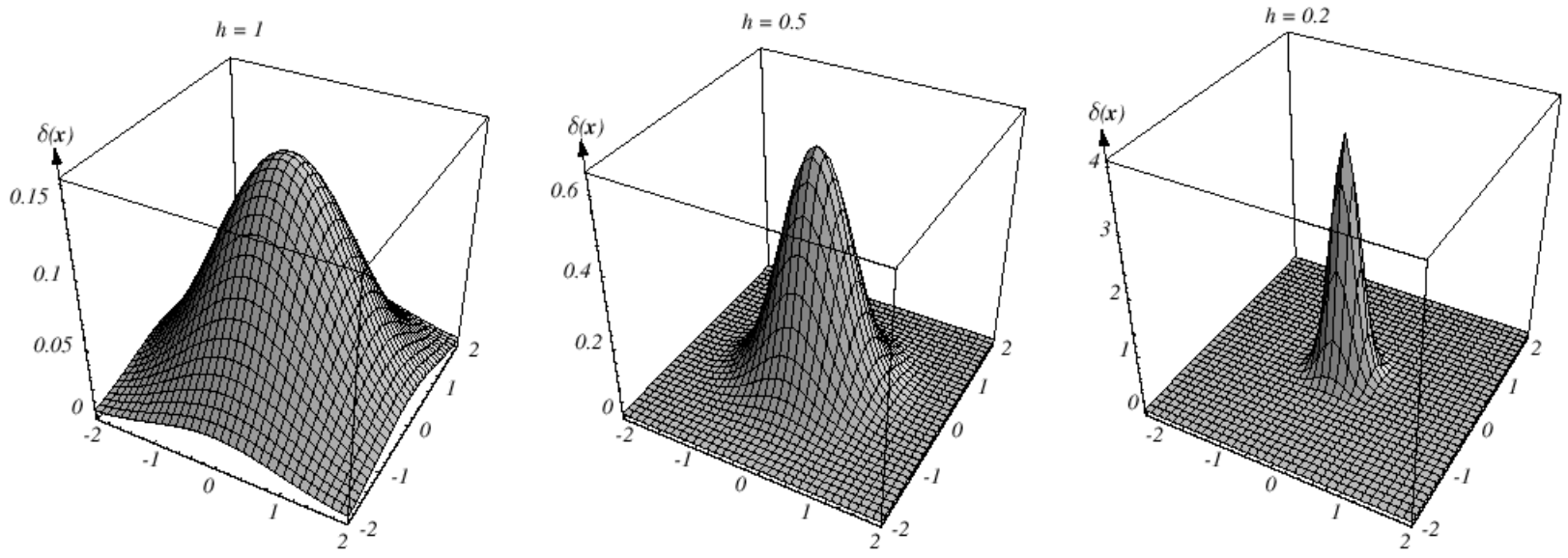
Parzen Windows



Training Samples

Effect of Window Width h_n

$$\delta_n(\mathbf{x}) = \frac{1}{V_n}\varphi\left(\frac{\mathbf{x}}{h_n}\right), \quad V_n = h_n^d$$

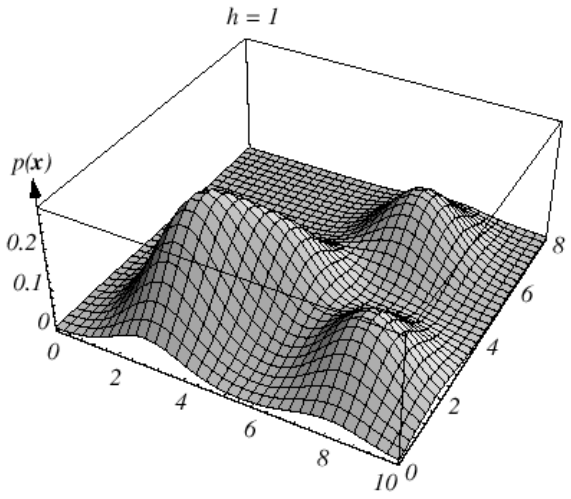


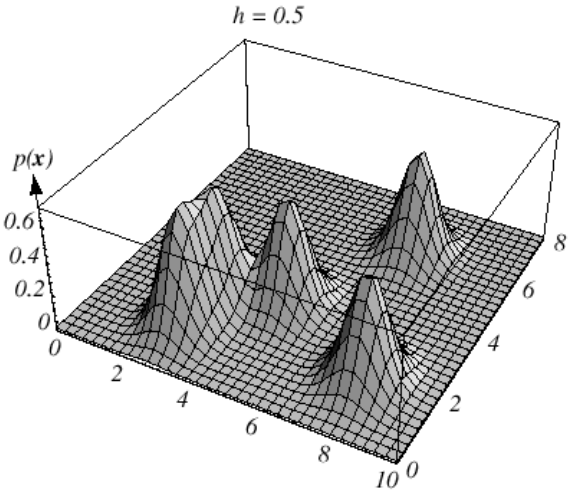
Parameter *hn* has an effect on both the window width and its amplitude:

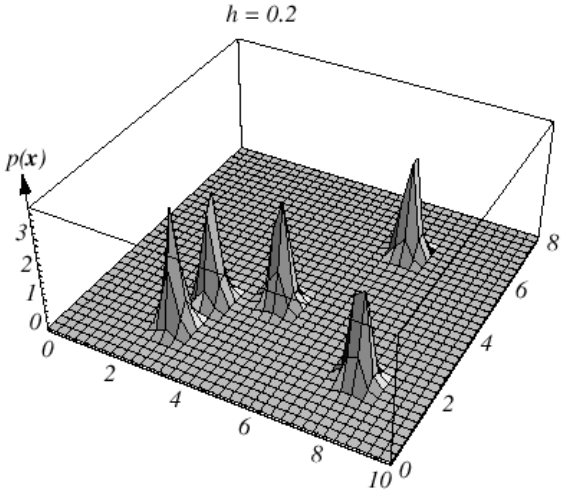
- **When *hn* is large (small), window becomes wide (narrow), window amplitude is small (large) and *x* must be far from (close to) *xi* for function *δn(x-xi)* to change drastically compared to its value at *δn(0)*.**

Effect of Window Width h_n

Hence







How the window width affects the estimation of p.d.f. *p(x)* :

- ✓ **If *hn* is large, estimate *pn(x)* results as superposition of *n* "wide" functions centered at the training samples and constitutes a smooth, "out-of-focus" estimation of *p(x)*, with low resolution.**
- ✓ **If *hn* is small, *pn(x)* results as superposition of *n* "narrow" functions, i.e. "erratic or noisy" estimation of *p(x).***

Convergence Properties of $p_n(\mathbf{x})$

- Mean value of random variable $p_n(x)$:

$$\bar{p}_n(\mathbf{x}) = E\left[p_n(\mathbf{x})\right] = \int \delta_n(\mathbf{x}-\mathbf{v})p(\mathbf{v})d\mathbf{v}$$

is the convolution of $p(x)$ with the window function, hence it is a blurred version of $p(x)$.

- It holds,

$$\delta_n(\mathbf{x}-\mathbf{v}) \xrightarrow[V_n \to 0]{} \delta(\mathbf{x}-\mathbf{v}) \quad \text{Hence} \quad \bar{p}_n(\mathbf{x}) \xrightarrow[n\to\infty]{} p(\mathbf{x})$$

- Variance of random variable $p_n(x)$:

$$\mathrm{var}\left(p_n(\mathbf{x})\right) = \frac{1}{nV_n}\int \frac{1}{V_n}\varphi^2\left(\frac{\mathbf{x}-\mathbf{v}}{h_n}\right)p(\mathbf{v})d\mathbf{v} - \frac{1}{n}\bar{p}_n^2(\mathbf{x}) \le \frac{\sup(\varphi(\cdot))\bar{p}_n(\mathbf{x})}{nV_n}$$

- Accordingly, we have small variance for large V_n! In the limit $n \to \infty$, we may have V_n approach 0 and variance will also approach 0, given that $nV_n \to \infty$.
- Valid choices:

