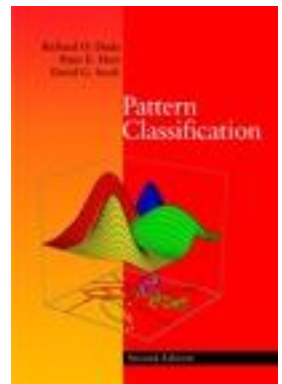
Chapter 4 (part 2): Non-Parametric Classification

- k_n–Nearest Neighbor Estimation
- The Nearest-Neighbor Rule
- Relaxation methods



All materials used in this course were taken from the textbook *"Pattern Classification"* by Duda et al., John Wiley & Sons, 2001 with the permission of the authors and the publisher

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**

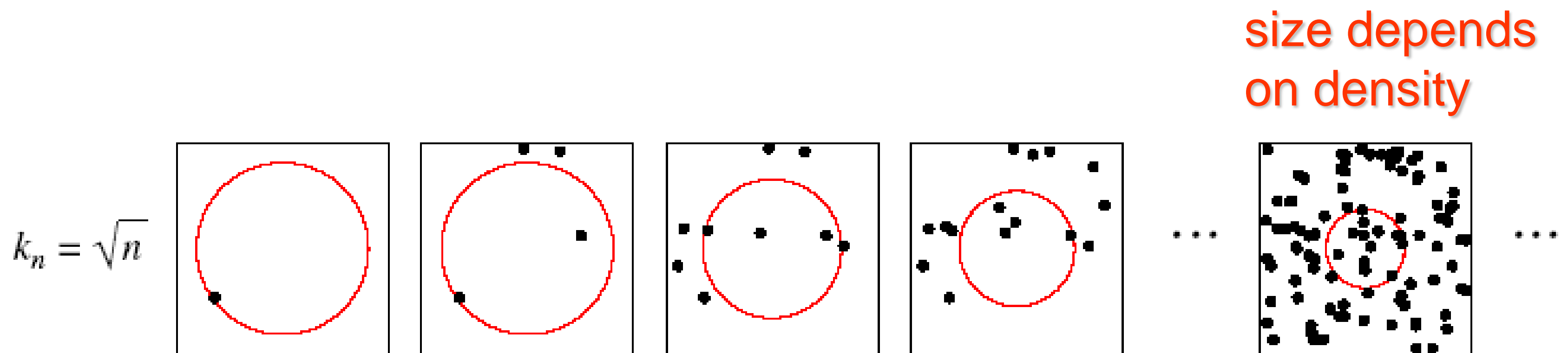
Two possibilities can occur:

- 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

If we take $p_n(\mathbf{x}) \cong \dfrac{k_n / n}{V_n}$ (31)

we want k_n to go to infinity as n goes to infinity, since this assures us that k_n/n will be a good estimate of the probability that a point will fall in the cell of volume V_n. However, we also want k_n to grow sufficiently slowly that the size of the cell needed to capture k_n training samples will shrink to zero. Thus, it is clear from Eq. 31 that the ratio k_n/n must go to zero.



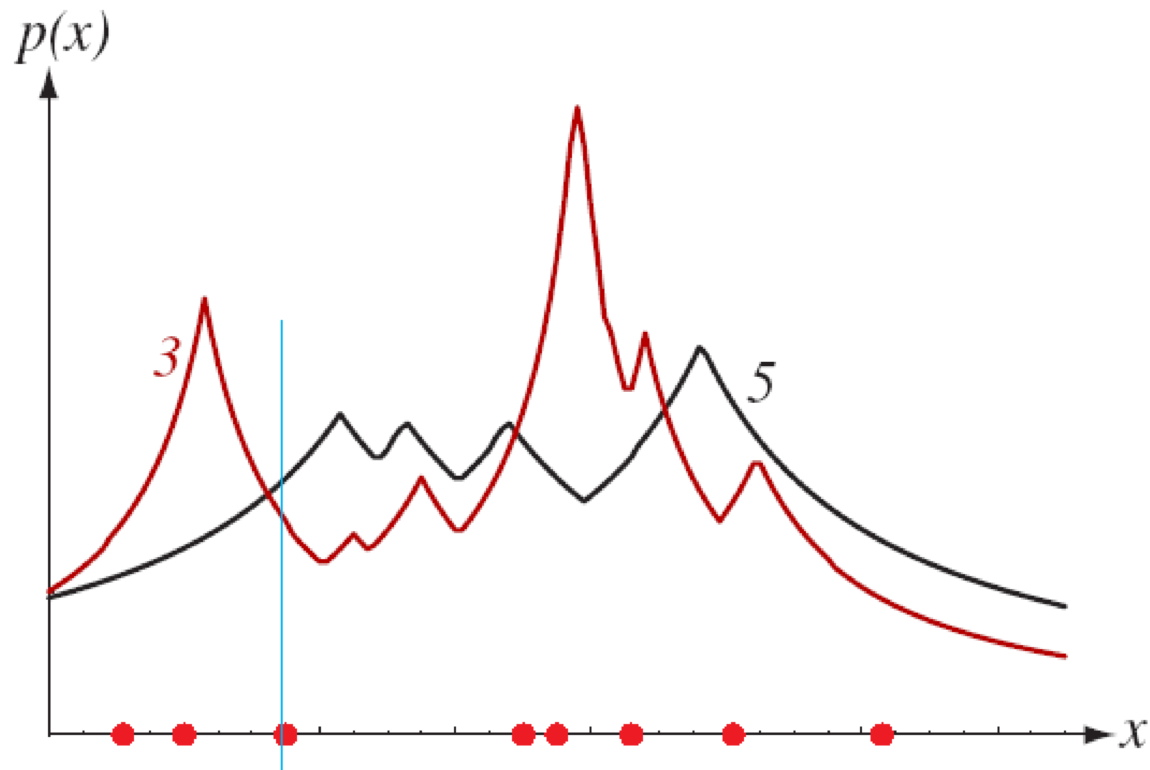


FIGURE 4.10. Eight points in one dimension and the k-nearest-neighbor density estimates, for $k = 3$ and 5. Note especially that the discontinuities in the slopes in the estimates generally lie away from the positions of the prototype points.

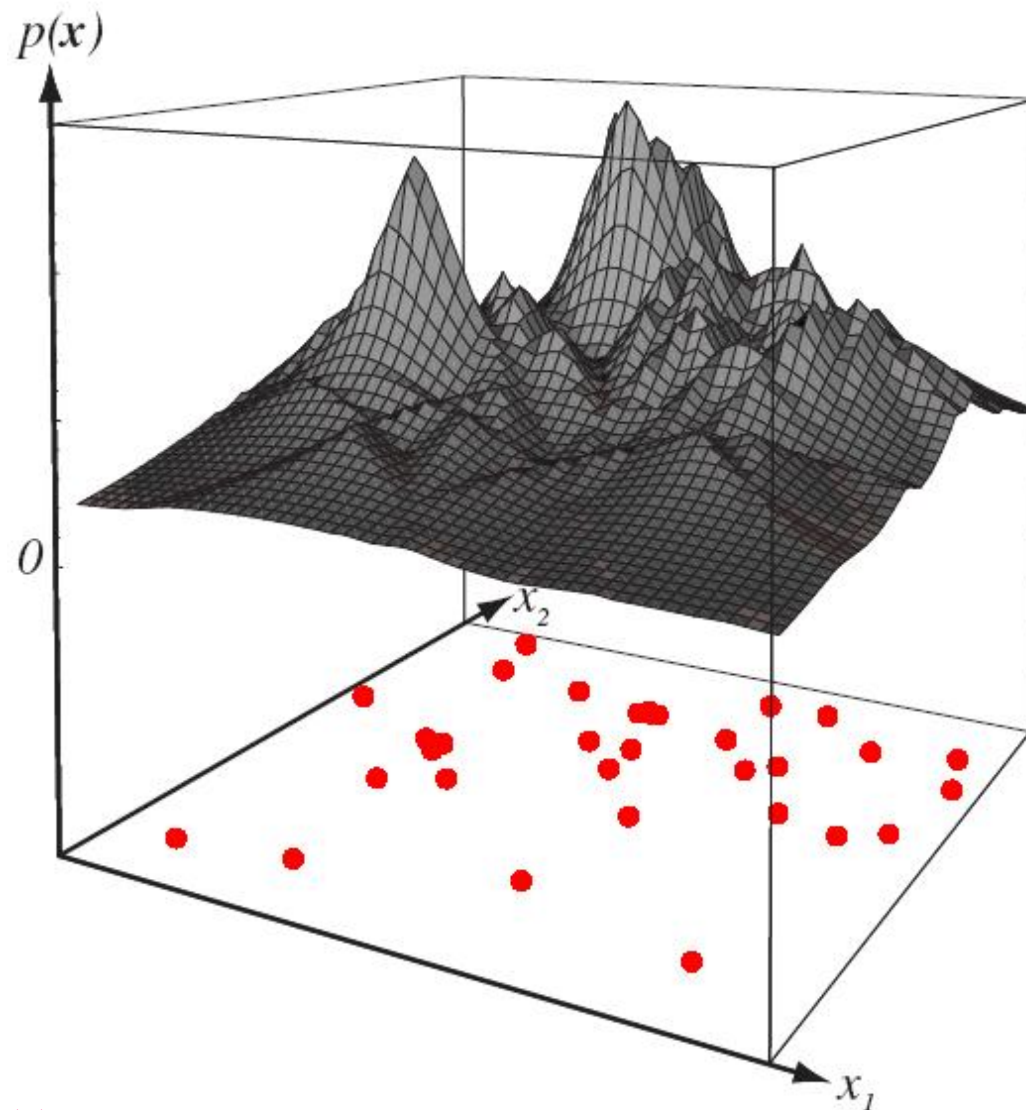
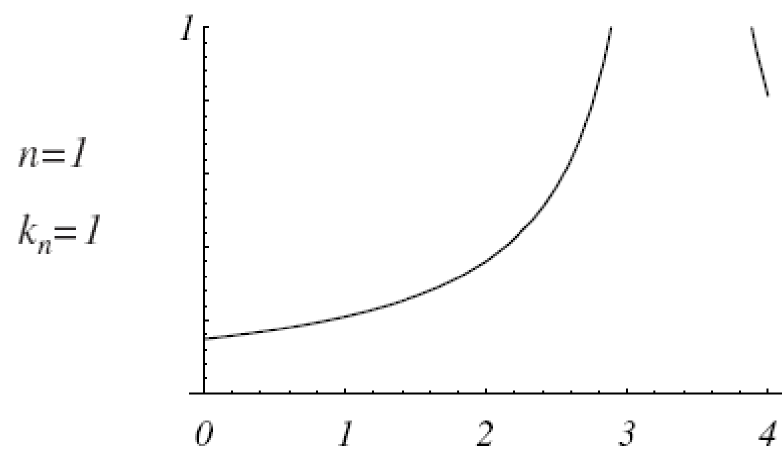
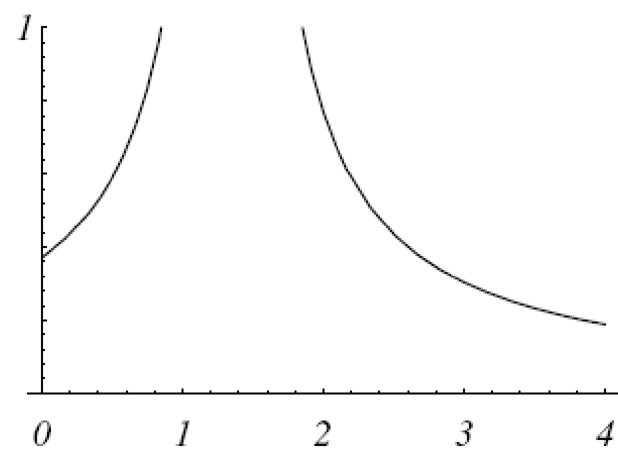


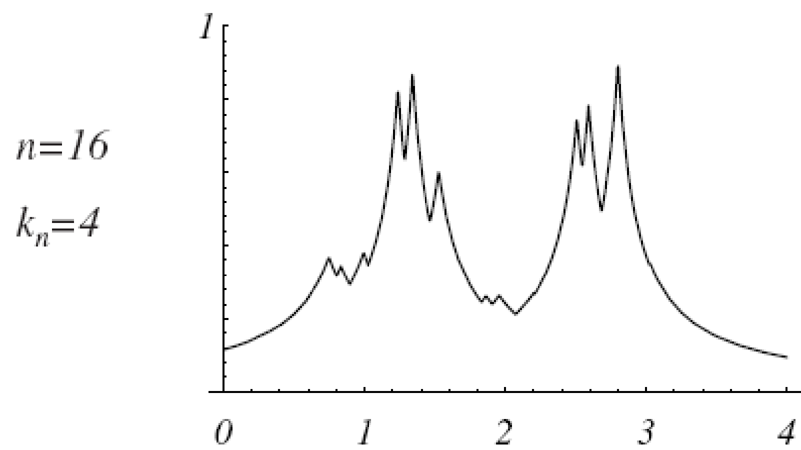
FIGURE 4.11. The *k*-nearest-neighbor estimate of a two-dimensional density for $k = 5$. Notice how such a finite *n* estimate can be quite "jagged," and notice that discontinuities in the slopes generally occur along lines away from the positions of the points themselves.



