

Pattern Recognition

Chapter 4

Non-Parametric Density Estimation

Non-Parametric Density Estimation

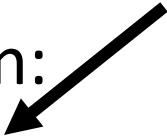
- Model the probability density function without making any assumption about its functional form.
- Any non-parametric density estimation technique has to deal with the choice of “*smoothing*” parameters that govern the smoothness of the estimated density.

- Discuss three types of methods based on:

(1) Histograms

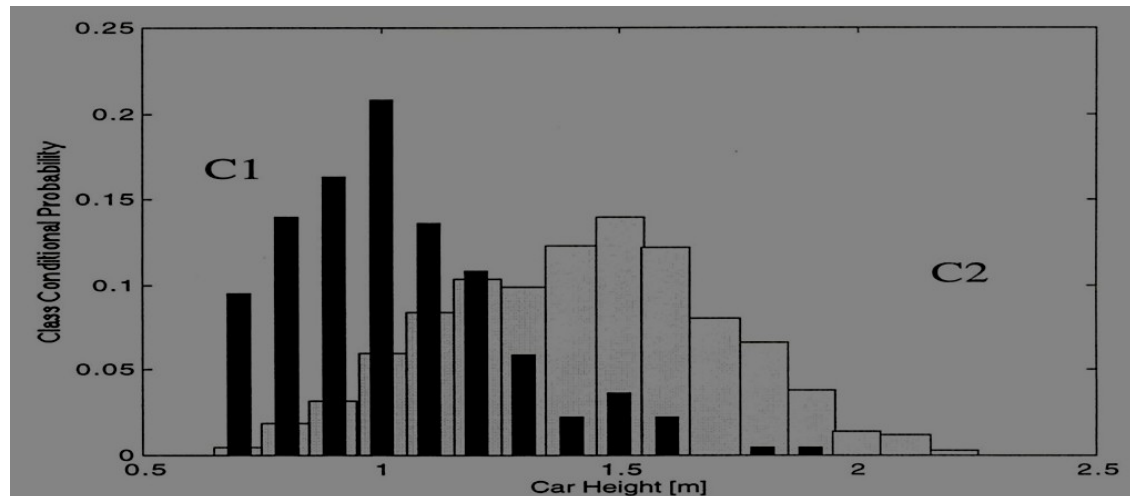
(2) Kernels

(3) K-nearest neighbors


$$P(\omega_j / x) = \frac{p(x / \omega_j)P(\omega_j)}{p(x)}$$

Histogram-Based Density Estimation

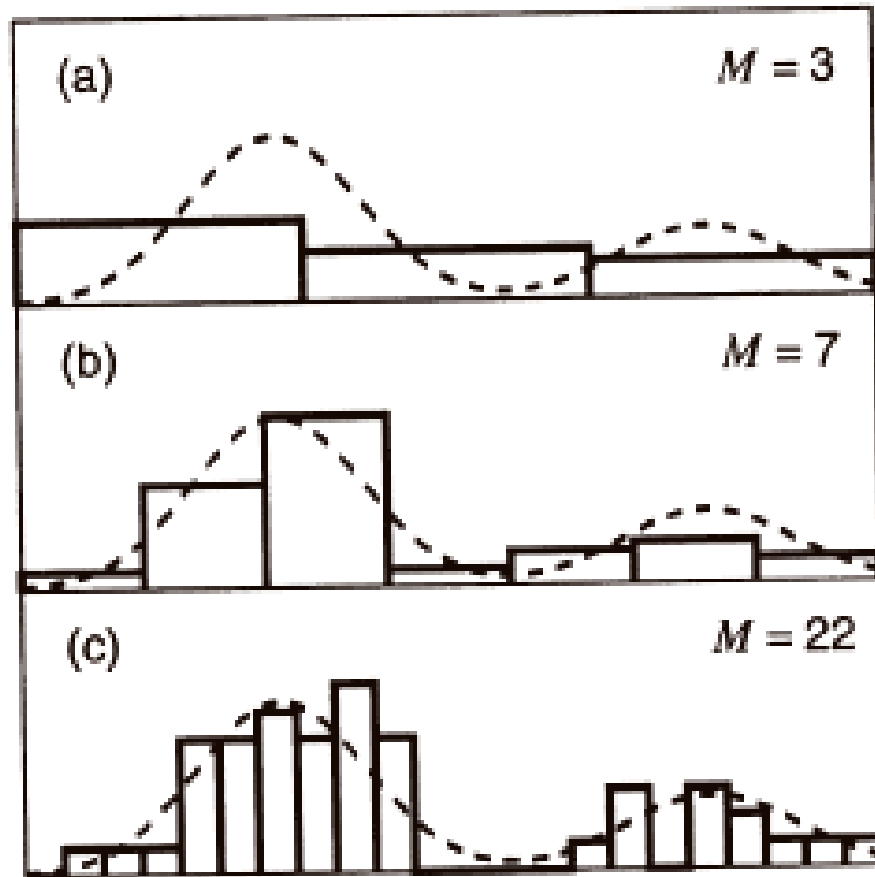
- Suppose each data point \mathbf{x} is represented by an n -dimensional feature vector (x_1, x_2, \dots, x_n) .
- The histogram is obtained by dividing each x_i -axis into a number of bins M and approximating the density at each value of x_i by the fraction of the points that fall inside the corresponding bin.



Histogram-Based Density Estimation (cont'd)

- The number of bins M (or bin size) is acting as a **smoothing** parameter.
 - If bin width is small (i.e., big M), then the estimated density is very spiky (i.e., noisy).
 - If bin width is large (i.e., small M), then the true structure of the density is smoothed out.
- In practice, we need to find an optimal value for M that compromises between these two issues.

Histogram-Based Density Estimation (cont'd)



Advantages of Histogram-Based Density Estimation

- Once the histogram has been constructed, the data is not needed anymore (i.e., memory efficient)
- Retain only info on the sizes and locations of histogram bins.
- Histogram can be built sequentially ... (i.e., consider the data one at a time and then discard).

Drawbacks of Histogram-Based Density Estimation

- The estimated density is not smooth and has discontinuities at the boundaries of the histogram bins.
- They do not generalize well in high dimensions.
 - Consider a d -dimensional feature space.
 - If we divide each variable in M intervals, we will end up with M^d bins.
 - A huge number of examples would be required to obtain good estimates (i.e., otherwise, most bins would be empty and the density will be approximated by zero).

Density Estimation

- The probability that a given vector \mathbf{x} , drawn from the unknown density $p(\mathbf{x})$, will fall inside some region R in the input space is given by:

$$P = \int_R p(\mathbf{x}') d\mathbf{x}'$$

- If we have n data points $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ drawn independently from $p(\mathbf{x})$, the probability that k of them will fall in R is given by the binomial law:

$$P(k) = P_k = \binom{n}{k} P^k (1 - P)^{n-k}$$

Density Estimation (cont'd)

- The expected value of k is:

$$E[k] = nP$$

- The expected percentage of points falling in R is:

$$E[k / n] = P$$

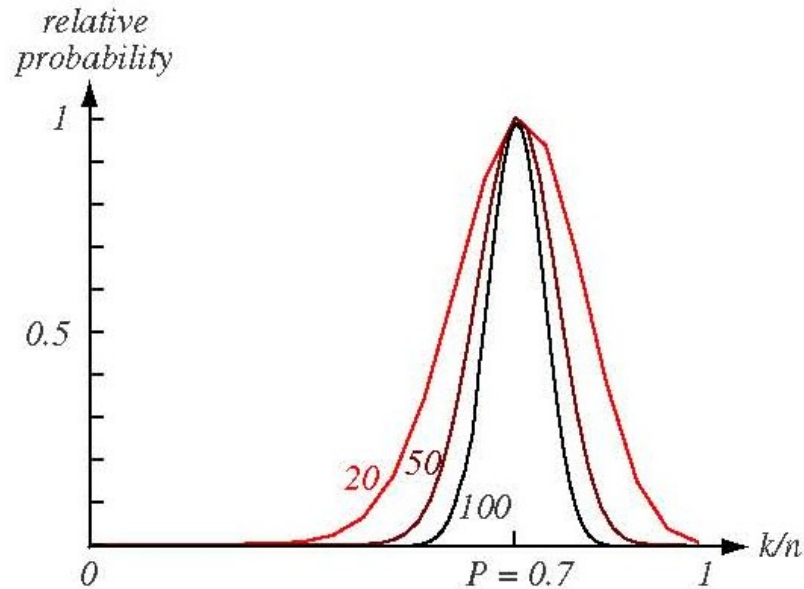
- The variance is given by:

$$\text{Var}[k / n] = E[(k / n - P)^2] = \frac{P(1 - P)}{n}$$

Density Estimation (cont'd)

- The distribution is sharply peaked as $n \rightarrow \infty$, thus:

$$P \approx k/n \quad \text{Approximation 1}$$



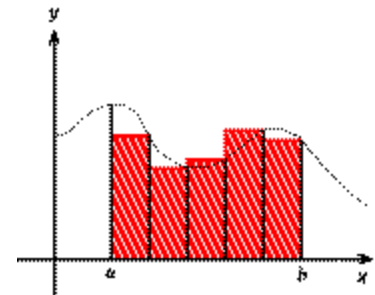
Density Estimation (cont'd)

- If we assume that $p(\mathbf{x})$ is continuous and does not vary significantly over the region R , we can approximate P by:

$$P = \int_R p(\mathbf{x}') dx' \approx p(\mathbf{x})V$$

Approximation 2

where V is the volume enclosed by R .



Density Estimation (cont'd)

- Combining these two approximations we have:

$$p(\mathbf{x}) \cong \frac{k/n}{V}$$

- The above approximation is based on **contradictory** assumptions:
 - R is relatively **large** (i.e., it contains many samples so that P_k is sharply peaked) – **Approximation 1**
 - R is relatively **small** so that $p(\mathbf{x})$ is approximately constant inside the integration region – **Approximation 2**
- We need to choose an optimum R in practice ...