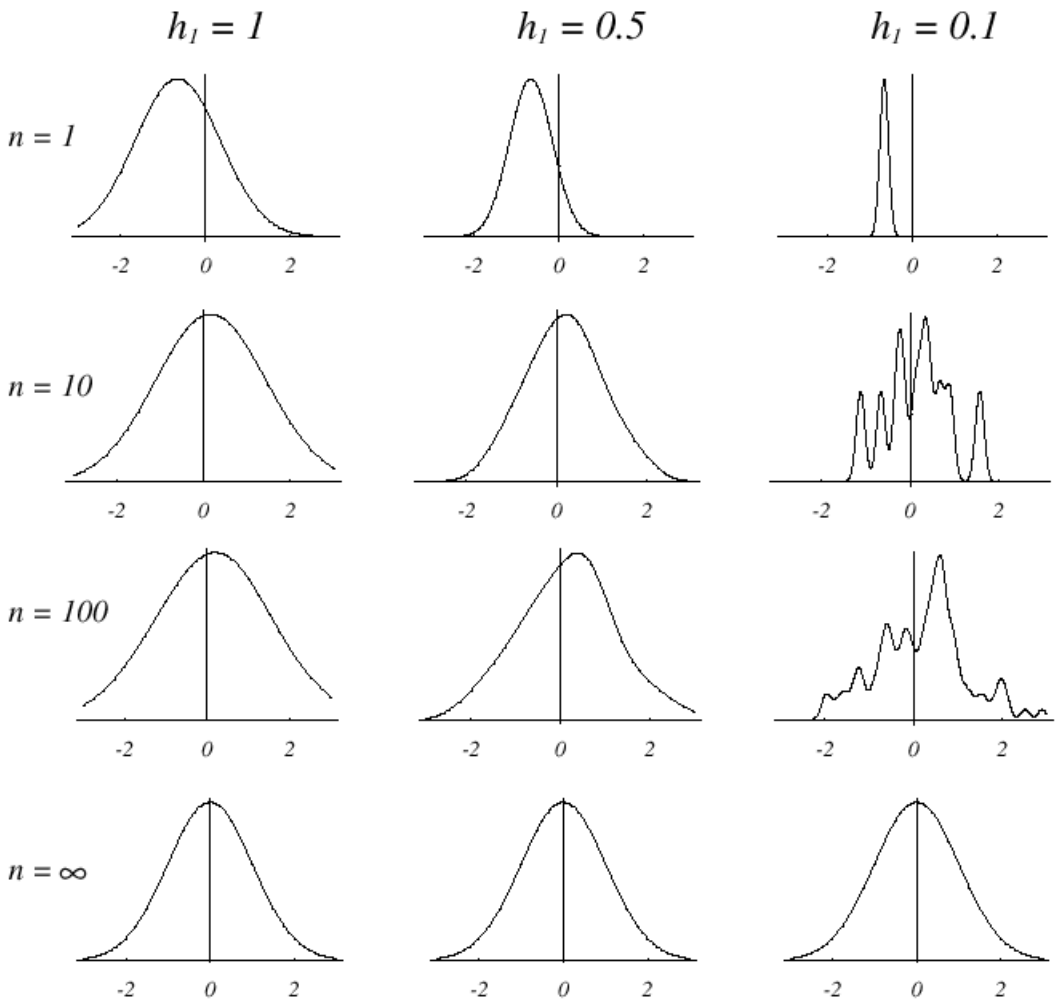
$$V_n = V_1/\sqrt{n} \quad \text{ή} \quad V_n = V_1/\ln n$$

Parzen Windows Examples

$$p_n(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h_n}\varphi\left(\frac{x - x_i}{h_n}\right);$$

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

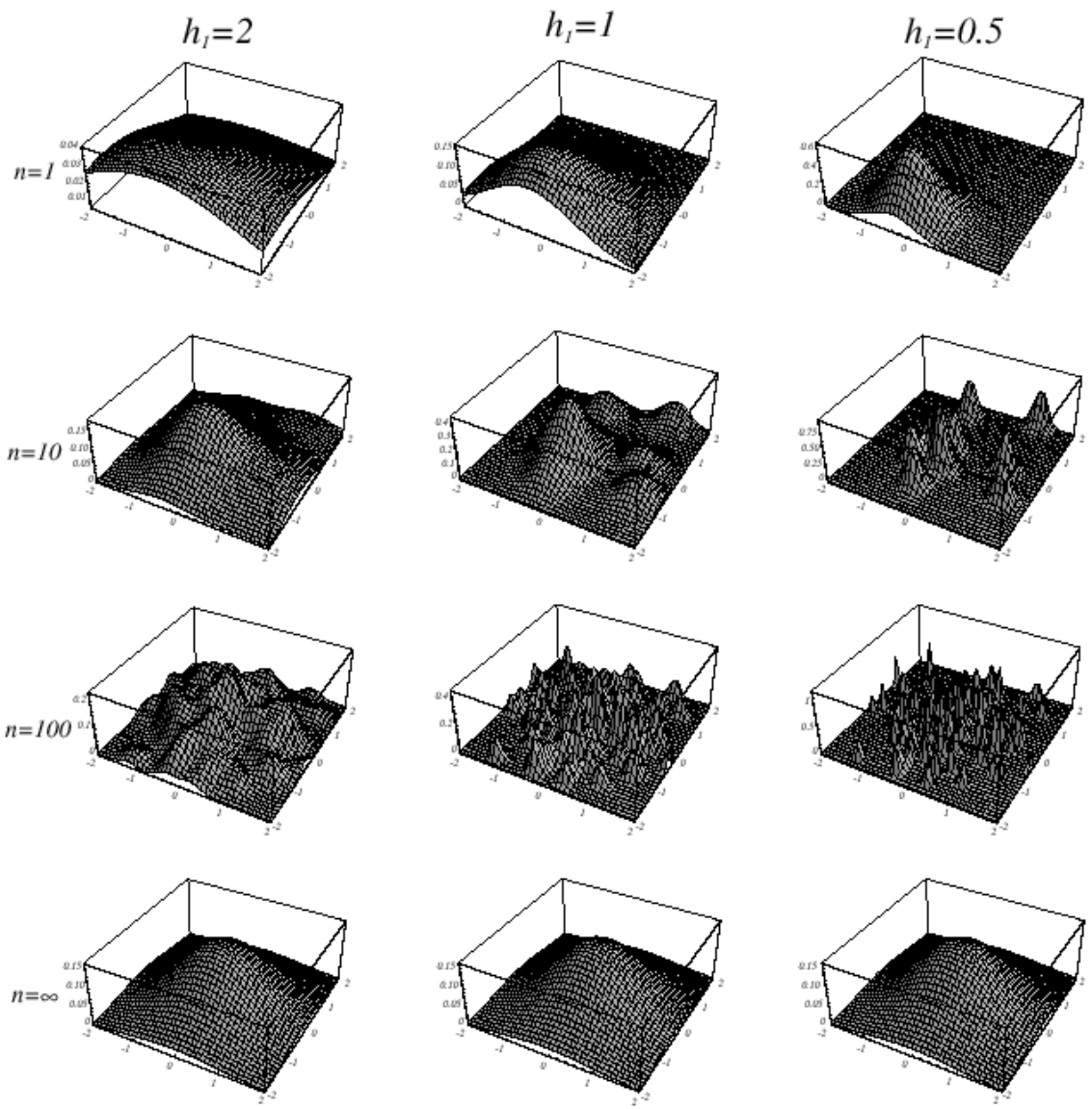
As n approaches infinity, estimation becomes accurate, regardless of the window width.



Parzen Windows Examples

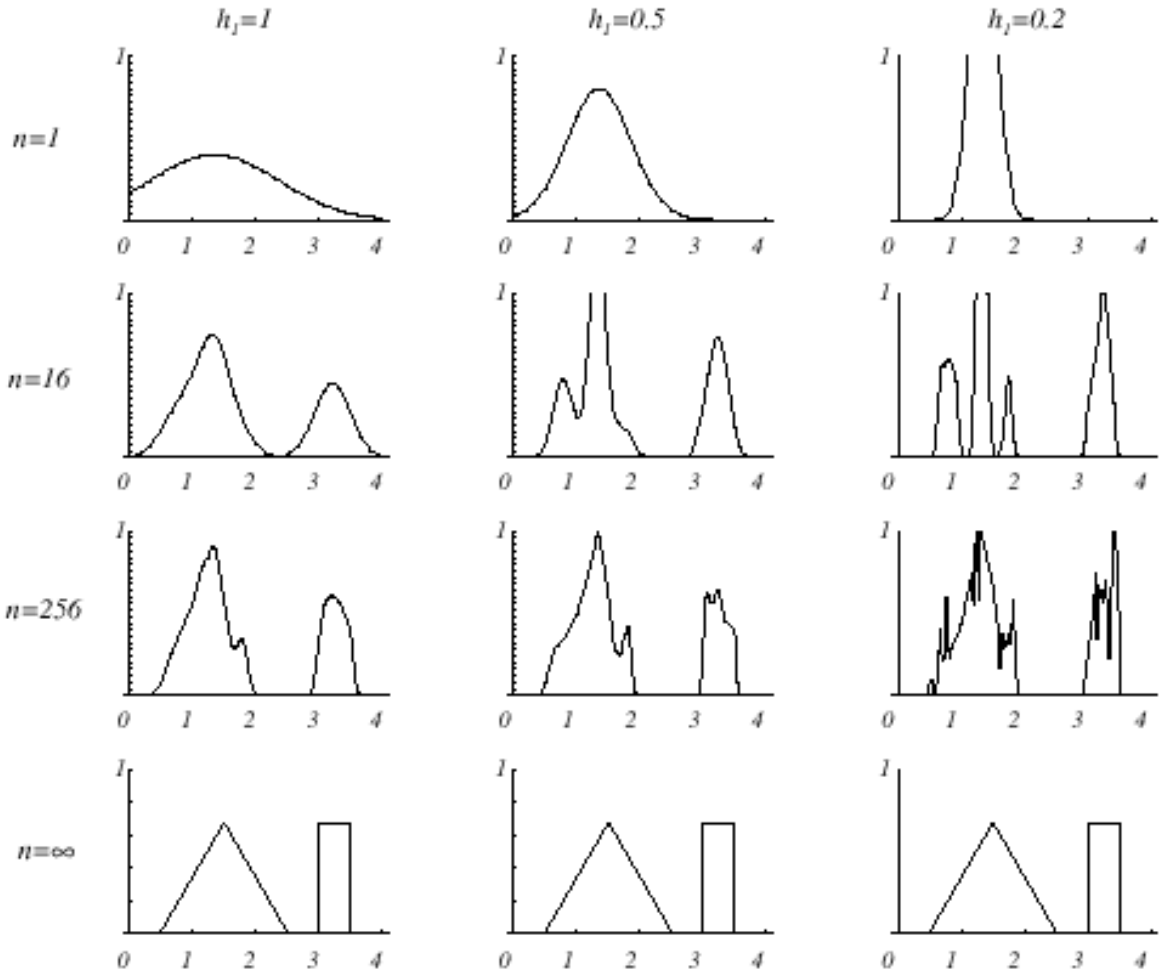
$$p_n(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h_n^2}\varphi\left(\frac{\mathbf{x}-\mathbf{x}_i}{h_n}\right);$$