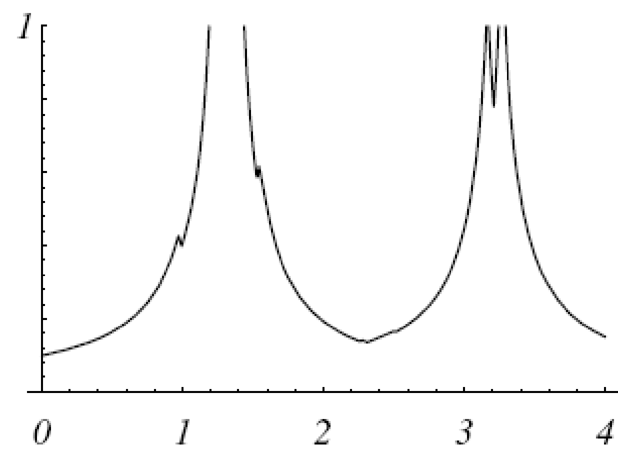
n=1
k_n=1
1
0
1
2
3
4



1
0
1
2
3
4



n=16
k_n=4
1
0
1
2
3
4



1
0
1
2
3
4

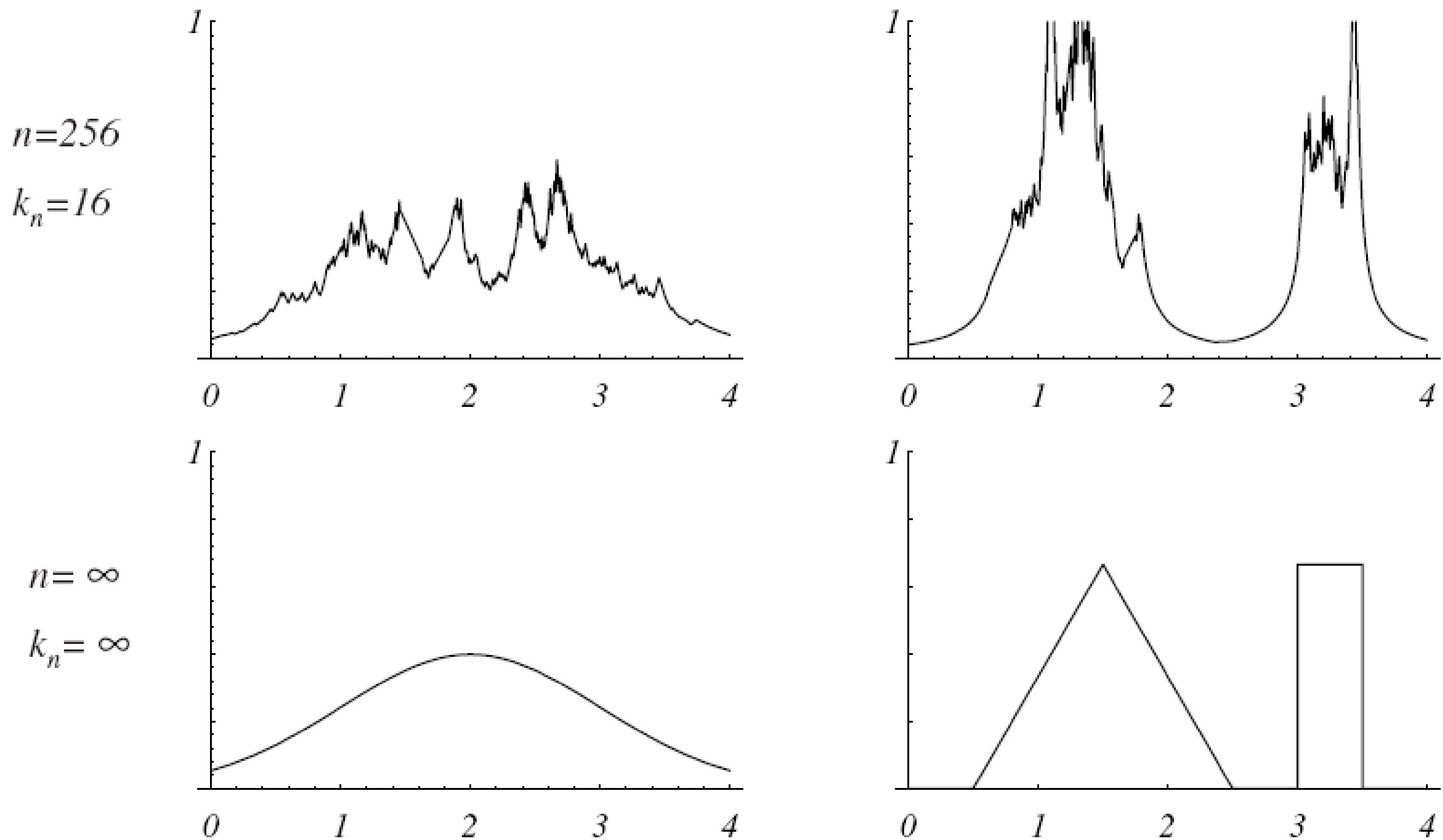
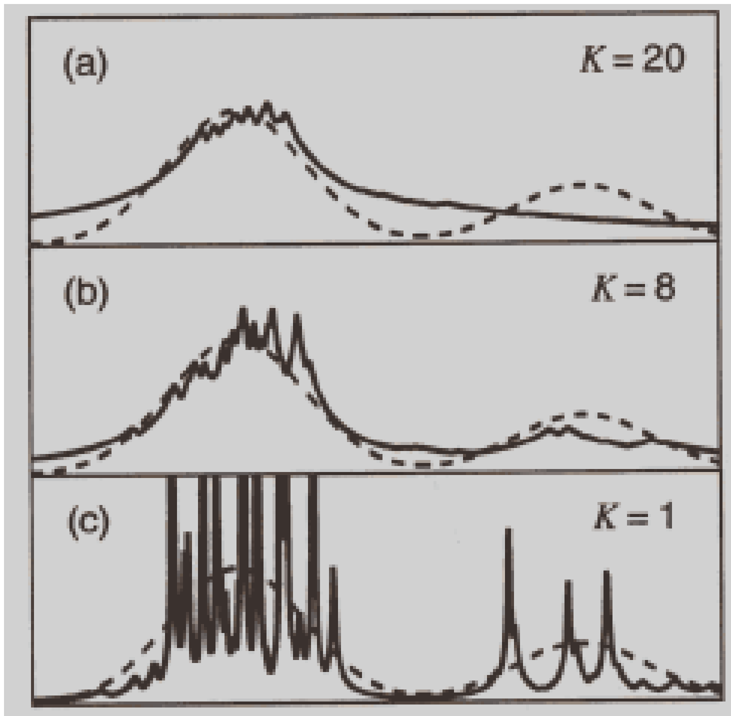
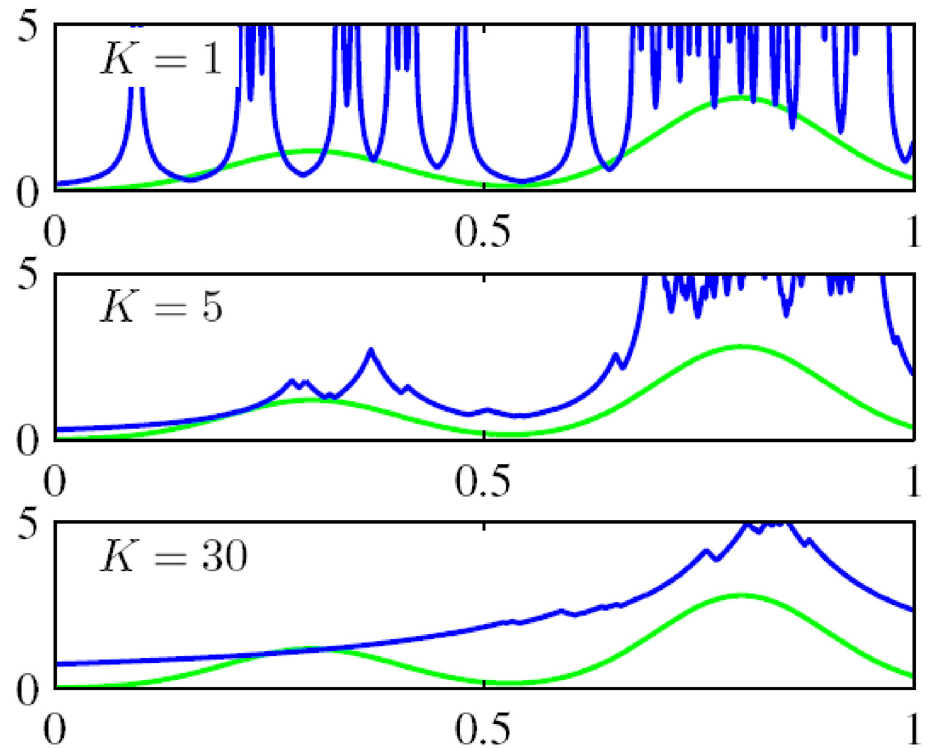


FIGURE 4.12. Several *k*-nearest-neighbor estimates of two unidimensional densities: a Gaussian and a bimodal distribution. Notice how the finite *n* estimates can be quite "spiky."





The parameter k_n acts as a smoothing parameter and needs to be optimized.

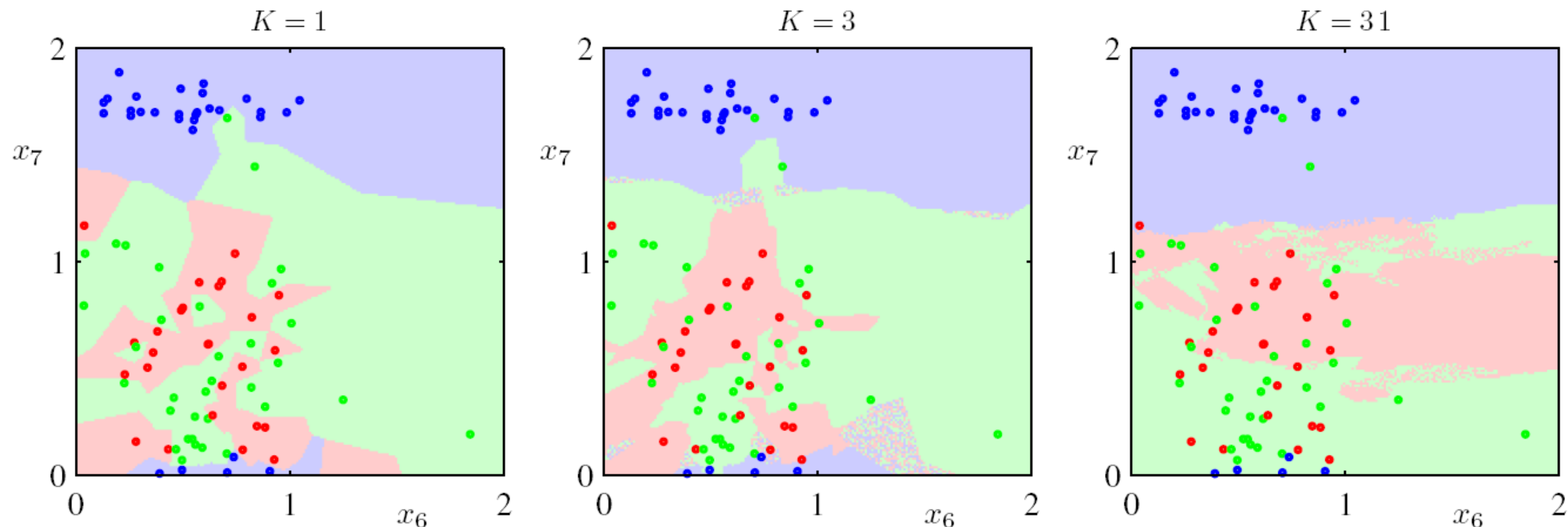
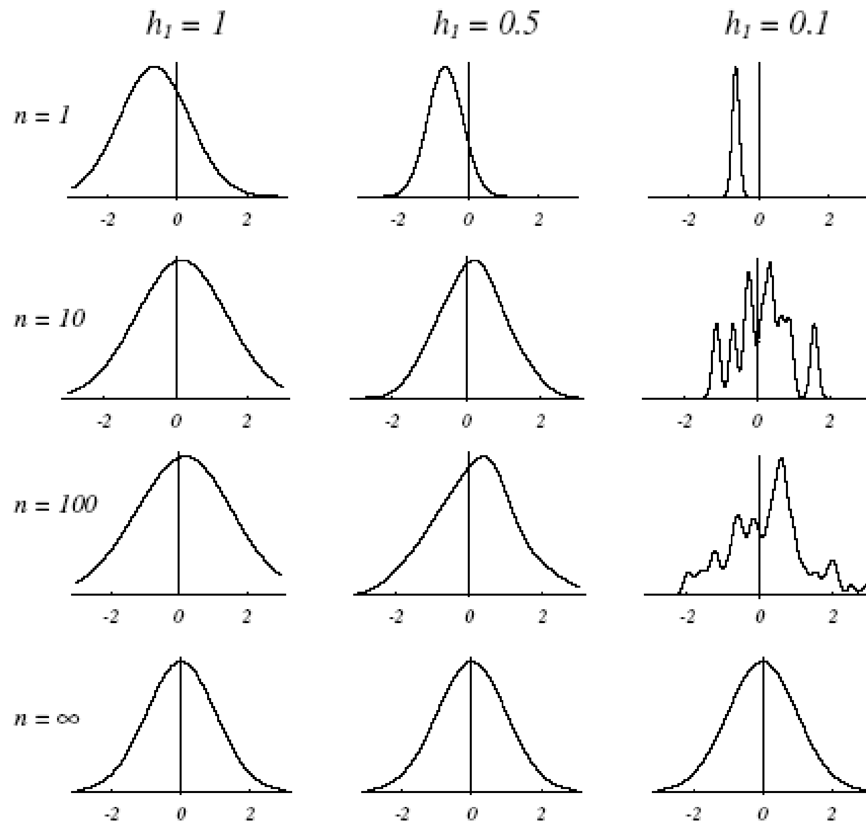


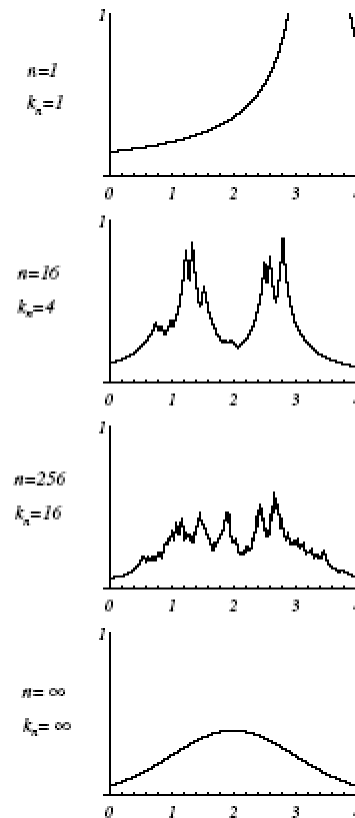
Figure 2.28 Plot of 200 data points from the oil data set showing values of x_6 plotted against x_7, where the red, green, and blue points correspond to the 'laminar', 'annular', and 'homogeneous' classes, respectively. Also shown are the classifications of the input space given by the K-nearest-neighbor algorithm for various values of K.