Notation

- Suppose we form regions R_1, R_2, \dots containing \mathbf{x} .
 - R_1 contains 1 sample, R_2 contains 2 samples, etc.
- R_i has volume V_i and contains k_i samples.
- The n -th estimate $p_n(\mathbf{x})$ of $p(\mathbf{x})$ is given by:

$$p_n(\mathbf{x}) \cong \frac{k_n / n}{V_n}$$

Leading Methods for Density Estimation

- How to choose the optimum values for V_n and k_n ?

$$p_n(\mathbf{x}) \cong \frac{k_n / n}{V_n}$$

- Two leading approaches:

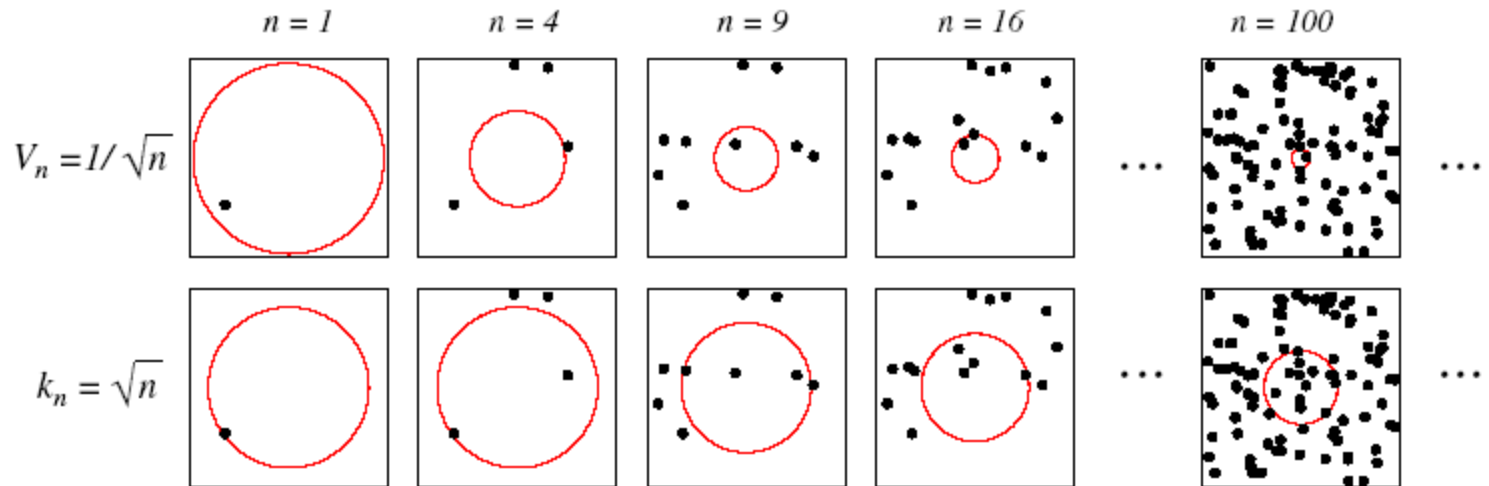
(1) Fix the volume V_n and determine k_n from the data (kernel-based density estimation methods), e.g.,

$$V_n = 1 / \sqrt{n}$$

(2) Fix the value of k_n and determine the corresponding volume V_n from the data (k -nearest neighbor method), e.g.,

$$k_n = \sqrt{n}$$

Leading Methods for Density Estimation (cont'd)



Kernel Density Estimation (Parzen Windows)

- **Problem:** Given a vector \mathbf{x} , estimate $p(\mathbf{x})$
- Assume R_n to be a hypercube with sides of length h_n , centered on the point \mathbf{x} :

$$V_n = h_n^d$$

$$p_n(\mathbf{x}) \cong \frac{k_n / n}{V_n}$$

- To find an expression for k_n (i.e., # points in the hypercube) let us define a kernel function:

$$\varphi(\mathbf{u}) = \begin{cases} 1 & |u_j| \leq \frac{1}{2} \quad j = 1, \dots, d \\ 0 & \text{otherwise} \end{cases}$$

Kernel Density Estimation (cont'd)

- The total number of points \mathbf{x}_i falling inside the hypercube is:

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

← equals 1 if \mathbf{x}_i falls within hypercube centered at \mathbf{x}

- Then, the estimate

$$p_n(\mathbf{x}) \cong \frac{k_n / n}{V_n}$$

becomes

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

Parzen windows estimate

Kernel Density Estimation (cont'd)

- The density estimate is a superposition of kernel functions and the samples \mathbf{x}_i .

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

- $\varphi(\mathbf{u})$ interpolates the density between samples.
- Each sample \mathbf{x}_i contributes to the estimate based on its distance from \mathbf{x} .

Properties of $\varphi(\mathbf{u})$

- The kernel function $\varphi(\mathbf{u})$ can have a more general form (i.e., not just hypercube).
- In order for $p_n(\mathbf{x})$ to be a legitimate estimate, $\varphi(\mathbf{u})$ must be a valid density itself:

$$\varphi(\mathbf{u}) \geq 0$$

$$\int \varphi(\mathbf{u}) d\mathbf{u} = 1$$

The role of h_n

- The parameter h_n acts as a smoothing parameter that needs to be optimized.
 - When h_n is too large, the estimated density is over-smoothed (i.e., superposition of “broad” kernel functions).
 - When h_n is too small, the estimate represents the properties of the data rather than the true density (i.e., superposition of “narrow” kernel functions)

$\varphi(\mathbf{u})$ as a function of h_n

- $\varphi(\mathbf{u})$ assuming different h_n values:

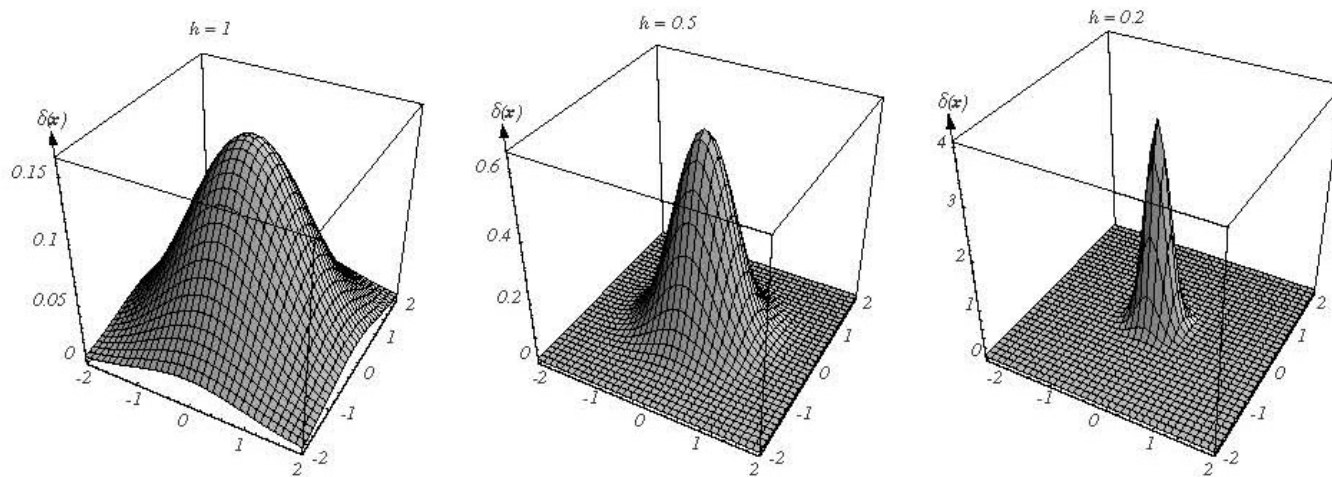


FIGURE 4.3. Examples of two-dimensional circularly symmetric normal Parzen windows for three different values of h . Note that because the $\delta(\mathbf{x})$ are normalized, different vertical scales must be used to show their structure. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

$p_n(\mathbf{x})$ as a function of h_n

- *Example: $p_n(\mathbf{x})$ estimates assuming 5 samples:*

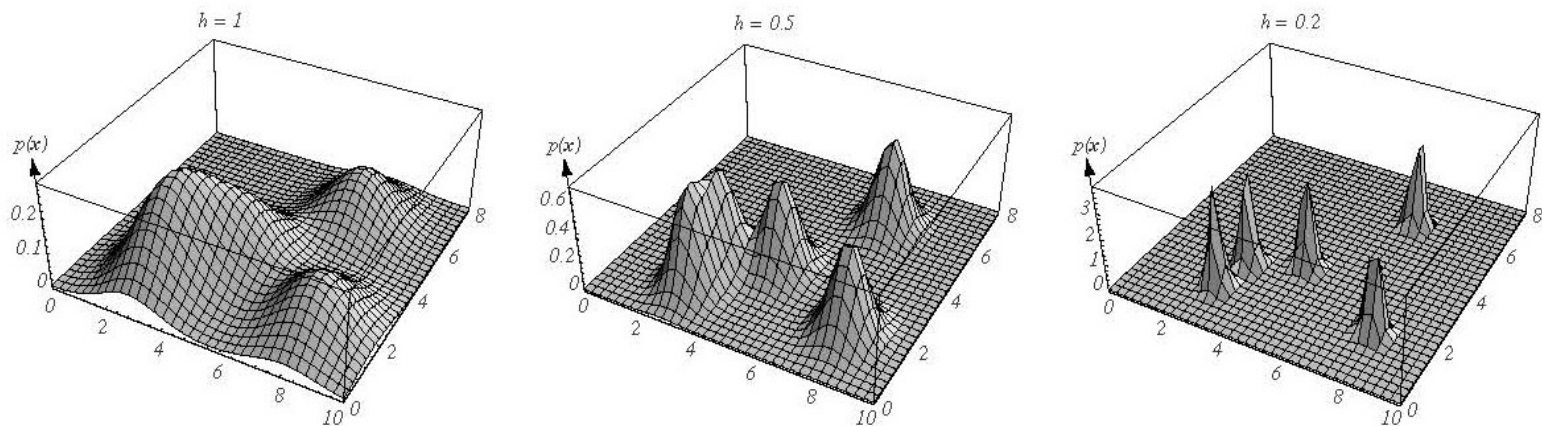
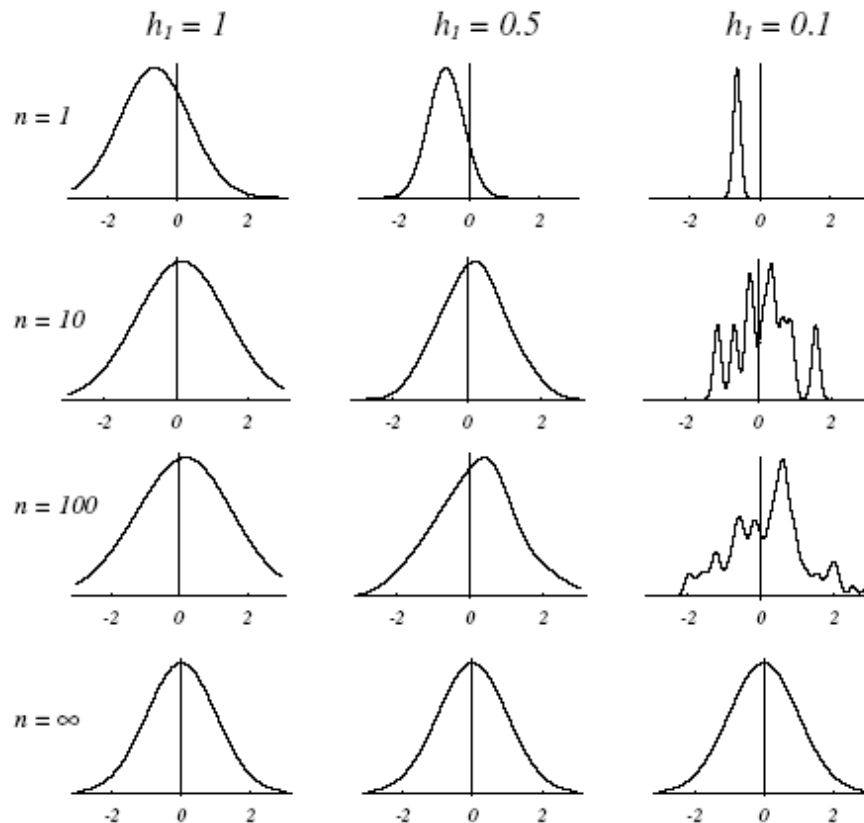


FIGURE 4.4. Three Parzen-window density estimates based on the same set of five samples, using the window functions in Fig. 4.3. As before, the vertical axes have been scaled to show the structure of each distribution. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

$p_n(\mathbf{x})$ as a function of h_n (cont'd)