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



Parzen Windows Examples

As n approaches infinity, estimation becomes accurate, regardless of the window width.



Parzen Windows for Pattern Classification

- Estimation of likelihood $p(\boldsymbol{x}|\omega_i)$ from training data via the Parzen windows approach, and use of Bayes rule for classification, e.g. computation of posterior probabilities and classification based on largest posterior probability.
- Pros: No need for problem specific assumptions, only the existence of training data set!
- Cons: Requires $(many)^d$ data so that estimate converges to actual distribution.
  - Moreover, as the dimension increases, the requirement for $(many)^d$ data becomes $((many)^{many\ (n)})^n$!!!! ⇒ Curse of dimensionality !
  - The only way to overcome the above is the prior knowledge about "good" training data!
- The training error can become arbitrarily small (even zero), by choosing small windows! Still this is not desirable, since it'll result in overfitting and it'll reduce performance in the classification of test data.

Parzen Windows for Pattern Classification

Very small window ⇒

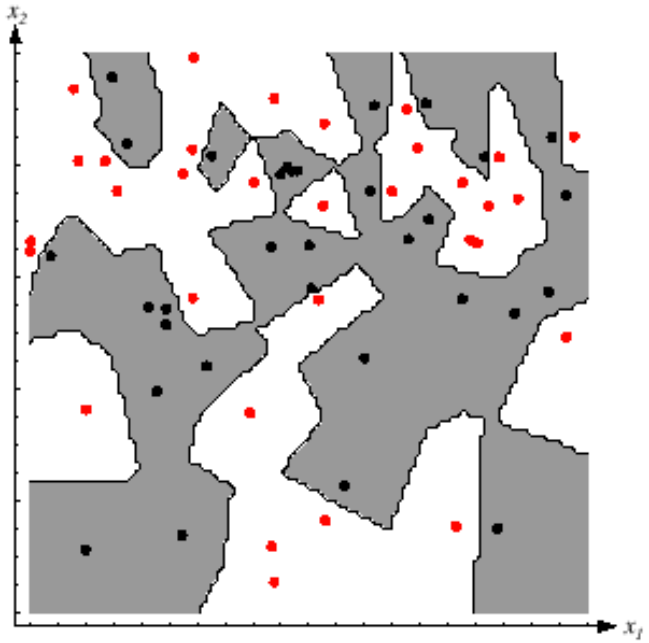
very small/detailed separation in the

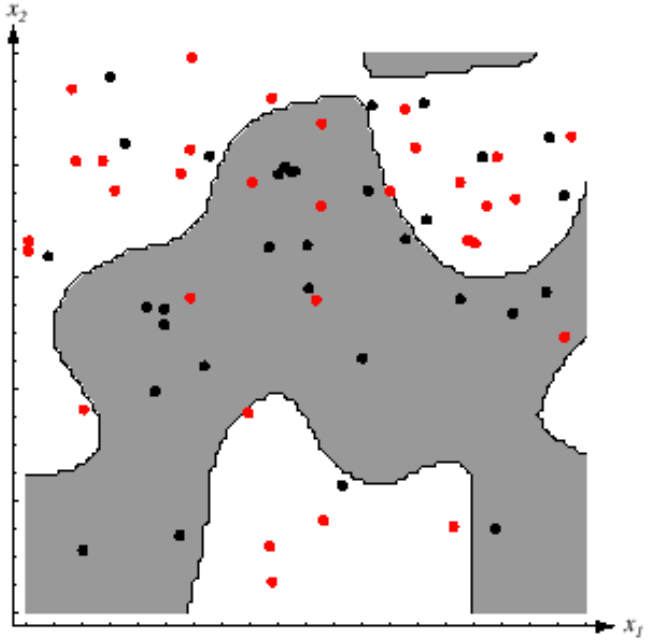
feature space, not desirable!

Larger window ⇒ Larger training error, but better generalization

performance!

Desirable property.





In practice we're looking at small windows in areas with many (dense) training samples, and large windows in areas with little (sparse) training samples! How can this be accomplished…?

Probabilistic Neural Networks (PNN)

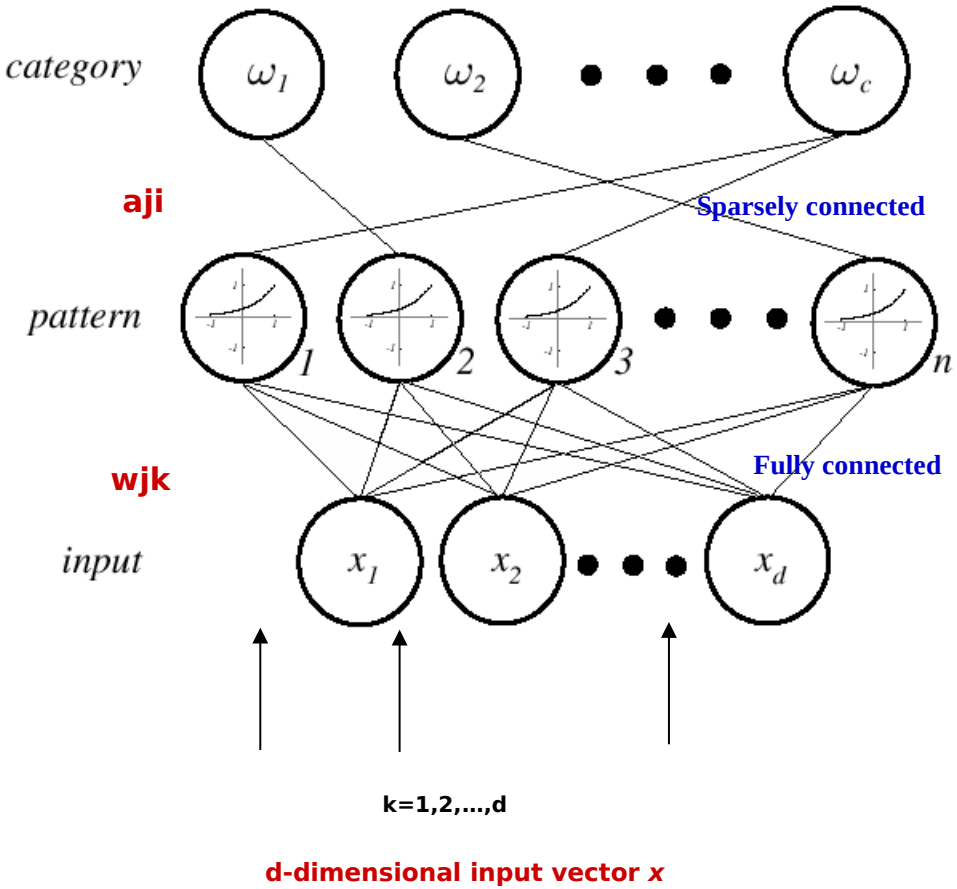
- Input: $\{xk; k=1,\ldots,d\}$ d nodes, each corresponds to a feature.

- wjk: weights that link k–th input node with j–th node of hidden layer (pattern node).

