VU Pattern Recognition II

Helmut A. Mayer

Department of Computer Sciences
University of Salzburg

WS 13/14

Human vs. Machine

- Human Perception

Human vs. Machine

- Human Perception
  - Senses to neural patterns

Human vs. Machine

- Human Perception
  - Senses to neural patterns
- Machine Perception

Human vs. Machine

- Human Perception
  - Senses to neural patterns
- Machine Perception
  - Sensors to value patterns

Human vs. Machine

- Human Perception
  - Senses to neural patterns
- Machine Perception
  - Sensors to value patterns
- Patterns are everywhere...

Human vs. Machine

- Human Perception
  - Senses to neural patterns
- Machine Perception
  - Sensors to value patterns
- Patterns are everywhere...
- Images, Time Series, Medical Diagnosis, Customer Analysis (only a few examples)

Human vs. Machine

- Human Perception
  - Senses to neural patterns
- Machine Perception
  - Sensors to value patterns
- Patterns are everywhere...
- Images, Time Series, Medical Diagnosis, Customer Analysis (only a few examples)
- Features build Model

Human vs. Machine

- Human Perception
  - Senses to neural patterns
- Machine Perception
  - Sensors to value patterns
- Patterns are everywhere...
- Images, Time Series, Medical Diagnosis, Customer Analysis (only a few examples)
- Features build Model
- Fish Example

Salmon or Sea Bass

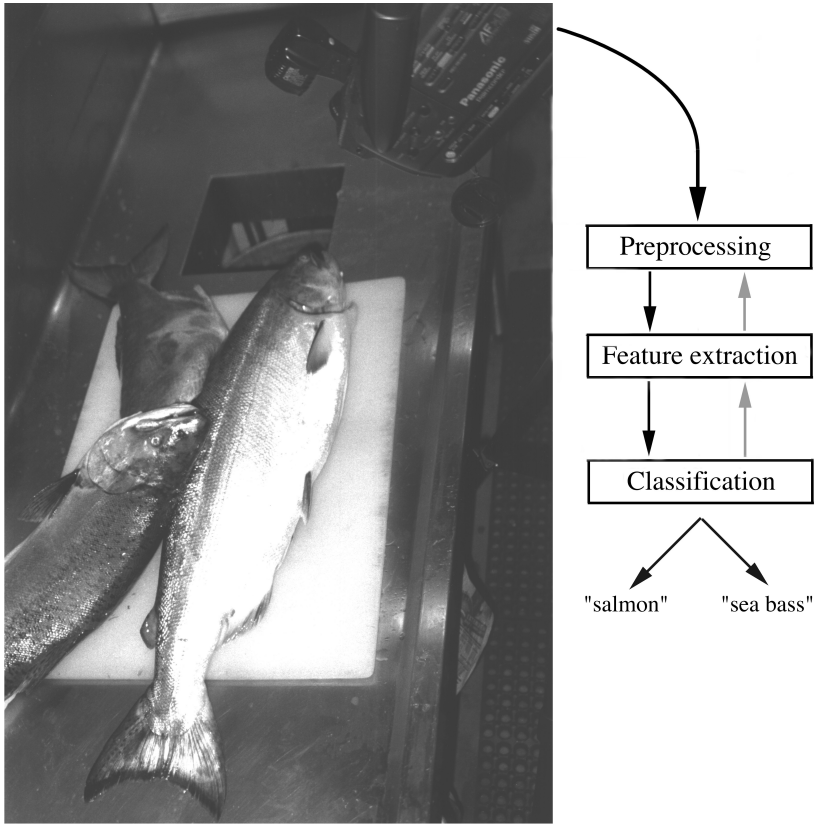


FIGURE 1.1. The objects to be classified are first sensed by a transducer (camera), whose signals are preprocessed. Next the features are extracted and finally the classification is emitted, here either "salmon" or "sea bass." Although the information flow is often chosen to be from the source to the classifier, some systems employ information flow in which earlier levels of processing can be altered based on the tentative or preliminary response in later levels (gray arrows). Yet others combine two or more stages into a unified step, such as simultaneous segmentation and feature extraction. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Fish Length Histogram

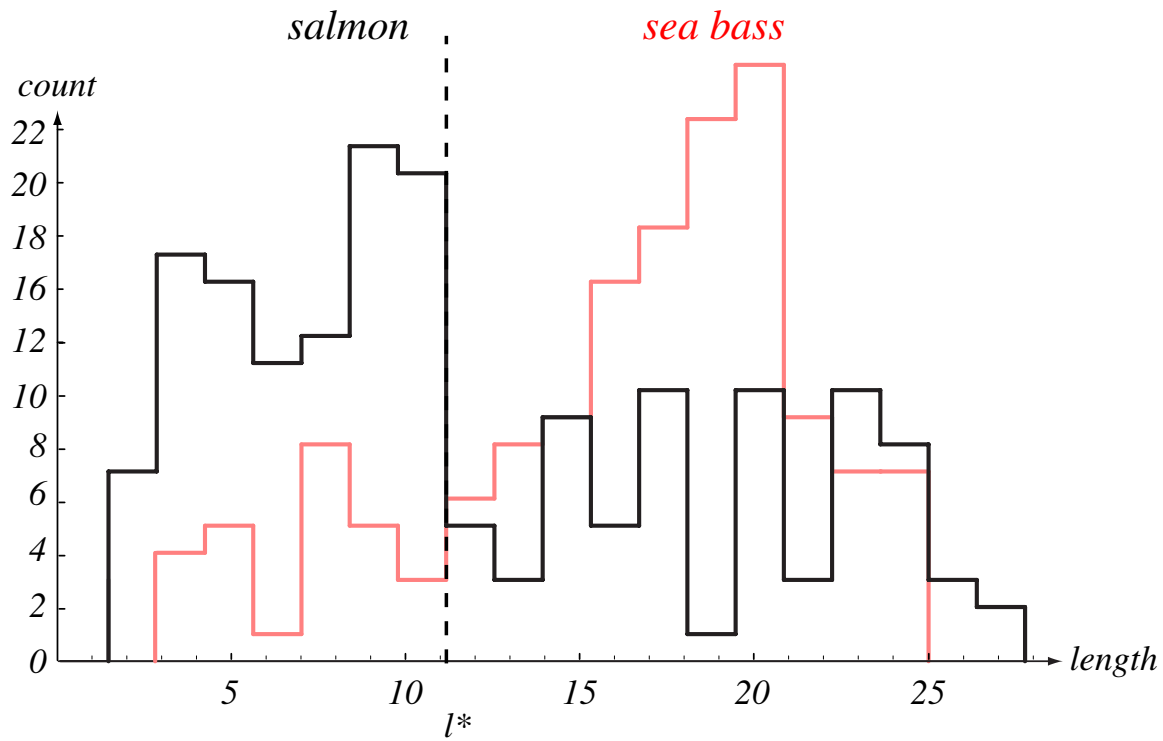


FIGURE 1.2. Histograms for the length feature for the two categories. No single threshold value of the length will serve to unambiguously discriminate between the two categories; using length alone, we will have some errors. The value marked l^* will lead to the smallest number of errors, on average. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Fish Lightness Histogram

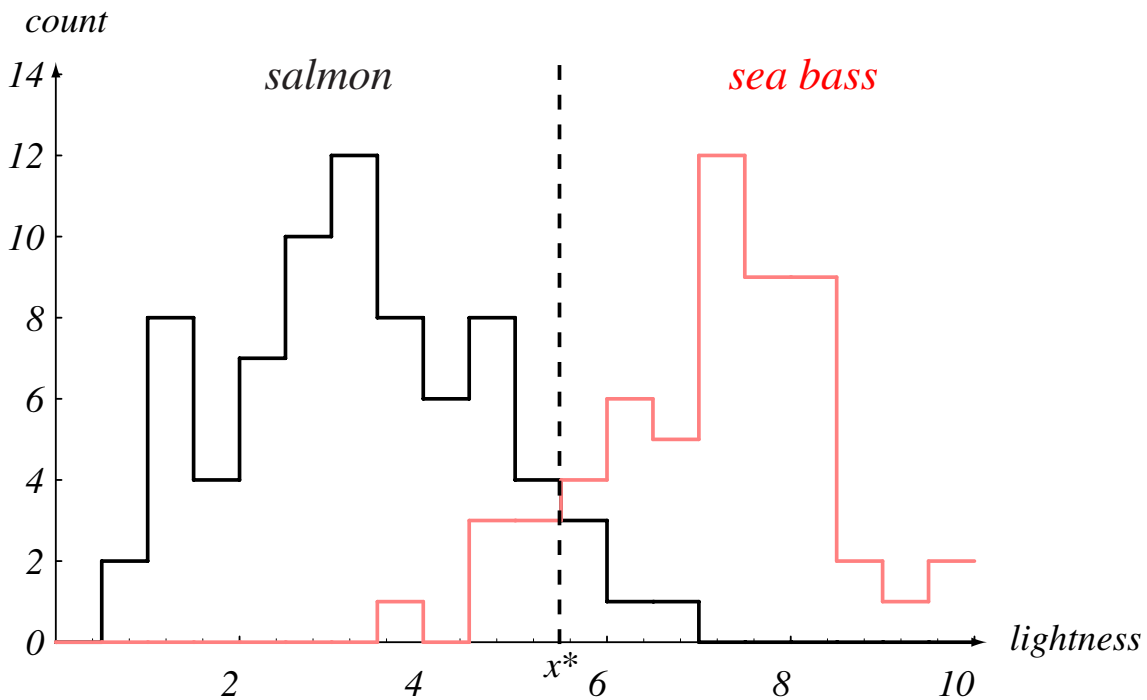


FIGURE 1.3. Histograms for the lightness feature for the two categories. No single threshold value x^* (decision boundary) will serve to unambiguously discriminate between the two categories; using lightness alone, we will have some errors. The value x^* marked will lead to the smallest number of errors, on average. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Decision Theory

- Cost of an Error?

Decision Theory

- Cost of an Error?
  - Salmon tastes better..;-)

Decision Theory

- Cost of an Error?
  - Salmon tastes better..;-)
  - Minimization of cost (risk)

Decision Theory

- Cost of an Error?
  - Salmon tastes better..;-)
  - Minimization of cost (risk)
  - Decision Rule/Boundary

Decision Theory

- Cost of an Error?
  - Salmon tastes better..;-)
  - Minimization of cost (risk)
  - Decision Rule/Boundary
- Improving Recognition

Decision Theory

- Cost of an Error?
  - Salmon tastes better..;-)
  - Minimization of cost (risk)
  - Decision Rule/Boundary
- Improving Recognition
  - Feature Vector $\vec{x} = \begin{pmatrix} lightness \\ width \end{pmatrix}$

Decision Theory

- Cost of an Error?
  - Salmon tastes better..;-)
  - Minimization of cost (risk)
  - Decision Rule/Boundary
- Improving Recognition
  - Feature Vector $\vec{x} = \begin{pmatrix} lightness \\ width \end{pmatrix}$
- 2D Decision Boundary

2D Feature Space

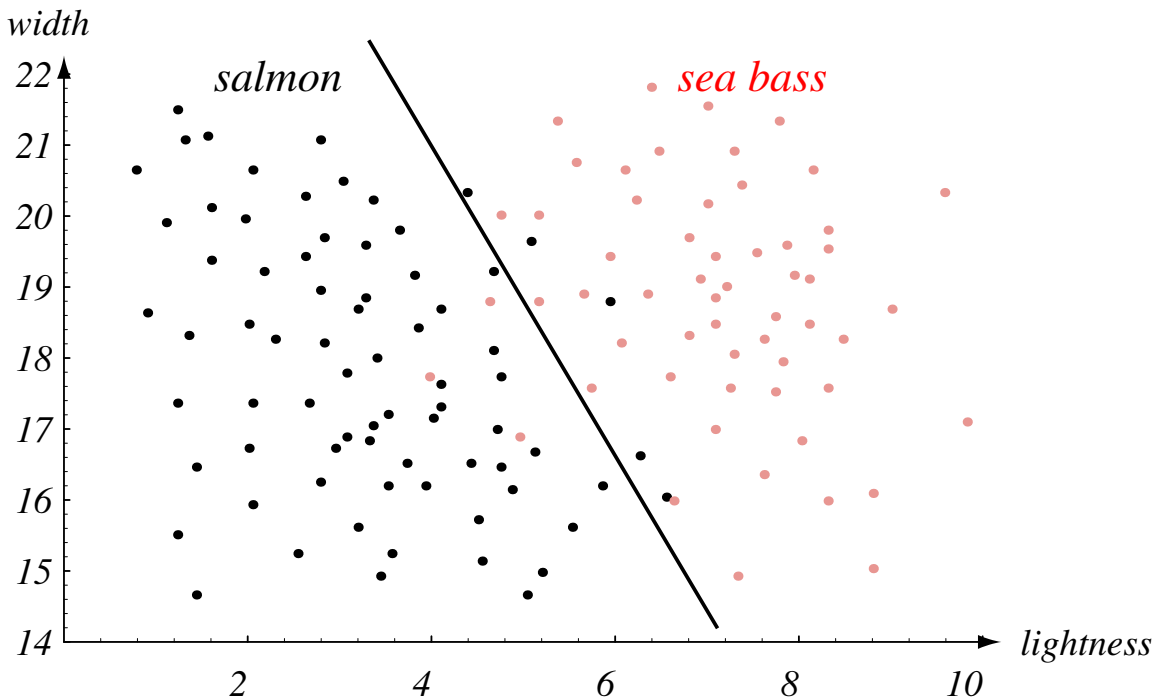


FIGURE 1.4. The two features of lightness and width for sea bass and salmon. The dark line could serve as a decision boundary of our classifier. Overall classification error on the data shown is lower than if we use only one feature as in Fig. 1.3, but there will still be some errors. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Overfitting

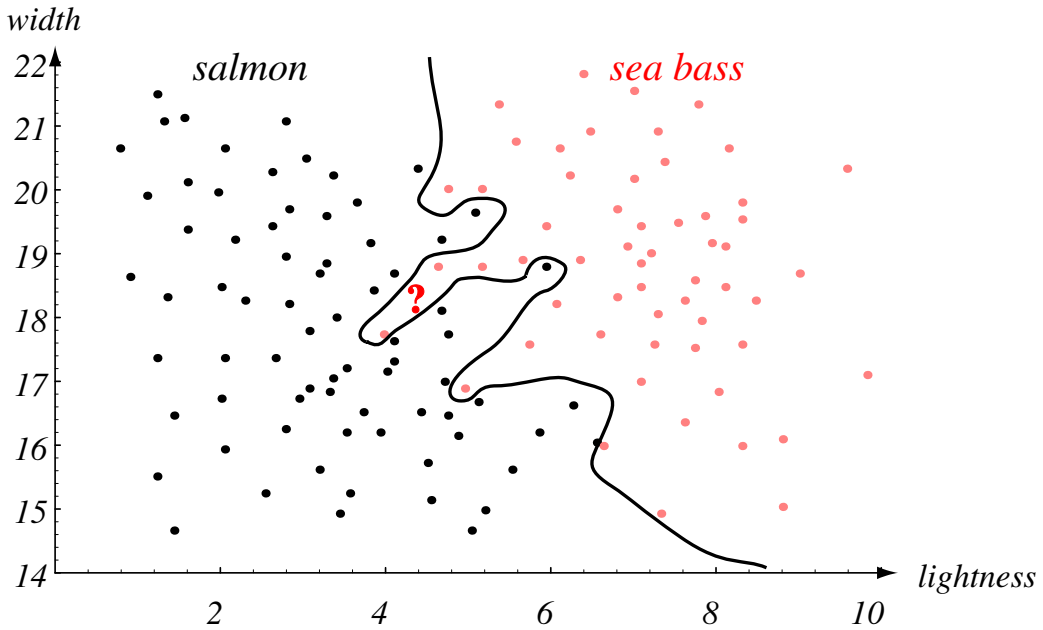


FIGURE 1.5. Overly complex models for the fish will lead to decision boundaries that are complicated. While such a decision may lead to perfect classification of our training samples, it would lead to poor performance on future patterns. The novel test point marked **?** is evidently most likely a salmon, whereas the complex decision boundary shown leads it to be classified as a sea bass. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Generalization