- *Example:* both $p(\mathbf{x})$ and $\varphi(\mathbf{u})$ are Gaussian

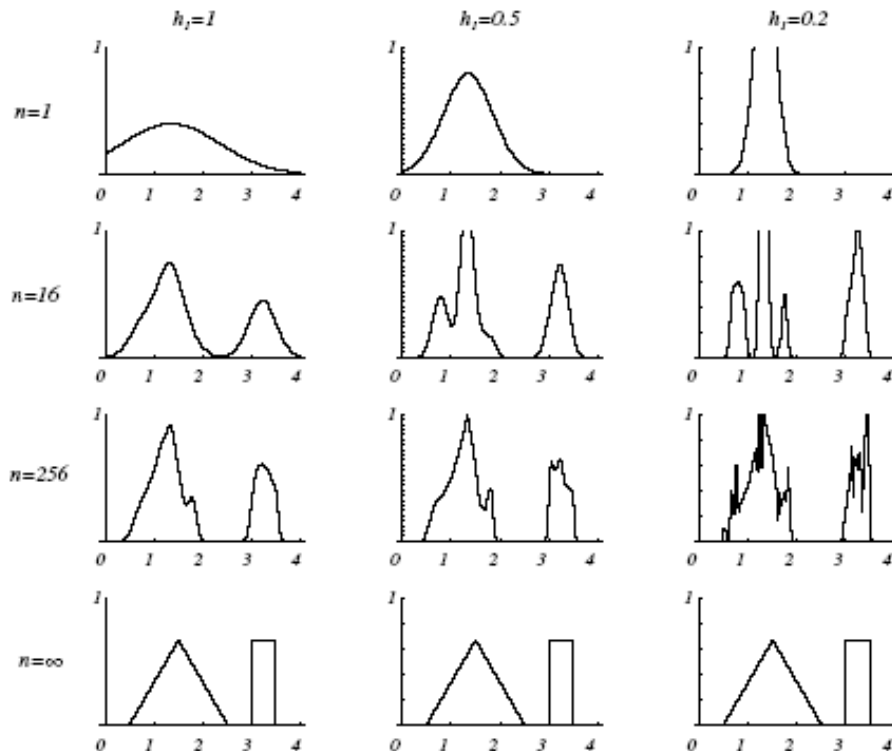


$$h_n = h_1 / \sqrt{n}$$

$$p_n(\mathbf{x})$$

$p_n(\mathbf{x})$ as a function of h_n (cont'd)

- *Example:* $p(\mathbf{x})$ consists of a uniform and triangular density and $\varphi(\mathbf{u})$ is Gaussian.



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

$$p_n(\mathbf{x})$$

Classification using kernel-based density estimation

- Estimate density for each class.
- Classify a test point by computing the posterior probabilities and picking the max.
- The decision regions depend on the choice of the kernel function and h_n .

Decision boundary

very low
error on
training
examples

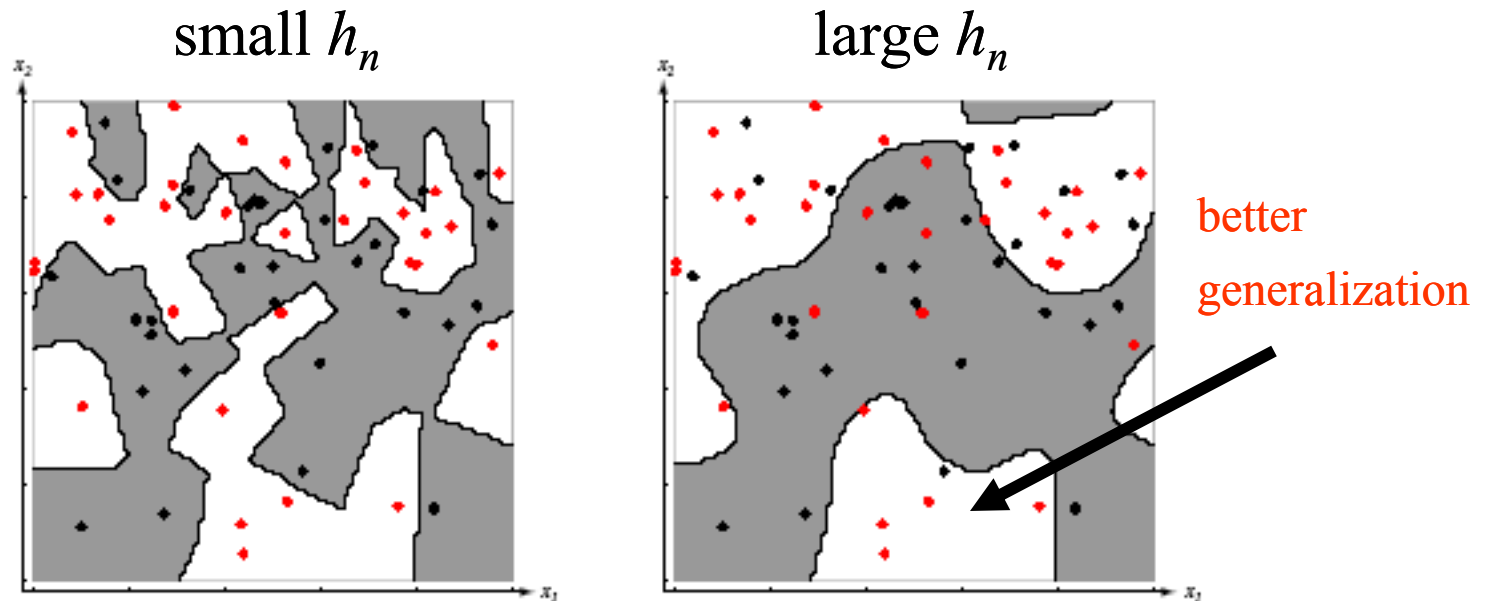


FIGURE 4.8. The decision boundaries in a two-dimensional Parzen-window dichotomizer depend on the window width h . At the left a small h leads to boundaries that are more complicated than for large h on same data set, shown at the right. Apparently, for these data a small h would be appropriate for the upper region, while a large h would be appropriate for the lower region; no single window width is ideal overall. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Drawbacks of kernel-based methods

- Require a large number of samples.
- Require all the samples to be stored.
- Evaluation of the density could be very slow if the number of data points is large.

- **K_n - Nearest neighbor estimation**

- **Goal:** a solution for the problem of the unknown “best” window function

- Let the cell volume be a function of the training data
 - Center a cell about x and let it grows until it captures k_n samples ($k_n = f(n)$)
 - k_n are called the k_n nearest-neighbors of x

2 possibilities can occur:

- Density is high near x ; therefore the cell will be small which provides a good resolution
 - Density is low; therefore the cell will grow large and stop until higher density regions are reached
 - We can obtain a family of estimates by setting $k_n = k_1 / \sqrt{n}$ and choosing different values for k_1

KNN – Number of Neighbors

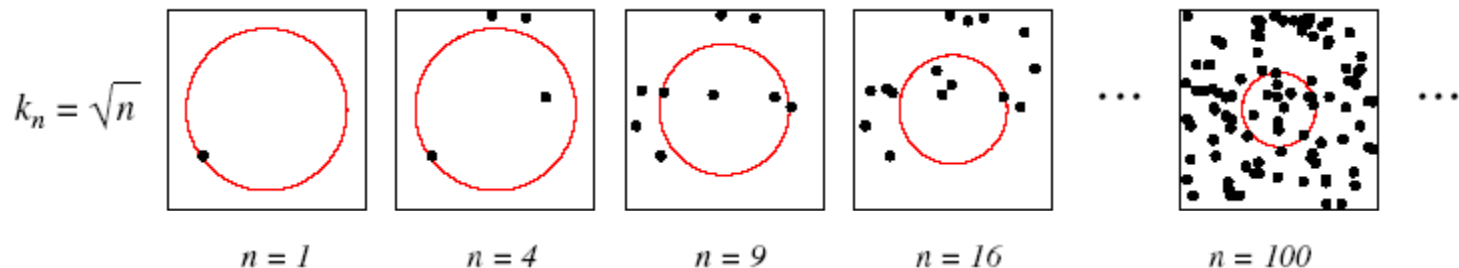
- If $K=1$, select the nearest neighbor
- If $K>1$,
 - For classification select the most frequent neighbor.
 - For regression calculate the average of K neighbors.

k_n -nearest-neighbor estimation

- Fix k_n and allow V_n to vary:
 - Consider a hypersphere around \mathbf{x} .
 - Allow the radius of the hypersphere to grow until it contains k_n data points.
 - V_n is determined by the volume of the hypersphere.

$$p_n(\mathbf{x}) \cong \frac{k_n / n}{V_n}$$

size depends
on density



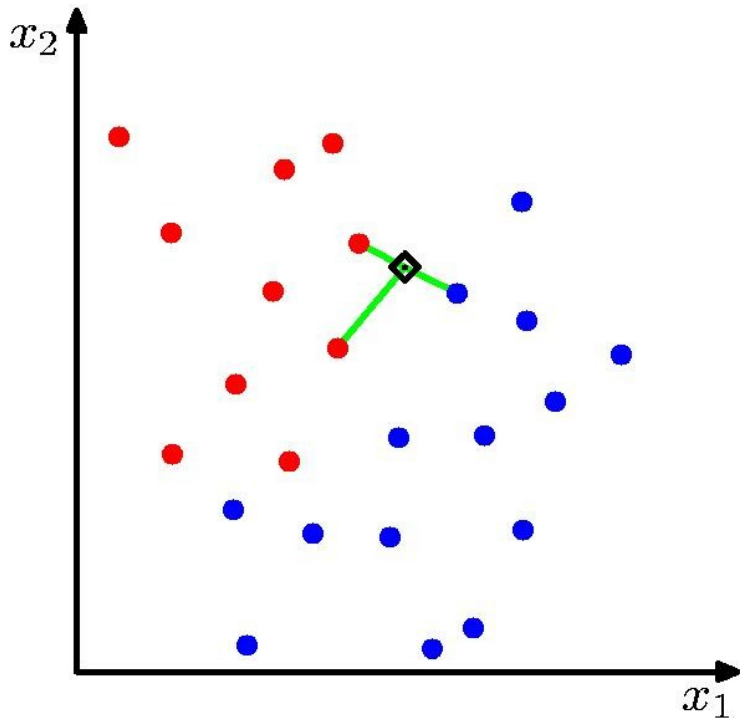
Nearest neighbor methods for classification and regression

- Nearest neighbors is usually used for classification or regression:
- For regression, average the predictions of the K nearest neighbors.
- For classification, pick the class with the most votes.
 - Let the k 'th nearest neighbor contribute a count that falls off with k . For example,

$$1 + \frac{1}{2^k}$$

The decision boundary implemented by 3NN

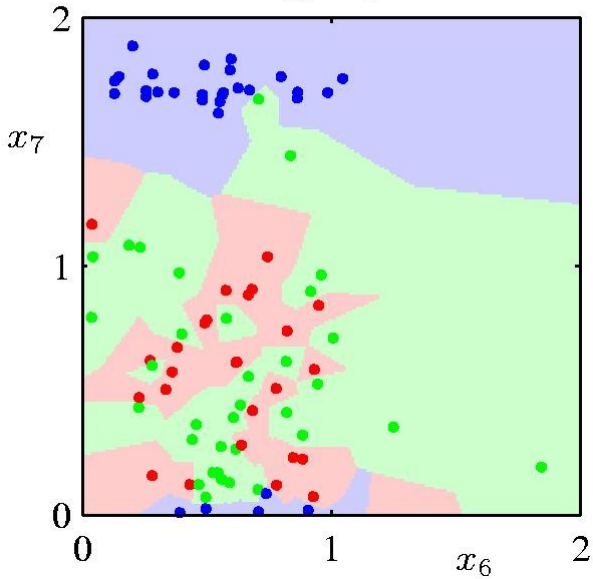
- The boundary is always the perpendicular bisector of the line between two points