- Hidden layer: n nodes, each corresponds to a pattern, i.e. training sample, $j=1,2,\ldots,n$.

- Output layer: c nodes, each represents a class.

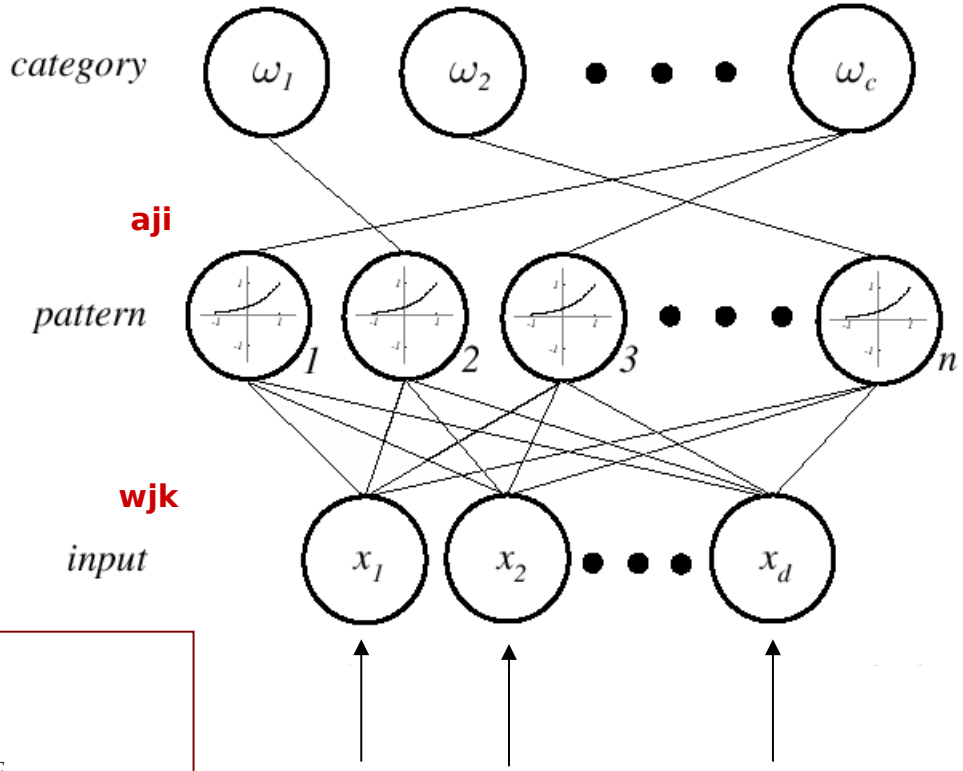
- aji: weights that link j–th hidden node with i–th output node, $i=1,2,\ldots,c$



PNN-Training

Training

- *j-th* training sample (pattern) is normalized to unit length.
- Given as input (input nodes).
- Weights wjk set as wjk=xjk.
- A unique link with weight aji=1 is established from the *j-th* hidden node to the output node that corresponds to the (known) class of **x**j.



k=1,2,…,d

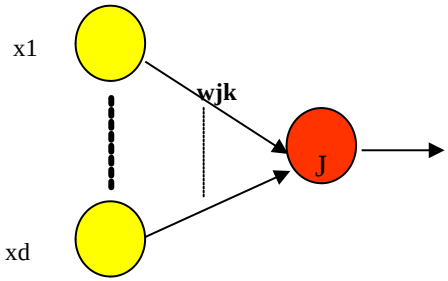
d-dimensional input vector *x*

Algorithm 1 (PNN training)

1 **begin initialize** $j = 0$ *j=0, aji=0* for *j=1,…,n; i=1,…,c*
2 **do** $j \leftarrow j + 1$
3 normalize : $x_{jk} \leftarrow x_{jk} / \left(\sum_{i}^{d} x_{ji}^2 \right)^{1/2}$
4 train : $w_{jk} \leftarrow x_{jk}$
5 **if** $\mathbf{x} \in \omega_i$ **then** $a_{aji} \leftarrow 1$
6 **until** $j = n$

PNN-Classification

- Each pattern node gives rise to the inner product of the vector of its weights with the normalized input $\boldsymbol{x}$ to compute $net_J = \mathbf{w}^t\mathbf{x}$, and output $e^{[(netJ\ -1)\ /\sigma 2]}$.



$$e^{\left((net_J - 1)/\sigma^2\right)}$$

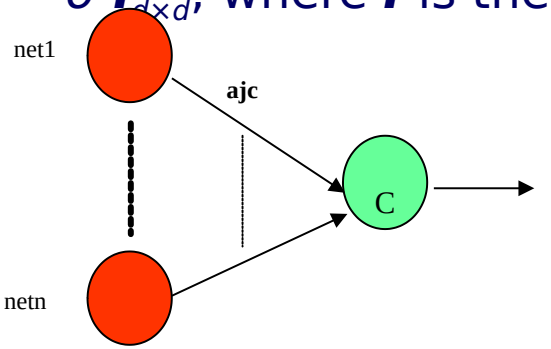
Width of Gaussian Parzen window

$$\varphi\left(\frac{\mathbf{x} - \mathbf{w}_k}{h_n}\right) \propto \overbrace{e^{-(\mathbf{x}-\mathbf{w}_k)^t(\mathbf{x}-\mathbf{w}_k)/2\sigma^2}}^{\text{desired Gaussian}}$$

$$= e^{-(\mathbf{x}^t\mathbf{x}+\mathbf{w}_k^t\mathbf{w}_k-2\mathbf{x}^t\mathbf{w}_k)/2\sigma^2} = \underbrace{e^{(z_k-1)/\sigma^2}}_{\text{Activation Function}}$$

function

- Each class node sums the results of the pattern nodes that are connected to it. Accordingly, the activation of each class represents the p.d.f. estimate with a Gaussian Parzen window with covariance matrix $\sigma^2\boldsymbol{I}_{d\times d}$, where $\boldsymbol{I}$ is the ide



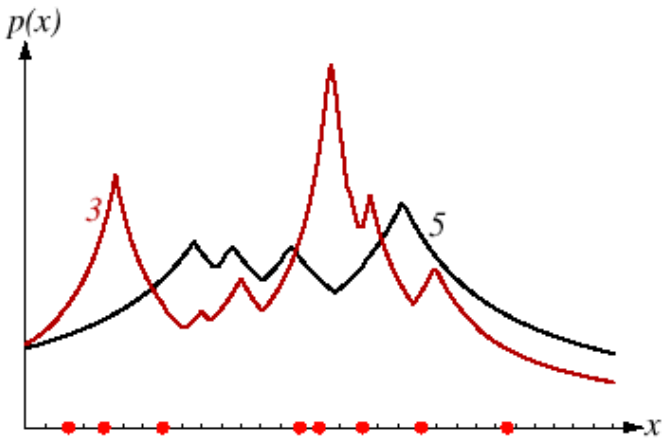
Algorithm 2 (PNN classification)

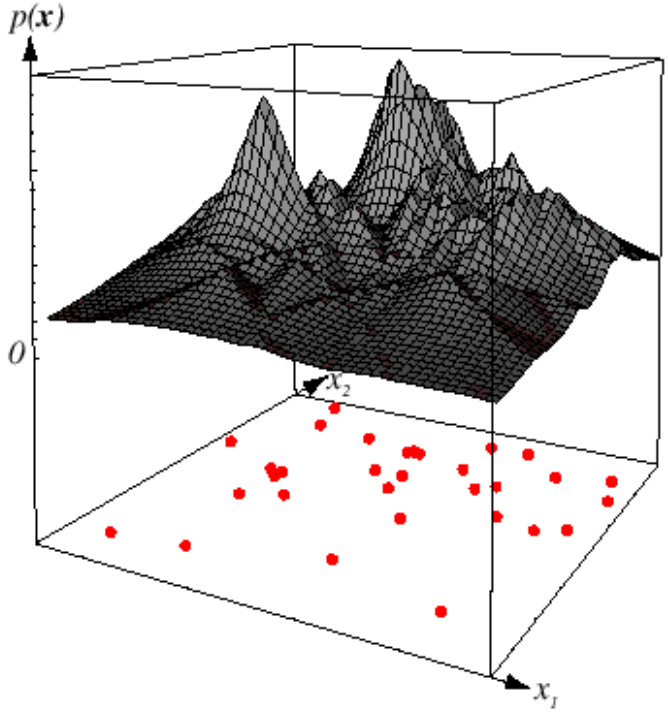
1 **begin initialize** $k = 0, \mathbf{x} =$ test pattern
2 **do** $k \leftarrow k + 1$
3 $z_k \leftarrow \mathbf{w}_k^t\mathbf{x}$
4 **if** $a_{ki} = 1$ **then** $g_i \leftarrow g_i + \exp[(z_k - 1)/\sigma^2]$
5 **until** $k = n$
6 **return** $class \leftarrow \arg\max_i g_i(\mathbf{x})$
7 **end**

k_n-Nearest Neighbor (KNN)