Parzen windows vs k_n-nearest-neighbor estimation

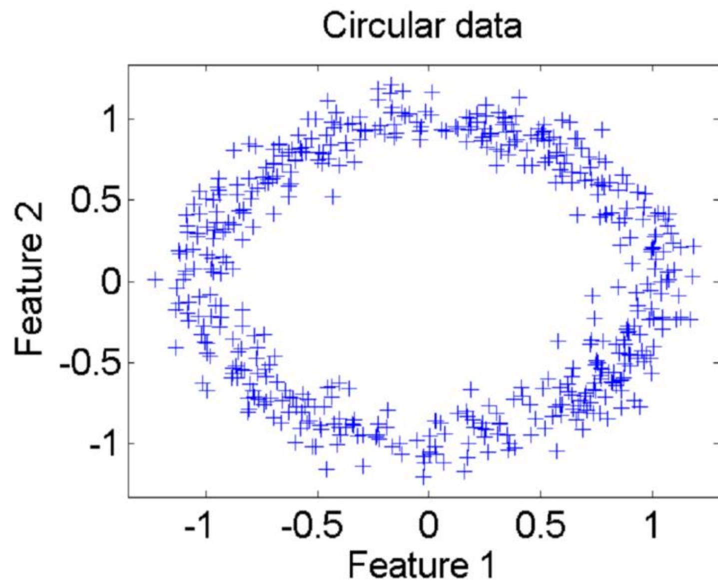
Parzen windows

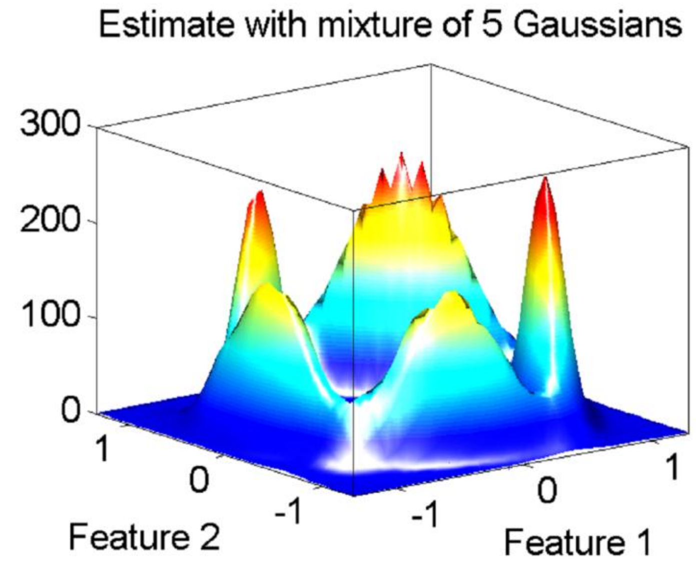


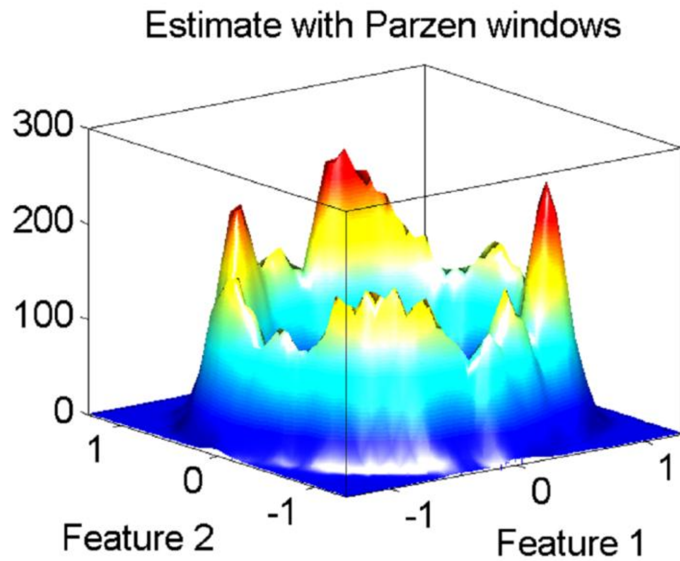
k_n-nearest-neighbor

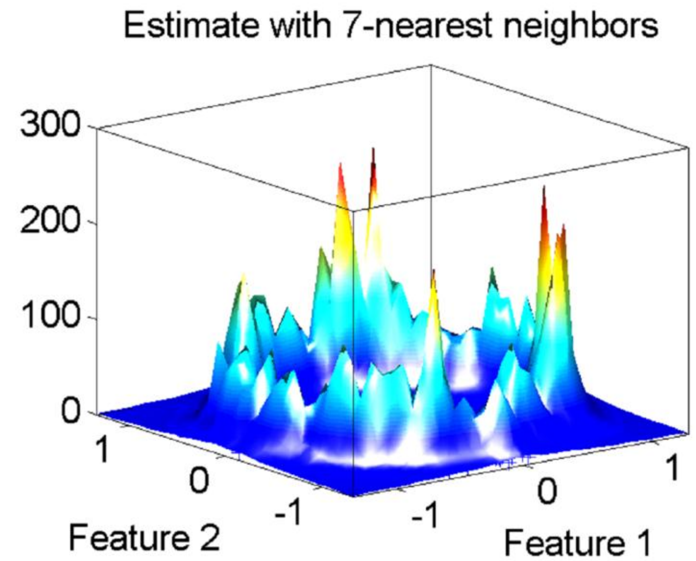


$$k_n = k_1\sqrt{n}$$

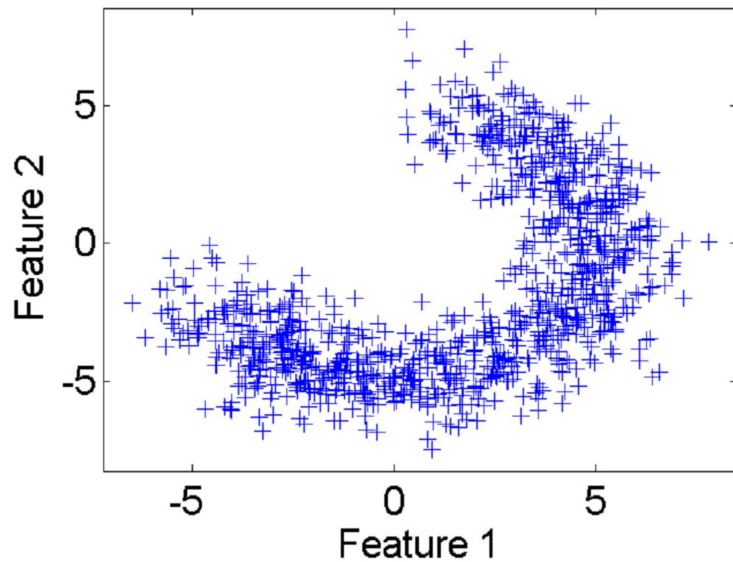


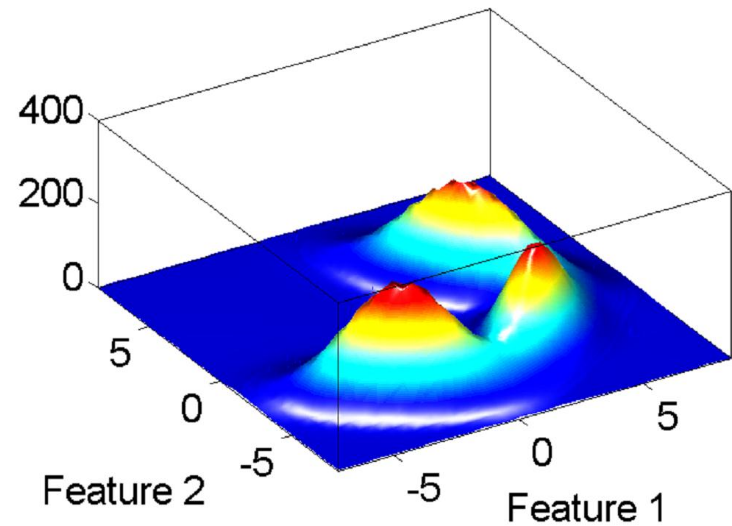


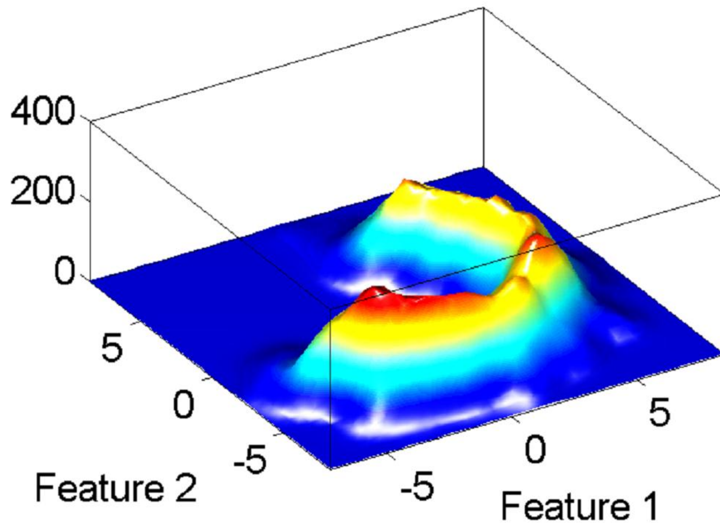


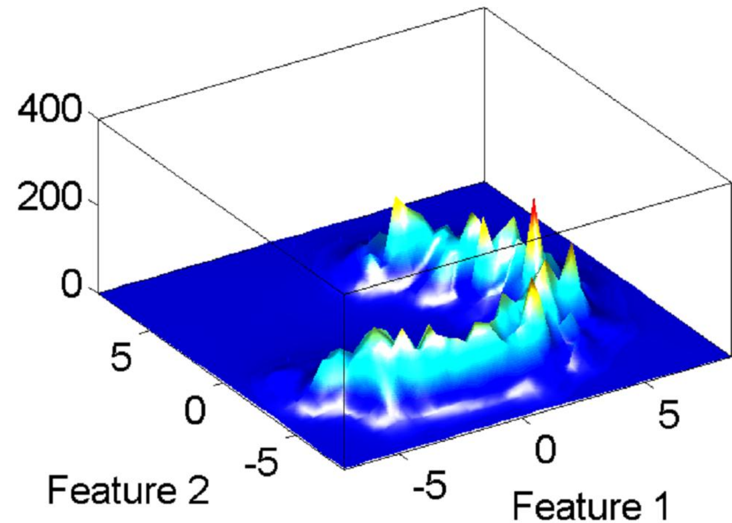


Density estimation examples for 2-D circular data.









Density estimation examples for 2-D banana shaped data.

- Estimation of *a*-posteriori probabilities

Goal: estimate $P(\omega_i|\mathbf{x})$ from a set of n labeled samples

- Let's place a cell of volume V around $\mathbf{x}$ and capture k samples
- k_i samples amongst k turned out to be labeled ω_i then:

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

An estimate for $p_n(\omega_i/\mathbf{x})$ is:

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

- k_i/k is the fraction of the samples within the cell that are labeled ω_i

- For minimum error rate, the most frequently represented category within the cell is selected

- If k is large and the cell sufficiently small, the performance will approach the best possible

x_2

1.3

0.9

0.6

0.3

0

0 0.4 0.7 1 1.4 x_1

$k=5$

Black ω_1

Blue ω_2

Full line circle

$$\rho = \sqrt{0.1^2 + 0.1^2} = 0.1\sqrt{2}$$

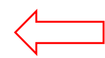
Dash line circle

$$\sqrt{0.2^2 + 0.2^2} = 0.2\sqrt{2} = 2\rho$$

$N_1=59$ $N_2=61$

$$V_1 = 4\pi\rho^2, V_2 = \pi\rho^2,$$

$$\frac{V_2}{V_1} = \frac{\pi\rho^2}{4\pi\rho^2} = 0.25$$

✓ ω_2  $0.25 < \frac{59}{61}$

- The nearest –neighbor rule

  – Let $D_n = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ be a set of n labeled prototypes

  – Let $\mathbf{x}' \in D_n$ be the closest prototype to a test point $\mathbf{x}$ then the nearest-neighbor rule for classifying $\mathbf{x}$ is to assign it the label associated with $\mathbf{x}'$

  – The nearest-neighbor rule leads to an error rate greater than the minimum possible; the Bayes rate

  – If the number of prototype is large (unlimited), the error rate of the nearest-neighbor classifier is never worse than twice the Bayes rate (it can be demonstrated!)

  – If $n \to \infty$, it is always possible to find $\mathbf{x}'$ sufficiently close so that: $P(\omega_i \mid \mathbf{x}') \cong P(\omega_i \mid \mathbf{x})$

If we define $\omega_m(\mathbf{x})$ by $P(\omega_m|\mathbf{x}) = \max_i P(\omega_i|\mathbf{x})$,

then the Bayes decision rule always selects ω_m.

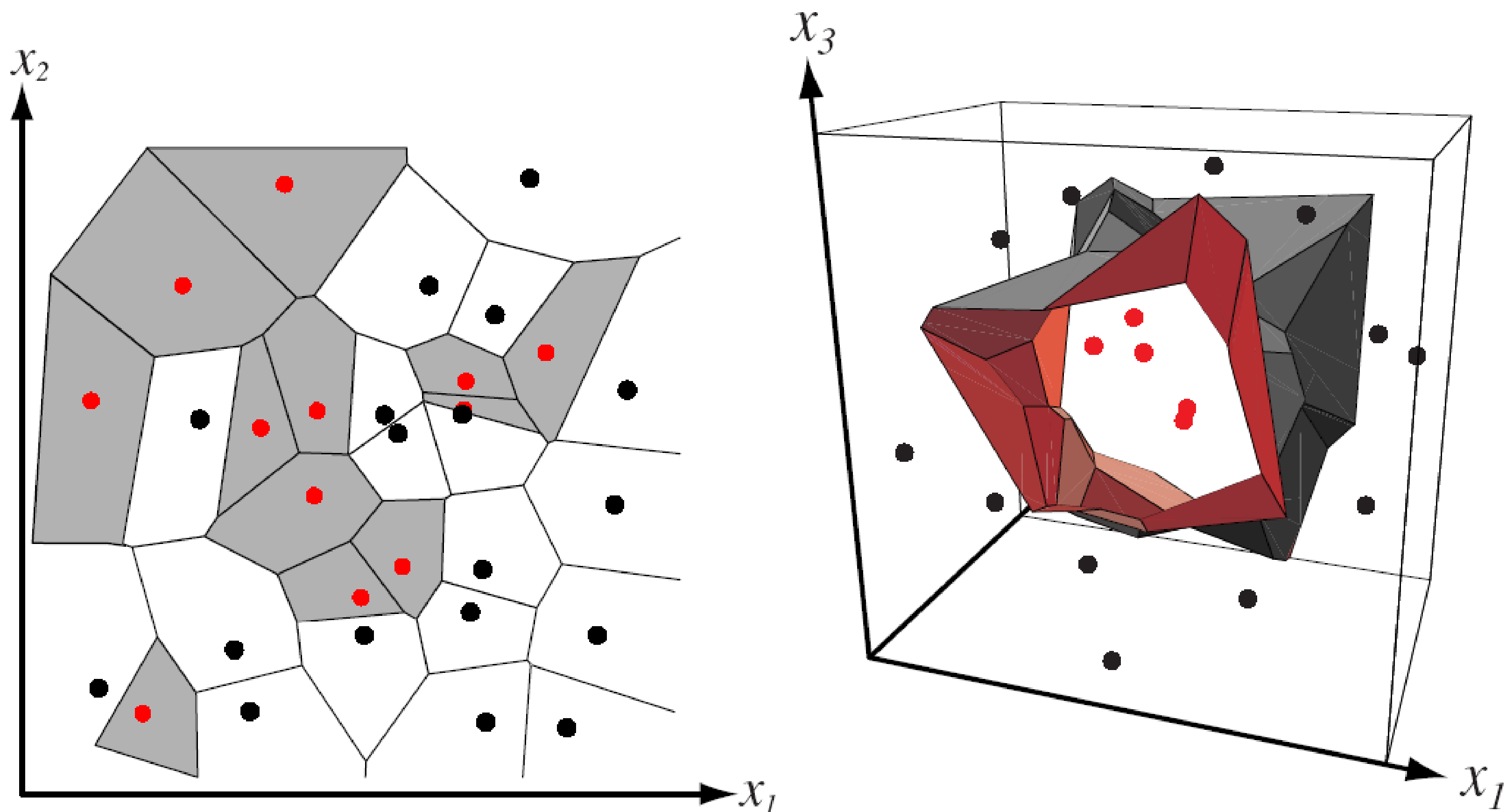


FIGURE 4.13. In two dimensions, the nearest-neighbor algorithm leads to a partitioning of the input space into Voronoi cells, each labeled by the category of the training point it contains. In three dimensions, the cells are three-dimensional, and the decision boundary resembles the surface of a crystal.

- If $P(\omega_m \mid \mathbf{x}) \cong 1$, then the nearest neighbor selection is almost always the same as the Bayes selection
- When $P(\omega_m|\mathbf{x})$ is close to $1/c$, so that all classes are essentially equally likely, the selections made by the nearest-neighbor rule and the Bayes decision rule are rarely the same, but the probability of error is approximately $1 - 1/c$ for both.
- The unconditional average probability of error will then be found by averaging $P(e|\mathbf{x})$ over all $\mathbf{x}$:

$$P(e) = \int P(e|\mathbf{x})p(\mathbf{x})\ d\mathbf{x}.$$

We should recall that the Bayes decision rule minimizes P(e) by minimizing P($e|\mathbf{x}$) for every $\mathbf{x}$. If we let $P^*(e|\mathbf{x})$ be the minimum possible value of P($e|\mathbf{x}$), and P^* be the minimum possible value of P(e), then

$$P^*(e|\mathbf{x}) = 1 - P(\omega_m|\mathbf{x})$$

and

$$P^* = \int P^*(e|\mathbf{x})p(\mathbf{x})\ d\mathbf{x}.$$

Convergence of the Nearest Neighbor

- If $P_n(e)$ is the n-sample error rate, and if

$$P = \lim_{n \to \infty} P_n(e),$$

then we want to show that

$$P^* \leq P \leq P^*\left(2 - \frac{c}{c-1}P^*\right).$$

$$P(e|\mathbf{x}) = \int P(e|\mathbf{x}, \mathbf{x}')p(\mathbf{x}'|\mathbf{x})\ d\mathbf{x}'.$$

$\mathbf{x}'$ is the nearest neighbor of $\mathbf{x}$

$p(\mathbf{x}'|\mathbf{x})$ difficult to obtain
$p(\mathbf{x}'|\mathbf{x}) \to \delta$ if $n \to \infty$

Error Rate for the Nearest-Neighbor Rule

Calculation of the conditional probability of error $P_n(e|\mathbf{x}, \mathbf{x}')$.