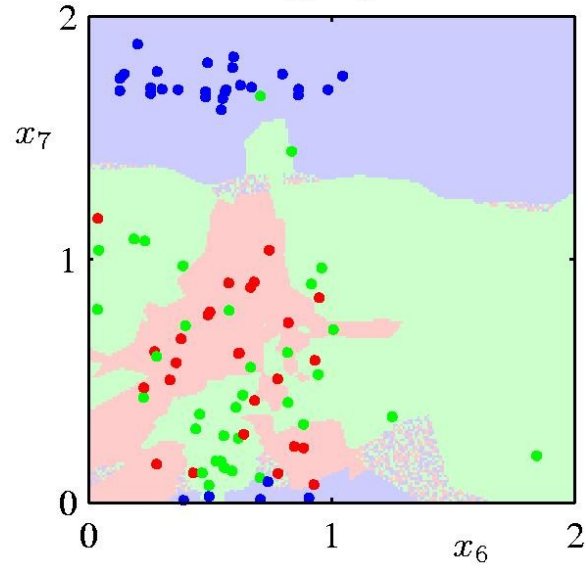
(a)

Regions defined by using various numbers of neighbors

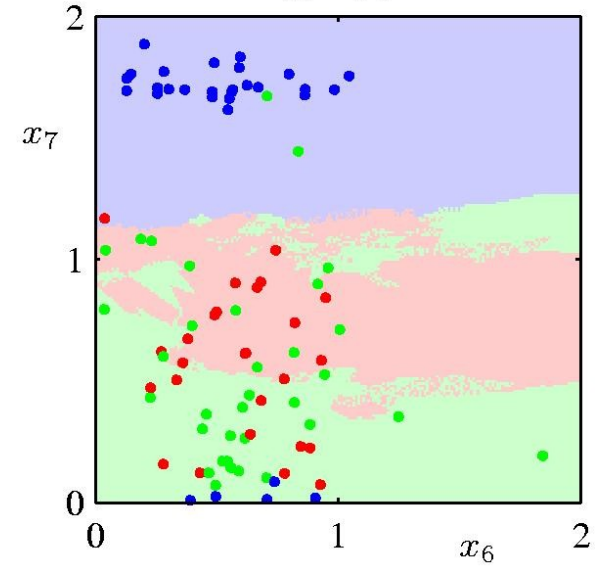
$K = 1$



$K = 3$

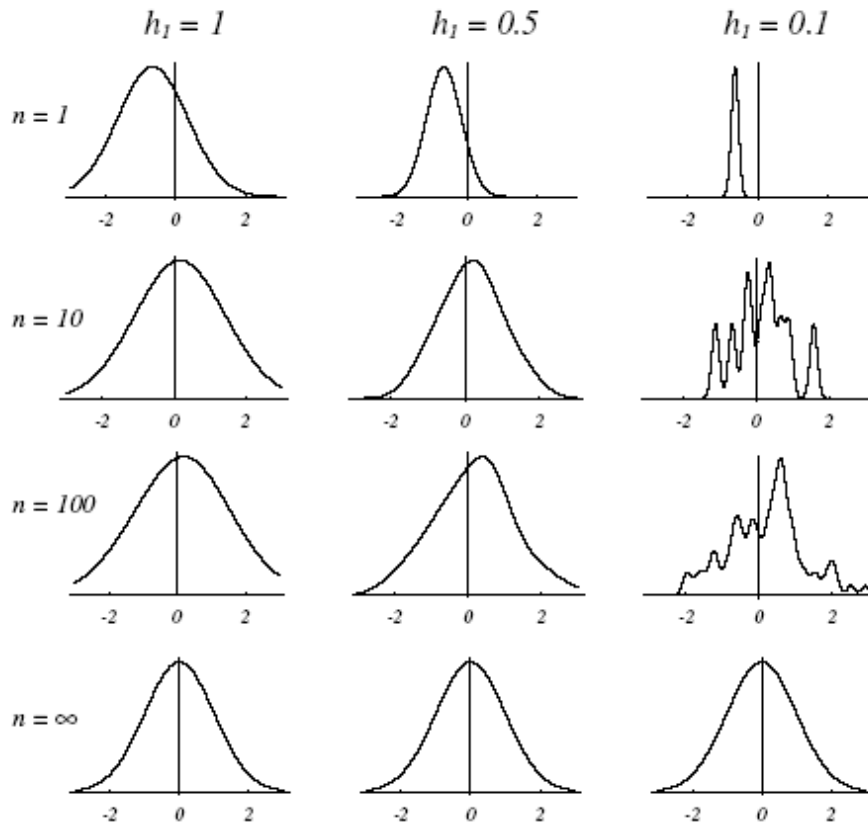


$K = 31$

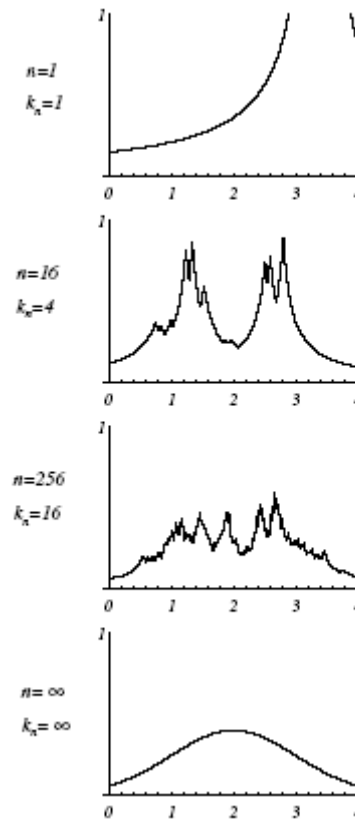


Parzen windows vs k_n -nearest-neighbor estimation

Parzen windows



k_n -nearest-neighbor



$$k_n = k_1 \sqrt{n}$$

k_n -nearest-neighbor classification

- Suppose that we have c classes and that class ω_i contains n_i points with $n_1+n_2+\dots+n_c=n$

$$P(\omega_i / \mathbf{x}) = \frac{p_n(\mathbf{x} / \omega_i)P(\omega_i)}{p_n(\mathbf{x})}$$

- Given a point \mathbf{x} , we find the k_n nearest neighbors
Suppose that k_i points from k_n belong to class ω_i ,
then:

$$p_n(\mathbf{x} / \omega_i) = \frac{k_i}{n_i V_n}$$

k_n -nearest-neighbor classification (cont'd)

- The prior probabilities can be computed as:

$$P(\omega_i) = \frac{n_i}{n}$$

- Using the Bayes' rule, the posterior probabilities can be computed as follows:

$$P(\omega_i / \mathbf{x}) = \frac{p_n(\mathbf{x} / \omega_i)P(\omega_i)}{p_n(\mathbf{x})} = \frac{k_i}{k_n}$$

where $p_n(\mathbf{x}) = \frac{k_n}{nV_n}$

k_n -nearest-neighbor rule

- k -nearest-neighbor classification rule:

Given a data point \mathbf{x} , find a hypersphere around it that contains k points and assign \mathbf{x} to the class having the largest number of representatives inside the hypersphere.

$$P(\omega_i / \mathbf{x}) = \frac{p_n(\mathbf{x} / \omega_i)P(\omega_i)}{p_n(\mathbf{x})} = \frac{k_i}{k_n}$$

- When $k=1$, we get the nearest-neighbor rule.

Example

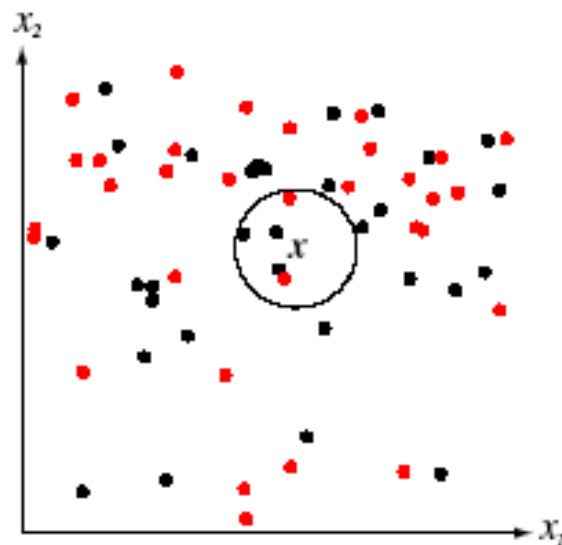


FIGURE 4.15. The k -nearest-neighbor query starts at the test point x and grows a spherical region until it encloses k training samples, and it labels the test point by a majority vote of these samples. In this $k = 5$ case, the test point x would be labeled the category of the black points. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Example

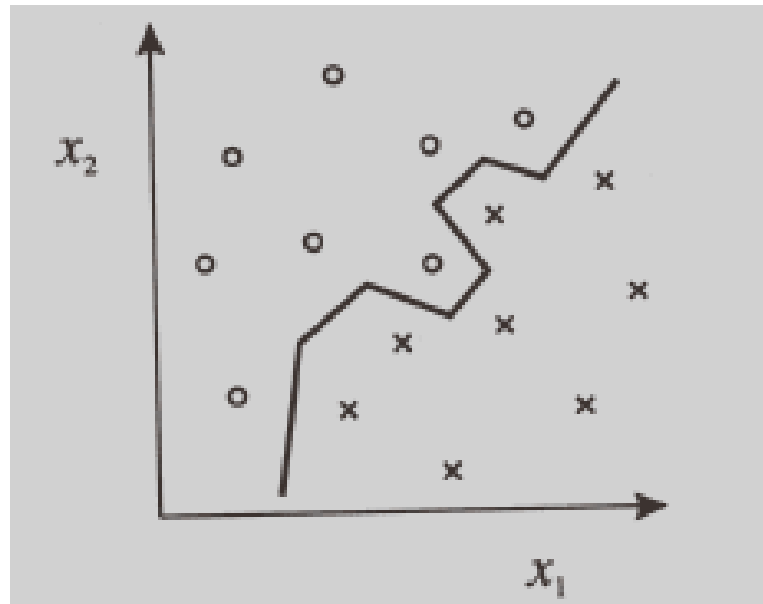
$k = 3$ (odd value)
and $x = (0.10, 0.25)^t$

Prototypes	Labels
$(0.15, 0.35)$	ω_1
$(0.10, 0.28)$	ω_2
$(0.09, 0.30)$	ω_5
$(0.12, 0.20)$	ω_2

- Closest vectors to \mathbf{x} with their labels are:
 $\{(0.10, 0.28, \omega_2); (0.12, 0.20, \omega_2); (0.15, 0.35, \omega_1)\}$
- Assign the label ω_2 to \mathbf{x} since ω_2 is the most frequently represented.