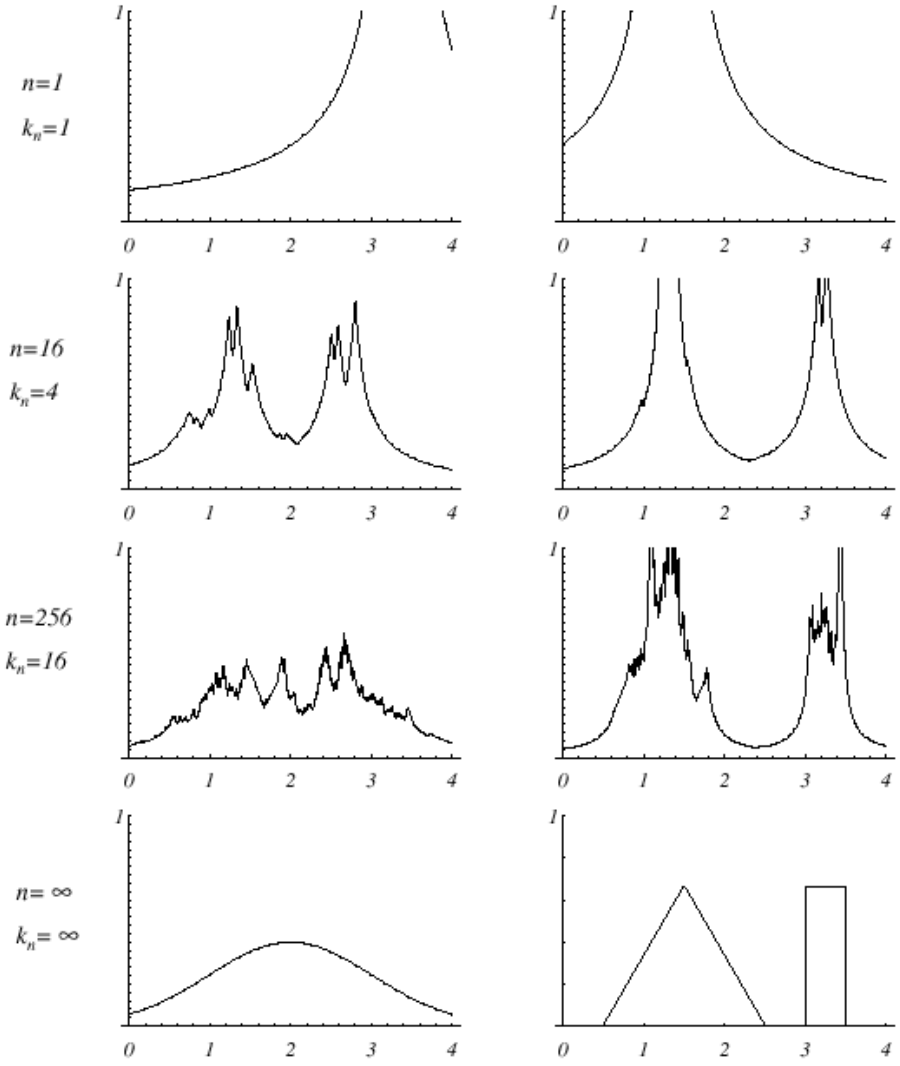
- Window width: instead of choosing it as a function of the number of training samples ($V_n = V_1/\sqrt{n}$), why not choosing it as a function of the training data?
- Note: a large window is desirable in areas with small number of data, whereas a small window is appropriate in dense (with data) areas!
- k-nearest neighbor estimation algorithm:
  - Choose an initial area centered at $\boldsymbol{x}$ where $p(\boldsymbol{x})$ is estimated.
  - Increase window until a predefined number of k_n samples falls inside the window. Those are the k_n **nearest neighbors** of $\boldsymbol{x}$.
  - Pdf (density) is estimated as $\frac{k_n/n}{V_n}$
  - Convergence conditions of $p_n(x)$: $\lim_{n\to\infty} k_n = \infty; \quad \lim_{n\to\infty} k_n/n = 0$

If $k_n = \sqrt{n}$ and assuming that *pn(**x**)* is a good estimate of *p(**x**)*, then $V_n \approx 1/\left(\sqrt{n}\, p(\mathbf{x})\right)$. Accordingly *Vn* is once more $V_1/\sqrt{n}$ but this time the initial area *V1* is identified from the pdf *p(**x**)* of the training data and is not an arbitrary choice. Moreover, for each *n*, the size of area *Vn* is a function of ***x***, i.e. *Vn= Vn(**x**)*.





How to Choose k_n



Note that as *kn* increases, estimation accuracy increases…!

In classification problems, we usually tune *kn* (or *hn* for Parzen windows), so that the classifier operates with the lowest error for the validation test dataset.

KNN Classification

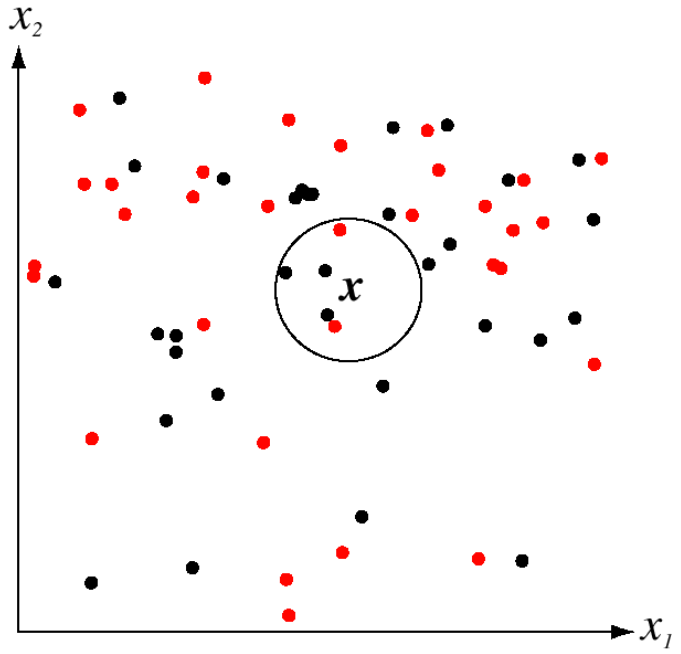
- KNN can be employed to estimate the posterior probabilities: Actually, posterior probabilities in an area around ***x*** are calculated as the ratio of samples within the area with class label ω_i.
  - n: total number of patterns from all classes
  - k_i: number of patterns from class i in the area around ***x***
  - k: total number of patterns from all classes in the area around ***x***

$$p_n(\mathbf{x}, \omega_i) = \frac{k_i / n}{V}$$

$$P_n(\omega_i \mid \mathbf{x}) = \frac{p_n(\mathbf{x}, \omega_i)}{\sum_{j=1}^{c} p_n(\mathbf{x}, \omega_j)} = \frac{k_i}{k}$$

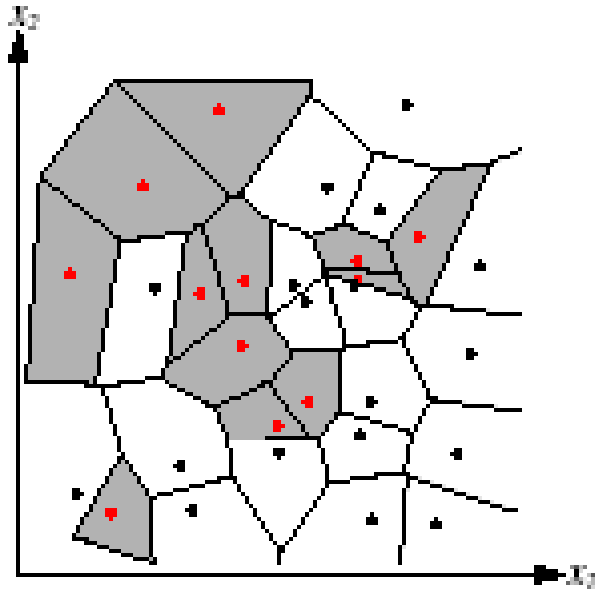
KNN Classification

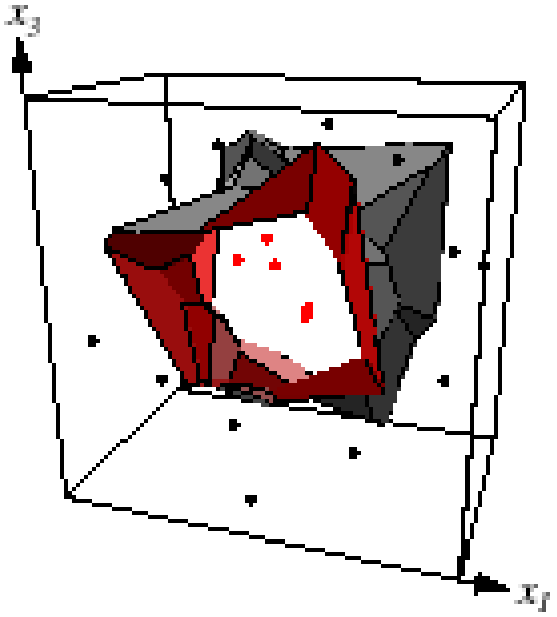
- Classification of a pattern $\boldsymbol{x}$,
  - From the n training samples, the k nearest neighbors are identified regardless of their class, (where k is odd for a two-class problem).
  - Let k_i the number of samples from class i, $\sum_i k_i = k$
  - Classify $\boldsymbol{x}$ to the class with the largest k_i !