When we say that we have n independently drawn labelled samples, we are talking about n pairs of random variables $(\mathbf{x}_1, \theta_1), (\mathbf{x}_2, \theta_2), \ldots, (\mathbf{x}_n, \theta_n)$, where θ_j may be any of the c states of nature $\omega_1, \ldots, \omega_c$. We assume that these pairs were generated by selecting a state of nature ω_j for θ_j with probability $P(\omega_j)$ and then selecting an $\mathbf{x}_j$ according to the probability law $p(\mathbf{x}|\omega_j)$, with each pair being selected independently.

Since the state of nature when $\mathbf{x}_j$ was drawn is independent of the state of nature when $\mathbf{x}$ is drawn, we have

$$P(\theta, \theta'_j | \mathbf{x}, \mathbf{x}'_j) = P(\theta|\mathbf{x})P(\theta'_j|\mathbf{x}'_j).$$

If we use the nearest-neighbor decision rule, we commit an error whenever $\theta \neq \theta'_j$.

$$\begin{aligned} P_n(e|\mathbf{x}, \mathbf{x}'_j) &= 1 - \sum_{i=1}^{c} P(\theta = \omega_i, \theta' = \omega_i | \mathbf{x}, \mathbf{x}'_j) \\ &= 1 - \sum_{i=1}^{c} P(\omega_i|\mathbf{x})P(\omega_i|\mathbf{x}'_j). \end{aligned}$$

We had $$P(e|\mathbf{x}) = \int P(e|\mathbf{x}, \mathbf{x}')p(\mathbf{x}'|\mathbf{x})\, d\mathbf{x}'.$$

As n goes to infinity and p($\mathbf{x}'|\mathbf{x}$) approaches a delta function.

$$\lim_{n\to\infty} P_n(e|\mathbf{x}) = \int \left[1 - \sum_{i=1}^{c} P(\omega_i|\mathbf{x})P(\omega_i|\mathbf{x}')\right]\delta(\mathbf{x}' - \mathbf{x})\ d\mathbf{x}'$$

$$= 1 - \sum_{i=1}^{c} P^2(\omega_i|\mathbf{x}).$$

Therefore, provided we can exchange some limits and integrals, the asymptotic nearest neighbor error rate is given by

$$P = \lim_{n\to\infty} P_n(e)$$

$$= \lim_{n\to\infty} \int P_n(e|\mathbf{x})p(\mathbf{x})\ d\mathbf{x}$$

$$= \int \left[1 - \sum_{i=1}^{c} P^2(\omega_i|\mathbf{x})\right]p(\mathbf{x})\ d\mathbf{x}.$$

Error Bounds

It can be shown

$$P^* \leq P \leq P^*\left(2 - \frac{c}{c-1}P^*\right).$$

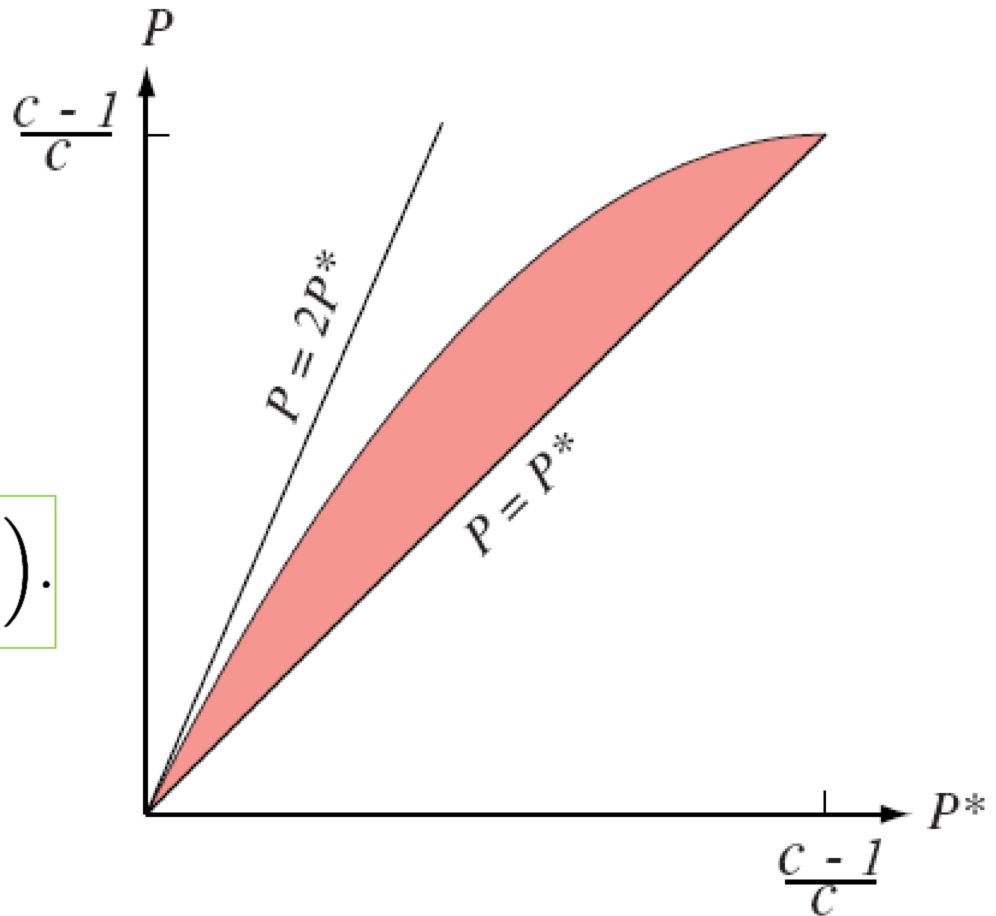


FIGURE 4.14. Bounds on the nearest-neighbor error rate P in a *c*-category problem given infinite training data, where P^* is the Bayes error. At low error rates, the nearest-neighbor error rate is bounded above by twice the Bayes rate.

- **The k–nearest-neighbor rule**

  - Goal: Classify **x** by assigning it the label most frequently represented among the *k* nearest samples and use a voting scheme
  - The single-nearest-neighbor rule selects ω_m with probability $P(\omega_m|\mathbf{x})$. The *k*-nearest neighbor rule selects ω_m if a majority of the *k* nearest neighbors are labeled ω_m, an event of probability

$$\sum_{i=(k+1)/2}^{k} \binom{k}{i} P(\omega_m|\mathbf{x})^i [1 - P(\omega_m|\mathbf{x})]^{k-i}.$$

In general, the larger the value of *k*, the greater the probability that ω_m will be selected.

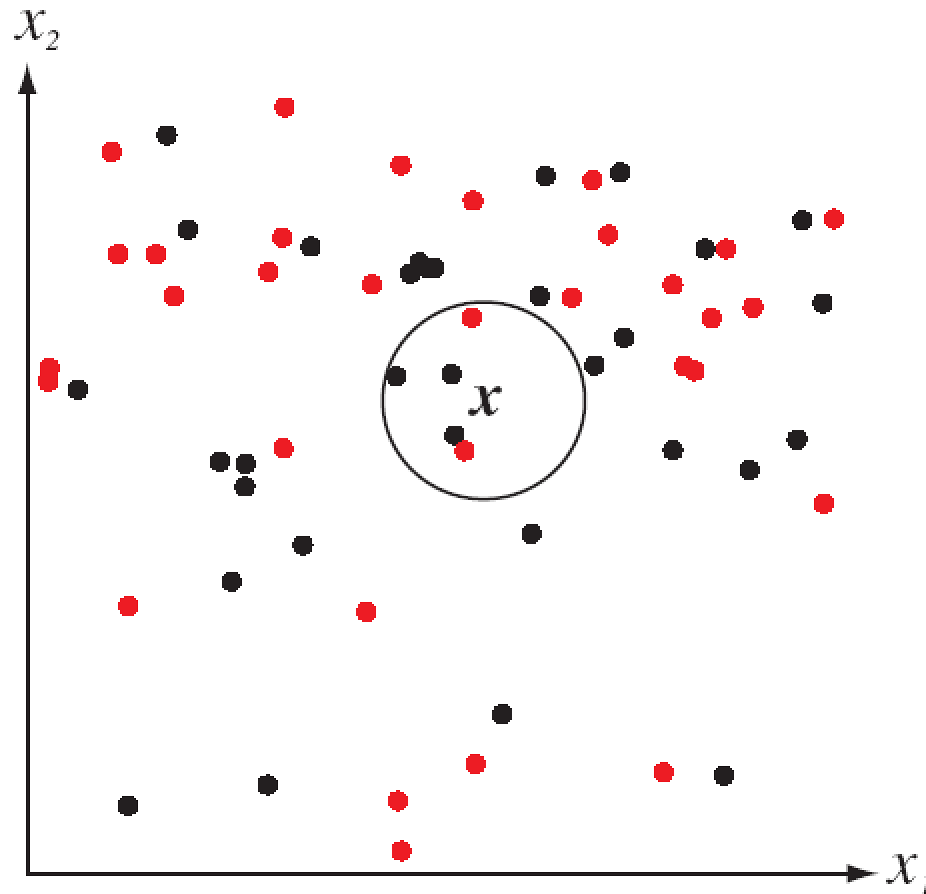


FIGURE 4.15. The k-nearest-neighbor query starts at the test point $\mathbf{x}$ and grows a spherical region until it encloses k training samples, and it labels the test point by a majority vote of these samples. In this $k = 5$ case, the test point $\mathbf{x}$ would be labeled the category of the black points.

$$\sum_{i=0}^{(k-1)/2} \binom{k}{i} \left[(P^*)^{i+1}(1-P^*)^{k-i} + (P^*)^{k-i}(1-P^*)^{i+1} \right]. \quad (54)$$

It can be shown that if k is odd, the large-sample two-class error rate for the k-nearest-neighbor rule is bounded above by the function $C_k(P^*)$, where $C_k(P^*)$ is defined to be the smallest concave function of P greater than the above expression.

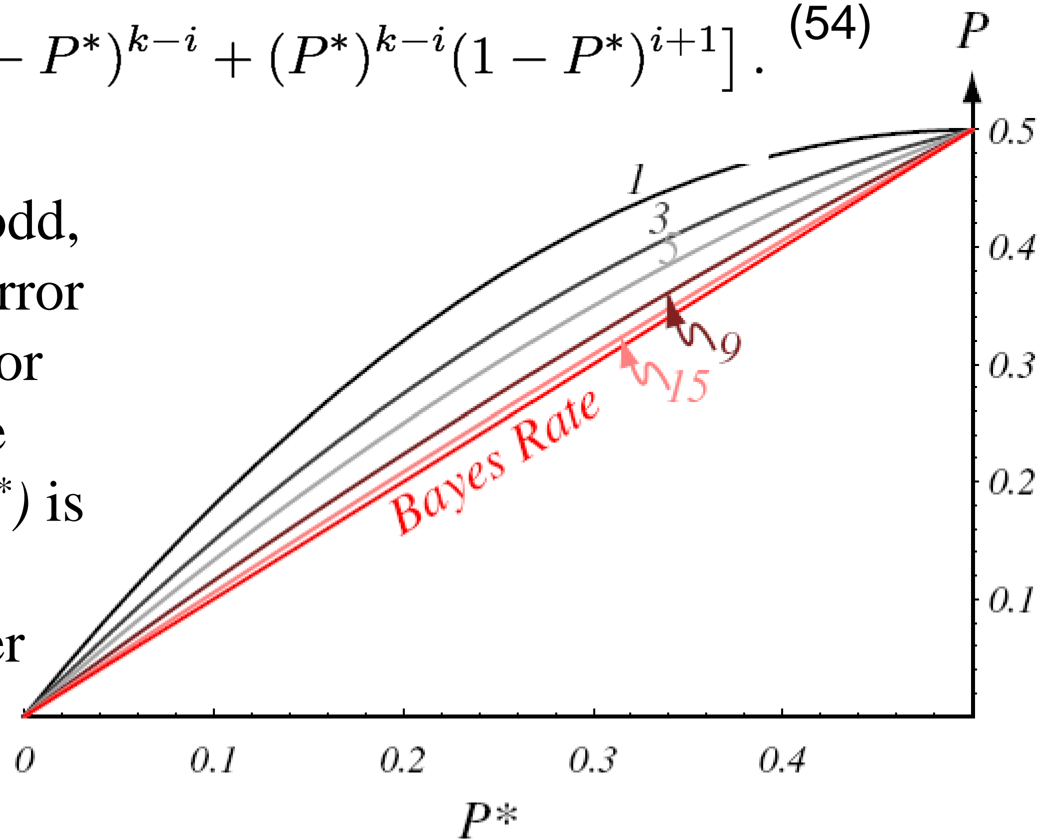


FIGURE 4.16. The error rate for the k-nearest-neighbor rule for a two-category problem is bounded by $C_k(P^*)$ in Eq. 54. Each curve is labeled by k; when $k = \infty$, the estimated probabilities match the true probabilities and thus the error rate is equal to the Bayes rate, that is, $P = P^*$.

Example:

$k = 3$ (odd value) and $\mathbf{x} = (0.10, 0.25)^t$

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

Closest vectors to **x** with their labels are:

$\{(0.10, 0.28, \omega_2); (0.12, 0.20, \omega_2); (0.15, 0.35, \omega_1)\}$

One voting scheme assigns the label ω_2 to **x** since ω_2 is the most frequently represented

Example: Digit Recognition

Yann LeCunn – 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

Example: Face Recognition

- In appearance-based face recognition, each person is represented by a few typical faces under different lighting and expression conditions.
- The recognition is then to decide the identify of a person of a given image.
- The nearest neighbor classifier could be used.

Example: Face Recognition (cont'd)

- ORL dataset
  - Consists of 40 subjects with 10 images each
  - Images were taken at different times with different lighting conditions
  - Limited side movement and tilt, no restriction on facial expression

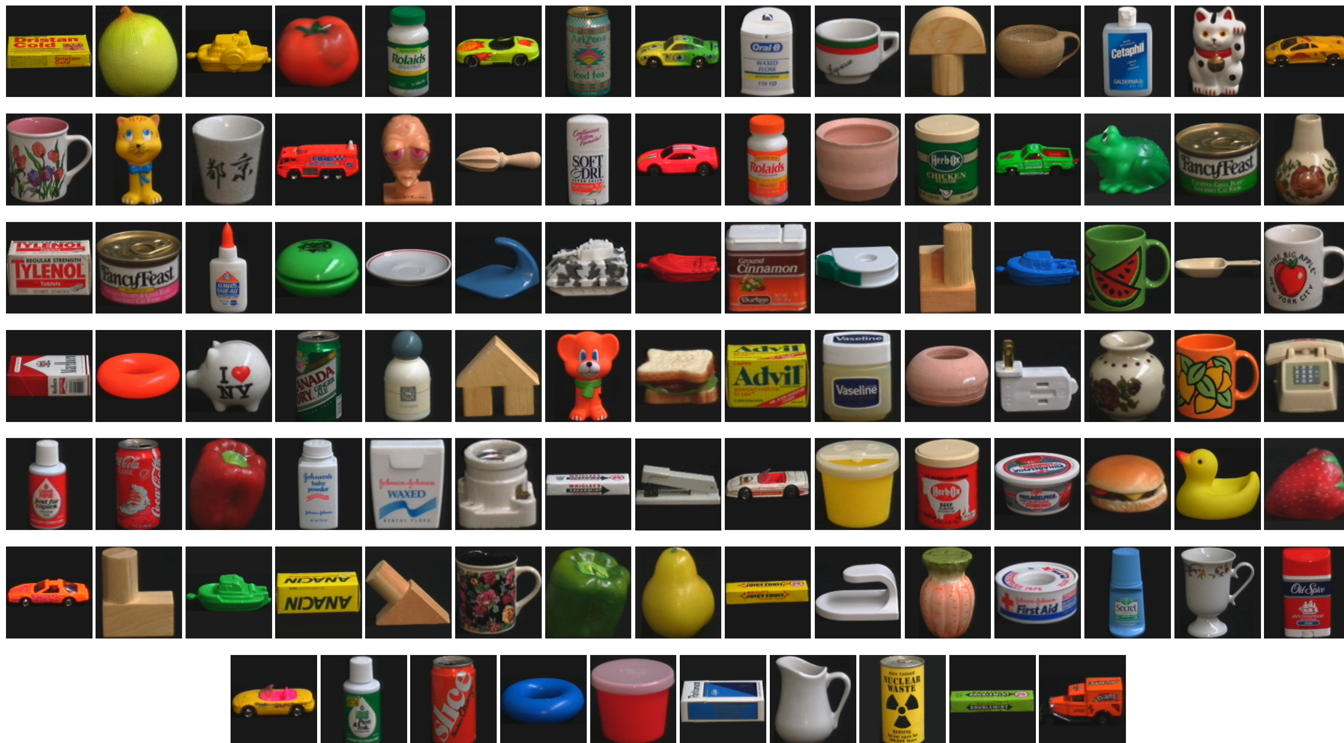
Example: Face Recognition (cont'd)

- The following table shows the result of 100 trials.