Decision boundary for k_n -nearest-neighbor rule

- The decision boundary is piece-wise linear.
- Each line segment corresponds to the perpendicular bisector of two points belonging to different classes.



Drawbacks of k -nearest-neighbor rule

- The resulting estimate is not a true density (i.e., its integral diverges).

e.g., if $n=1$ and $k_n = \sqrt{n}$,
$$p_1(x) = \frac{1}{2|x - x_1|}$$

- Require all the data points to be stored.
- Computing the closest neighbors could be time consuming (i.e., efficient algorithms are required).

Nearest-neighbor rule

$$(k_n=1)$$

- Suppose we have $D^n = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ labeled training samples (i.e., known classes).
- Let \mathbf{x}' in D^n be the closest point to \mathbf{x} , which needs to be classified.
- The nearest neighbor rule is to assign \mathbf{x} the class associated with \mathbf{x}' .

Example

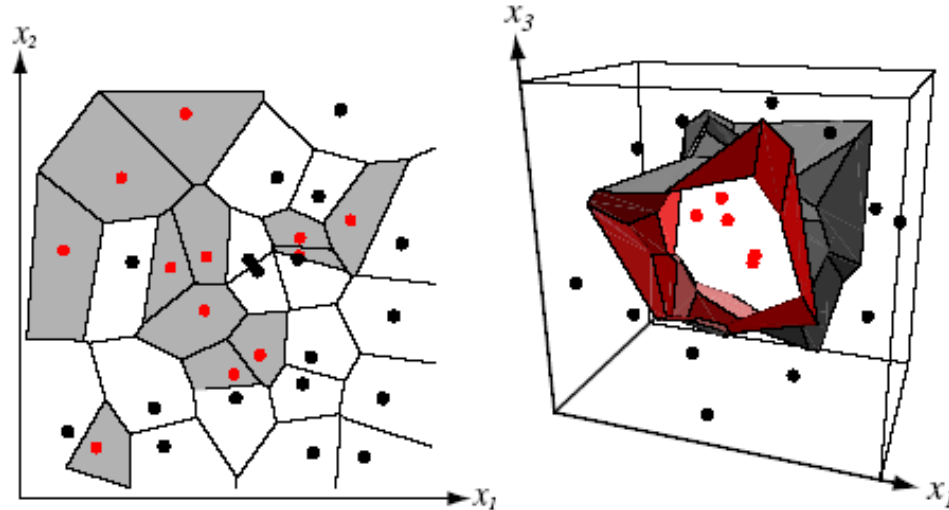
- $x = (0.10, 0.25)^t$

Training Examples	Labels	Distance
(0.15, 0.35)	ω_1	0.118
(0.10, 0.28)	ω_2	0.030
(0.09, 0.30)	ω_5	0.051
(0.12, 0.20)	ω_2	0.054

Decision boundary (nearest-neighbor rule)

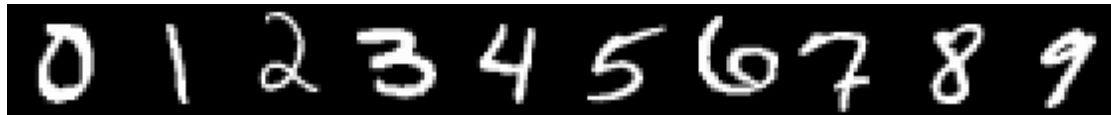
- The nearest neighbor rule leads to a Voronoi tessellation of the feature space.
- Each cell contains all the points that are closer to a given training point \mathbf{x} than to any other training points.
- All the points in a cell are labeled by the category of the training point in that cell.

Decision boundary (nearest-neighbor rule) (cont'd)



- Knowledge of this boundary is sufficient to classify new points.
- The boundary itself is rarely computed...
 - Many algorithms seek to retain only those points necessary to generate an identical boundary.

Example: Digit Recognition



- Yann LeCun – MNIST Digit Recognition
 - Handwritten digits
 - 28x28 pixel images ($d = 784$)
 - 60,000 training samples
 - 10,000 test samples
- **Nearest neighbor is competitive!!**

	Test Error Rate (%)
Linear classifier (1-layer NN)	12.0
K-nearest-neighbors, Euclidean	5.0
K-nearest-neighbors, Euclidean, deskewed	2.4
K-NN, Tangent Distance, 16x16	1.1
K-NN, shape context matching	0.67
1000 RBF + linear classifier	3.6
SVM deg 4 polynomial	1.1
2-layer NN, 300 hidden units	4.7
2-layer NN, 300 HU, [deskewing]	1.6
LeNet-5, [distortions]	0.8
Boosted LeNet-4, [distortions]	0.7

Computational complexity (nearest-neighbor rule)

- Assuming n training examples in d dimensions, a straightforward implementation would take $O(dn^2)$
- A parallel implementation would take $O(1)$

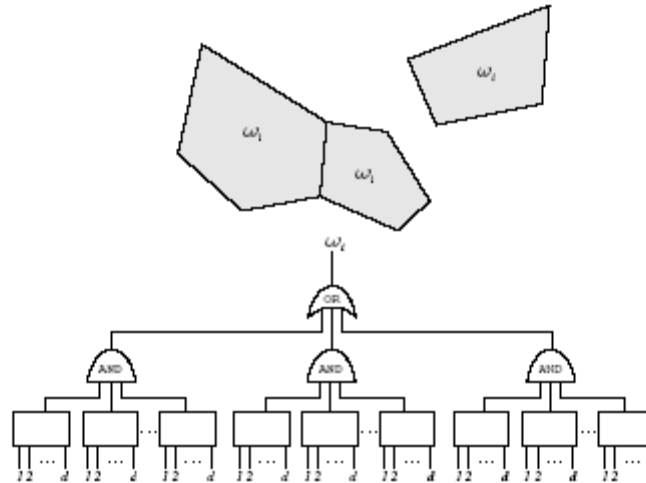


FIGURE 4.17. A parallel nearest-neighbor circuit can perform search in constant—that is, $O(1)$ —time. The d -dimensional test pattern x is presented to each box, which calculates which side of a cell's face x lies on. If it is on the "close" side of every face of a cell, it lies in the Voronoi cell of the stored pattern, and receives its label. In the case shown, each of the three AND gates corresponds to a single Voronoi cell. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Reducing computational complexity

- Three generic approaches:
 - Computing partial distances
 - Pre-structuring (e.g., search tree)
 - Editing the stored prototypes

Partial distances

- Compute distance using first r dimensions only:

$$D_r(\mathbf{x}, \mathbf{x}') = \left(\sum_{k=1}^r (x_k - x'_k)^2 \right)^{1/2}$$

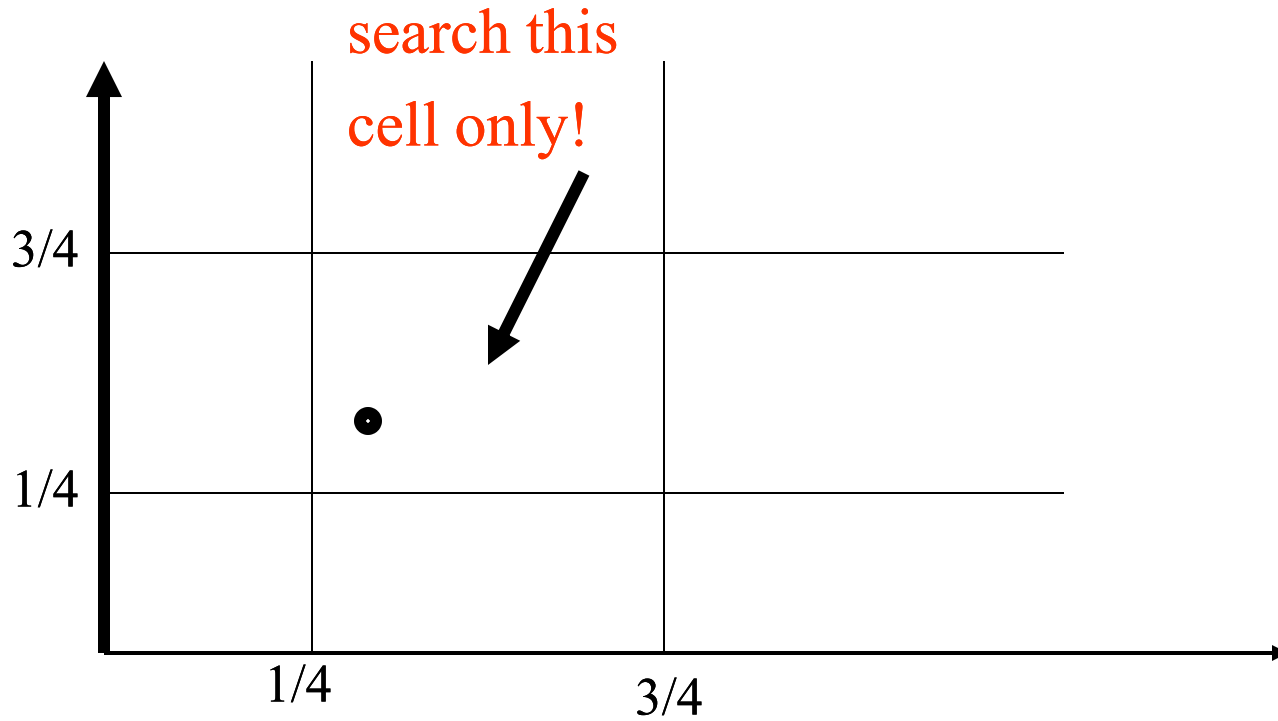
where $r < d$.

- If the partial distance is too great (i.e., greater than the distance of \mathbf{x} to current closest prototype), there is no reason to compute additional terms.

Pre-structuring: Bucketing

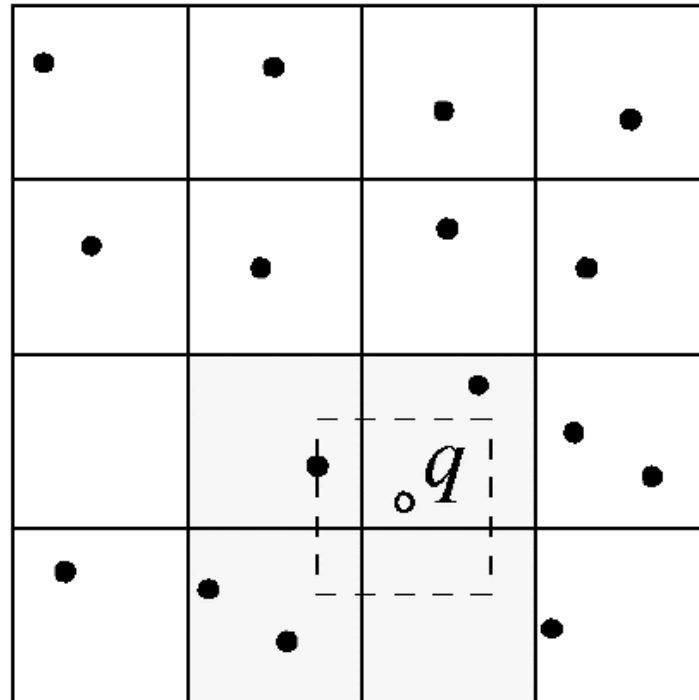
- In the Bucketing algorithm, the space is divided into identical cells.
 - For each cell the data points inside it are stored in a list.
 - Given a test point \mathbf{x} , find the cell that contains it.
 - Search only the points inside that cell!
 - Does not guarantee to find the true nearest neighbor(s) !