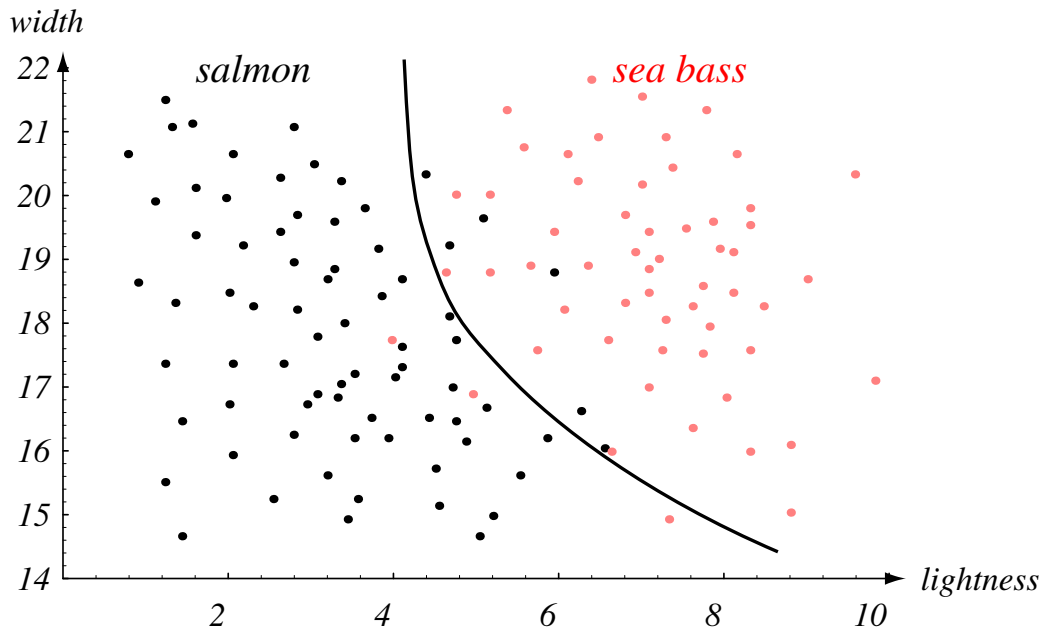


FIGURE 1.6. The decision boundary shown might represent the optimal tradeoff between performance on the training set and simplicity of classifier, thereby giving the highest accuracy on new patterns. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Related Fields

- Statistical Hypothesis Testing

Related Fields

- Statistical Hypothesis Testing
- Image Processing

Related Fields

- Statistical Hypothesis Testing
- Image Processing
- Regression (age $\leftrightarrow$ weight)

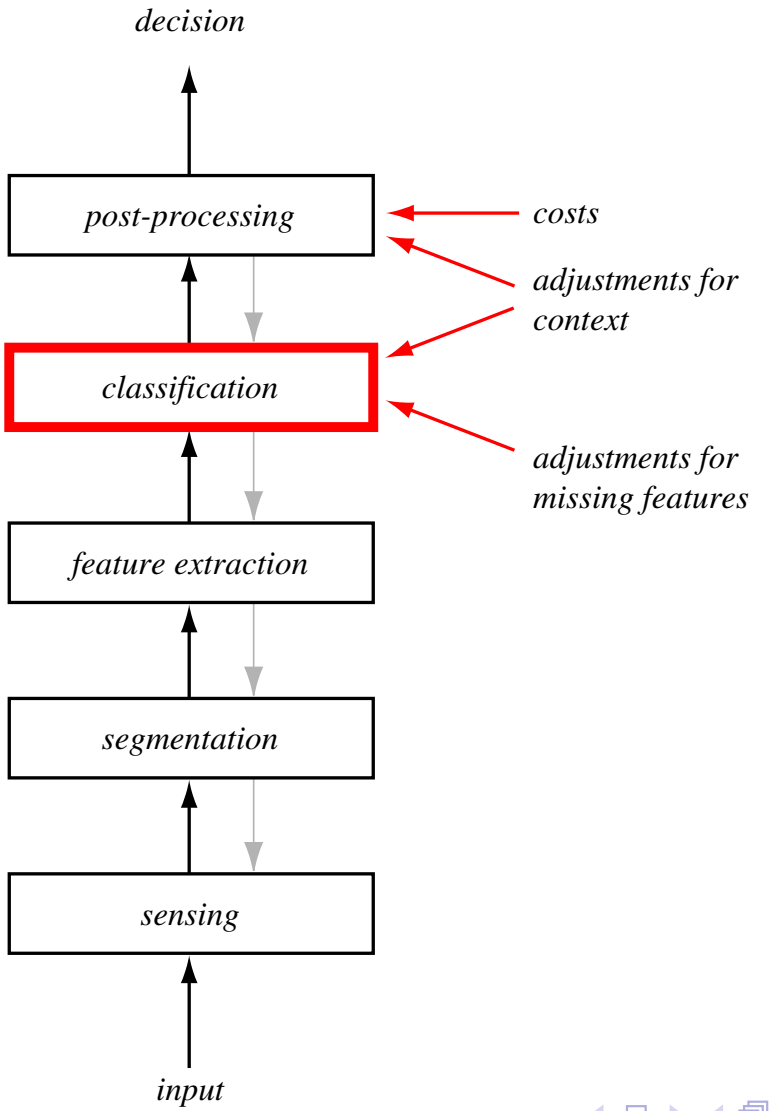
Related Fields

- Statistical Hypothesis Testing
- Image Processing
- Regression (age $\leftrightarrow$ weight)
- Interpolation

Related Fields

- Statistical Hypothesis Testing
- Image Processing
- Regression (age $\leftrightarrow$ weight)
- Interpolation
- Density Estimation

Pattern Recognition Systems



Feature Extraction

- Features $\leftrightarrow$ Classification

Feature Extraction

- Features $\leftrightarrow$ Classification
- Invariant Features (translation, rotation, scale)

Feature Extraction

- Features $\leftrightarrow$ Classification
- Invariant Features (translation, rotation, scale)
- Deformation (e.g. Cropping)

Feature Extraction

- Features $\leftrightarrow$ Classification
- Invariant Features (translation, rotation, scale)
- Deformation (e.g. Cropping)
- Feature Selection (Filter, Wrapper)

Post Processing

- Error Rate, Risk (weighted error)

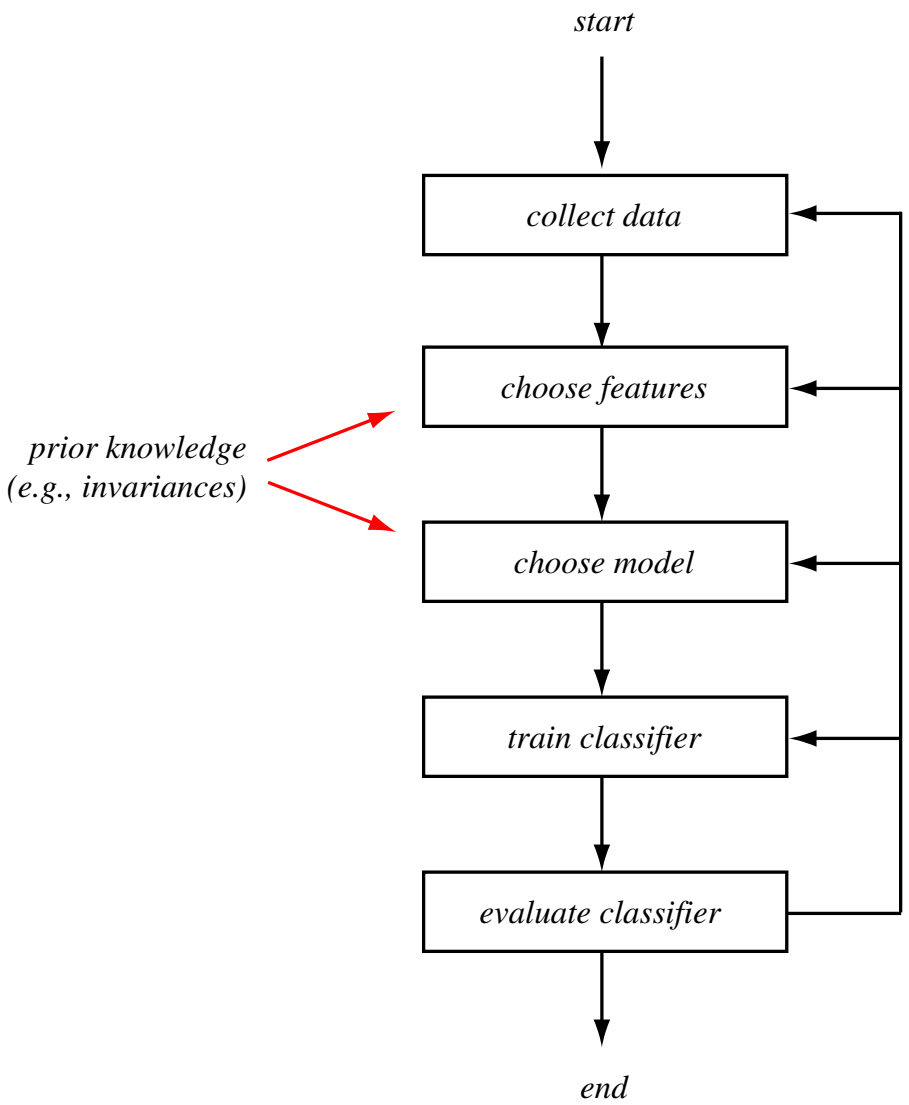
Post Processing

- Error Rate, Risk (weighted error)
- Context (IC* *IN)

Post Processing

- Error Rate, Risk (weighted error)
- Context (IC* *IN)
- Multiple Classifiers (subspaces, fusion)

Design Cycle



Learning and Adaptation

- Learning is Parameter Tuning

Learning and Adaptation

- Learning is Parameter Tuning
- Supervised Learning (teacher)

Learning and Adaptation

- Learning is Parameter Tuning
- Supervised Learning (teacher)
- Reinforcement Learning (critic)

Learning and Adaptation

- Learning is Parameter Tuning
- Supervised Learning (teacher)
- Reinforcement Learning (critic)
- Unsupervised Learning (clustering)

Probabilities

- State of Nature $\omega = \omega_1$ (class)

Probabilities

- State of Nature $\omega = \omega_1$ (class)
- A Priori Probability $P(\omega_1)$ (prior)

Probabilities

- State of Nature $\omega = \omega_1$ (class)
- A Priori Probability $P(\omega_1)$ (prior)
- Decision Rule $P(\omega_1) > P(\omega_2) \rightarrow \omega_1$

Probabilities

- State of Nature $\omega = \omega_1$ (class)
- A Priori Probability $P(\omega_1)$ (prior)
- Decision Rule $P(\omega_1) > P(\omega_2) \rightarrow \omega_1$
- Class–Conditional Probability Density Function $p(x|\omega)$

Class–Conditional Probability Density

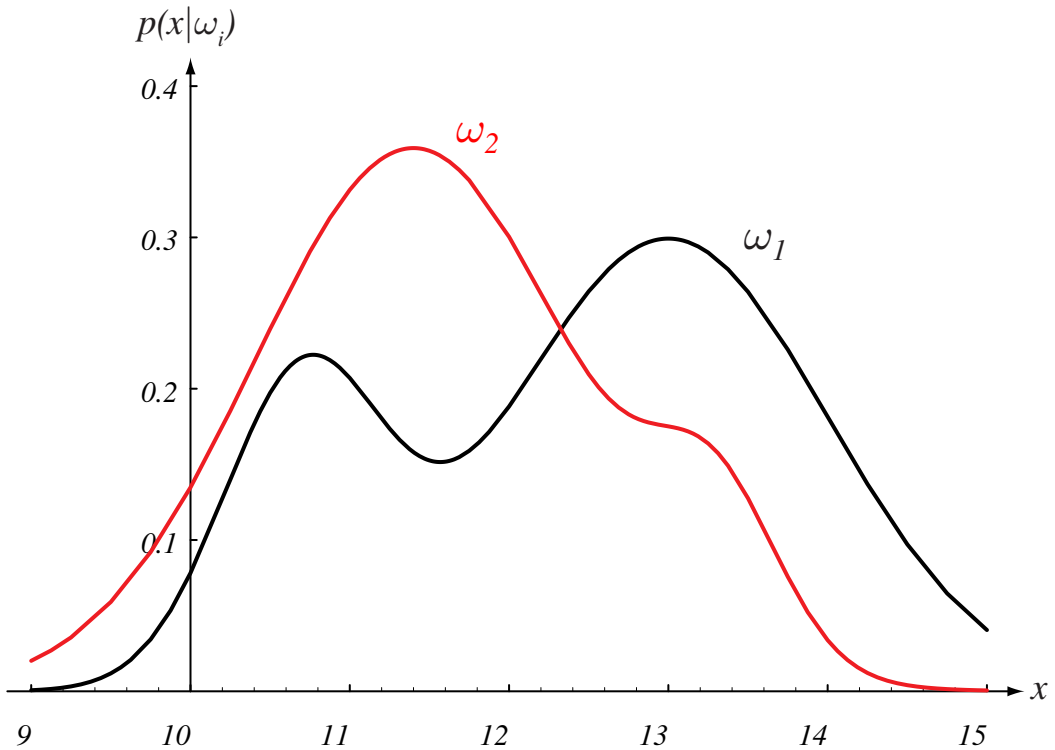


FIGURE 2.1. Hypothetical class-conditional probability density functions show the probability density of measuring a particular feature value x given the pattern is in category ω_i. If x represents the lightness of a fish, the two curves might describe the difference in lightness of populations of two types of fish. Density functions are normalized, and thus the area under each curve is 1.0. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Bayes Decision Rule

- Joint Probability Density
  $p(\omega_j, x) = P(\omega_j|x)p(x) = p(x|\omega_j)P(\omega_j)$

Bayes Decision Rule

- Joint Probability Density
  $p(\omega_j, x) = P(\omega_j|x)p(x) = p(x|\omega_j)P(\omega_j)$
- Bayes Formula $P(\omega_j|x) = \frac{p(x|\omega_j)P(\omega_j)}{p(x)}$

Bayes Decision Rule

- Joint Probability Density
  $p(\omega_j, x) = P(\omega_j|x)p(x) = p(x|\omega_j)P(\omega_j)$
- Bayes Formula $P(\omega_j|x) = \frac{p(x|\omega_j)P(\omega_j)}{p(x)}$
- Decision Rule $P(\omega_1|x) > P(\omega_2|x) \rightarrow \omega_1$