# of training faces	Average error rate	Best error rate	Worst error rate
1	30.15%	25.28%	36.67%
2	16.01%	7.81%	22.50%
3	8.04%	2.86%	17.14%
4	3.88%	0.42%	9.58%
5	2.06%	0.00%	5.50%

3D Object Recognition

- COIL Dataset



3D Object Recognition (cont'd)

Training/test views

Methods	36/36	18/54	8/64	4/68
Spectral histogram	0.08%	0.67%	4.67%	10.71%
Spectral histogram without background	0.00%	0.13%	1.89%	7.96%
SNoW (Yang et al.,2000)	4.19%	7.69%	14.87%	18.54%
Linear SVM (Yang et al.,2000)	3.97%	8.70%	15.20%	21.50%
Nearest Neighbor (Yang et al.,2000)	1.50%	12.46%	20.52%	25.37%

Computational complexity (nearest-neighbor rule)

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

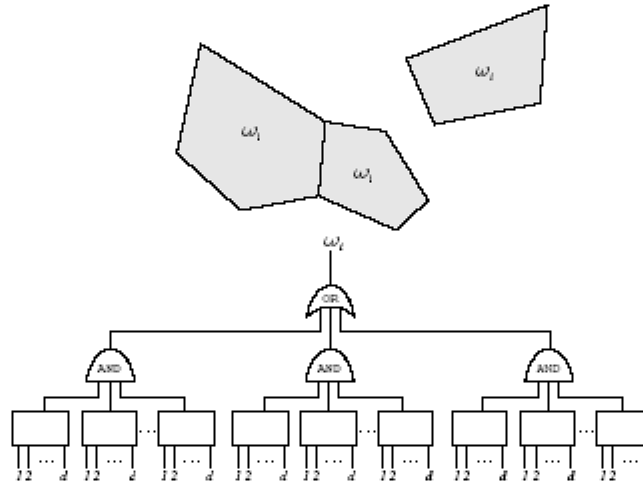


FIGURE 4.17. A parallel nearest-neighbor circuit can perform search in constant—that is, $O(1)$—time. The d-dimensional test pattern $\mathbf{x}$ is presented to each box, which calculates which side of a cell's face $\mathbf{x}$ lies on. If it is on the "close" side of every face of a cell, it lies 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:

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

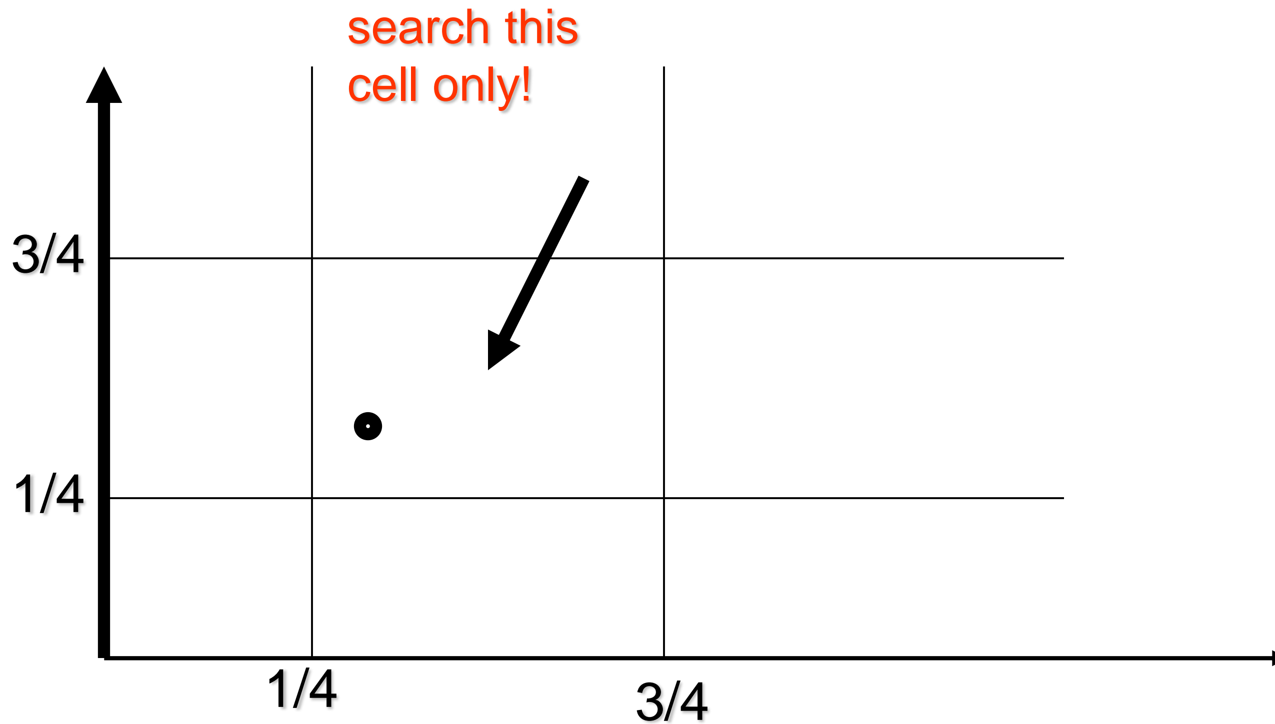
where $r<d$.

- If the partial distance is too great (i.e., greater than the distance of **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 <u>identical cells</u>.
  - For each cell the data points inside it are stored in a list.
  - Given a test point **x**, find the cell that contains it.
  - Search only the points inside that cell!
  - Does not guarantee to find the true nearest neighbor(s) !

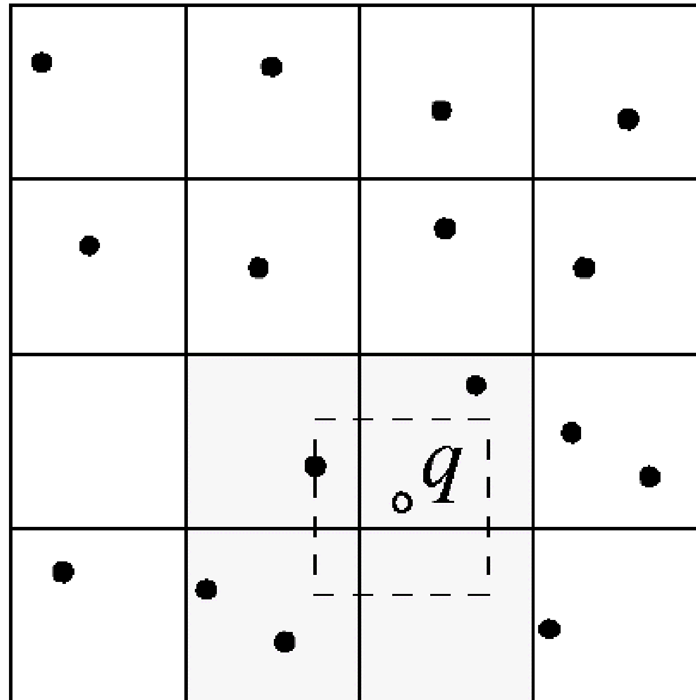
Pre-structuring: Bucketing (cont'd)



Pre-structuring: Bucketing (cont'd)

Tradeoff:

– speed vs accuracy

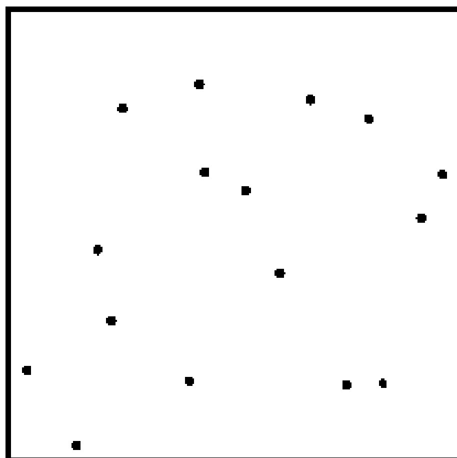


Pre-structuring: Search Trees (*k-d* tree)

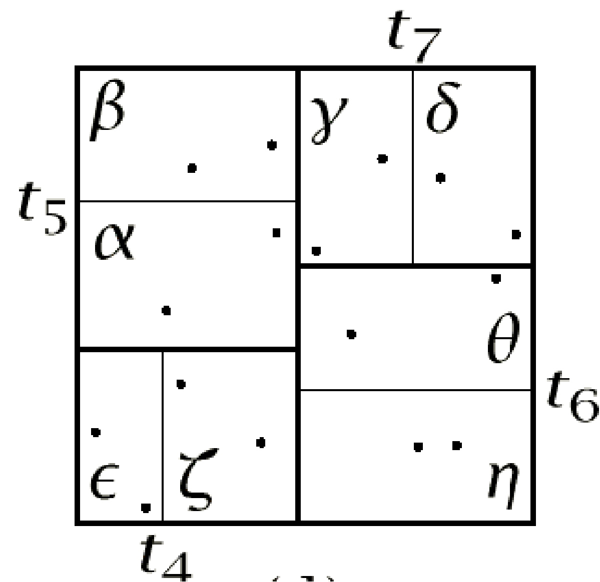
- A *k*-d tree is a data structure for storing a finite set of points from a *k*-dimensional space.
- Generalization of binary search ...
- **Goal:** hierarchically decompose space into a relatively small number of cells such that no cell contains too many points.

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

input



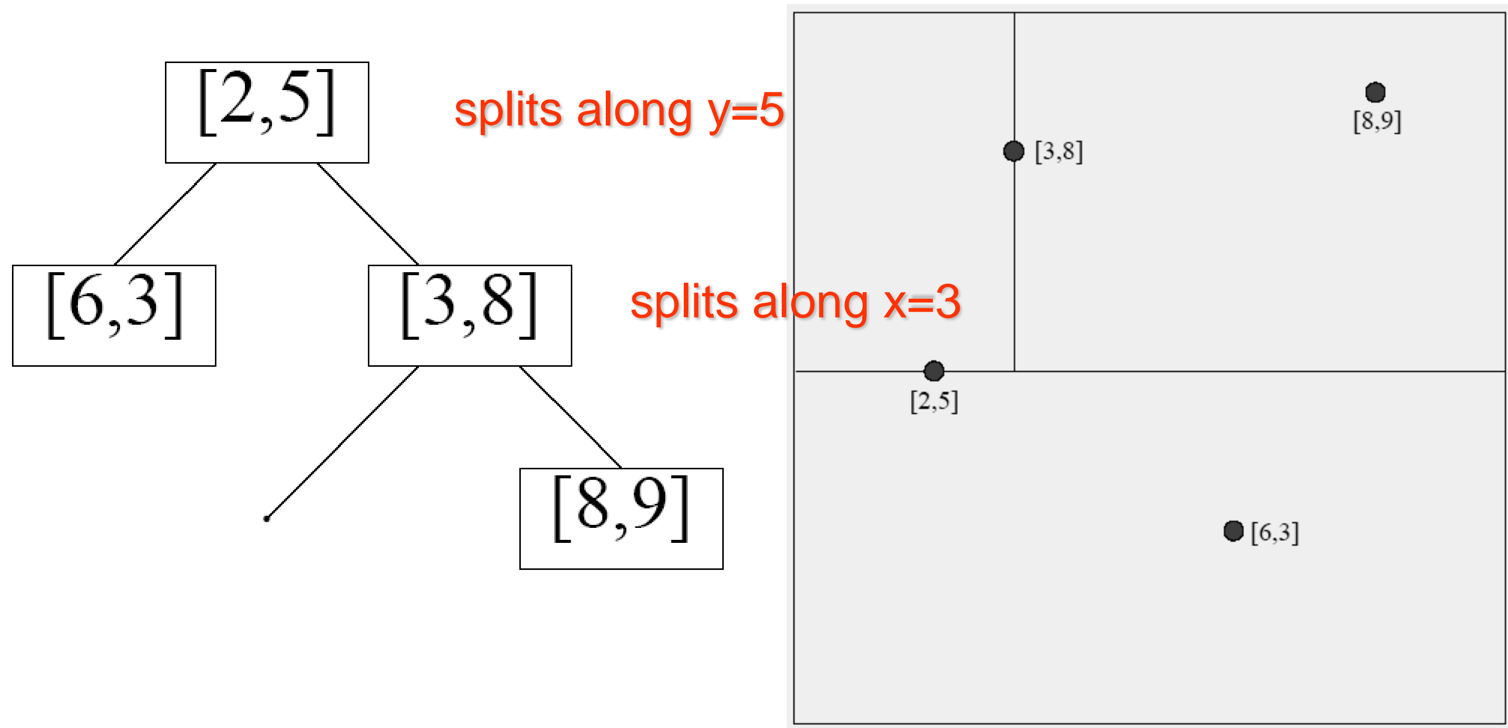
output



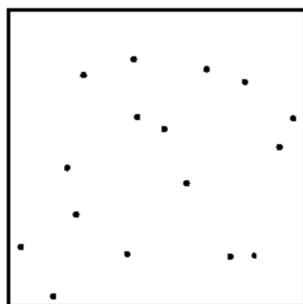
Pre-structuring: Search Trees (how to build a *k*-d tree)

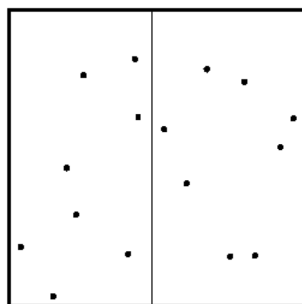
- Each internal node in a *k*-d tree is associated with a hyper-rectangle and a hyper-plane orthogonal to one of the coordinate axis.
  - The hyper-plane splits the hyper-rectangle into two parts, which are associated with the child nodes.
  - The partitioning process goes on until the number of data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

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

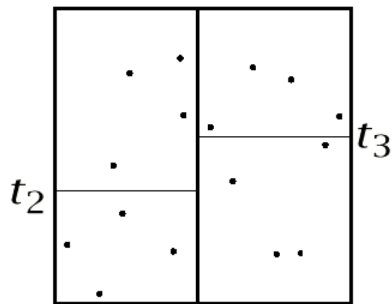


Pre-structuring: Search Trees (how to build a *k*-d tree) (cont'd)

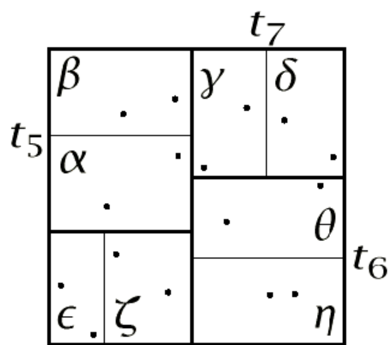

(a)



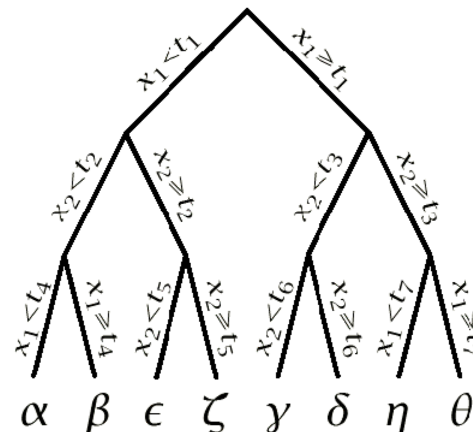
(b)