Pre-structuring: Bucketing (cont'd)



Pre-structuring: Bucketing (cont'd)

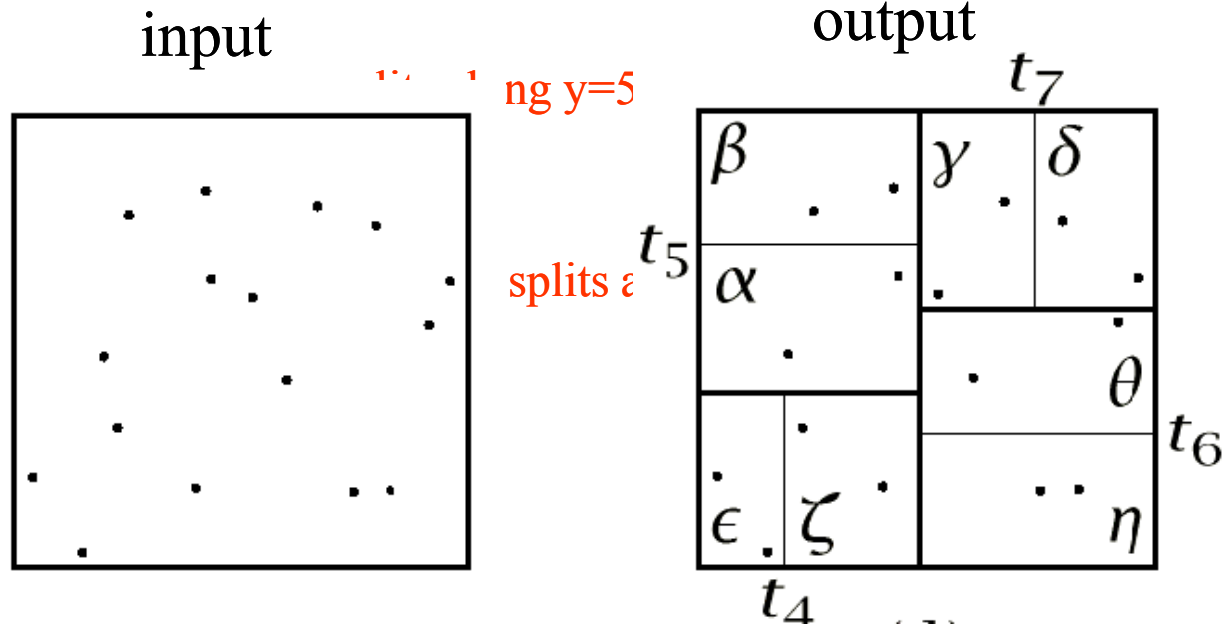
- Tradeoff:
 - speed vs accuracy



Pre-structuring: Search Trees (k -d tree)

- A k -d tree is a data structure for storing a finite set of points from a k -dimensional space.
- Generalization of binary search ...
- **Goal:** hierarchically decompose space into a relatively small number of cells such that no cell contains too many points.

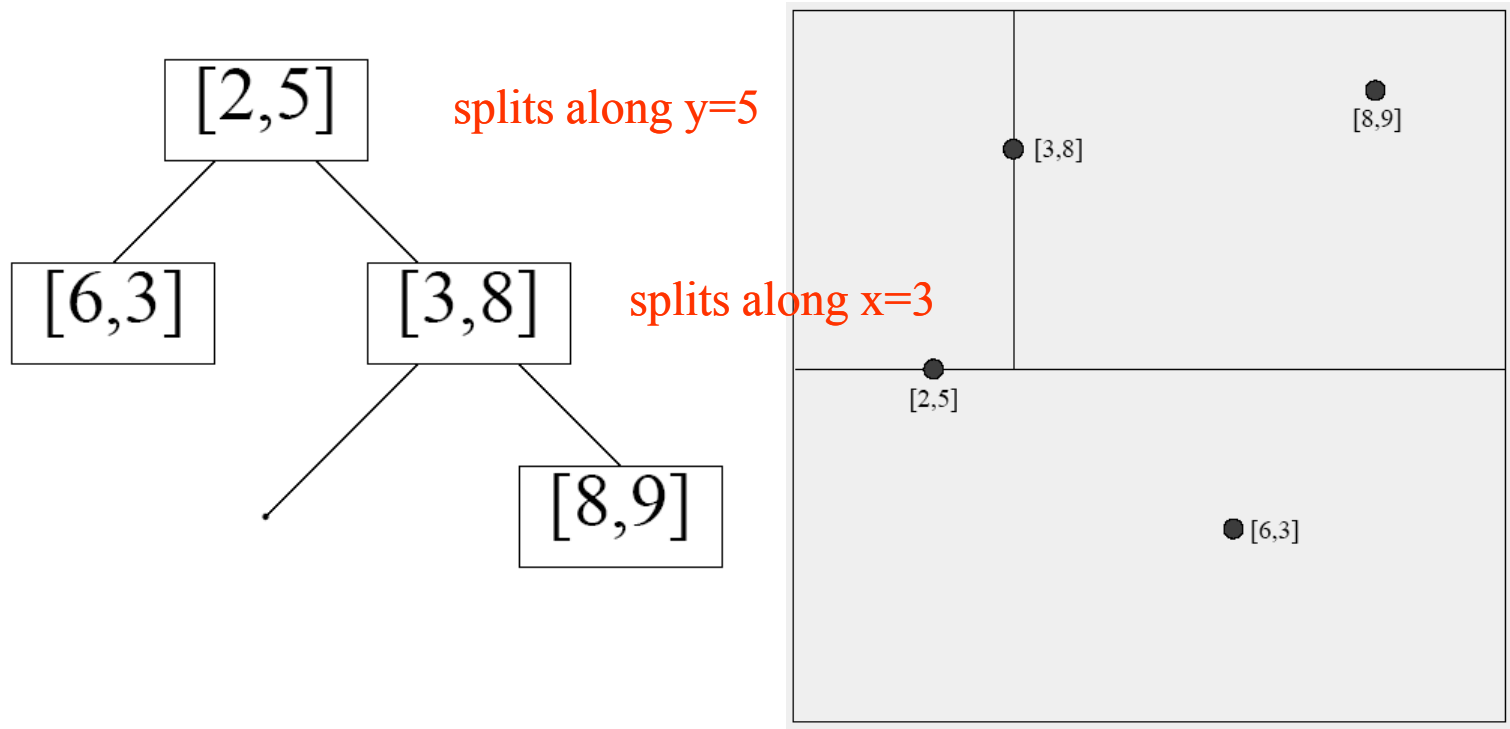
Pre-structuring: Search Trees (k -d tree) (cont'd)



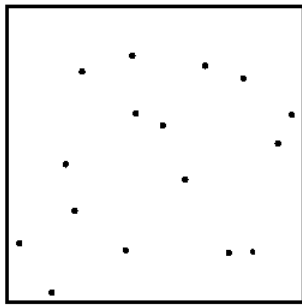
Pre-structuring: Search Trees (how to build a k -d tree)

- Each internal node in a k -d tree is associated with a hyper-rectangle and a hyper-plane orthogonal to one of the coordinate axis.
 - The hyper-plane splits the hyper-rectangle into two parts, which are associated with the child nodes.
 - The partitioning process goes on until the number of data points in the hyper-rectangle falls below some given threshold.

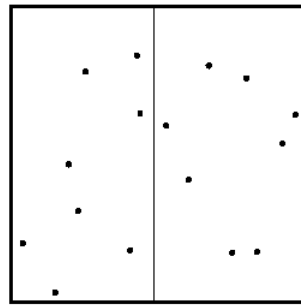
Pre-structuring: Search Trees (how to build a k-d tree) (cont'd)



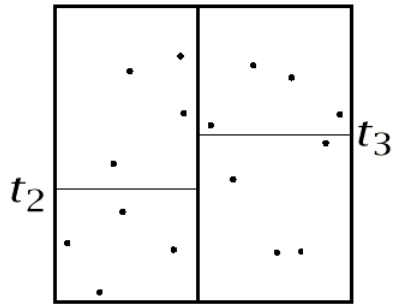
Pre-structuring: Search Trees (how to build a k -d tree) (cont'd)



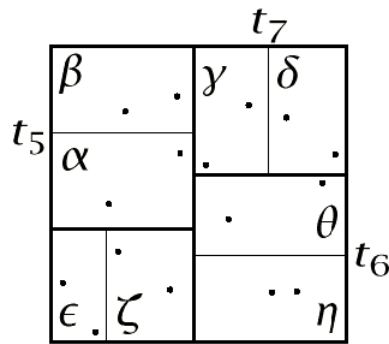
(a)



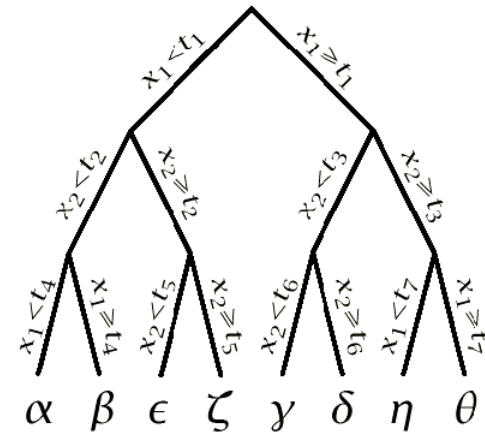
(b)



(c)



(d)



(e)

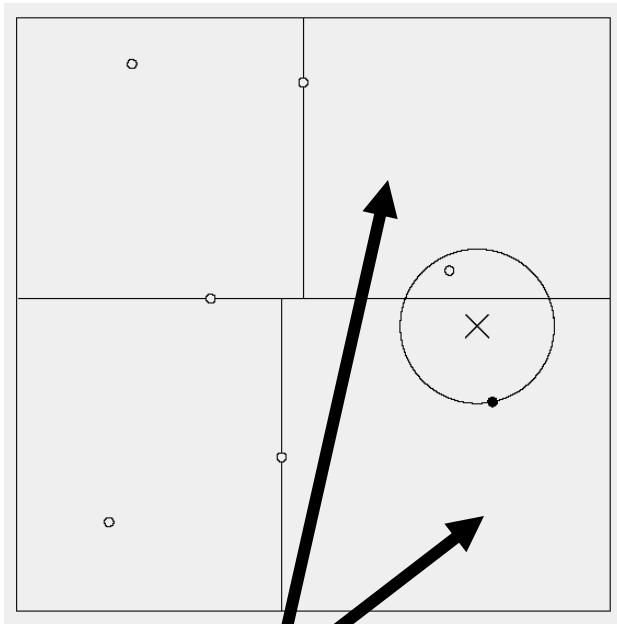
Pre-structuring: Search Trees (how to search using k -d trees)

- For a given query point, the algorithm works by first descending the tree to find the data points lying in the cell that contains the query point.
- Then it examines surrounding cells if they overlap the ball centered at the query point and the closest data point so far.