Posterior Probabilities

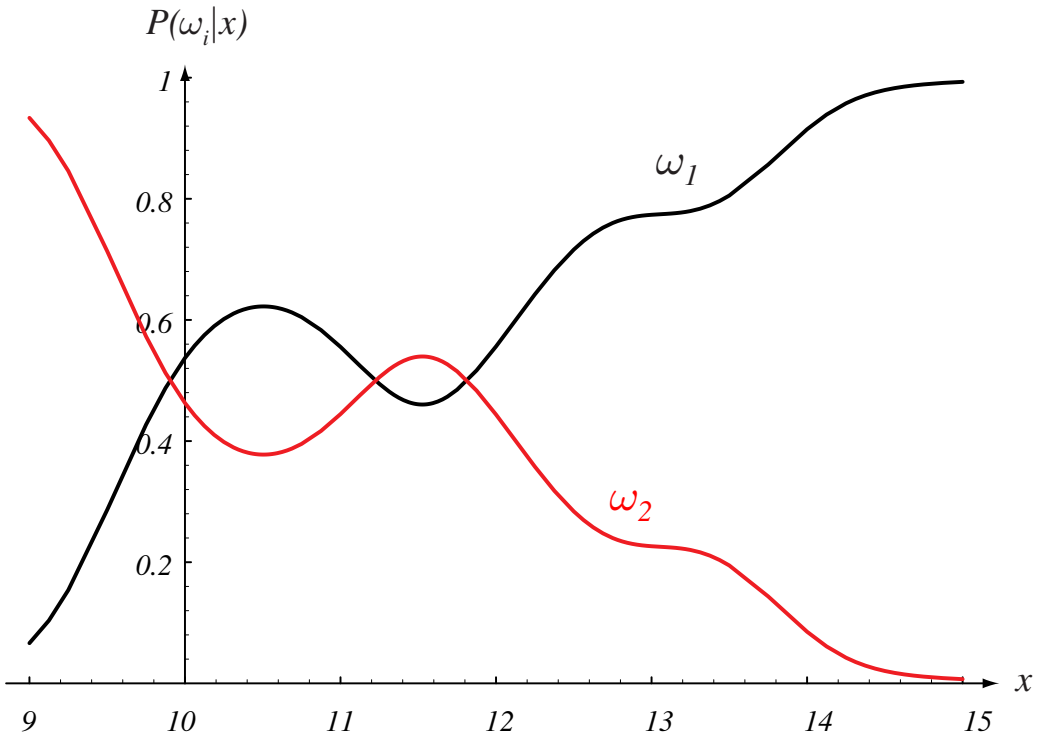


FIGURE 2.2. Posterior probabilities for the particular priors $P(\omega_1) = 2/3$ and $P(\omega_2) = 1/3$ for the class-conditional probability densities shown in Fig. 2.1. Thus in this case, given that a pattern is measured to have feature value $x = 14$, the probability it is in category ω_2 is roughly 0.08, and that it is in ω_1 is 0.92. At every x, the posteriors sum to 1.0. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Error Probabilities

- Error $P(error|x) = \begin{cases} P(\omega_1|x) & if\, \omega_2 \\ P(\omega_2|x) & if\, \omega_1 \end{cases}$

Error Probabilities

- Error $P(error|x) = \begin{cases} P(\omega_1|x) & if\, \omega_2 \\ P(\omega_2|x) & if\, \omega_1 \end{cases}$
- Average Error Probability
  $P(error) = \int_{-\infty}^{\infty} p(error, x)dx = \int_{-\infty}^{\infty} P(error|x)p(x)dx$

Error Probabilities

- Error $P(error|x) = \begin{cases} P(\omega_1|x) & if\, \omega_2 \\ P(\omega_2|x) & if\, \omega_1 \end{cases}$
- Average Error Probability
  $P(error) = \int_{-\infty}^{\infty} p(error, x)dx = \int_{-\infty}^{\infty} P(error|x)p(x)dx$
- Bayes Rule minimizes $P(error)$

Generalized Bayes Rule

- Feature vector $\vec{x} \in \mathbb{R}^d$

Generalized Bayes Rule

- Feature vector $\vec{x} \in \mathbb{R}^d$
- Classes $\omega_1 \ldots \omega_c$

Generalized Bayes Rule

- Feature vector $\vec{x} \in \mathbb{R}^d$
- Classes $\omega_1 \dots \omega_c$
- Bayes Formula $P(\omega_j|\vec{x}) = \frac{p(\vec{x}|\omega_j)P(\omega_j)}{p(\vec{x})}$

Dichotomizer

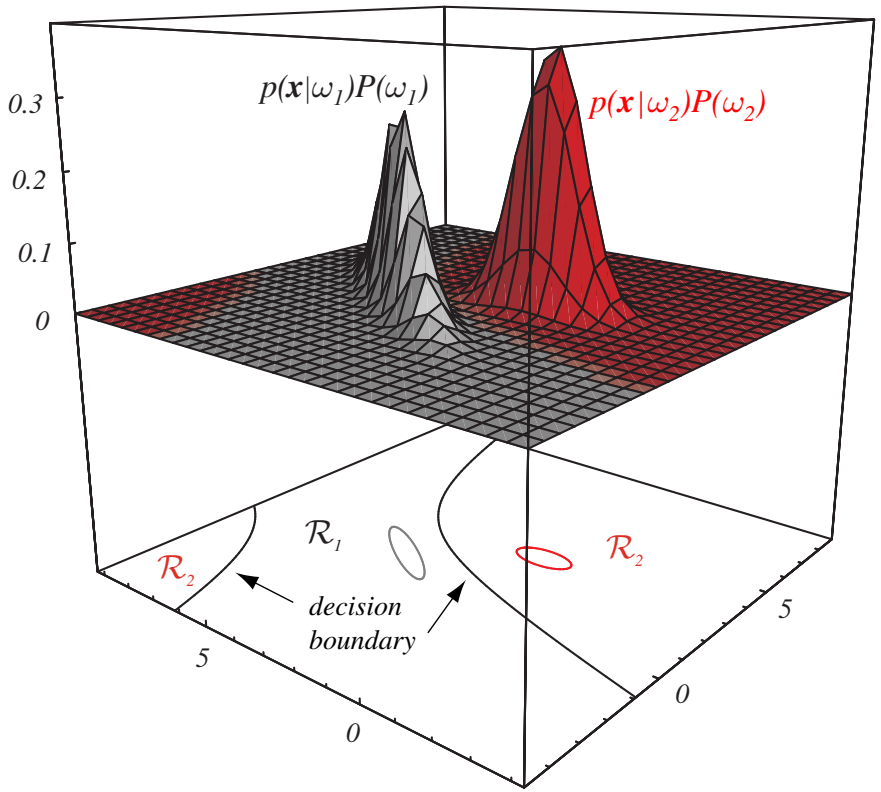


FIGURE 2.6. In this two-dimensional two-category classifier, the probability densities are Gaussian, the decision boundary consists of two hyperbolas, and thus the decision region $\mathcal{R}_2$ is not simply connected. The ellipses mark where the density is $1/e$ times that at the peak of the distribution. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

The Normal Density

- Randomized Prototype Vectors with Mean $\vec{\mu} \rightarrow$ Normal Distribution

The Normal Density

- Randomized Prototype Vectors with Mean $\vec{\mu} \rightarrow$ Normal Distribution
- Expected Value
  $\mathcal{E}[f(x)] = \int_{-\infty}^{\infty} f(x)p(x)dx$ (continous)
  $\mathcal{E}[f(x)] = \sum_{x \in \mathcal{D}} f(x)P(x)$ (discrete)

The Normal Density

- Randomized Prototype Vectors with Mean $\vec{\mu} \rightarrow$ Normal Distribution
- Expected Value
  $\mathcal{E}[f(x)] = \int_{-\infty}^{\infty} f(x)p(x)dx$ (continous)
  $\mathcal{E}[f(x)] = \sum_{x \in \mathcal{D}} f(x)P(x)$ (discrete)
- Univariate Normal Density
  $p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$
  $\mathcal{E}[x] = \mu \quad \mathcal{E}[(x-\mu)^2] = \sigma^2$

Normal Distribution

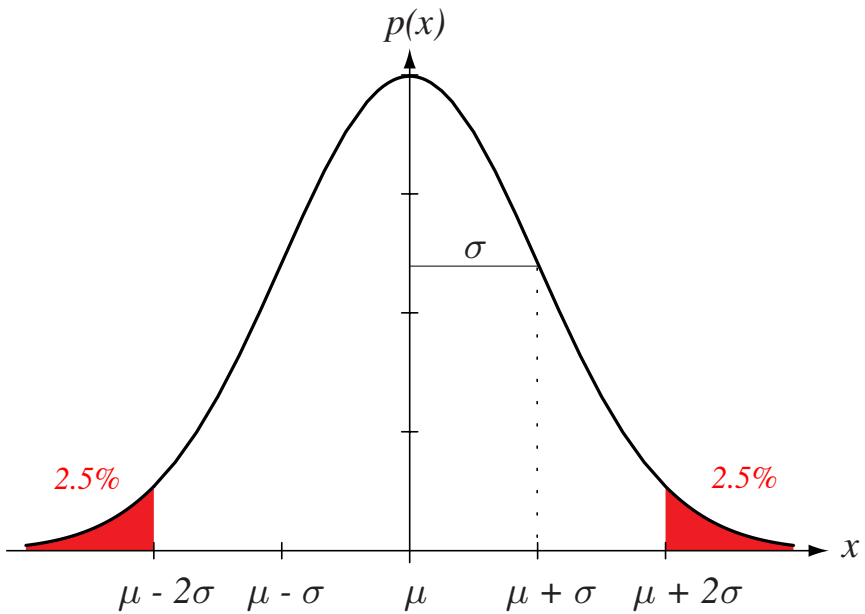


FIGURE 2.7. A univariate normal distribution has roughly 95% of its area in the range $|x - \mu| \leq 2\sigma$, as shown. The peak of the distribution has value $p(\mu) = 1/\sqrt{2\pi}\sigma$. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Multivariate Density

- Multivariate Normal Density

$$p(x) = \frac{1}{2\pi^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^t \Sigma^{-1}(\vec{x}-\vec{\mu})}$$

Multivariate Density

- Multivariate Normal Density
  $$p(x) = \frac{1}{2\pi^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^t \Sigma^{-1}(\vec{x}-\vec{\mu})}$$
- Covariance Matrix Σ $(d \times d)$
  $$\mathcal{E}[\vec{x}] = \vec{\mu} \quad \mathcal{E}[(\vec{x}-\vec{\mu})(\vec{x}-\vec{\mu})^t] = \Sigma$$
  $$\mathcal{E}[x_i] = \mu_i \quad \mathcal{E}[(x_i-\mu_i)(x_j-\mu_j)] = \sigma_{ij}$$

2D Gaussian

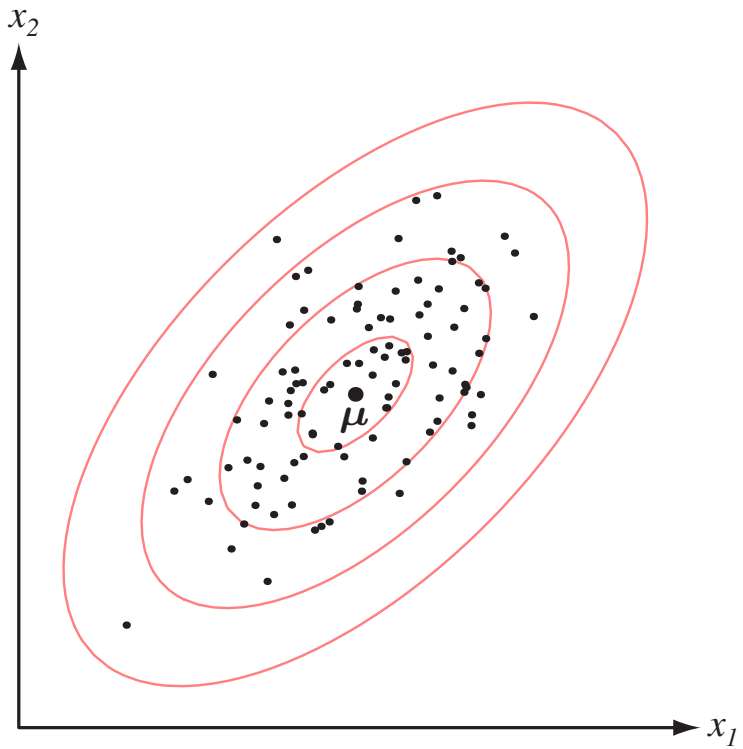


FIGURE 2.9. Samples drawn from a two-dimensional Gaussian lie in a cloud centered on the mean $\boldsymbol{\mu}$. The ellipses show lines of equal probability density of the Gaussian. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Four Categories

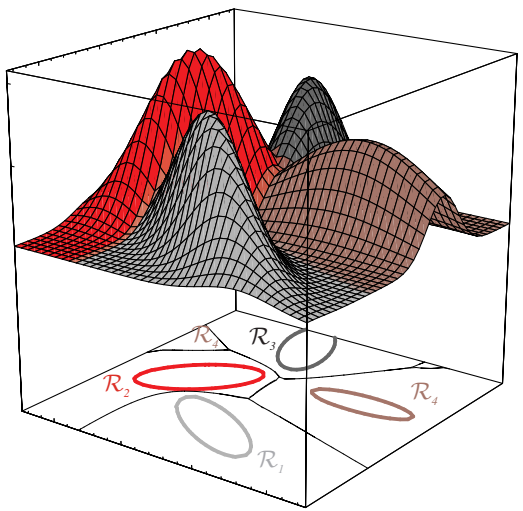


FIGURE 2.16. The decision regions for four normal distributions. Even with such a low number of categories, the shapes of the boundary regions can be rather complex. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Classification Errors

- Bayes Error: overlapping densities, inherent problem property

Classification Errors

- Bayes Error: overlapping densities, inherent problem property
- Model Error: incorrect model

Classification Errors

- Bayes Error: overlapping densities, inherent problem property
- Model Error: incorrect model
- Estimation Error: finite sample of training data

Bayes Error and Dimensionality

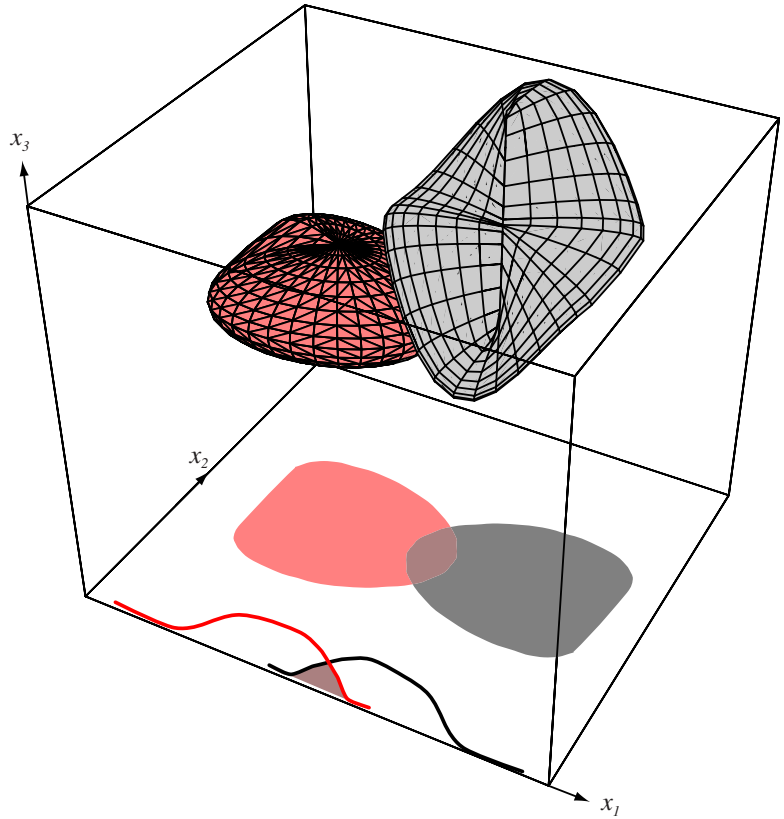


FIGURE 3.3. Two three-dimensional distributions have nonoverlapping densities, and thus in three dimensions the Bayes error vanishes. When projected to a subspace—here, the two-dimensional $x_1 - x_2$ subspace or a one-dimensional x_1 subspace—there can be greater overlap of the projected distributions, and hence greater Bayes error. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Unknown Densities

- Real problems: multi–modal, parametric densities: uni–modal
  $\rightarrow$ estimation of densities directly from data

Unknown Densities

- Real problems: multi–modal, parametric densities: uni–modal $\rightarrow$ estimation of densities directly from data
- P that pattern $\vec{x}$ falls in region $\mathcal{R}$, $P = \int_{\mathcal{R}} p(\vec{x}) d\vec{x}$