Nearest Neighbor Classifier

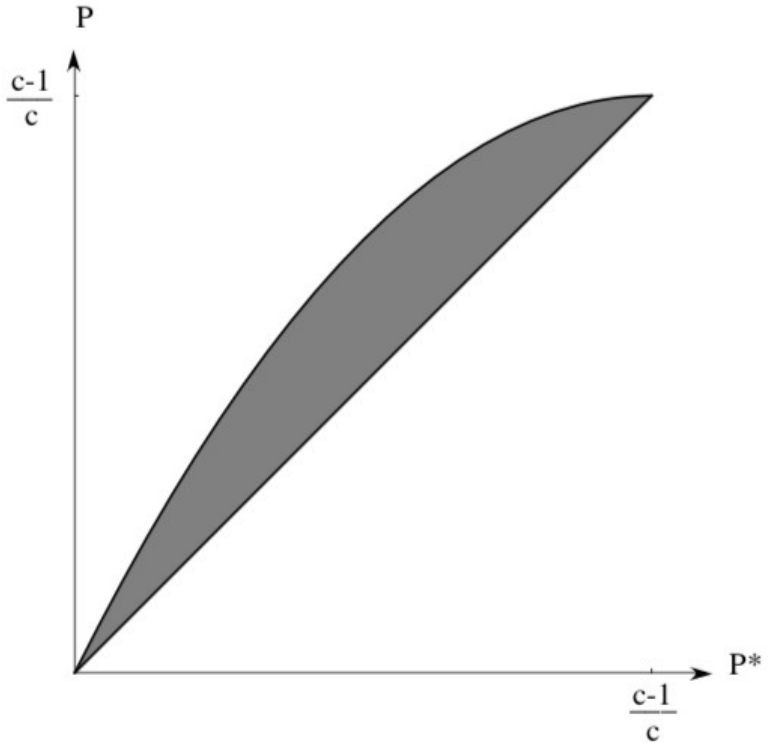
- The simplest KNN classifier is for *k=1*! In this case ***x*** assumes the class of its **nearest neighbor**! Is this a good classifier…?



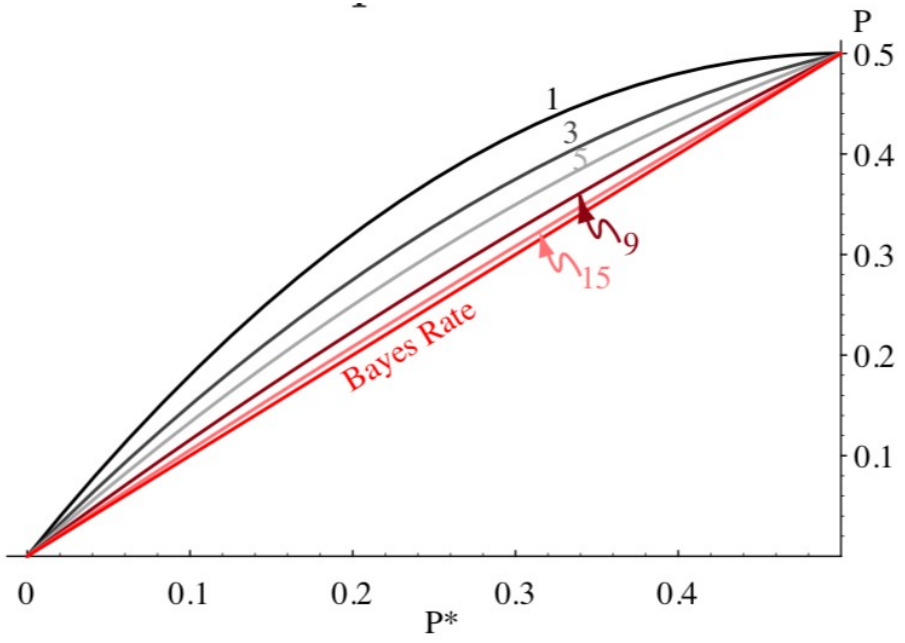


- **NN classifier partitions the feature space as a Voronoi tessellation, where each cell assumes the label of the class within it.**
- **Given infinite number of training samples, the classification error probability is bounded by twice the error of the Bayesian classifier (valid for small error probabilities).**

Error Rates



NN Classifier



K-NN Classifier

Computational Issues

- Partial distances
  - Use a subset r of d dimensions, to compute the distance of the unknown pattern from the training samples:

$$D_r(\mathbf{a},\mathbf{b}) = \left(\sum_{k=1}^{r} (a_k - b_k)^2 \right)^{1/2}$$

- Search trees
  - Establish “search trees” where training samples are selectively linked so that, for the classification of a new pattern, distance computation is required only to some **entry/root** patterns and the patterns linked to them.
- Editing, pruning, or condensing
  - Prune patterns that are surrounded from patterns of the same class.
    1. Compute the Voronoi diagram of the training patterns
    2. For each pattern, if any of its neighbors is not of the same class, mark it.
    3. Prune non-marked patterns and compute the revised Voronoi diagram

Metrics and KNN Classification

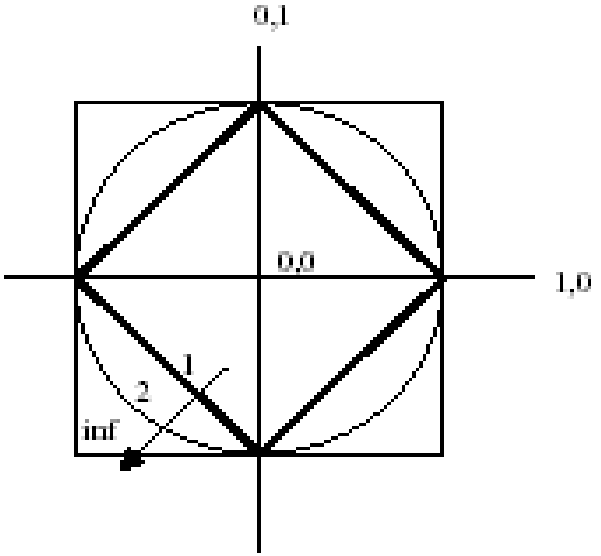
- Properties:
  - Nonegativity: $D(\mathbf{a},\mathbf{b}) \geq 0$
  - reflexivity: $D(\mathbf{a},\mathbf{b}) = 0 \quad \text{iff} \quad \mathbf{a} = \mathbf{b}$
  - symmetry: $D(\mathbf{a},\mathbf{b}) = D(\mathbf{b},\mathbf{a})$
  - triangle inequality: $D(\mathbf{a},\mathbf{b}) + D(\mathbf{b},\mathbf{c}) \geq D(\mathbf{a},\mathbf{c})$

- Euclidean Distance: $L_2(\mathbf{a},\mathbf{b}) = \left(\sum_{k=1}^{d} (a_k - b_k)^2 \right)^{1/2}$

- Minkowski Metric: $L_p(\mathbf{a},\mathbf{b}) = \left(\sum_{k=1}^{d} |a_k - b_k|^p \right)^{1/p}$

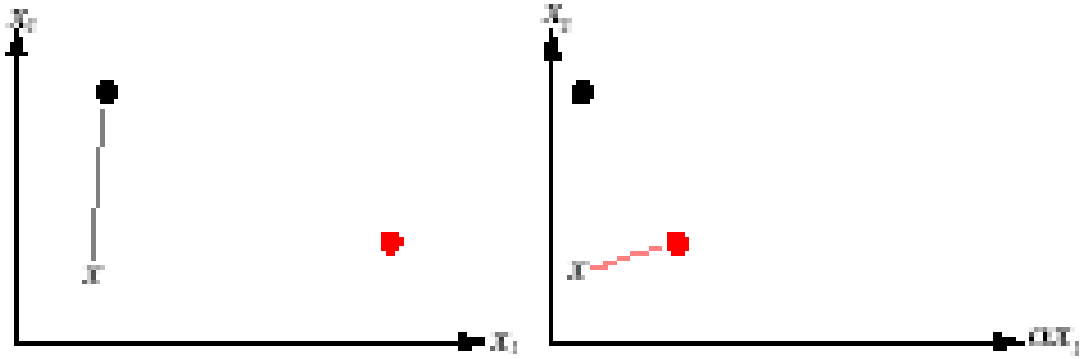
- Manhatan Distance: $L_1(\mathbf{a},\mathbf{b}) = \sum_{k=1}^{d} |a_k - b_k|$

- Chess-board Distance: $L_\infty(\mathbf{a},\mathbf{b}) = \max_{k=1,\ldots,d} \left(|a_k - b_k| \right)$

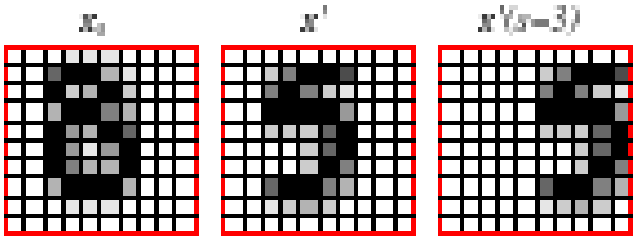


Distance 1 from the center via the use of each one of the metrics *Lp*.