(c)



(d)



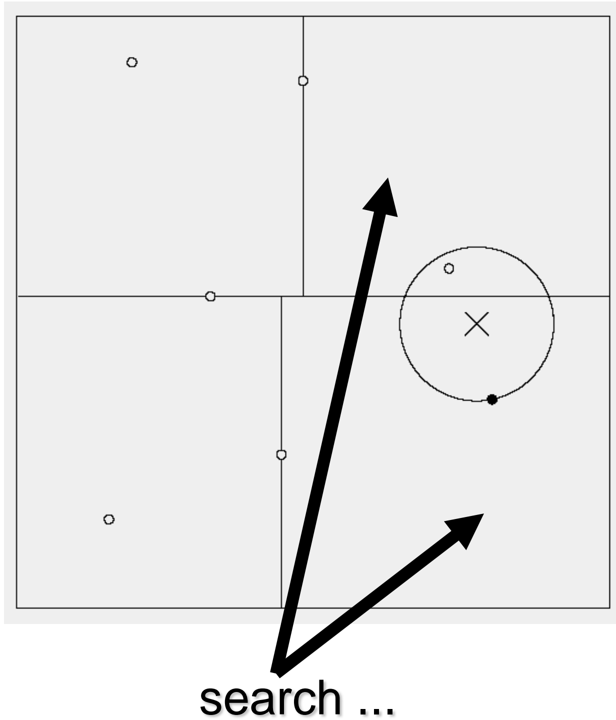
(e)

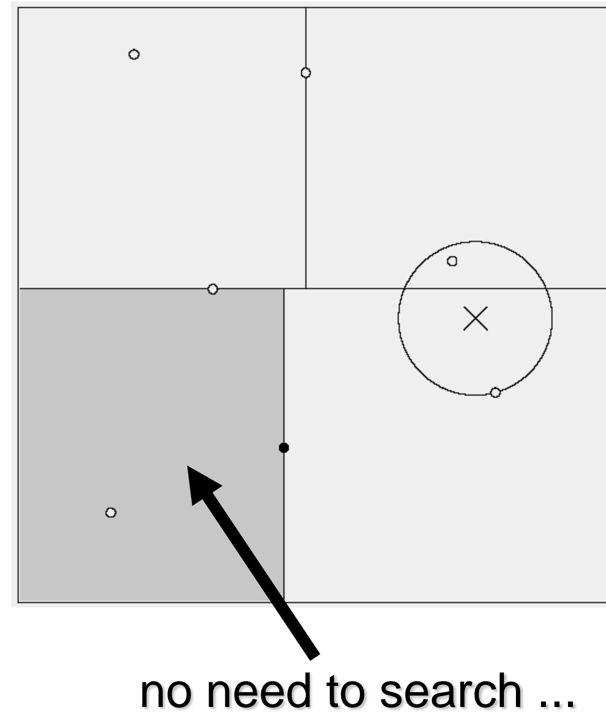
Pre-structuring: Search Trees (how to search using *k*-d trees)

- 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.

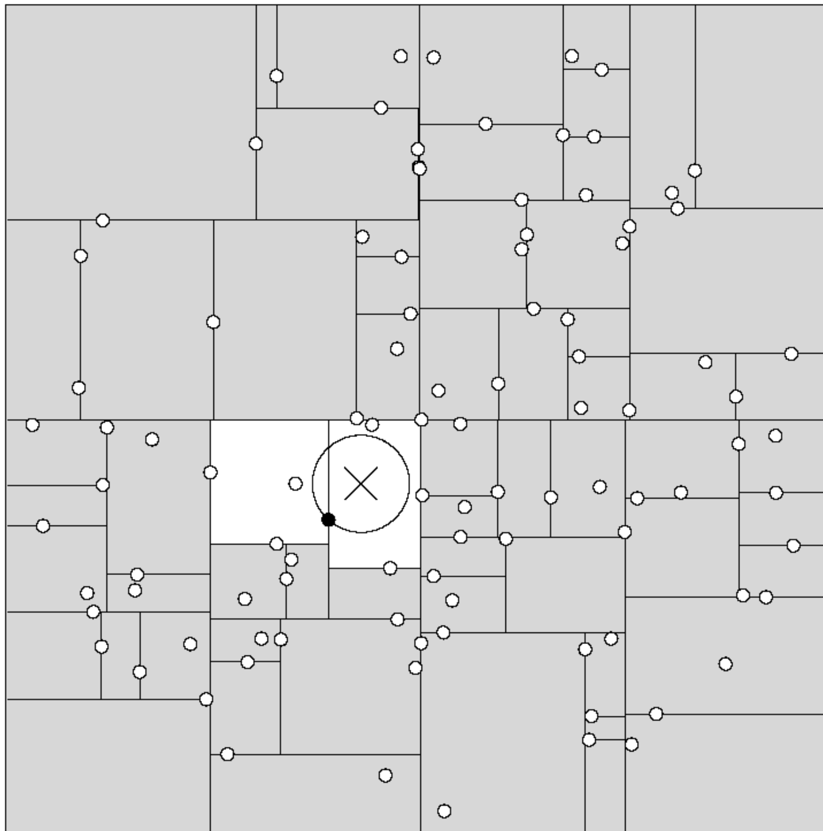
http://www-2.cs.cmu.edu/~awm/animations/kdtree/nn-vor.ppt

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



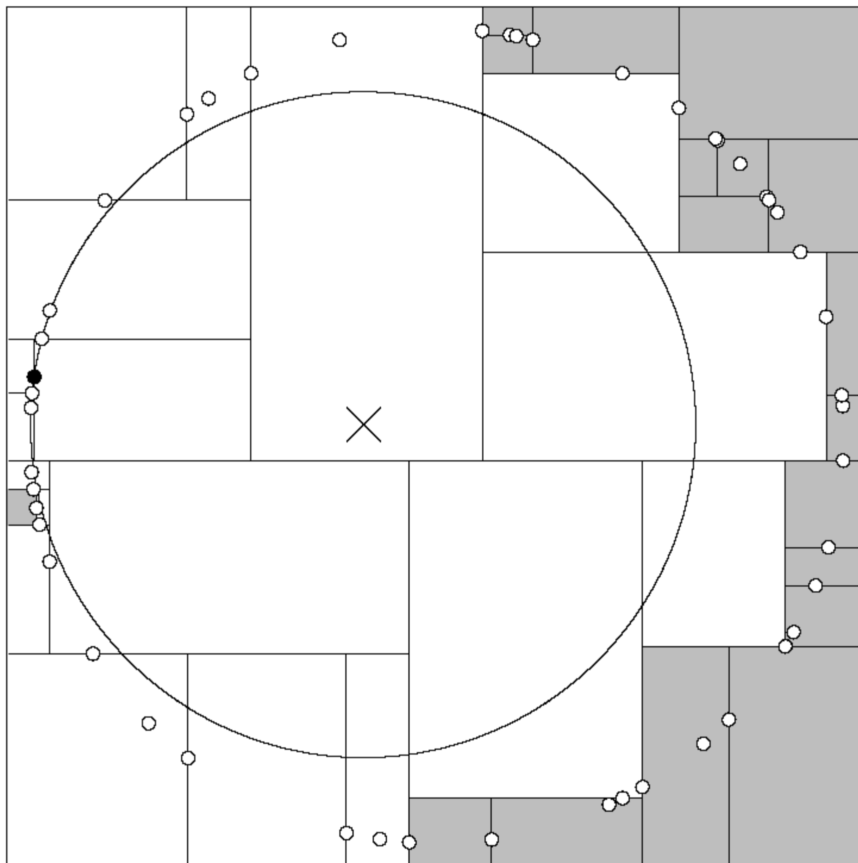


Pre-structuring: Search Trees (how to search using *k*-d trees) (cont'd)



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

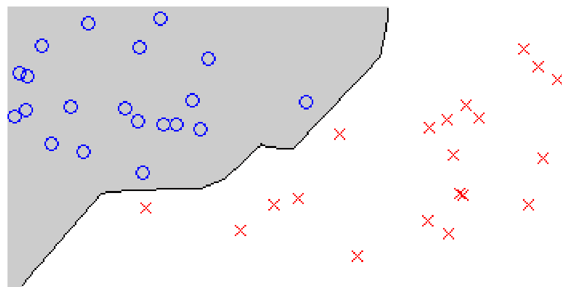
- **Goal:** reduce the number of training samples.
- Two main approaches:
  - **Condensing:** preserve decision boundaries.
  - **Pruning:** eliminate noisy examples to produce smoother boundaries and improve accuracy.

* Editing using condensing

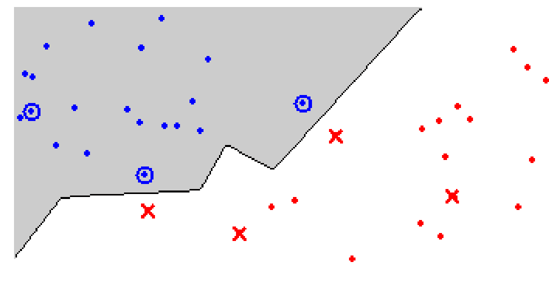
- Retain only the samples that are needed to define the decision boundary.
- <u>Decision Boundary Consistent</u> – a subset whose nearest neighbour decision boundary is close to the boundary of the entire training set.
- <u>Minimum Consistent Set</u> – the smallest subset of the training data that correctly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

* Editing using condensing (cont'd)

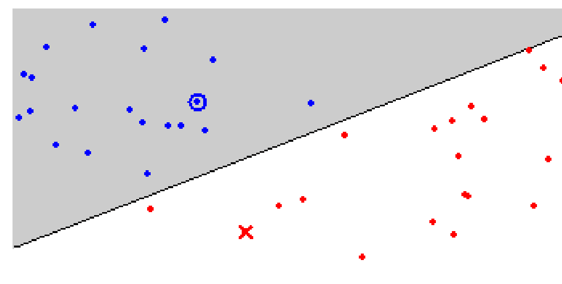
- Retain mostly points along the decision boundary.



Original data



Condensed data



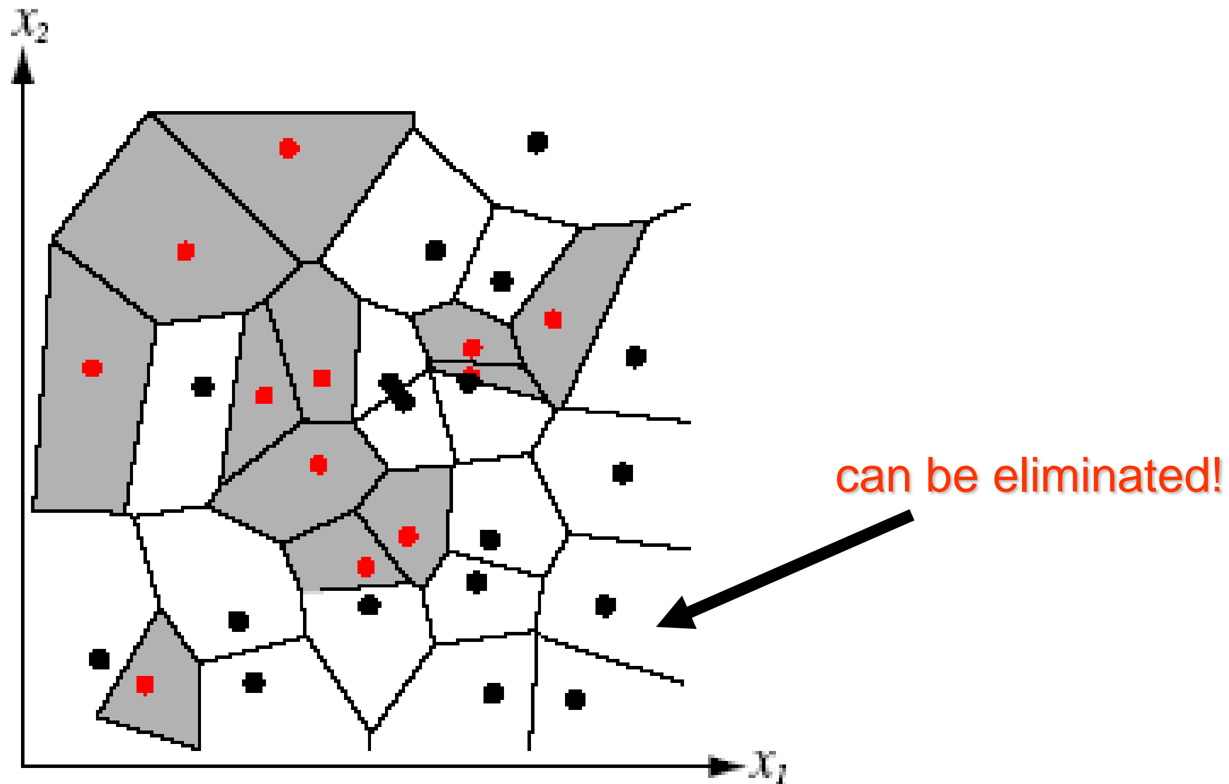
Minimum Consistent Set

* 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 <u>surrounded by samples of the same category</u>.

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

* Editing using condensing (cont’d)



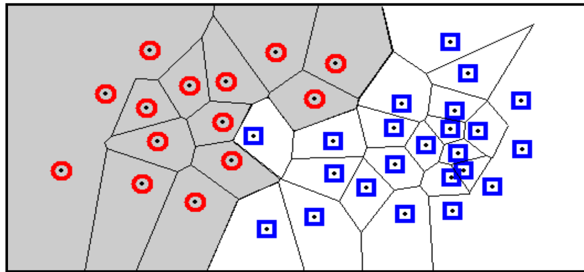
* Editing using pruning

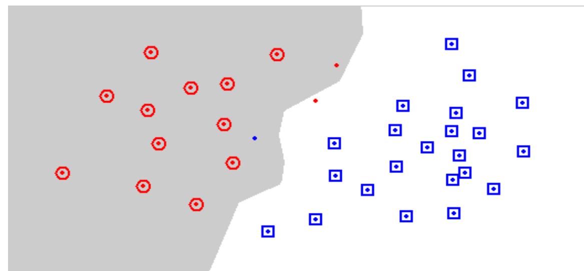
- Pruning seeks to remove "noisy" points and produces smooth decision boundaries.