Unknown Densities

- Real problems: multi–modal, parametric densities: uni–modal
  $\rightarrow$ estimation of densities directly from data
- P that pattern $\vec{x}$ falls in region $\mathcal{R}$, $P = \int_{\mathcal{R}} p(\vec{x}) d\vec{x}$
- n patterns, probability that k patterns are in $\mathcal{R}$
  $P_k = \binom{n}{k} P^k (1-P)^{n-k} \qquad \mathcal{E}[k] = nP$

Unknown Densities

- Real problems: multi–modal, parametric densities: uni–modal $\rightarrow$ estimation of densities directly from data
- P that pattern $\vec{x}$ falls in region $\mathcal{R}$, $P = \int_{\mathcal{R}} p(\vec{x}) d\vec{x}$
- n patterns, probability that k patterns are in $\mathcal{R}$
  $P_k = \binom{n}{k} P^k (1-P)^{n-k} \qquad \mathcal{E}[k] = nP$
- Assuming small region $\mathcal{R} \rightarrow$
  $p(\vec{x}) \simeq const \rightarrow \int_{\mathcal{R}} p(\vec{x}) d\vec{x} \simeq p(\vec{x}) V \rightarrow p(\vec{x}) \simeq \frac{\frac{k}{n}}{V}$

Relative Probability

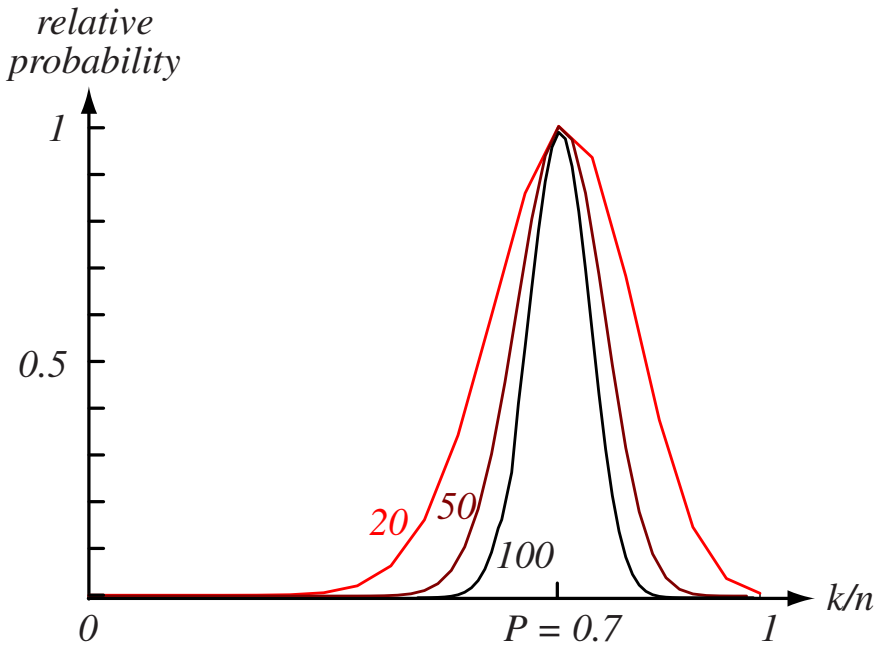


FIGURE 4.1. The relative probability an estimate given by Eq. 4 will yield a particular value for the probability density, here where the true probability was chosen to be 0.7. Each curve is labeled by the total number of patterns n sampled, and is scaled to give the same maximum (at the true probability). The form of each curve is binomial, as given by Eq. 2. For large n, such binomials peak strongly at the true probability. In the limit $n \to \infty$, the curve approaches a delta function, and we are guaranteed that our estimate will give the true probability. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Sample Size

- Estimate $p(\vec{x}) \simeq \frac{\frac{k}{n}}{V}$ is dependent on size of V
  if $V \to 0$, $p(\vec{x})$ would be exact, but no more samples in V

Sample Size

- Estimate $p(\vec{x}) \simeq \frac{\frac{k}{n}}{V}$ is dependent on size of V
  if $V \to 0$, $p(\vec{x})$ would be exact, but no more samples in V
- Assuming infinite pattern set with decreasing V_n
  n–th estimate $p_n(\vec{x}) = \frac{\frac{k_n}{n}}{V_n}$

Sample Size

- Estimate $p(\vec{x}) \simeq \frac{\frac{k}{n}}{V}$ is dependent on size of V
  if $V \to 0$, $p(\vec{x})$ would be exact, but no more samples in V
- Assuming infinite pattern set with decreasing V_n
  n–th estimate $p_n(\vec{x}) = \frac{\frac{k_n}{n}}{V_n}$
- For convergence of $p_n(\vec{x}) \to p(\vec{x})$
  $\lim_{n\to\infty} V_n = 0 \quad \lim_{n\to\infty} k_n = \infty \quad \lim_{n\to\infty} \frac{k_n}{n} = 0$

Sample Size

- Estimate $p(\vec{x}) \simeq \frac{\frac{k}{n}}{V}$ is dependent on size of V
  if $V \to 0$, $p(\vec{x})$ would be exact, but no more samples in V
- Assuming infinite pattern set with decreasing V_n
  n–th estimate $p_n(\vec{x}) = \frac{\frac{k_n}{n}}{V_n}$
- For convergence of $p_n(\vec{x}) \to p(\vec{x})$
  $\lim_{n\to\infty} V_n = 0 \quad \lim_{n\to\infty} k_n = \infty \quad \lim_{n\to\infty} \frac{k_n}{n} = 0$
- Decreasing V_n, e.g., $V_n = \frac{1}{\sqrt{n}} \to$ Parzen Windows
  Increasing k_n, e.g., $k_n = \sqrt{n} \to k_n$–Nearest–Neighbors

Point Density Estimation

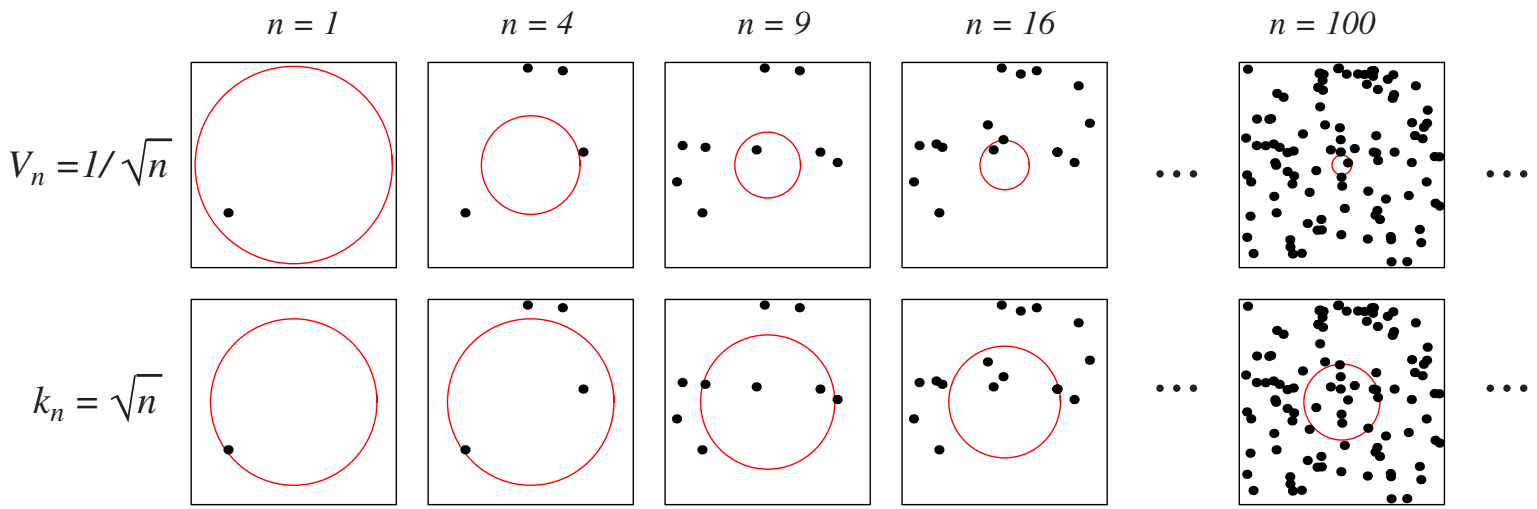


FIGURE 4.2. There are two leading methods for estimating the density at a point, here at the center of each square. The one shown in the top row is to start with a large volume centered on the test point and shrink it according to a function such as $V_n = 1/\sqrt{n}$. The other method, shown in the bottom row, is to decrease the volume in a data-dependent way, for instance letting the volume enclose some number $k_n = \sqrt{n}$ of sample points. The sequences in both cases represent random variables that generally converge and allow the true density at the test point to be calculated. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Prototype Estimation

- Estimate density at arbitrary $\vec{x}$ by k_n nearest neighbors of $\vec{x}$
  $p_n(\vec{x}) = \frac{\frac{k_n}{n}}{V_n}$ (neighbors are training patterns)

Prototype Estimation

- Estimate density at arbitrary $\vec{x}$ by k_n nearest neighbors of $\vec{x}$
  $p_n(\vec{x}) = \frac{\frac{k_n}{n}}{V_n}$ (neighbors are training patterns)
- Dense neighbors $\rightarrow$ small $V_n \rightarrow$ good resolution
  Sparse neighbors $\rightarrow$ large $V_n \rightarrow$ bad resolution

Prototype Estimation

- Estimate density at arbitrary $\vec{x}$ by k_n nearest neighbors of $\vec{x}$
  $p_n(\vec{x}) = \frac{\frac{k_n}{n}}{V_n}$ (neighbors are training patterns)
- Dense neighbors $\rightarrow$ small $V_n \rightarrow$ good resolution
  Sparse neighbors $\rightarrow$ large $V_n \rightarrow$ bad resolution
- Problem: often $\int p_n(\vec{x}) d\vec{x} > 1$

1D kNN Estimate

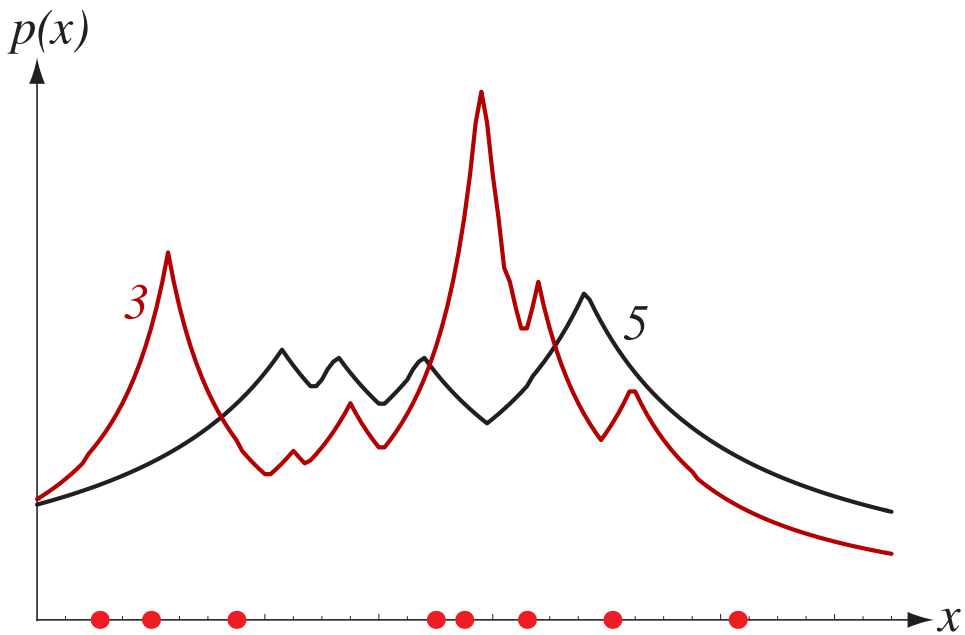


FIGURE 4.10. Eight points in one dimension and the k-nearest-neighbor density estimates, for $k = 3$ and 5. Note especially that the discontinuities in the slopes in the estimates generally lie *away* from the positions of the prototype points. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

2D kNN Estimate

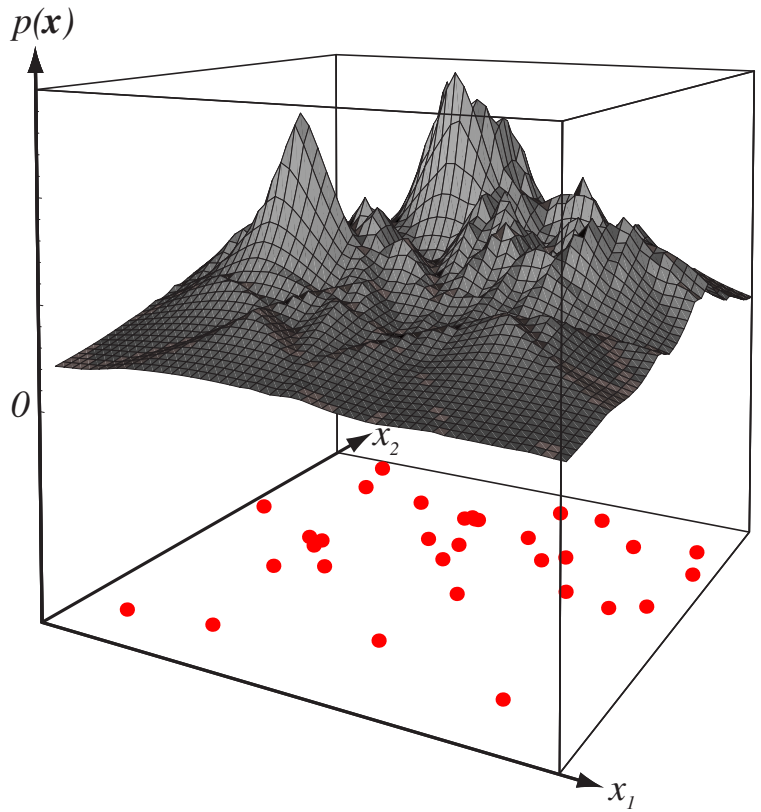


FIGURE 4.11. The k-nearest-neighbor estimate of a two-dimensional density for $k = 5$. Notice how such a finite n estimate can be quite "jagged," and notice that discontinuities in the slopes generally occur along lines away from the positions of the points themselves. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Unimodal and Bimodal 1D kNN Estimates

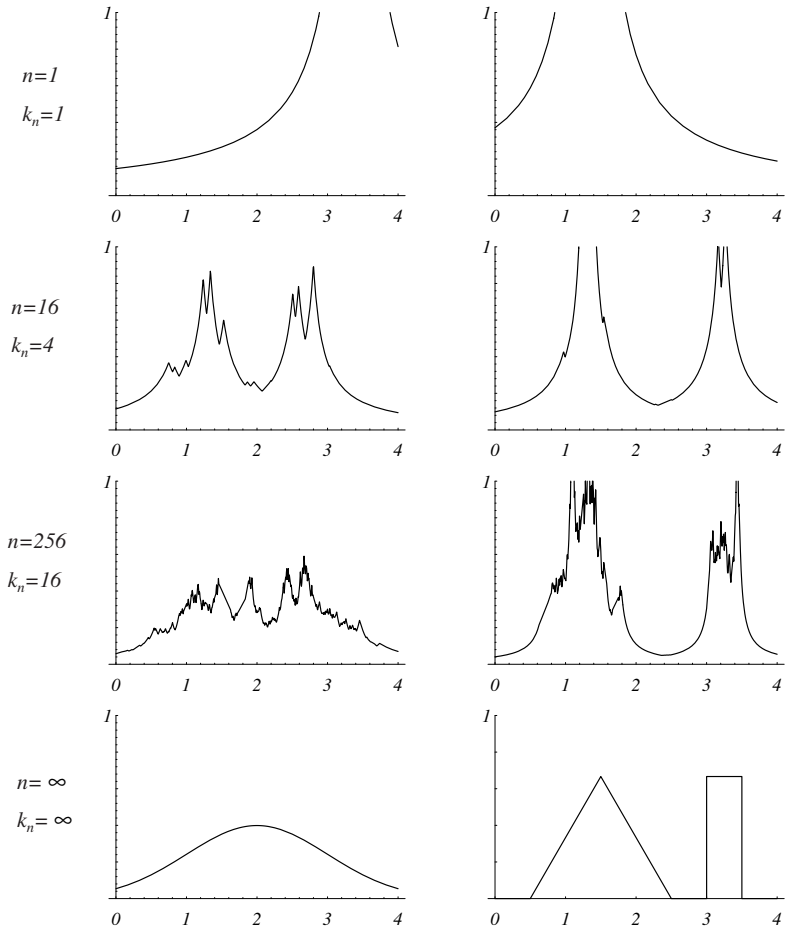


FIGURE 4.12. Several k-nearest-neighbor estimates of two unidimensional densities: a Gaussian and a bimodal distribution. Notice how the finite n estimates can be quite "spiky." From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Estimation of A Posteriori Probabilities

- Samples of different classes, what is $P(\omega_i|\vec{x})$?