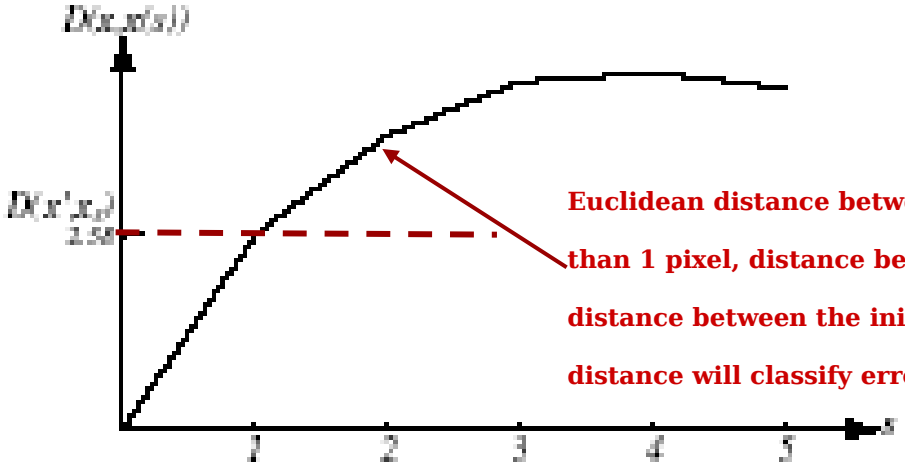
Metrics and KNN Classification



When the pattern space is transformed after multiplying each feature by a constant, distances in the transformed space may vary significantly from the initial ones. Evidently, this effects the performance of the KNN classifier.



It is of utmost importance to employ metrics that are invariant under basic transformations, such as translation (shift), scaling, rotation, line thickness, shear. Example case: optical character recognition – OCR.



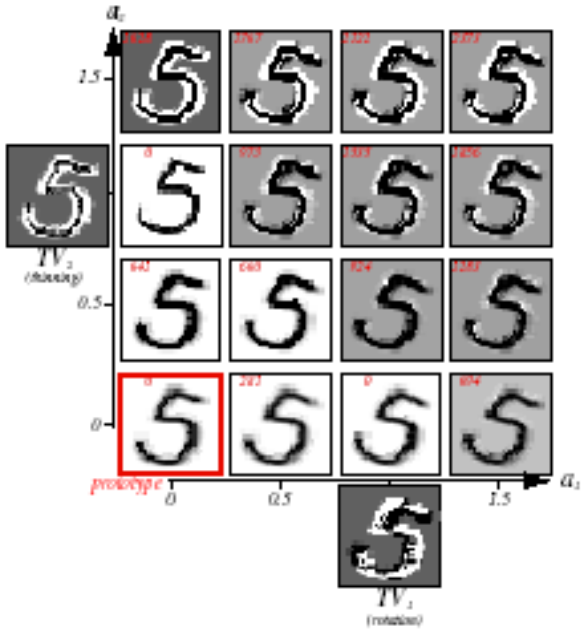
Euclidean distance between 5 and a translated 5 by *s* pixels. For translations larger than 1 pixel, distance between the initial 5 and translated 5 is larger than the distance between the initial 5 and 8, and hence NN classifier that employs Euclidean distance will classify erroneously.

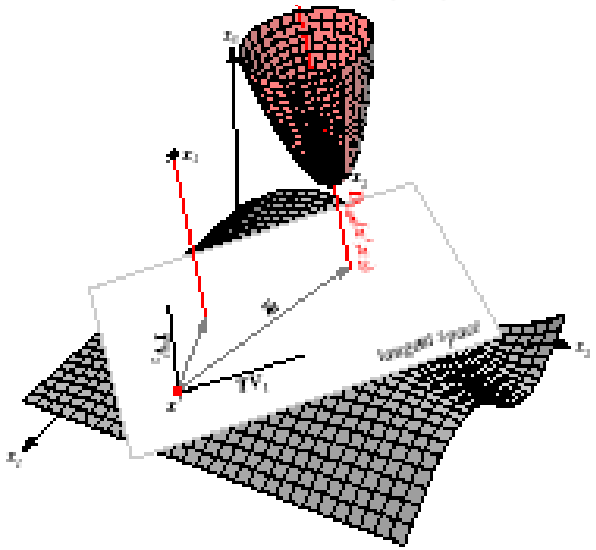
Tangent Distance

- Let r transformations, αi.
- Let $\boldsymbol{x}'$ a pattern.
- Transformed pattern, $Fi(\boldsymbol{x}';\alpha i)$.
- Tangent vector: $\boldsymbol{TV}i=Fi(\boldsymbol{x}';\alpha i)-\boldsymbol{x}'$.
- Tangent matrix: $\boldsymbol{T}dxr=[\boldsymbol{TV}1,\ldots,\boldsymbol{TV}r]$.
- Tangent space: space defined by the r linearly independent tangent vectors $\boldsymbol{TV}i$ that pass from $\boldsymbol{x}'$. Forms a linear approximation of the space of transformed $\boldsymbol{x}'$.
- Tangent distance:

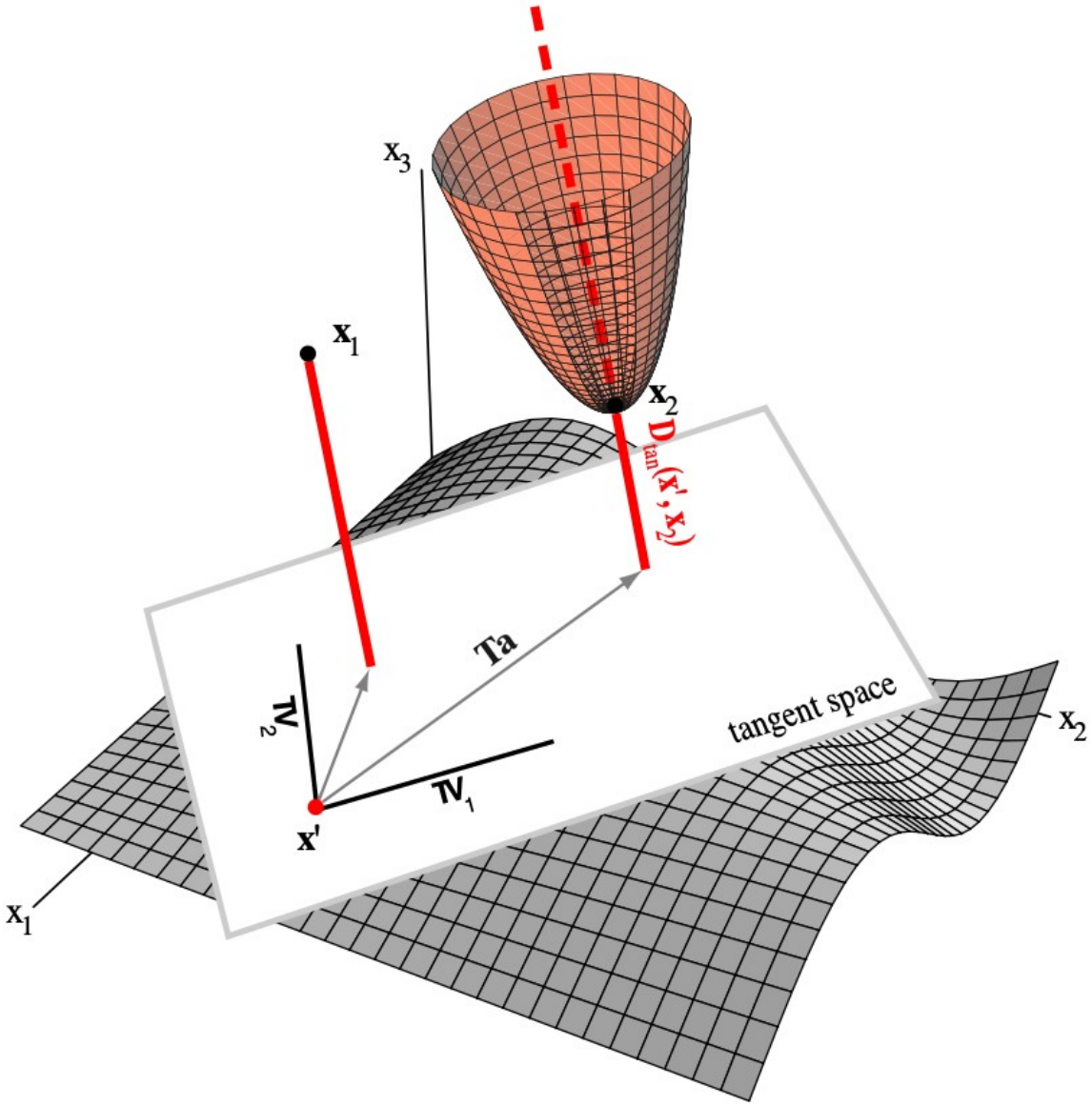
$$D_{tan}(\mathbf{x}',\mathbf{x}) \equiv \min_{\mathbf{a}} \left[\left\| \left(\mathbf{x}' + \mathbf{T}\mathbf{a} \right) - \mathbf{x} \right\| \right]$$