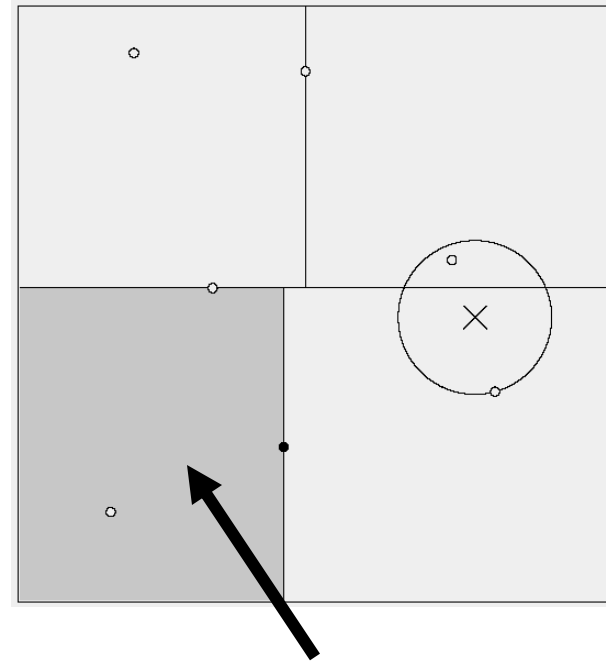
<http://www-2.cs.cmu.edu/~awm/animations/kdtree/nn-vor.ppt>

Pre-structuring: Search Trees

(how to search using k -d trees) (cont'd)



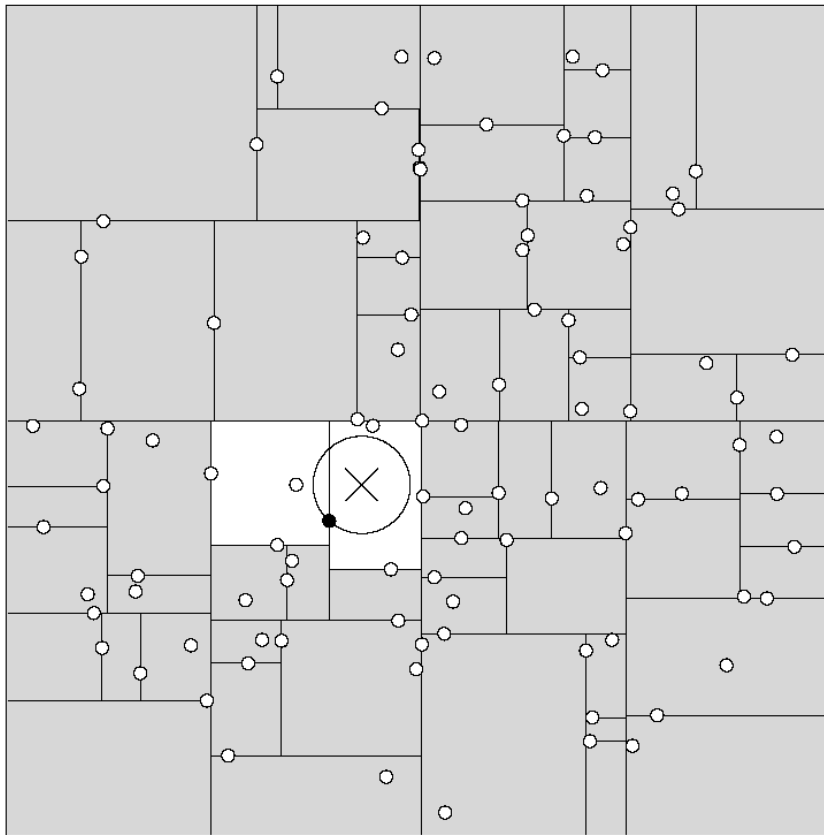
search ...



no need to search ...

Pre-structuring: Search Trees

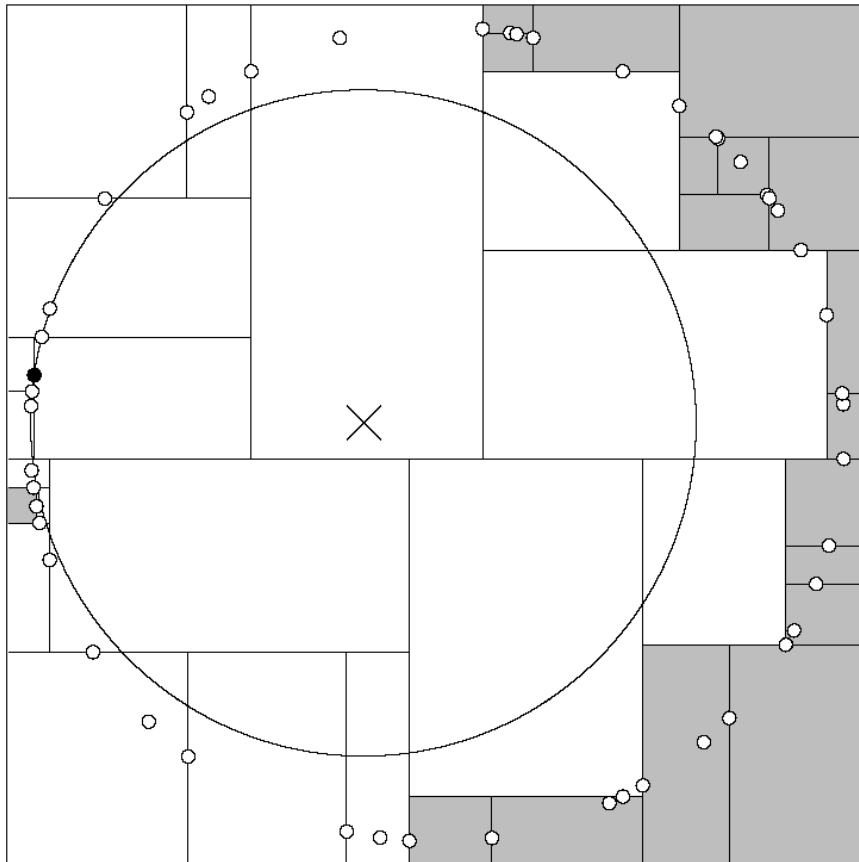
(how to search using k -d trees) (cont'd)



Generally during a nearest neighbour search only a few leaf nodes need to be inspected.

Pre-structuring: Search Trees

(how to search using k -d trees) (cont'd)



A bad distribution which forces almost all nodes to be inspected.

Editing

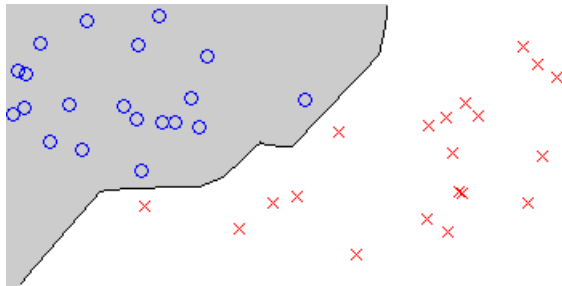
- **Goal:** reduce the number of training samples.
- Two main approaches:
 - **Condensing:** preserve decision boundaries.
 - **Pruning:** eliminate noisy examples to produce smoother boundaries and improve accuracy.

Editing using condensing

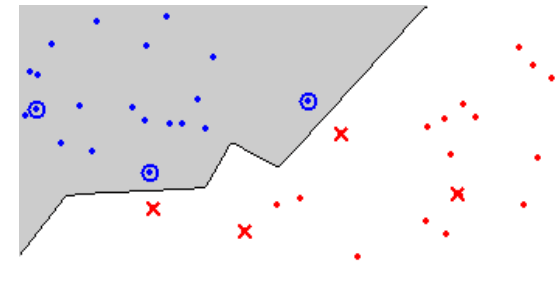
- Retain only the samples that are needed to define the decision boundary.
- Decision Boundary Consistent – a subset whose nearest neighbour decision boundary is close to the boundary of the entire training set.
- Minimum Consistent Set – the smallest subset of the training data that correctly classifies all of the original training data.

Editing using condensing (cont'd)

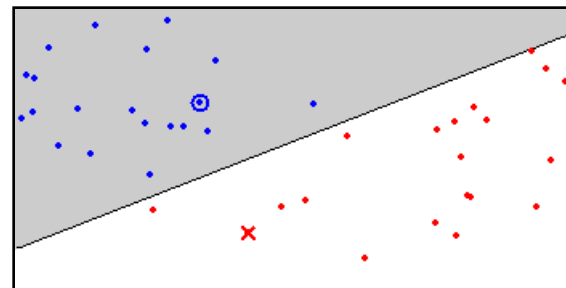
- Retain mostly points along the decision boundary.



Original data



Condensed data



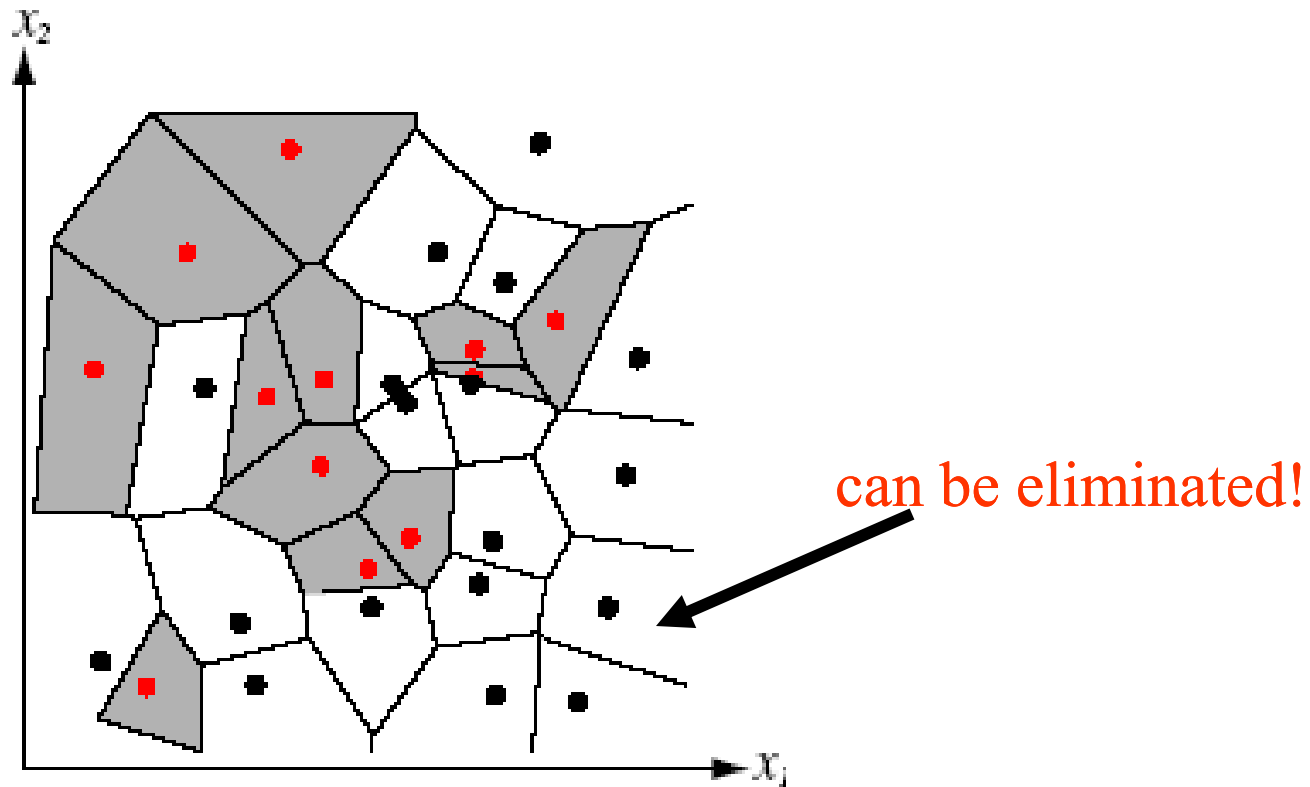
Minimum Consistent Set

Editing using condensing (cont'd)