  Estimate for $p_n(\vec{x}, \omega_i) = \frac{\frac{k_i}{n}}{V}$ (in arbitrary V)

Estimation of A Posteriori Probabilities

- Samples of different classes, what is $P(\omega_i|\vec{x})$?

  Estimate for $p_n(\vec{x}, \omega_i) = \frac{\frac{k_i}{n}}{V}$ (in arbitrary V)

- Estimate for $P(\omega_i|\vec{x}) = \frac{p_n(\vec{x},\omega_i)}{\sum_{j=1}^{c} p_n(\vec{x},\omega_j)} = \frac{k_i}{k}$

Estimation of A Posteriori Probabilities

- Samples of different classes, what is $P(\omega_i|\vec{x})$?

  Estimate for $p_n(\vec{x},\omega_i) = \frac{\frac{k_i}{n}}{V}$ (in arbitrary V)

- Estimate for $P(\omega_i|\vec{x}) = \frac{p_n(\vec{x},\omega_i)}{\sum_{j=1}^{c} p_n(\vec{x},\omega_j)} = \frac{k_i}{k}$

- With $n \to \infty$ and Bayes Rule: optimal performance (Parzen and kNN)

Nearest Neigbor Rule

- Single nearest neigbor is $\vec{x'}$ $(k = 1)$
  Class label of $\vec{x'}$ is θ' (random variable)
  $P(\theta' = \omega_i) = P(\omega_i|\vec{x'}) \simeq P(\omega_i|\vec{x})$ (for large n)

Nearest Neigbor Rule

- Single nearest neigbor is $\vec{x'}$ $(k = 1)$
  Class label of $\vec{x'}$ is θ' (random variable)
  $P(\theta' = \omega_i) = P(\omega_i|\vec{x'}) \simeq P(\omega_i|\vec{x})$ (for large n)
- Assumption of 1NN: $P(\omega_i|\vec{x'})$ is largest probability
  If true (e.g., $P \simeq 1$, or $P \simeq \frac{1}{c}$), then 1NN close to Bayes Error

Nearest Neigbor Rule

- Single nearest neigbor is $\vec{x'}$ $(k = 1)$
  Class label of $\vec{x'}$ is θ' (random variable)
  $P(\theta' = \omega_i) = P(\omega_i|\vec{x'}) \simeq P(\omega_i|\vec{x})$ (for large n)
- Assumption of 1NN: $P(\omega_i|\vec{x'})$ is largest probability
  If true (e.g., $P \simeq 1$, or $P \simeq \frac{1}{c}$), then 1NN close to Bayes Error
- Average error probability $P(e) = \int P(e|\vec{x})p(\vec{x})d\vec{x}$
  $P(e|\vec{x}) = 1 - P(\omega_i|\vec{x'})$ is "minimum" $P^*(e|\vec{x})$
  $P^*(e) = \int P^*(e|\vec{x})p(\vec{x})d\vec{x}$

Nearest Neigbor Rule

- Single nearest neigbor is $\vec{x'}$ $(k = 1)$
  Class label of $\vec{x'}$ is θ' (random variable)
  $P(\theta' = \omega_i) = P(\omega_i|\vec{x'}) \simeq P(\omega_i|\vec{x})$ (for large n)
- Assumption of 1NN: $P(\omega_i|\vec{x'})$ is largest probability
  If true (e.g., $P \simeq 1$, or $P \simeq \frac{1}{c}$), then 1NN close to Bayes Error
- Average error probability $P(e) = \int P(e|\vec{x})p(\vec{x})d\vec{x}$
  $P(e|\vec{x}) = 1 - P(\omega_i|\vec{x'})$ is "minimum" $P^*(e|\vec{x})$
  $P^*(e) = \int P^*(e|\vec{x})p(\vec{x})d\vec{x}$
- 1NN error $P = \lim_{n\to\infty} P_n(e)$
  $P^* \leq P \leq P^*(2 - \frac{c}{c-1}P^*)$

Voronoi Tesselation

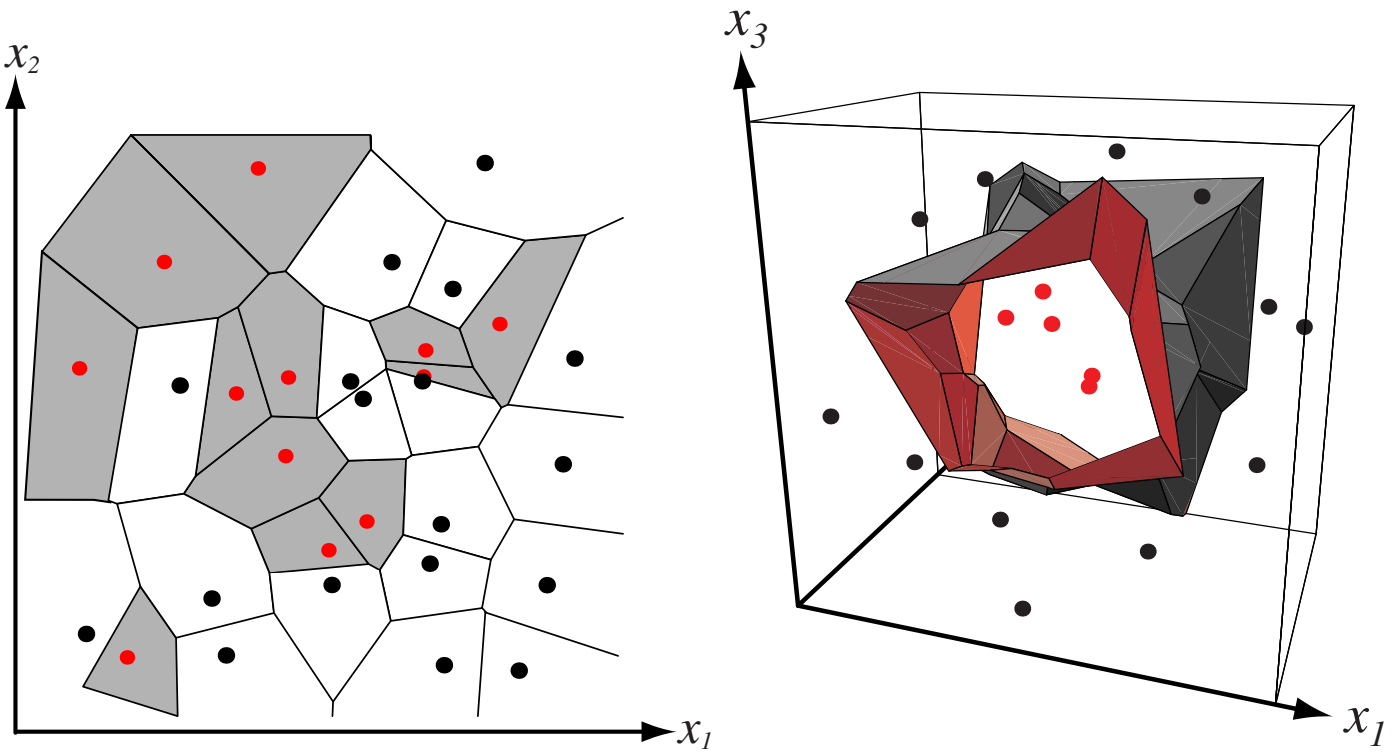


FIGURE 4.13. In two dimensions, the nearest-neighbor algorithm leads to a partitioning of the input space into Voronoi cells, each labeled by the category of the training point it contains. In three dimensions, the cells are three-dimensional, and the decision boundary resembles the surface of a crystal. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

1NN Error Rate Bounds

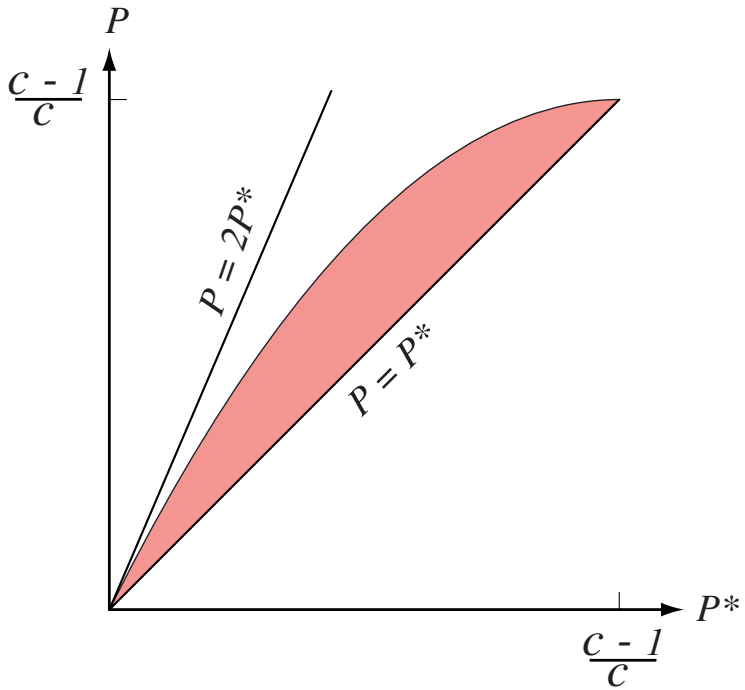


FIGURE 4.14. Bounds on the nearest-neighbor error rate P in a c-category problem given infinite training data, where P^* is the Bayes error (Eq. 52). At low error rates, the nearest-neighbor error rate is bounded above by twice the Bayes rate. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

k–Nearest–Neigbor Rule

- Straight–forward extension: k neighbors

k–Nearest–Neigbor Rule

- Straight–forward extension: k neighbors
- Majority voting: $P(\omega_m|\vec{x})$ is largest probability (most prototypes in class m)

k–Nearest–Neigbor Rule

- Straight–forward extension: k neighbors
- Majority voting: $P(\omega_m|\vec{x})$ is largest probability (most prototypes in class m)
- If $k \to \infty$ then k–NN rule becomes optimal

5NN in 2D

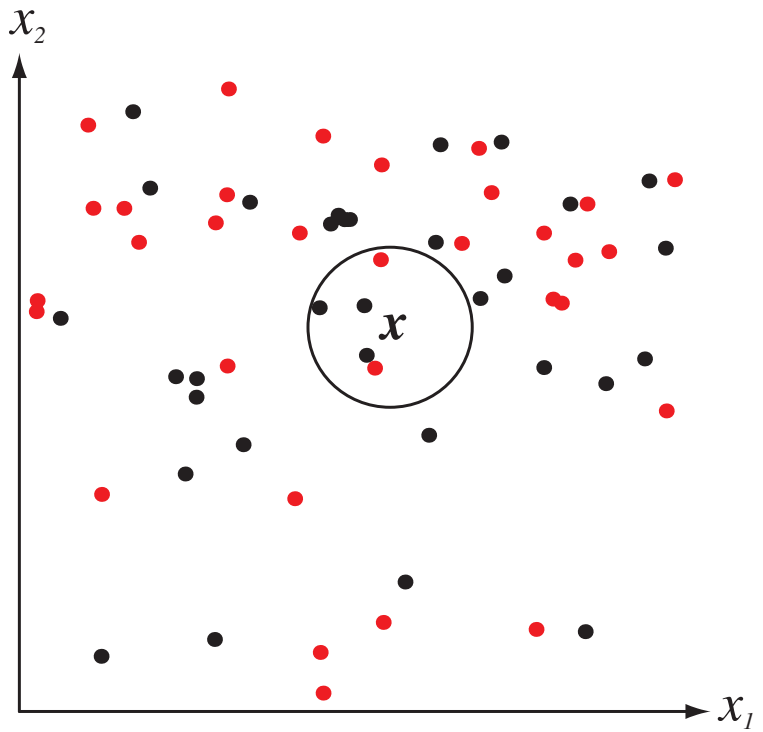


FIGURE 4.15. The k-nearest-neighbor query starts at the test point **x** and grows a spherical region until it encloses k training samples, and it labels the test point by a majority vote of these samples. In this $k = 5$ case, the test point **x** would be labeled the category of the black points. From: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

kNN Error Rate Bounds

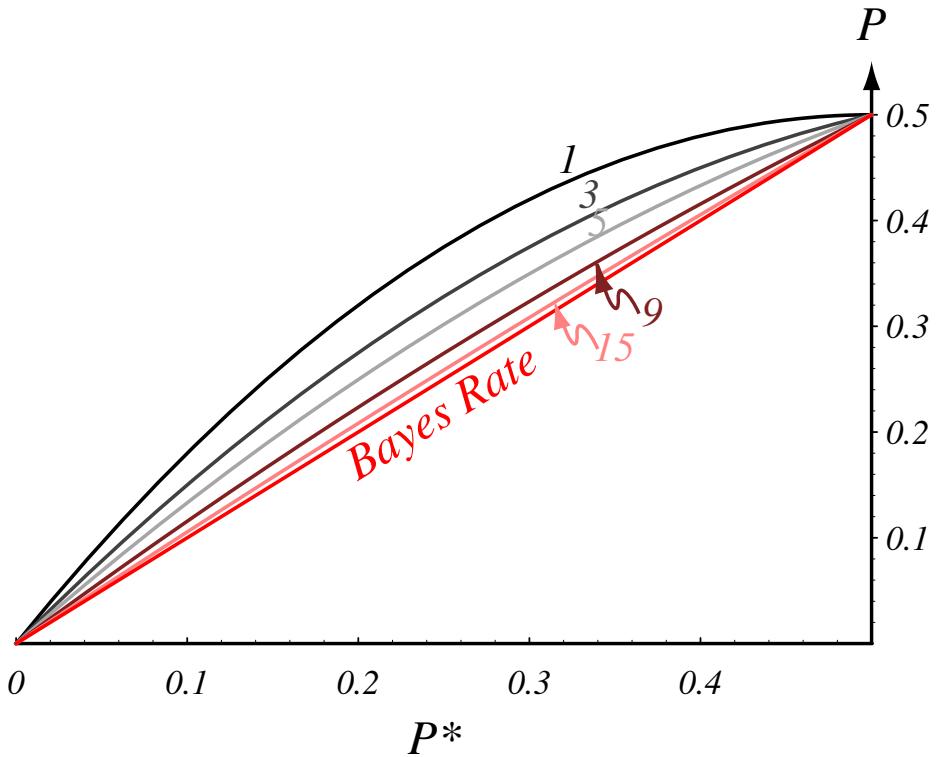


FIGURE 4.16. The error rate for the k-nearest-neighbor rule for a two-category problem is bounded by $C_k(P^*)$ in Eq. 54. Each curve is labeled by k; when $k = \infty$, the estimated probabilities match the true probabilities and thus the error rate is equal to the Bayes rate, that is, $P = P^*$. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Metrics

- What is a distance?