- Actually, it is Euclidean distance of $\boldsymbol{x}$ from the tangent space of $\boldsymbol{x}'$





Tangent Distance



Tangent Distance

We find the optimal $\mathbf{a}$ via iterative gradient descent. For gradient descent we need the derivative of the (squared) Euclidean distance. The Euclidean distance in Eq. 61 obeys

$$D^2(\mathbf{x}' + \mathbf{Ta}, \mathbf{x}) = \|(\mathbf{x}' + \mathbf{Ta}) - \mathbf{x}\|^2, \tag{62}$$

and we compute the gradient with respect to the vector of parameters $\mathbf{a}$ — the projections onto the tangent vectors — as

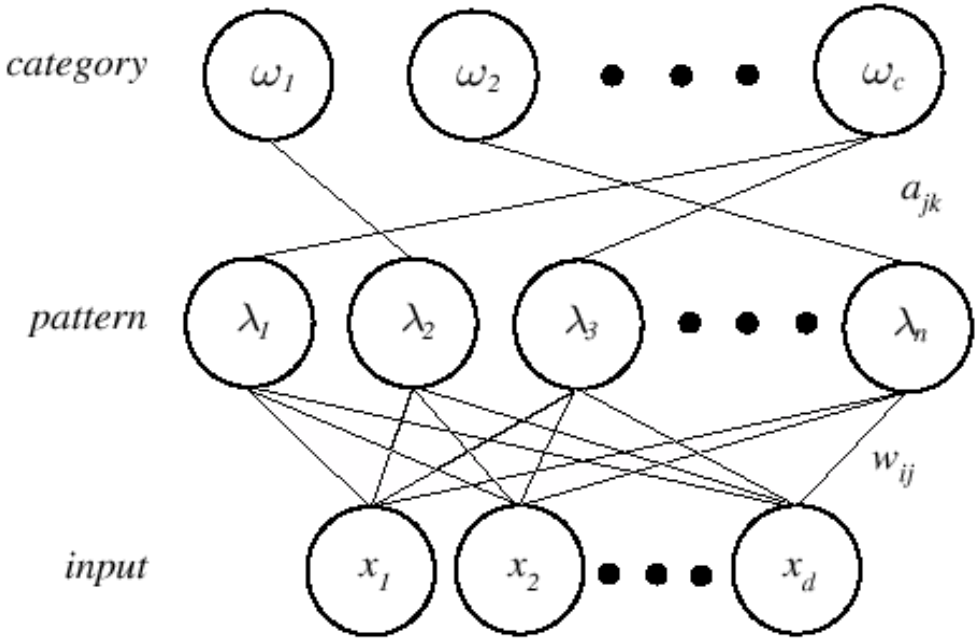
$$\nabla_{\mathbf{a}} D^2(\mathbf{x}' + \mathbf{Ta}, \mathbf{x}) = 2\mathbf{T}^t(\mathbf{x}' + \mathbf{Ta} - \mathbf{x}). \tag{63}$$

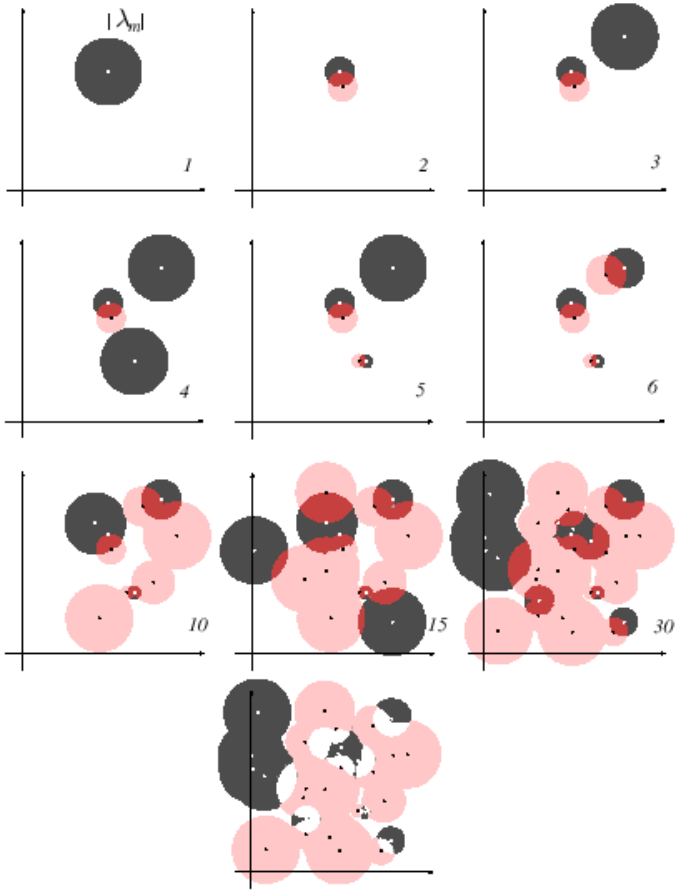
Thus we can start with an arbitrary $\mathbf{a}$ and take a step in the direction of the negative gradient, updating our parameter vector as

$$\mathbf{a}(t+1) = \mathbf{a}(t) - \eta\mathbf{T}^t(\mathbf{Ta}(t) + \mathbf{x}' - \mathbf{x}), \tag{64}$$

Reduced Coulomb Energy (RCE) Networks

- An RCE network, during training tunes the window width around each pattern according to its distance from the nearest pattern in a different class.
- During training, each pattern initiates a new circle and circle radii are adapted so that they do not contain patterns of different classes.
- Black circles represent class 1, pink circles class 2, whereas dark red areas represent ambiguous regions where no classification decision can be made.





Reduced Coulomb Energy (RCE) Networks

Algorithm 4 (RCE training)

1 **begin initialize** $j = 0, n = \#\text{patterns}, \epsilon = \text{small param}, \lambda_m = \text{max radius}$
2 **do** $j \leftarrow j + 1$
3 **train weight:** $w_{jk} \leftarrow x_k$
4 **find nearest pt not in** ω_i: $\hat{\mathbf{x}} \leftarrow \arg\min_{\mathbf{x} \notin \omega_i} D(\mathbf{x}, \mathbf{x}')$
5 **set radius:** $\lambda_j \leftarrow Min[D(\hat{\mathbf{x}}, \mathbf{x}') - \epsilon, \lambda_m]$
6 **if** $\mathbf{x} \in \omega_i$ **then** $a_{ic} \leftarrow 1$
7 **until** $j = n$
8 **end**

Algorithm 5 (RCE classification)

1 **begin initialize** $j = 0, k = 0, \mathbf{x} = \text{test pattern}, \mathcal{D}_t = \{\}$
2 **do** $j \leftarrow j + 1$
3 **if** $D(\mathbf{x}, \mathbf{x}'_j) < \lambda_j$ **then** $\mathcal{D}_t \leftarrow \mathcal{D}_t \cup \mathbf{x}'_j$
4 **until** $j = n$
5 **if** cat of all $\mathbf{x}'_j \in \mathcal{D}_t$ is the same **then** **return** label of all $\mathbf{x}_k \in \mathcal{D}_t$
6 **else** **return** "ambiguous" label
7 **end**