- 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)

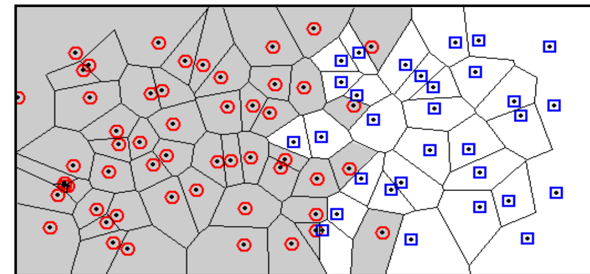
Original data

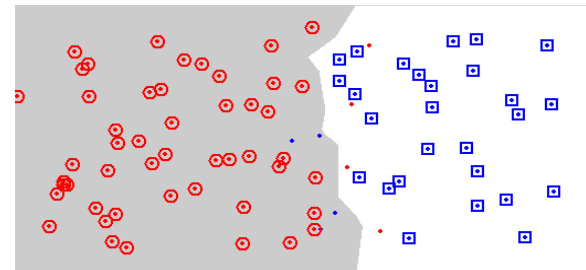




Wilson editing with k=7

Original data



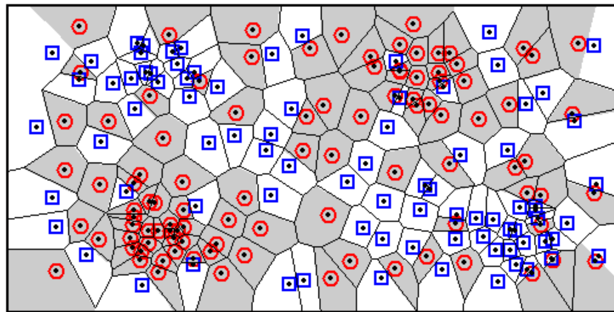


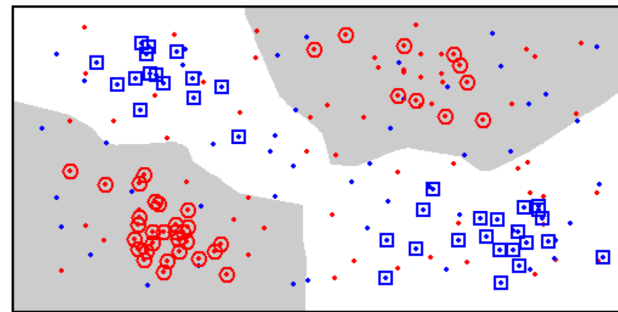
Wilson editing with k=7

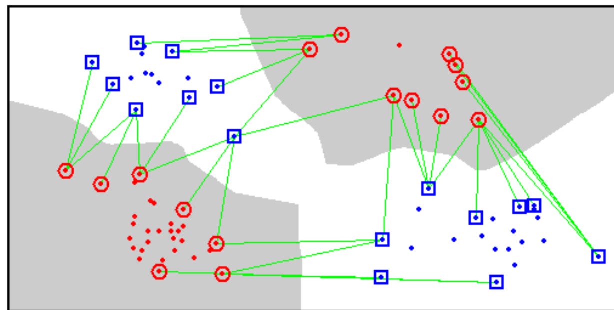
* Combined Editing/Condensing

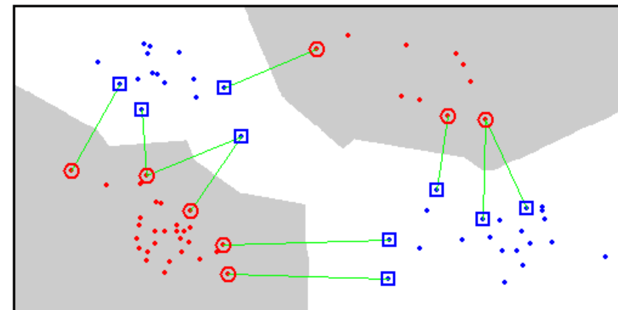
(1) Prune the data to remove noise and smooth the boundary.

(2) Condense to obtain a smaller subset.


Original data: (99,85)


Multi-edit: (39,36)


Delaunay edited: (16,16)


Gabriel edited: (8,9)

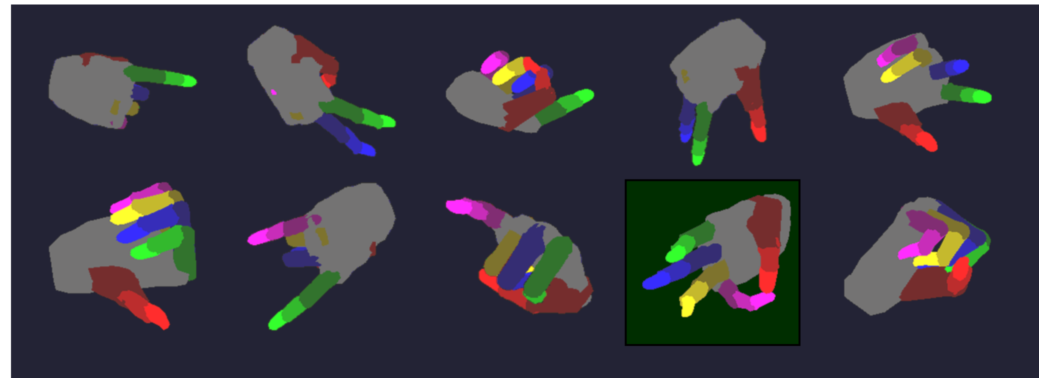
Nearest Neighbor Embedding

- Map the training examples to a low dimensional space such that distances between training examples are preserved as much as possible.
  - i.e., reduce *d* and at the same time keep all the nearest neighbors in the original space.

Example: 3D hand pose estimation

Database (107,328 images)

query



nearest neighbor

Athitsos and Sclaroff. ***Estimating 3D Hand Pose from a Cluttered Image,*** CVPR 2004

General comments (nearest-neighbor classifier)

- The nearest neighbor classifier provides a powerful tool.
  - Its error is bounded to be at most <u>two times</u> of the Bayes error (in the limiting case).
  - It is <u>easy</u> to implement and understand.
  - It can be implemented <u>efficiently</u>.
  - Its performance, however, relies on the <u>metric</u> 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: $D(\mathbf{a}, \mathbf{b}) + D(\mathbf{b}, \mathbf{c}) \geq D(\mathbf{a}, \mathbf{c})$

Metrics and Nearest-Neighbor Classification

- A metric $D(\cdot, \cdot)$ is merely a function that gives a generalized scalar distance between two argument patterns.
- Euclidean formula for distance in d dimensions

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

- Minkowski metric (L_k norm)

$$L_k(\mathbf{a}, \mathbf{b}) = \left(\sum_{i=1}^{d} |a_i - b_i|^k \right)^{1/k},$$

The L_1 norm is sometimes called the *Manhattan* or *city block* distance, the shortest path between ***a*** and ***b***, each segment of which is parallel to a coordinate axis.

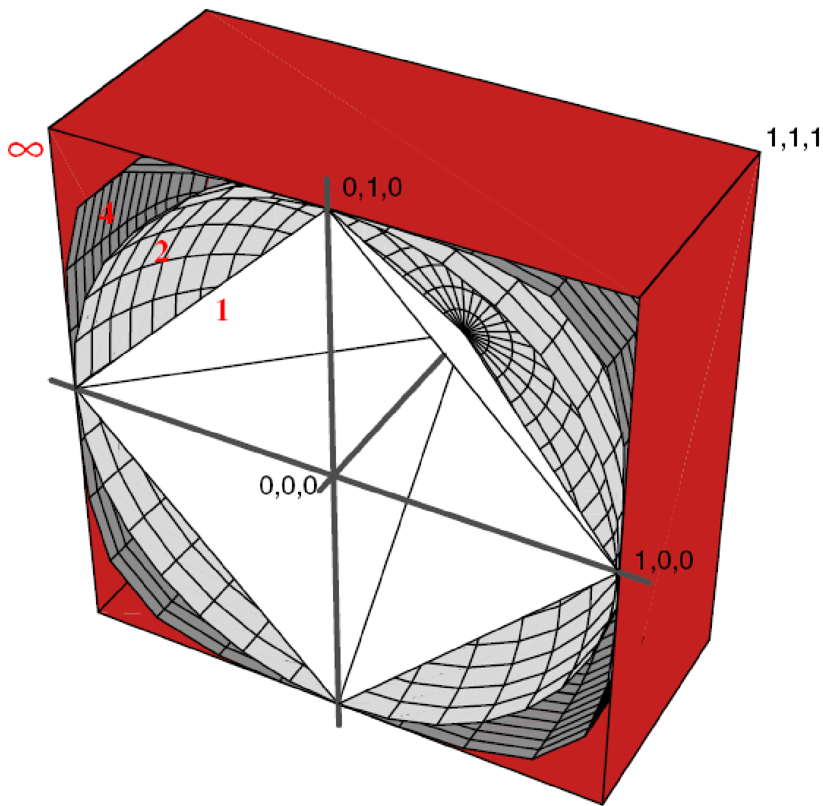


Figure 4.19: Each colored surface consists of points a distance 1.0 from the origin, measured using different values for *k* in the *Minkowski metric* (*k* is printed in red). Thus the white surfaces correspond to the L_1 norm (*Manhattan distance*), light gray the L_2 norm (*Euclidean distance*), dark gray the L_4 norm, and red the L_∞ norm.

Todeschini (1989) assesses six global metrics (Table 4.1) on 10 datasets after four ways of standardizing the data (Table4.2). The maximum scaling standardization procedure performed well, and was found to be robust to the choice of distance metric.

Table 4.1 Distance metrics assessed by Todeschini.

Cosine metric	$d(\boldsymbol{x},\boldsymbol{y}) = \frac{\boldsymbol{x}.\boldsymbol{y}}{\lvert\boldsymbol{x}\rvert\lvert\boldsymbol{y}\rvert} = \frac{\sum_{i=1}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} x_i^2}\sqrt{\sum_{i=1}^{d} y_i^2}}$
Canberra metric	$d(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{d}\sum_{i=1}^{d} \frac{\lvert x_i - y_i\rvert}{x_i + y_i}$
Euclidean metric	$d(\boldsymbol{x},\boldsymbol{y}) = \lvert\boldsymbol{x}-\boldsymbol{y}\rvert = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$
Lagrange metric	$d(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1,\ldots,d} \lvert x_i - y_i\rvert$
Lance–Williams metric	$d(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i=1}^{d}\lvert x_i - y_i\rvert}{\sum_{i=1}^{d}(x_i + y_i)}$
Manhattan metric	$d(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{d}\lvert x_i - y_i\rvert$

Table 4.2 Standardisation approaches assessed by Todeschini. μ_j and σ_j are the training data estimates of the mean and standard deviation of the jth variable. U_j and L_j are the upper and lower bounds of the jth variable. $y_{i,j}$ is the jth variable of the ith training data example.

Autoscaling	$x'_j = \frac{x_j - \mu_j}{\sigma_j}$
Maximum scaling	$x'_j = \frac{x_j}{U_j}$
Range scaling	$x'_j = \frac{x_j - L_j}{U_j - L_j}$
Profiles	$x'_j = \frac{x_j}{\sqrt{\sum_{i=1}^{n}\left(y_{i,j}^2\right)}\sqrt{\sum_{j=1}^{d}\left(x_j^2\right)}}$

Tanimoto metric

The *Tanimoto* metric finds most use in taxonomy, where the distance between two sets is defined as

$$D_{Tanimoto}(\mathcal{S}_1, \mathcal{S}_2) = \frac{n_1 + n_2 - 2n_{12}}{n_1 + n_2 - n_{12}},$$

where n_1 and n_2 are the number of elements in sets S_1 and S_2, respectively, and n_{12} is the number that is in both sets.

* Tangent distance

There may be drawbacks inherent in the uncritical use of a particular metric in nearest-neighbor classifiers, and these drawbacks can be overcome by the careful use of more general measures of distance.

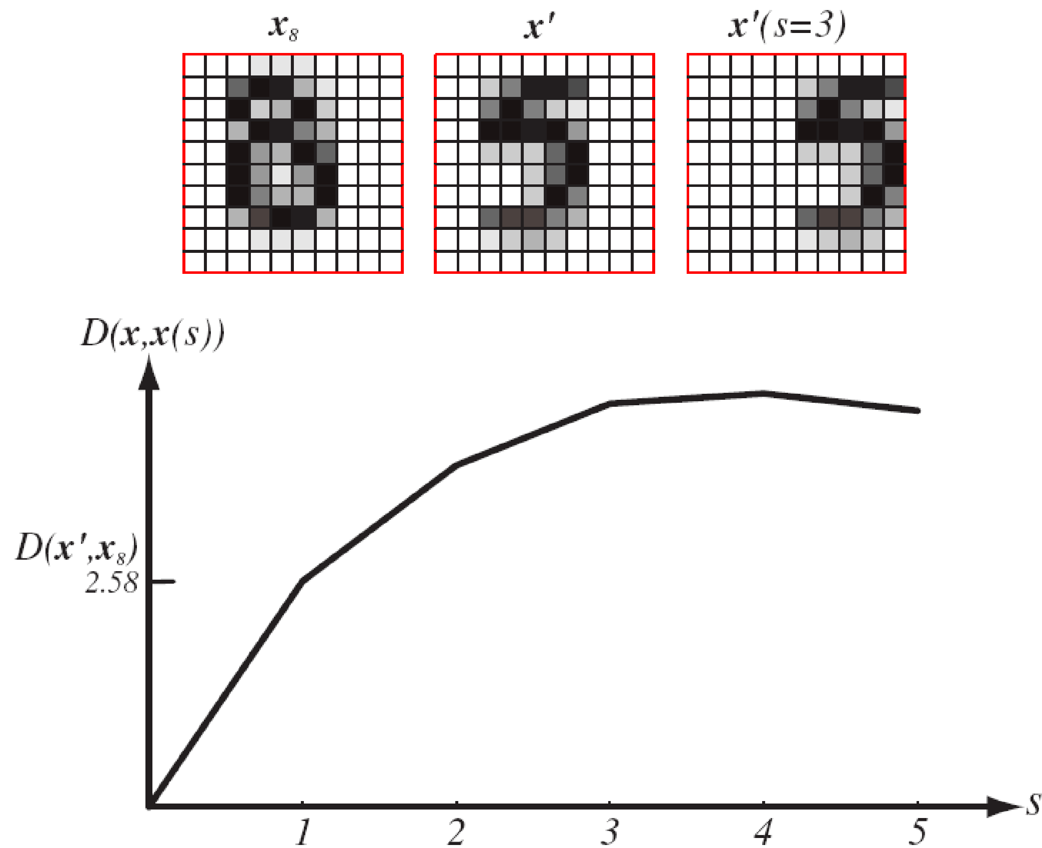


Figure 4.20: The uncritical use of Euclidean metric cannot address the problem of translation invariance. Pattern **x** represents a handwritten 5, and $\mathbf{x}'(s = 3)$ the same shape but shifted three pixels to the right. The Euclidean distance $D(\mathbf{x}', \mathbf{x}'(s = 3))$ is much larger than $D(\mathbf{x}', \mathbf{x}_8)$, where $\mathbf{x}_8$ represents the handwritten 8. Nearest-neighbor classification based on the Euclidean distance in this way leads to very large errors. Instead, we seek a distance measure that would be insensitive to such translations, or indeed other known invariances, such as scale or rotation.