Metrics

- What is a distance?
- Properties of Metrics
  - Nonnegativity: $D(\vec{a}, \vec{b}) \geq 0$
  - Reflexivity: $D(\vec{a}, \vec{b}) = 0$ iff $\vec{a} = \vec{b}$
  - Symmetry: $D(\vec{a}, \vec{b}) = D(\vec{b}, \vec{a})$
  - Triangle inequality: $D(\vec{a}, \vec{b}) + D(\vec{b}, \vec{c}) \geq D(\vec{a}, \vec{c})$

Metrics

- What is a distance?
- Properties of Metrics
  - Nonnegativity: $D(\vec{a}, \vec{b}) \geq 0$
  - Reflexivity: $D(\vec{a}, \vec{b}) = 0$ iff $\vec{a} = \vec{b}$
  - Symmetry: $D(\vec{a}, \vec{b}) = D(\vec{b}, \vec{a})$
  - Triangle inequality: $D(\vec{a}, \vec{b}) + D(\vec{b}, \vec{c}) \geq D(\vec{a}, \vec{c})$
- Scaling of feature values equivalent to changing the metric

Scaling is Change of Metric

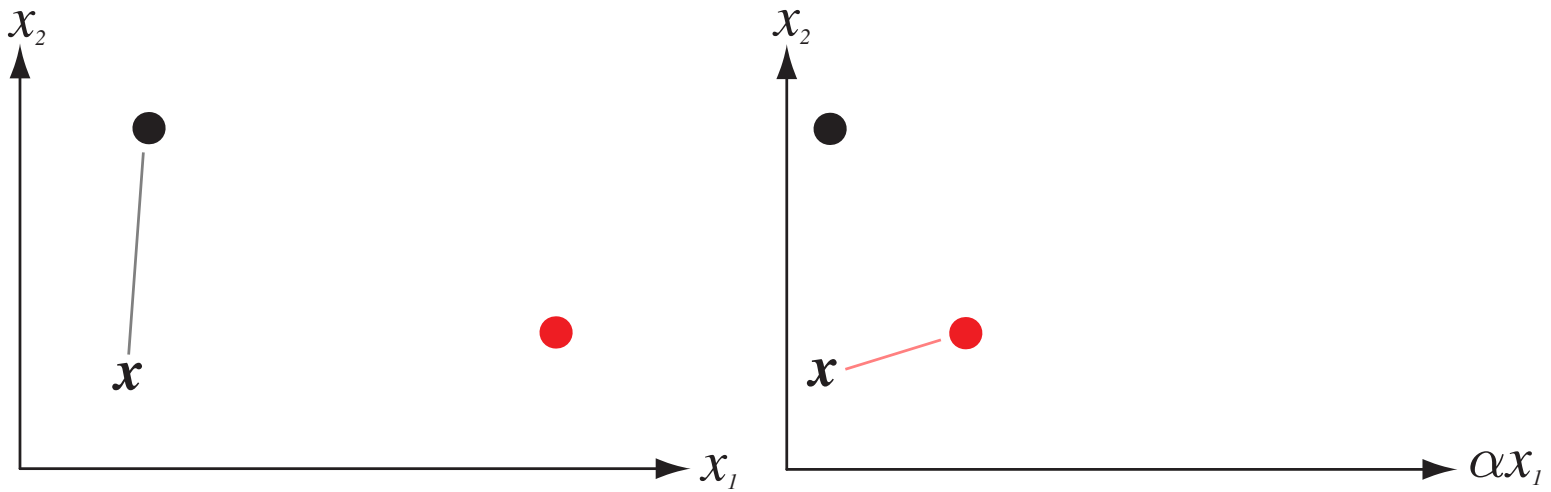


FIGURE 4.18. Scaling the coordinates of a feature space can change the distance relationships computed by the Euclidean metric. Here we see how such scaling can change the behavior of a nearest-neighbor classifer. Consider the test point **x** and its nearest neighbor. In the original space (left), the black prototype is closest. In the figure at the right, the x_1 axis has been rescaled by a factor 1/3; now the nearest prototype is the red one. If there is a large disparity in the ranges of the full data in each dimension, a common procedure is to rescale all the data to equalize such ranges, and this is equivalent to changing the metric in the original space. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Class of Metrics

- Minkowski Metric (L_k Norm) $L_k(\vec{a}, \vec{b}) = (\sum_{i=1}^{d} |a_i - b_i|^k)^{\frac{1}{k}}$

Class of Metrics

- Minkowski Metric (L_k Norm) $L_k(\vec{a}, \vec{b}) = (\sum_{i=1}^{d} |a_i - b_i|^k)^{\frac{1}{k}}$
- L_1 Norm: Manhattan distance

Class of Metrics

- Minkowski Metric (L_k Norm) $L_k(\vec{a}, \vec{b}) = (\sum_{i=1}^{d} |a_i - b_i|^k)^{\frac{1}{k}}$
- L_1 Norm: Manhattan distance
- L_2 Norm: Euclidean distance

Class of Metrics

- Minkowski Metric (L_k Norm) $L_k(\vec{a}, \vec{b}) = (\sum_{i=1}^{d} |a_i - b_i|^k)^{\frac{1}{k}}$
- L_1 Norm: Manhattan distance
- L_2 Norm: Euclidean distance
- L_∞ Norm: Maximum of projected distances

Minkowski Metric

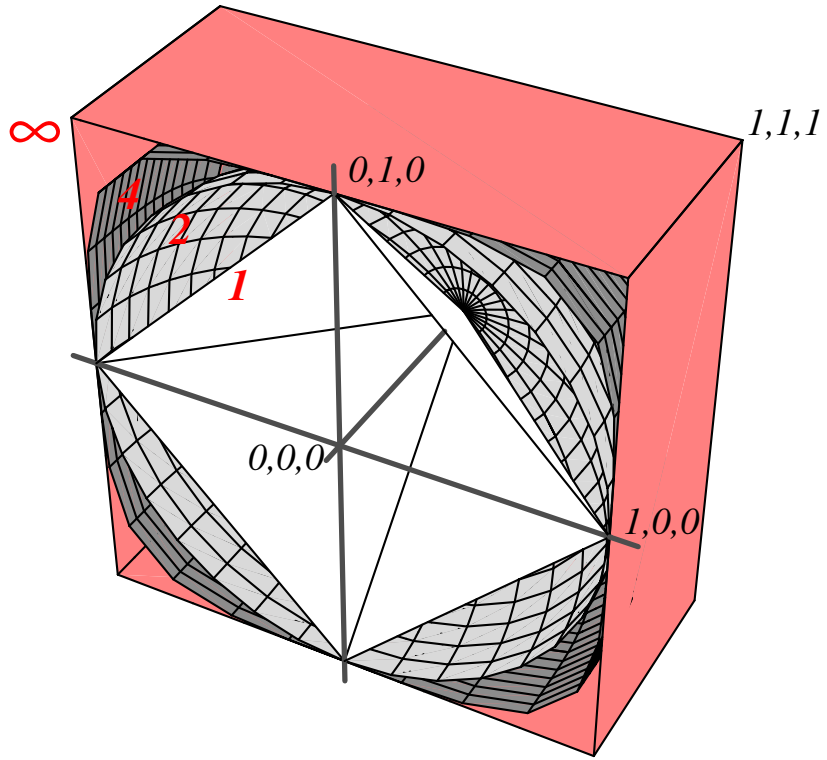


FIGURE 4.19. Each colored surface consists of points a distance 1.0 from the origin, measured using different values for k in the Minkowski metric (k is printed in red). Thus the white surfaces correspond to the L_1 norm (Manhattan distance), the light gray sphere corresponds to the L_2 norm (Euclidean distance), the dark gray ones correspond to the L_4 norm, and the pink box corresponds to the L_∞ norm. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Discriminant Functions

- Assumption: we know the form of discriminant functions (not probability densities)

Discriminant Functions

- Assumption: we know the form of discriminant functions (not probability densities)
- Problem: determine parameters of discriminant functions

Discriminant Functions

- Assumption: we know the form of discriminant functions (not probability densities)
- Problem: determine parameters of discriminant functions
- Method: gradient descent of criterion functions (based on training set)

Linear Classifier

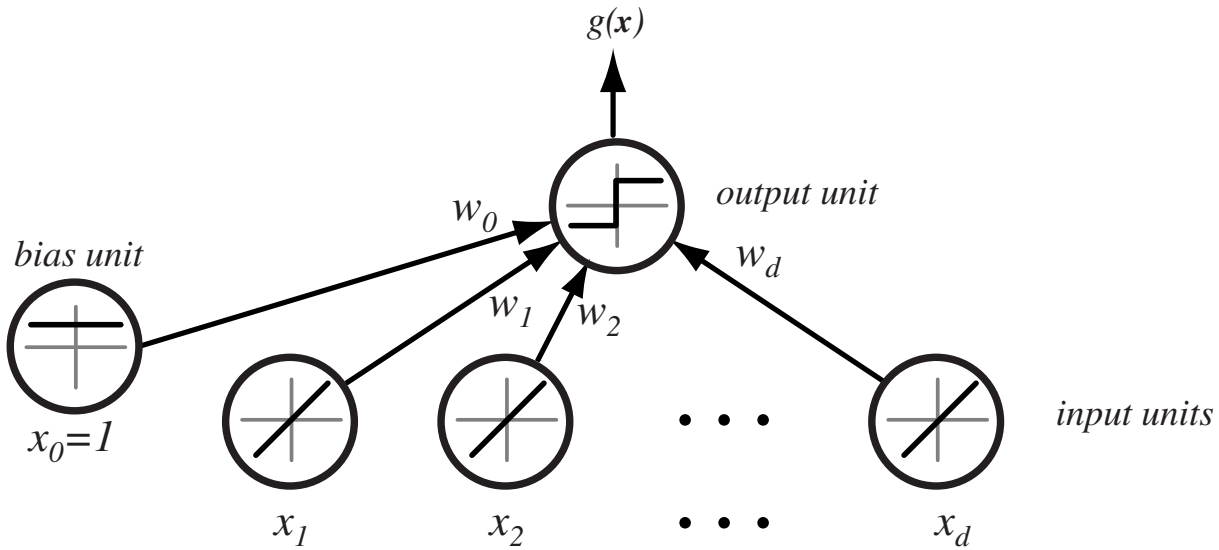


FIGURE 5.1. A simple linear classifier having d input units, each corresponding to the values of the components of an input vector. Each input feature value x_i is multiplied by its corresponding weight w_i; the effective input at the output unit is the sum all these products, $\sum w_i x_i$. We show in each unit its effective input-output function. Thus each of the d input units is linear, emitting exactly the value of its corresponding feature value. The single bias unit unit always emits the constant value 1.0. The single output unit emits a $+1$ if $\mathbf{w}^t\mathbf{x} + w_0 > 0$ or a -1 otherwise. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Linear Discriminant Functions

- Linear discriminant function $g(\vec{x}) = \vec{w}^t \vec{x} + w_0$ (weight vector $\vec{w}$, bias w_0)

Linear Discriminant Functions

- Linear discriminant function $g(\vec{x}) = \vec{w}^t\vec{x} + w_0$ (weight vector $\vec{w}$, bias w_0)
- Two classes: $g(\vec{x}) > 0 \rightarrow \omega_1$, else ω_2 or $\vec{w}^t\vec{x} > -w_0$

Linear Discriminant Functions

- Linear discriminant function $g(\vec{x}) = \vec{w}^t\vec{x} + w_0$
  (weight vector $\vec{w}$, bias w_0)
- Two classes: $g(\vec{x}) > 0 \rightarrow \omega_1$, else ω_2
  or $\vec{w}^t\vec{x} > -w_0$
- Decision surface is hyperplane, $\vec{x}_1$, $\vec{x}_2$ on boundary
  $\vec{w}^t\vec{x}_1 + w_0 = \vec{w}^t\vec{x}_2 + w_0 \rightarrow \vec{w}^t(\vec{x}_1 - \vec{x}_2) = 0$
  ($\vec{w}$ is normal vector)

Linear Discriminant Functions

- Linear discriminant function $g(\vec{x}) = \vec{w}^t\vec{x} + w_0$
  (weight vector $\vec{w}$, bias w_0)
- Two classes: $g(\vec{x}) > 0 \rightarrow \omega_1$, else ω_2
  or $\vec{w}^t\vec{x} > -w_0$
- Decision surface is hyperplane, $\vec{x}_1$, $\vec{x}_2$ on boundary
  $\vec{w}^t\vec{x}_1 + w_0 = \vec{w}^t\vec{x}_2 + w_0 \rightarrow \vec{w}^t(\vec{x}_1 - \vec{x}_2) = 0$
  ($\vec{w}$ is normal vector)
- Hyperplane H divides space in two half–spaces
  $\mathcal{R}_1$ is positive side ($g(\vec{x}) > 0$), $\mathcal{R}_2$ is negative side ($g(\vec{x}) < 0$)

Multiple Classes

- Variant: c dichotomizers (ω_i, not ω_i)

Multiple Classes

- Variant: c dichotomizers (ω_i, not ω_i)
- Variant: $\frac{c(c-1)}{2}$ dichotomizers (all class pairs)

Multiple Classes

- Variant: c dichotomizers (ω_i, not ω_i)
- Variant: $\frac{c(c-1)}{2}$ dichotomizers (all class pairs)
- Variant: linear machine, discriminant functions $g_i(\vec{x}), \quad i = 1, \ldots, c$

Multiple Classes

- Variant: c dichotomizers (ω_i, not ω_i)
- Variant: $\frac{c(c-1)}{2}$ dichotomizers (all class pairs)
- Variant: linear machine, discriminant functions
  $g_i(\vec{x}), \quad i = 1, \ldots, c$
- Decision boundary
  $g_i(\vec{x}) = g_j(\vec{x}) \rightarrow (\vec{w}_i - \vec{w}_j)^t \vec{x} + (w_{i0} - w_{j0}) = 0$
  $(\vec{w}_i - \vec{w}_j) \perp H_{ij}$, $r = \frac{g_i(\vec{x}) - g_j(\vec{x})}{||\vec{w}_i - \vec{w}_j||}$

Dichotomizers in a Four–class Problem

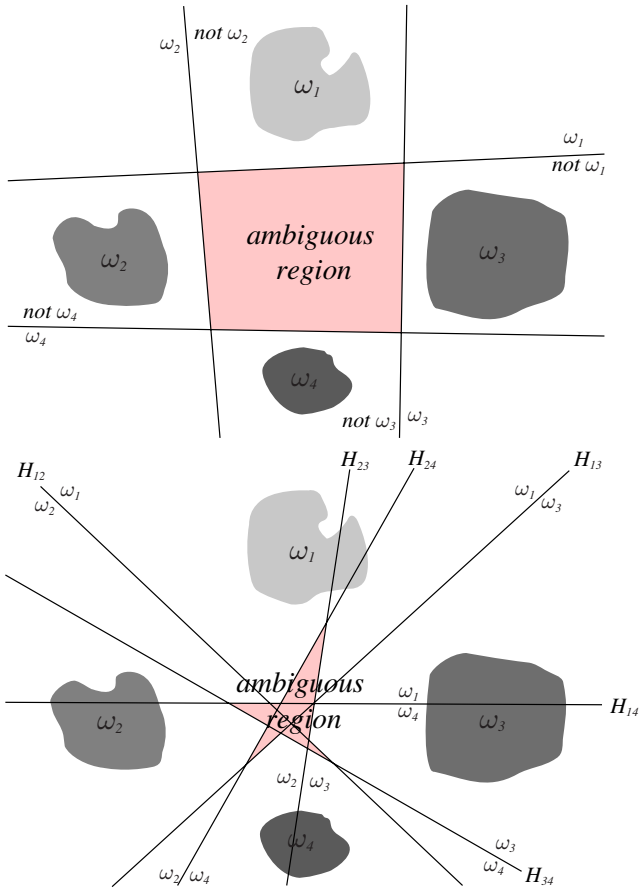


FIGURE 5.3. Linear decision boundaries for a four-class problem. The top figure shows ω_i/not ω_i dichotomies while the bottom figure shows ω_i/ω_j dichotomies and the corresponding decision boundaries H_{ij}. The pink regions have ambiguous category assignments. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Linear Machines in Multi–class Problems