- Keep points contributing to the boundary (i.e., at least one neighbor belongs to a different category).
- Eliminate prototypes that are surrounded by samples of the same category.

```
1 begin initialize  $j = 0$ ,  $\mathcal{D} =$  data set,  $n = \#$ prototypes
2     construct the full Voronoi diagram of  $\mathcal{D}$ 
3     do  $j \leftarrow j + 1$ ; for each prototype  $\mathbf{x}'_j$ 
4         Find the Voronoi neighbors of  $\mathbf{x}'_j$ 
5         if any neighbor is not from the same class as  $\mathbf{x}'_j$  then mark  $\mathbf{x}'_j$ 
6     until  $j = n$ 
7     Discard all points that are not marked
8     Construct the Voronoi diagram of the remaining (marked) prototypes
9 end
```

Editing using condensing (cont'd)

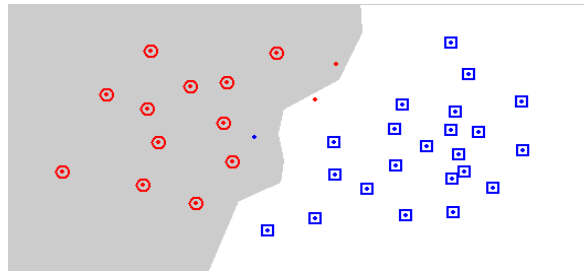
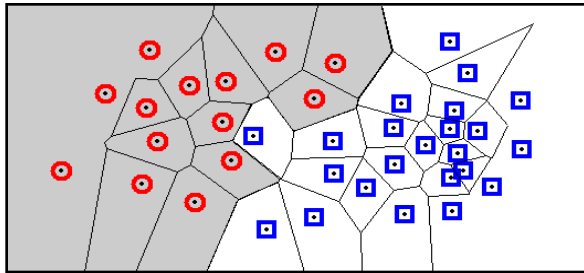


Editing using pruning

- Pruning seeks to remove “noisy” points and produces smooth decision boundaries.
- Often, it retains points far from the decision boundaries.
- **Wilson pruning:** remove points that do not agree with the majority of their k -nearest-neighbours.

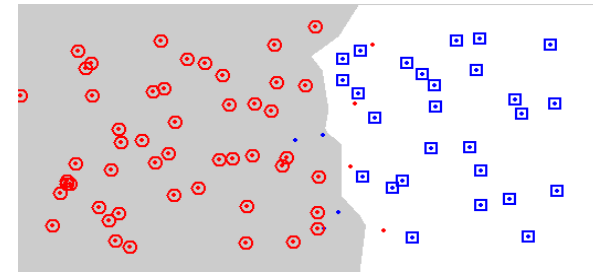
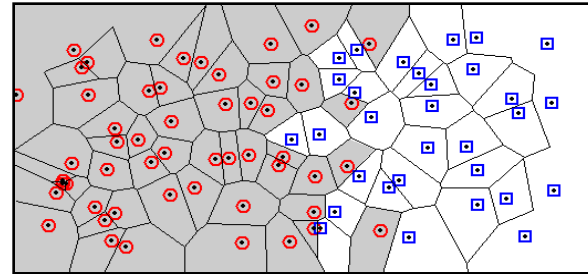
Editing using pruning (cont'd)

Original data



Wilson editing with $k=7$

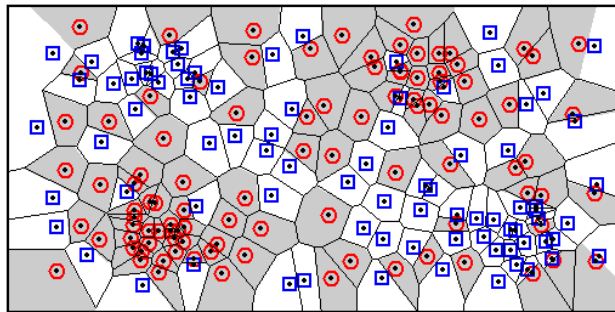
Original data



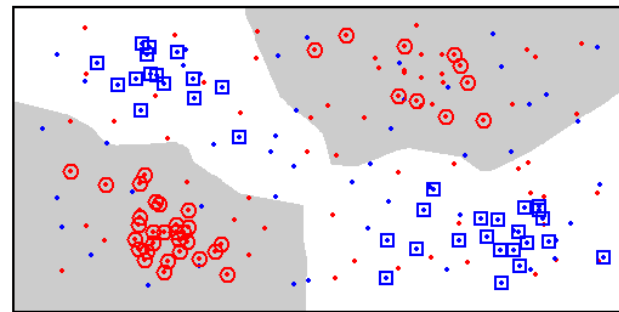
Wilson editing with $k=7$

Combined Editing/Condensing

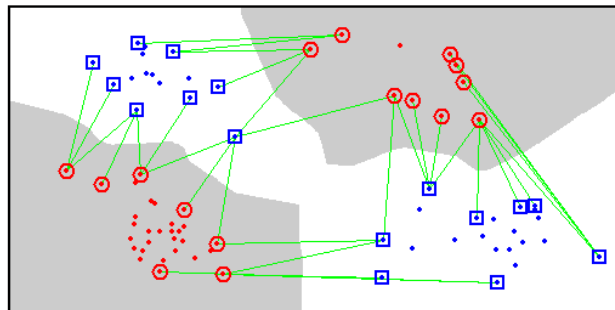
- (1) Prune the data to remove noise and smooth the boundary.
- (2) Condense to obtain a smaller subset.



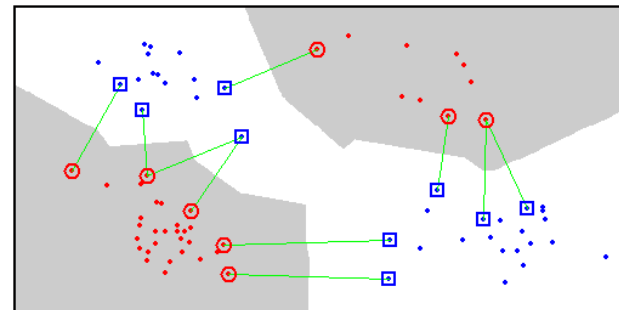
Original data: (99,85)



Multi-edit: (39,36)



Delaunay edited: (16,16)

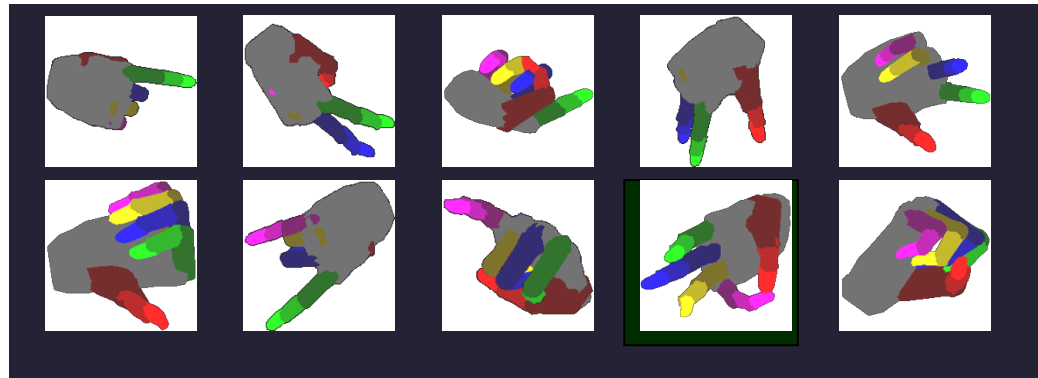


Gabriel edited: (8,9)

Example: 3D hand pose estimation



Database (107,328 images)



nearest
neighbor

General comments (nearest-neighbor classifier)

- The nearest neighbor classifier provides a powerful tool.
 - Its error is bounded to be at most two times of the Bayes error (in the limiting case).
 - It is easy to implement and understand.
 - It can be implemented efficiently.
 - Its performance, however, relies on the metric used to compute distances!

Properties of distance metrics

non-negativity: $D(\mathbf{a}, \mathbf{b}) \geq 0$

reflexivity: $D(\mathbf{a}, \mathbf{b}) = 0$ if and only if $\mathbf{a} = \mathbf{b}$

symmetry: $D(\mathbf{a}, \mathbf{b}) = D(\mathbf{b}, \mathbf{a})$

triangle inequality: $D(\mathbf{a}, \mathbf{b}) + D(\mathbf{b}, \mathbf{c}) \geq D(\mathbf{a}, \mathbf{c})$

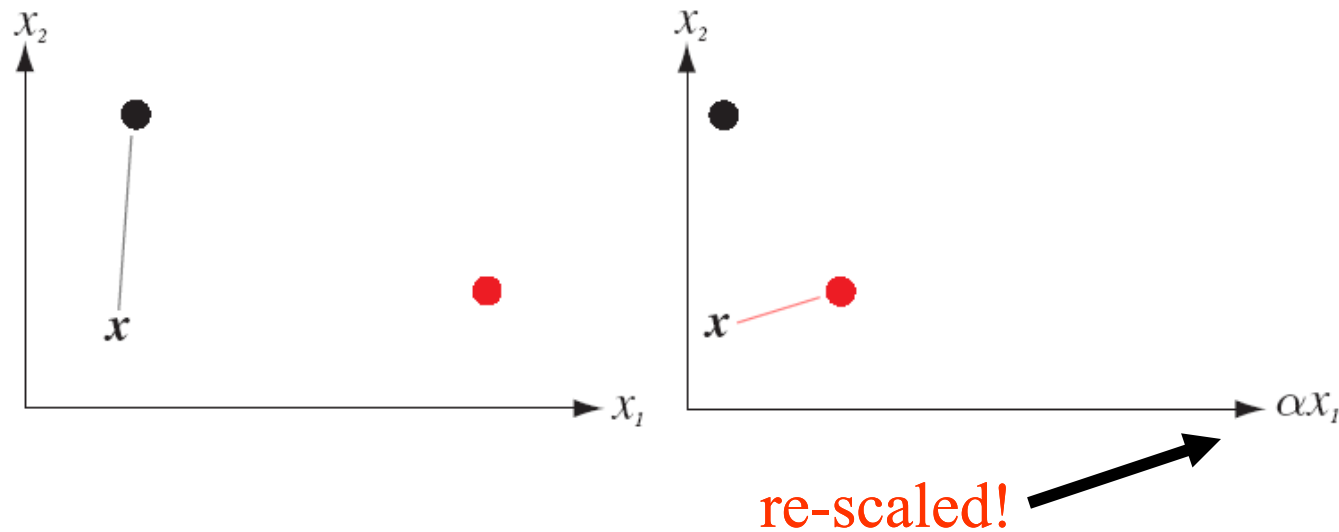
Distance metrics - Euclidean

- Euclidean distance:

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

- Distance relations can change by scaling (or other) transformations.
 - e.g., choose different units.

Distance metrics – Euclidean (cont'd)



- **Hint:** normalize data in each dimension if there is a large disparity in the ranges of values.