* Distance metrics - Invariance

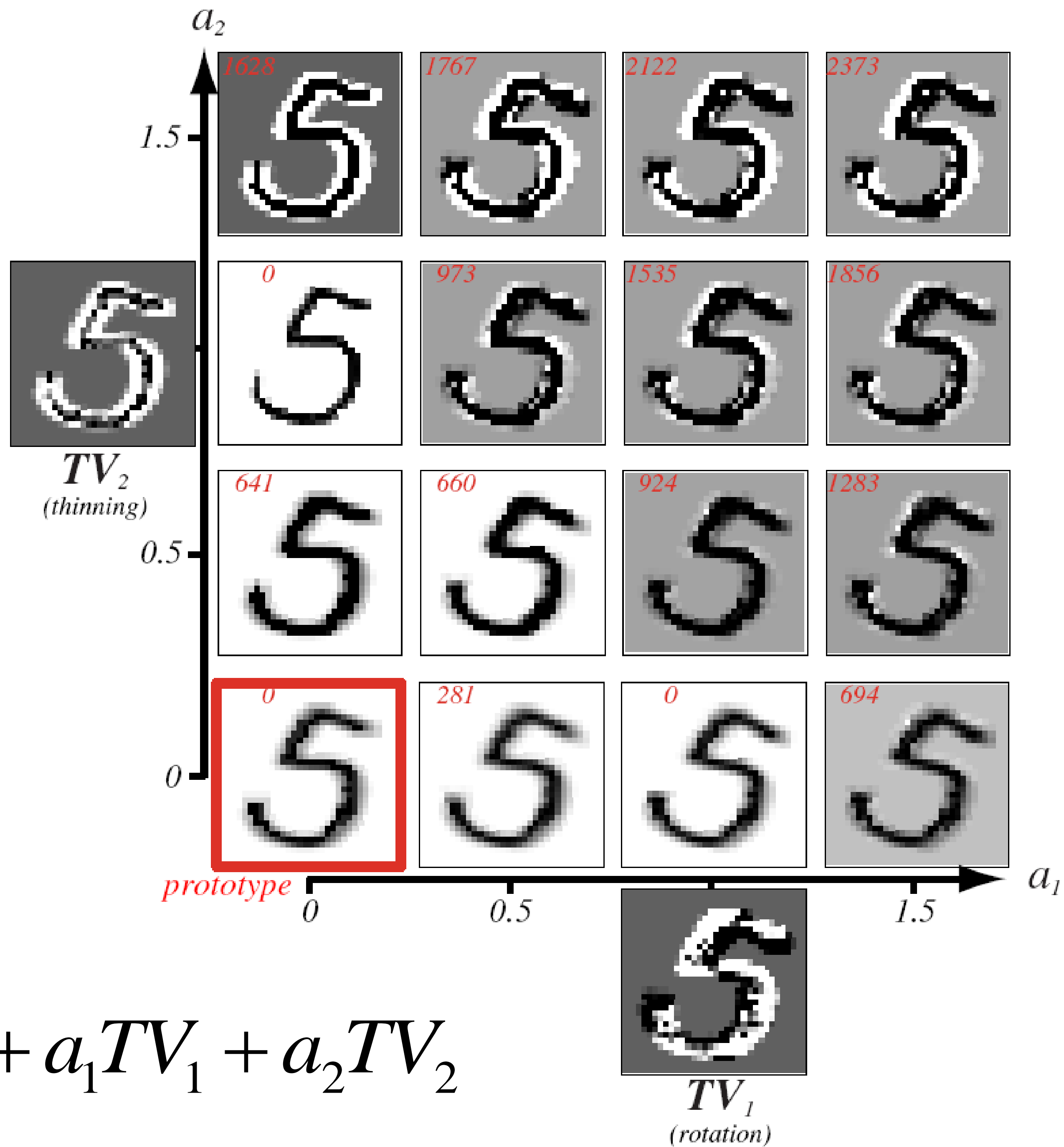
- How to deal with transformations?
  - Normalize data (e.g., shift center to a fixed location)
  - More difficult to normalize with respect to rotation and scaling ...
  - How to find the rotation/scaling factors?

Suppose we believe there are r transformations applicable to our problem, such as horizontal translation, vertical translation, shear, rotation, scale, and line thinning.

During construction of the classifier we take each stored prototype $\mathbf{x}'$ and perform each of the transformations $F_i(\mathbf{x}'; \alpha_i)$ on it. Thus $F_i(\mathbf{x}'; \alpha_i)$ could represent the image described by $\mathbf{x}'$, rotated by a small angle α_i. We then construct a tangent vector TV_i for each transformation:

$$\mathrm{TV}_i = \mathrm{F}_i(\mathbf{x}'; \alpha_i) - \mathbf{x}'.$$

While such a transformation may be compute intensive — as, for instance, the line thinning transform — it need be done only once, during training when computational constraints are lax. In this way we construct for each prototype $\mathbf{x}'$ an $r \times d$ matrix $\mathbf{T}$, consisting of the tangent vectors at $\mathbf{x}'$.



$$\hat{x}' = x' + a_1 TV_1 + a_2 TV_2$$

Figure 4.21: The pixel image of the handwritten 5 prototype at the lower left was subjected to two transformations, rotation, and line thinning, to obtain the tangent vectors TV_1 and TV_2; images corresponding to these tangent vectors are shown outside the axes. Each of the 16 images within the axes represents the prototype plus linear combination of the two tangent vectors with coefficients a_1 and a_2. The small red number in each image is the Euclidean distance between the tangent approximation and the image generated by the unapproximated transformations. Of course, this Euclidean distance is 0 for the prototype and for the cases $a_1 = 1, a_2 = 0$ and $a_1 = 0, a_2 = 1$. (The patterns generated with $a_1 + a_2 > 1$ have a gray background because of automatic grayscale conversion of images with negative pixel values.)

Now we turn to computing the tangent distance from a test point **x** to a particular stored prototype **x**′. Formally, given a matrix **T** consisting of the *r tangent vectors* at **x**′, the tangent distance from **x**′ *to* **x** *is:*

$$D_{tan}(\mathbf{x}', \mathbf{x}) = \min_{\mathbf{a}}[\|(\mathbf{x}' + \mathbf{Ta}) - \mathbf{x}\|],$$

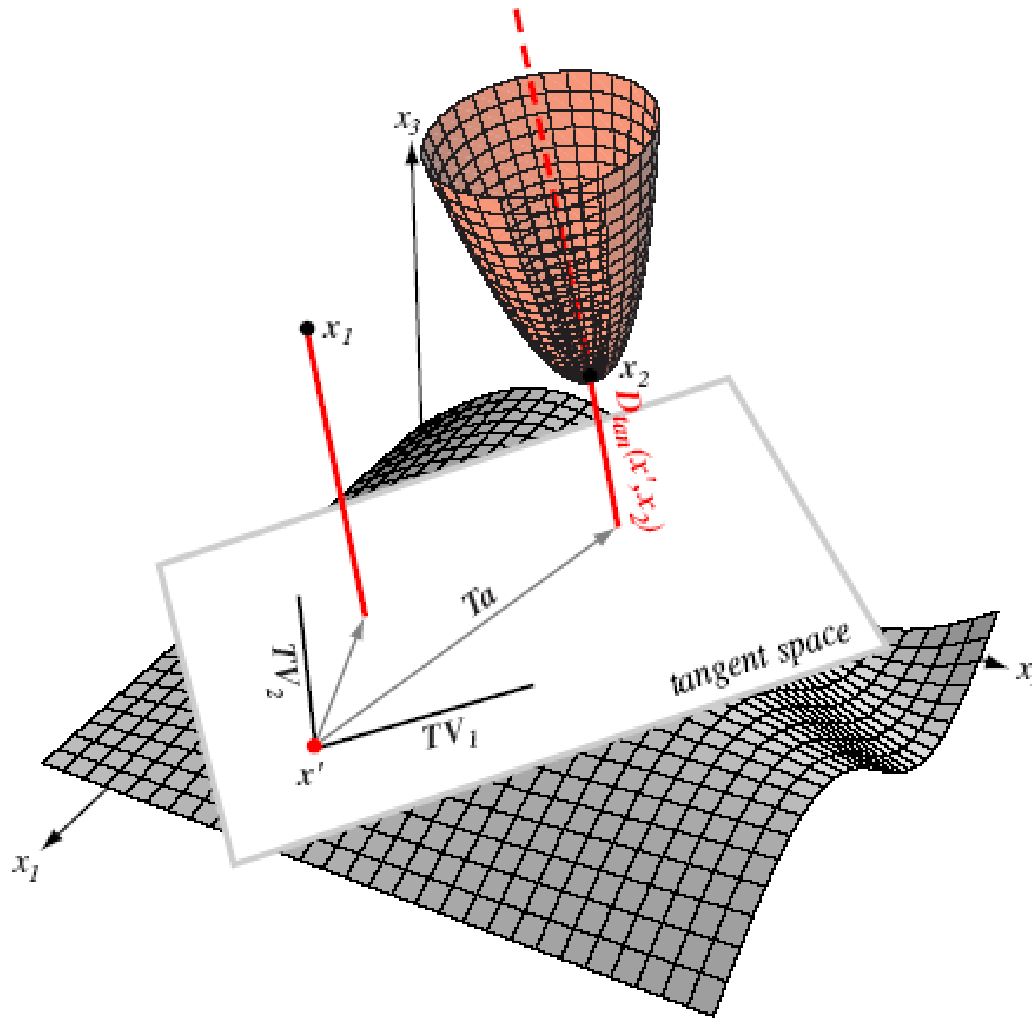
i.e., the Euclidean distance from **x** to the tangent space of **x**′.

During classification of **x** we will find its tangent distance to **x**′ by finding the optimizing value of **a** required by above eq.

.

.

.



- Fuzzy Classification …
- Approximations by Series Expansions ...
- Relaxation methods …

Relaxation methods

- Parzen-window method uses a fixed window throughout the feature space, and that this could lead to difficulties: in some regions a small window width was appropriate while elsewhere a large one would be best.
- The *k-nearest neighbor* method addressed this problem by adjusting the region based on the density of the points.
- Another approach that is intermediate between these two is to adjust the size of the window during training according to the distance to the nearest point of a *different category.* → *relaxation techniques.*

One representative method — called the Reduced Coulomb Energy or RCE network — has the form shown in the next Fig.

Algorithm 4 (RCE training)

1 **begin initialize** $j \leftarrow 0$, $n \leftarrow$ #patterns,
$\varepsilon \leftarrow$ small param, $\lambda_m \leftarrow$ max radius
2 **do** $j \leftarrow j + 1$
3 train weight: $w_{ij} \leftarrow x_i$
4 find nearest point not in ω_k: $\hat{\mathbf{x}} \leftarrow \arg\min_{\mathbf{x}' \notin \omega_k} D(\mathbf{x}, \mathbf{x}')$
5 set radius $\lambda_j \leftarrow \min[\max[D(\hat{\mathbf{x}}, \mathbf{x}'), \epsilon], \lambda_m]$
6 **if** $\mathbf{x} \in \omega_k$ **then** $a_{jk} \leftarrow 1$
7 **until** $j = n$
8 **end**

Let λ_j be the radius around stored prototype $\mathbf{x}_j$ and now let D_t be the set of stored prototypes in whose hypershperes test point **x** lies, then our classification algorithm is written as:

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**

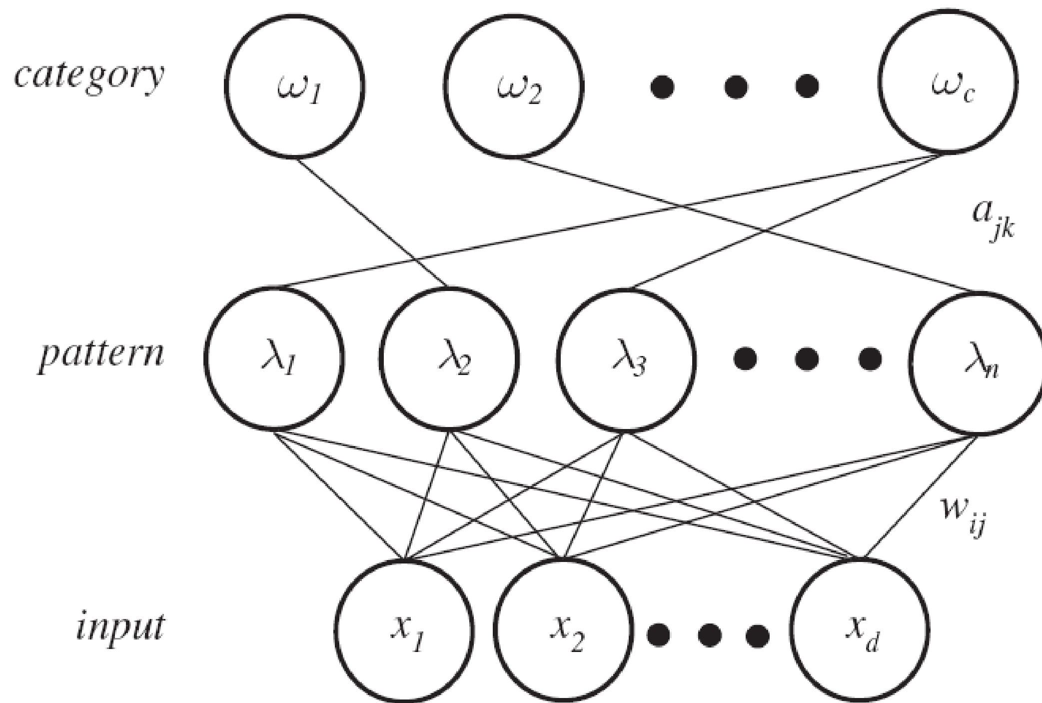


Figure 4.25: An RCE network is topologically equivalent to the PNN. During training the weights are adjusted to have the same values as the pattern presented, just as in a PNN. However, pattern units in an RCE network also have a modifiable "radius" parameter λ. During training, each λ is adjusted so that the region is as large as possible without containing training patterns from a different category.

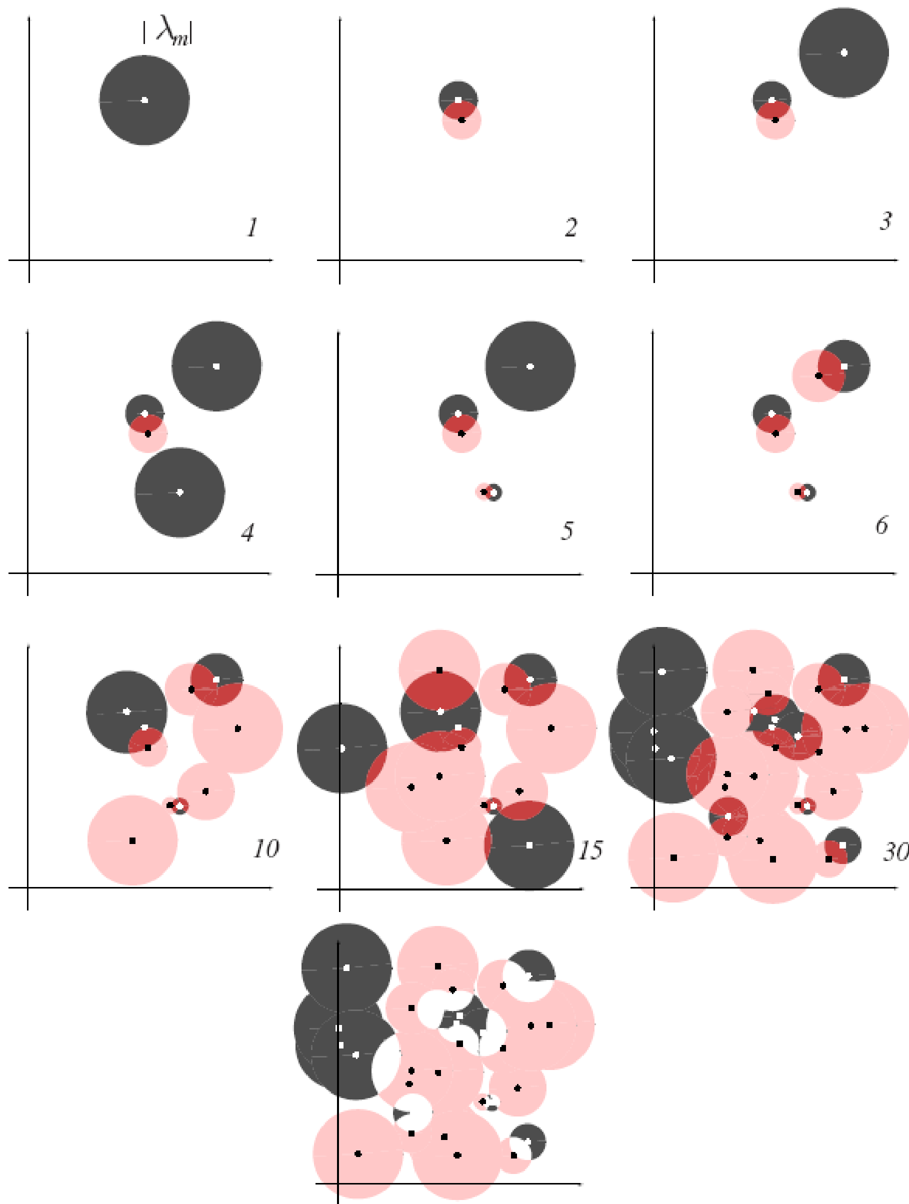


Figure 4.26: During training, each pattern has a parameter — equivalent to a radius in the *d-dimensional space* — that is adjusted to be as large as possible, without enclosing any points from a different category. As new patterns are presented, each such radius is decreased accordingly (and can never increase). In this way, each pattern unit can enclose several prototypes, but only those having the same category label. The number of points is shown in each component figure. The figure at the bottom shows the final complicated decision regions, colored by category.