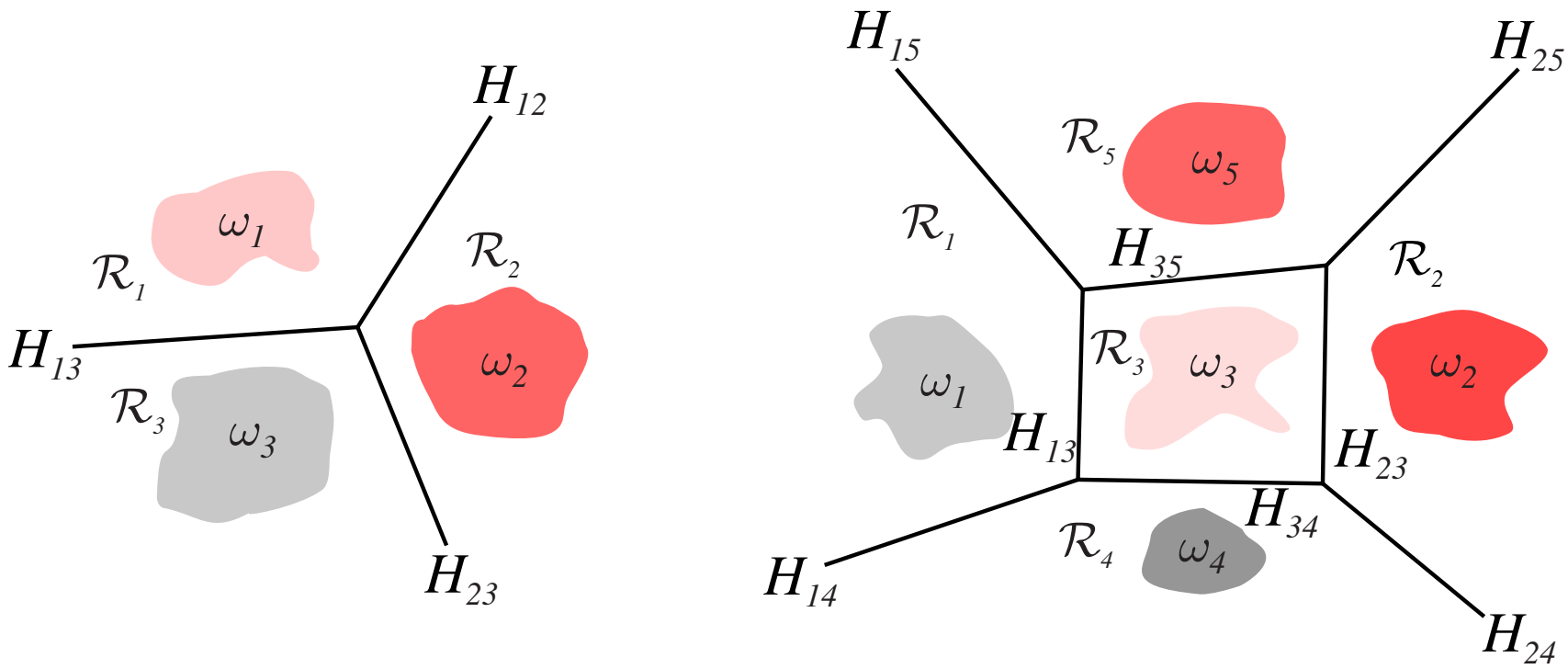


FIGURE 5.4. Decision boundaries produced by a linear machine for a three-class problem and a five-class problem. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Generalized Linear Discriminant Functions

- More complex decision boundaries
  e.g., quadratic discriminant
  $g(\vec{x}) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=1}^{d} w_{ij} x_i x_j$

Generalized Linear Discriminant Functions

- More complex decision boundaries
  e.g., quadratic discriminant
  $g(\vec{x}) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=1}^{d} w_{ij} x_i x_j$
- Generalized LDF $g(\vec{x}) = \sum_{i=1}^{\hat{d}} a_i y_i(\vec{x}) = \vec{a}^t \vec{y}$
  $\hat{d}$ $y_i(\vec{x})$ functions map points from d–dimensional $\vec{x}$–space to $\hat{d}$–dimensional $\vec{y}$–space

Generalized Linear Discriminant Functions

- More complex decision boundaries
  e.g., quadratic discriminant
  $g(\vec{x}) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=1}^{d} w_{ij} x_i x_j$
- Generalized LDF $g(\vec{x}) = \sum_{i=1}^{\hat{d}} a_i y_i(\vec{x}) = \vec{a}^t \vec{y}$
  $\hat{d}$ $y_i(\vec{x})$ functions map points from d–dimensional $\vec{x}$–space to $\hat{d}$–dimensional $\vec{y}$–space
- Example $g(x) = a_1 + a_2 x + a_3 x^2$, $\vec{y} = \begin{pmatrix} 1 \\ x \\ x^2 \end{pmatrix}$

Generalized Linear Discriminant Functions

- More complex decision boundaries
  e.g., quadratic discriminant
  $g(\vec{x}) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=1}^{d} w_{ij} x_i x_j$
- Generalized LDF $g(\vec{x}) = \sum_{i=1}^{\hat{d}} a_i y_i(\vec{x}) = \vec{a}^t \vec{y}$
  $\hat{d}$ $y_i(\vec{x})$ functions map points from d–dimensional $\vec{x}$–space to $\hat{d}$–dimensional $\vec{y}$–space
- Example $g(x) = a_1 + a_2 x + a_3 x^2$, $\vec{y} = \begin{pmatrix} 1 \\ x \\ x^2 \end{pmatrix}$
- Decision boundary is linear in $\vec{y}$–space
  Transformed density $p(x)$ is degenerate
  If d is large, huge number of parameters
  (requires large training data set)

From 1D to 3D

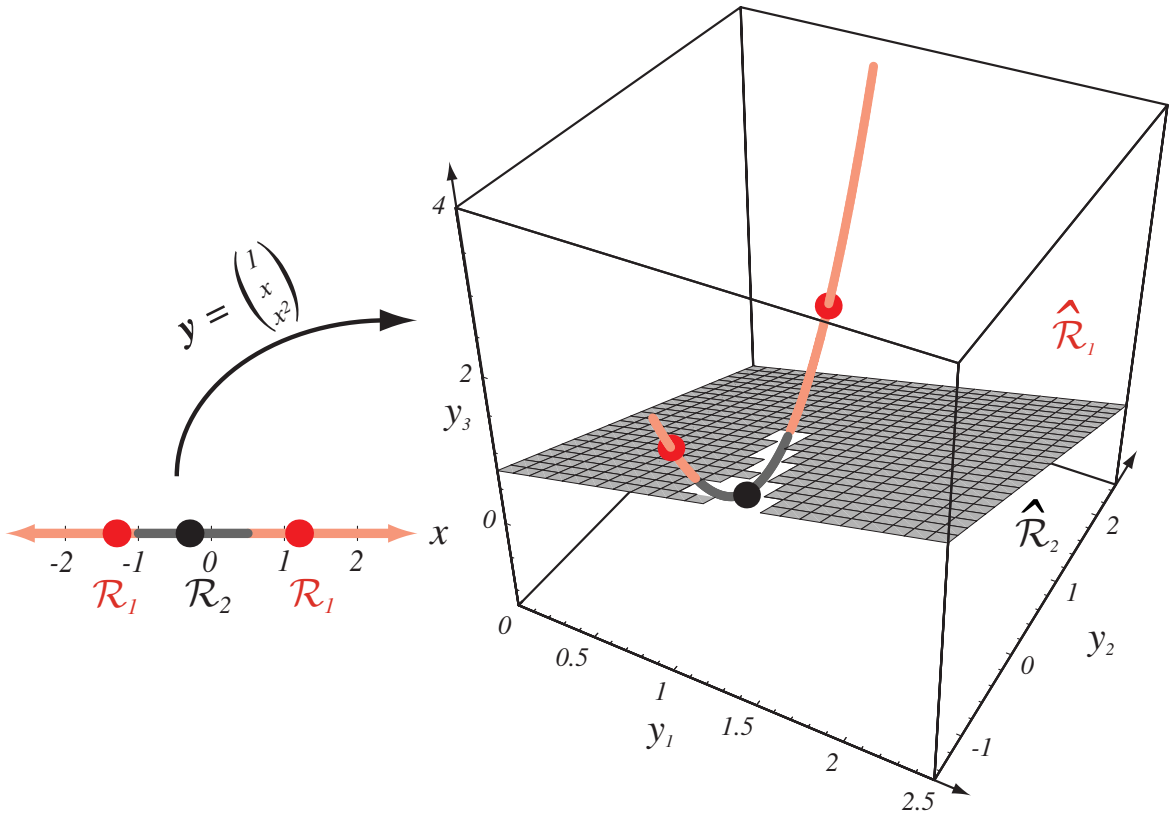


FIGURE 5.5. The mapping $\mathbf{y} = (1, x, x^2)^t$ takes a line and transforms it to a parabola in three dimensions. A plane splits the resulting $\mathbf{y}$-space into regions corresponding to two categories, and this in turn gives a nonsimply connected decision region in the one-dimensional x-space. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

From 2D to 3D

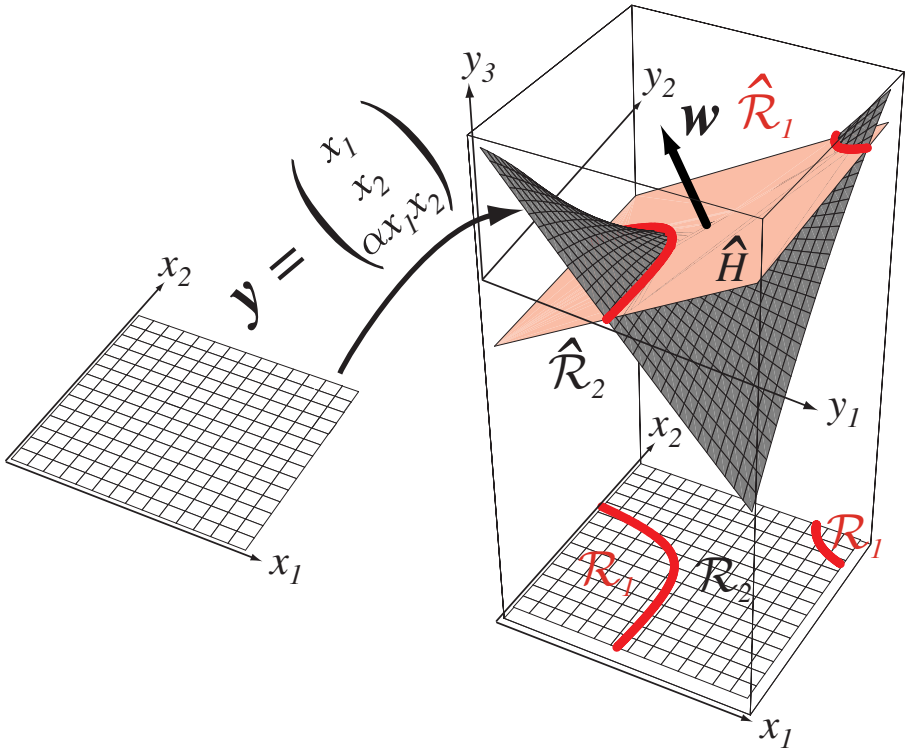


FIGURE 5.6. The two-dimensional input space **x** is mapped through a polynomial function f to **y**. Here the mapping is $y_1 = x_1$, $y_2 = x_2$ and $y_3 \propto x_1 x_2$. A linear discriminant in this transformed space is a hyperplane, which cuts the surface. Points to the positive side of the hyperplane $\hat{H}$ correspond to category ω_1, and those beneath it correspond to category ω_2. Here, in terms of the **x** space, $\mathcal{R}_1$ is a not simply connected. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Linearly Separable Dichotomy

- Two classes, samples $\vec{y}_i$, $\vec{a}^t\vec{y}_i > 0 \rightarrow \omega_1$, $\vec{a}^t\vec{y}_i < 0 \rightarrow \omega_2$

Linearly Separable Dichotomy

- Two classes, samples $\vec{y}_i$, $\vec{a}^t\vec{y}_i > 0 \rightarrow \omega_1$, $\vec{a}^t\vec{y}_i < 0 \rightarrow \omega_2$
- "Normalization" of ω_2: $\vec{y}_i = -\vec{y}_i \rightarrow \vec{a}^t\vec{y}_i > 0 \quad \forall\vec{y}_i$

Linearly Separable Dichotomy

- Two classes, samples $\vec{y}_i$, $\vec{a}^t\vec{y}_i > 0 \rightarrow \omega_1$, $\vec{a}^t\vec{y}_i < 0 \rightarrow \omega_2$
- "Normalization" of ω_2: $\vec{y}_i = -\vec{y}_i \rightarrow \vec{a}^t\vec{y}_i > 0 \quad \forall\vec{y}_i$
- Solution region defines all possible values of $\vec{a}$
  intersection of n half–spaces ($\vec{a}^t\vec{y}_i = 0$)

Linearly Separable Dichotomy

- Two classes, samples $\vec{y}_i$, $\vec{a}^t\vec{y}_i > 0 \rightarrow \omega_1$, $\vec{a}^t\vec{y}_i < 0 \rightarrow \omega_2$
- "Normalization" of ω_2: $\vec{y}_i = -\vec{y}_i \rightarrow \vec{a}^t\vec{y}_i > 0 \quad \forall\vec{y}_i$
- Solution region defines all possible values of $\vec{a}$
  intersection of n half–spaces ($\vec{a}^t\vec{y}_i = 0$)
- Margin $b > 0$, $\vec{a}^t\vec{y}_i \geq b$, new solution region has distance $\frac{b}{||\vec{y}_i||}$
  from old boundaries

Solution Region and Normalization

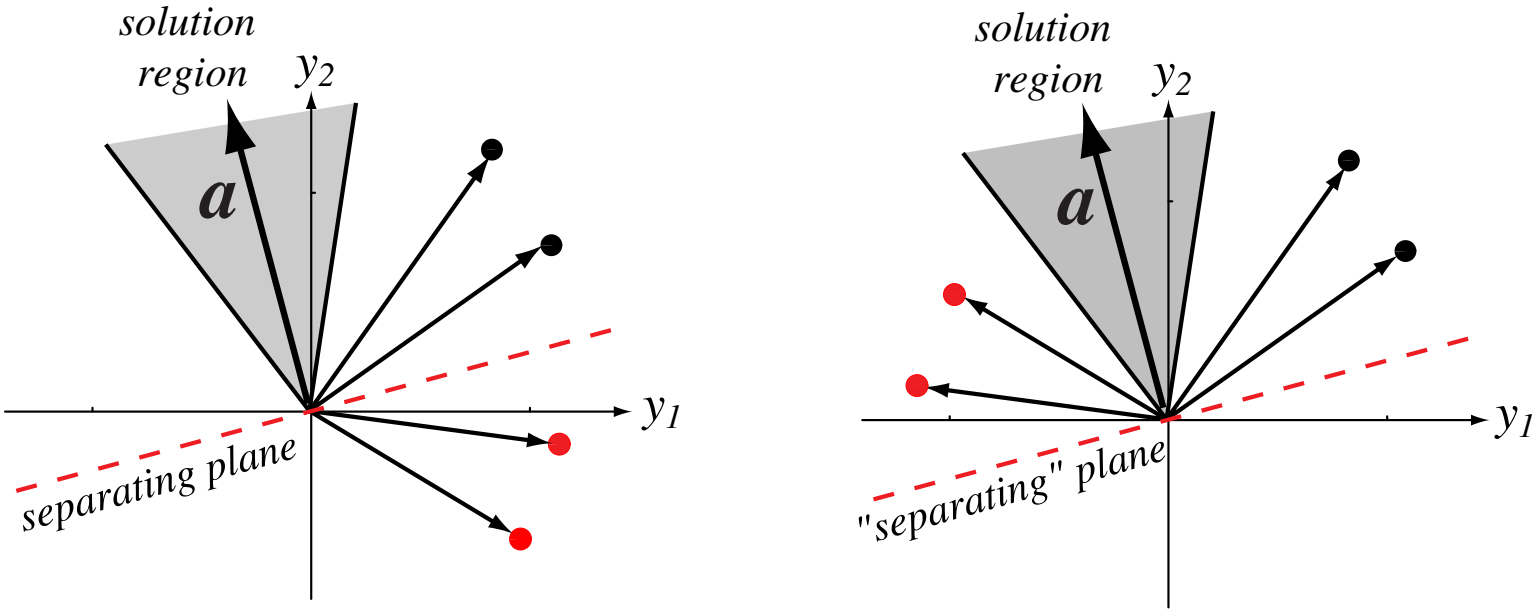


FIGURE 5.8. Four training samples (black for ω_1, red for ω_2) and the solution region in feature space. The figure on the left shows the raw data; the solution vectors leads to a plane that separates the patterns from the two categories. In the figure on the right, the red points have been "normalized"—that is, changed in sign. Now the solution vector leads to a plane that places all "normalized" points on the same side. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Solution Region with Margins

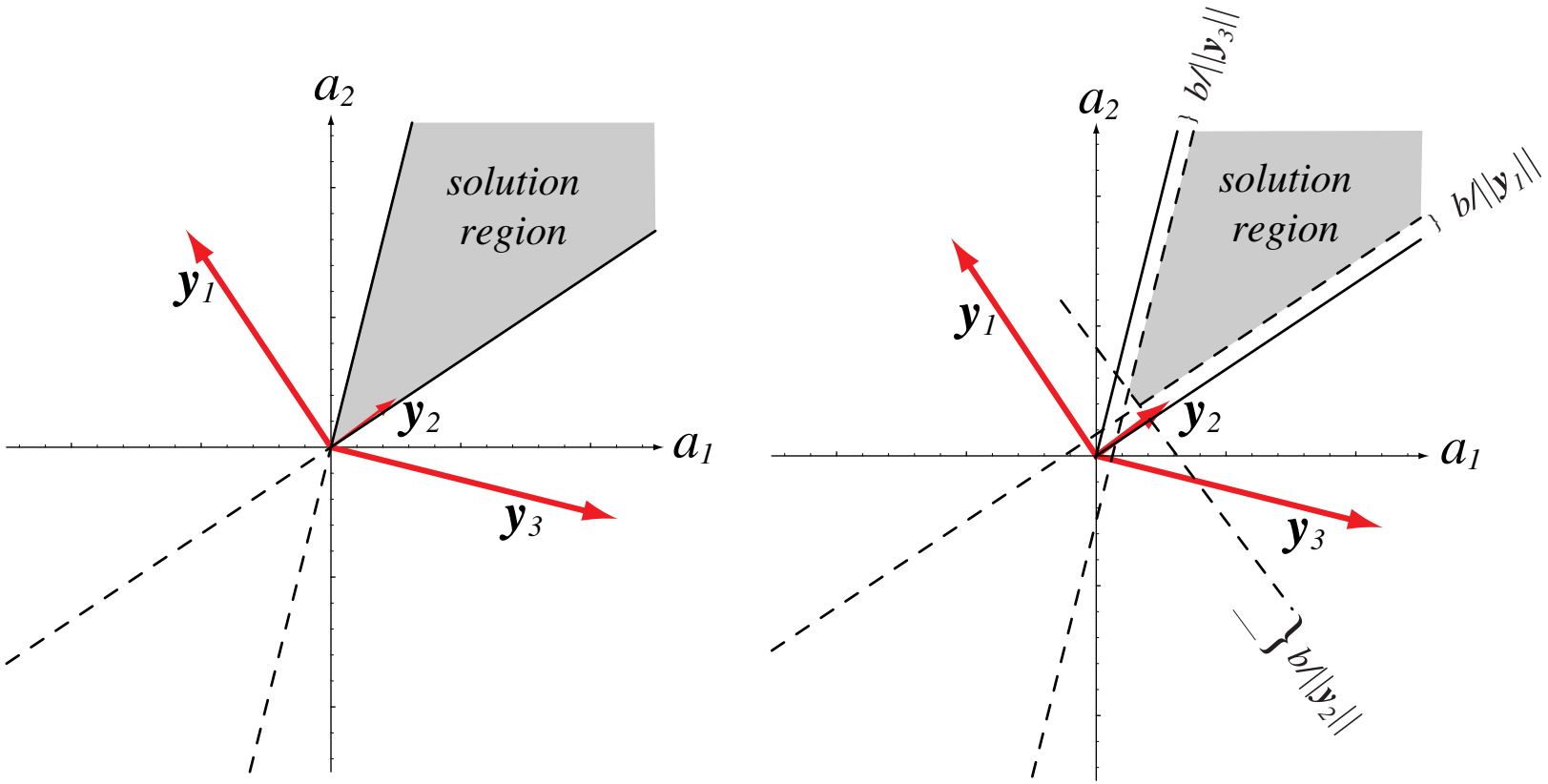


FIGURE 5.9. The effect of the margin on the solution region. At the left is the case of no margin ($b = 0$) equivalent to a case such as shown at the left in Fig. 5.8. At the right is the case $b > 0$, shrinking the solution region by margins $b/\|\mathbf{y}_i\|$. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Gradient Descent Solutions

- Set of linear inequalities $\vec{a}^t \vec{y}_i > 0$, define criterion function $J(\vec{a})$, which is minimized for a solution vector $\vec{a}^*$

Gradient Descent Solutions

- Set of linear inequalities $\vec{a}^t \vec{y}_i > 0$, define criterion function $J(\vec{a})$, which is minimized for a solution vector $\vec{a}^*$
- Minimizing a scalar function $J(\vec{a})$ by gradient descent $\vec{a}(k+1) = \vec{a}(k) - \eta(k)\vec{\nabla} J(\vec{a}(k))$

Gradient Descent Solutions

- Set of linear inequalities $\vec{a}^t \vec{y}_i > 0$, define criterion function $J(\vec{a})$, which is minimized for a solution vector $\vec{a}^*$
- Minimizing a scalar function $J(\vec{a})$ by gradient descent $\vec{a}(k+1) = \vec{a}(k) - \eta(k)\vec{\nabla} J(\vec{a}(k))$
- Second–order expansion $J(\vec{a}) \simeq J(\vec{a}(k)) + \vec{\nabla} J^t(\vec{a} - \vec{a}(k)) + \frac{1}{2}(\vec{a} - \vec{a}(k))^t H(\vec{a} - \vec{a}(k))$ H is Hessian Matrix

Gradient Descent Solutions

- Set of linear inequalities $\vec{a}^t \vec{y}_i > 0$, define criterion function $J(\vec{a})$, which is minimized for a solution vector $\vec{a}^*$
- Minimizing a scalar function $J(\vec{a})$ by gradient descent
  $\vec{a}(k+1) = \vec{a}(k) - \eta(k)\vec{\nabla} J(\vec{a}(k))$
- Second–order expansion
  $J(\vec{a}) \simeq J(\vec{a}(k)) + \vec{\nabla} J^t(\vec{a} - \vec{a}(k)) + \frac{1}{2}(\vec{a} - \vec{a}(k))^t H(\vec{a} - \vec{a}(k))$
  H is Hessian Matrix
- Minimize $J(\vec{a}(k+1))$ with $\eta(k) = \frac{||\vec{\nabla} J||^2}{\vec{\nabla} J^t H \vec{\nabla} J}$
  $J(\vec{a}) \sim \vec{a}^2 \rightarrow H = const. \rightarrow \eta = const.$

Gradient Descent Solutions

- Set of linear inequalities $\vec{a}^t \vec{y}_i > 0$, define criterion function $J(\vec{a})$, which is minimized for a solution vector $\vec{a}^*$
- Minimizing a scalar function $J(\vec{a})$ by gradient descent $\vec{a}(k+1) = \vec{a}(k) - \eta(k)\vec{\nabla} J(\vec{a}(k))$
- Second–order expansion $J(\vec{a}) \simeq J(\vec{a}(k)) + \vec{\nabla} J^t(\vec{a} - \vec{a}(k)) + \frac{1}{2}(\vec{a} - \vec{a}(k))^t H(\vec{a} - \vec{a}(k))$ H is Hessian Matrix
- Minimize $J(\vec{a}(k+1))$ with $\eta(k) = \frac{||\vec{\nabla} J||^2}{\vec{\nabla} J^t H \vec{\nabla} J}$ $J(\vec{a}) \sim \vec{a}^2 \rightarrow H = const. \rightarrow \eta = const.$
- Minimize second–order expansion with $\vec{a}(k+1) \rightarrow$ Newton Descent $\vec{a}(k+1) = \vec{a}(k) - H^{-1}\vec{\nabla} J$ (expensive)

Gradient and Newton Descent

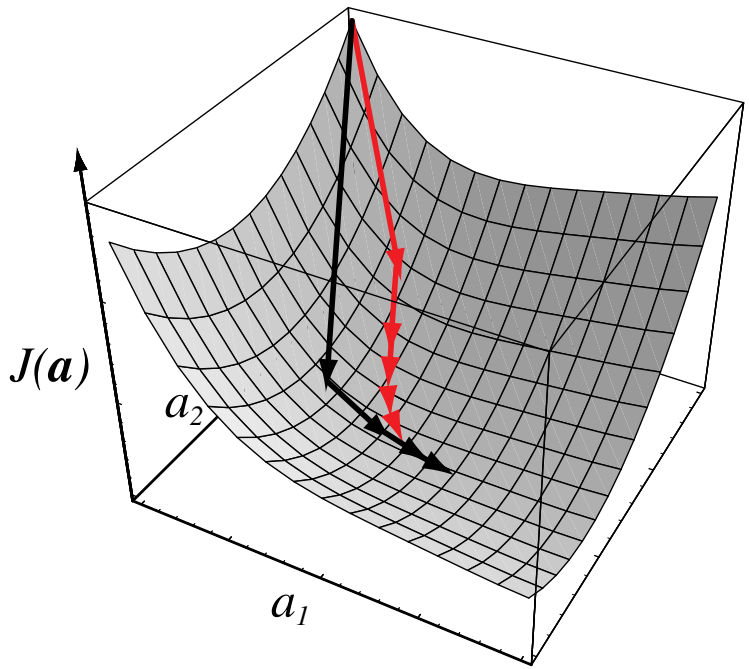


FIGURE 5.10. The sequence of weight vectors given by a simple gradient descent method (red) and by Newton's (second order) algorithm (black). Newton's method typically leads to greater improvement per step, even when using optimal learning rates for both methods. However the added computational burden of inverting the Hessian matrix used in Newton's method is not always justified, and simple gradient descent may suffice. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Perceptron Criterion Function

- "Normalized" inequalities $\vec{a}^t \vec{y}_i > 0$
  Perceptron criterion $J_p(\vec{a}) = \sum_{\vec{y} \in \mathcal{Y}} -\vec{a}^t \vec{y}$
  ($\mathcal{Y}$ is set of misclassified patterns)

Perceptron Criterion Function

- "Normalized" inequalities $\vec{a}^t \vec{y}_i > 0$
  Perceptron criterion $J_p(\vec{a}) = \sum_{\vec{y} \in \mathcal{Y}} -\vec{a}^t \vec{y}$
  ($\mathcal{Y}$ is set of misclassified patterns)
- Gradient $\vec{\nabla} J_p = \sum_{\vec{y} \in \mathcal{Y}} -\vec{y}$

Perceptron Criterion Function

- "Normalized" inequalities $\vec{a}^t \vec{y}_i > 0$
  Perceptron criterion $J_p(\vec{a}) = \sum_{\vec{y} \in \mathcal{Y}} -\vec{a}^t \vec{y}$
  ($\mathcal{Y}$ is set of misclassified patterns)
- Gradient $\vec{\nabla} J_p = \sum_{\vec{y} \in \mathcal{Y}} -\vec{y}$
- Update rule $\vec{a}(k+1) = \vec{a}(k) + \eta(k) \sum_{\vec{y} \in \mathcal{Y}_k} \vec{y}$

Perceptron Criterion Function

- "Normalized" inequalities $\vec{a}^t \vec{y}_i > 0$
  Perceptron criterion $J_p(\vec{a}) = \sum_{\vec{y} \in \mathcal{Y}} -\vec{a}^t \vec{y}$
  ($\mathcal{Y}$ is set of misclassified patterns)
- Gradient $\vec{\nabla} J_p = \sum_{\vec{y} \in \mathcal{Y}} -\vec{y}$
- Update rule $\vec{a}(k+1) = \vec{a}(k) + \eta(k) \sum_{\vec{y} \in \mathcal{Y}_k} \vec{y}$
- Batch vs. single–sample correction

Minimum Squared–Error Procedures

- Set of equalities $\vec{a}^t \vec{y}_i = b_i$
  $b_i > 0$ are arbitrary constants

Minimum Squared–Error Procedures

- Set of equalities $\vec{a}^t \vec{y}_i = b_i$
  $b_i > 0$ are arbitrary constants
- Solve $Y\vec{a} = \vec{b}$
  Y is $n \times (d+1)$ matrix containing all training vectors

Minimum Squared–Error Procedures

- Set of equalities $\vec{a}^t \vec{y}_i = b_i$
  $b_i > 0$ are arbitrary constants
- Solve $Y\vec{a} = \vec{b}$
  Y is $n \times (d+1)$ matrix containing all training vectors
- If Y nonsingular $\vec{a} = Y^{-1}\vec{b}$, however Y mostly rectangular!

Minimum Squared–Error Procedures

- Set of equalities $\vec{a}^t \vec{y}_i = b_i$
  $b_i > 0$ are arbitrary constants
- Solve $Y\vec{a} = \vec{b}$
  Y is $n \times (d+1)$ matrix containing all training vectors
- If Y nonsingular $\vec{a} = Y^{-1}\vec{b}$, however Y mostly rectangular!
- Minimizing $\vec{e} = Y\vec{a} - \vec{b}$ leads to
  $Y^t Y\vec{a} = Y^t\vec{b} \rightarrow \vec{a} = (Y^t Y)^{-1} Y^t \vec{b} = Y^\dagger \vec{b}$
  $Y^\dagger$ is pseudoinverse $(d+1) \times n$ matrix

Support Vector Machines

- Transform patterns to (much) higher dimension via nonlinear mapping $\varphi(.)$

Support Vector Machines

- Transform patterns to (much) higher dimension via nonlinear mapping $\varphi(.)$
- Linear discriminant $g(\vec{y}) = \vec{a}^t \vec{y}$

Support Vector Machines

- Transform patterns to (much) higher dimension via nonlinear mapping $\varphi(.)$
- Linear discriminant $g(\vec{y}) = \vec{a}^t \vec{y}$
- Distance of $\vec{y}_k$ to H is $\frac{z_k g(\vec{y}_k)}{||a||} \geq b$
  $z_k = \pm 1$ (normalization), b is margin

Support Vector Machines

- Transform patterns to (much) higher dimension via nonlinear mapping $\varphi(.)$
- Linear discriminant $g(\vec{y}) = \vec{a}^t \vec{y}$
- Distance of $\vec{y}_k$ to H is $\frac{z_k g(\vec{y}_k)}{||a||} \geq b$
  $z_k = \pm 1$ (normalization), b is margin
- Maximize b with constrained $||a|| = \frac{1}{b} \rightarrow$ minimize $||a||$ with inequality constraints

Support Vector Machines

- Transform patterns to (much) higher dimension via nonlinear mapping $\varphi(.)$
- Linear discriminant $g(\vec{y}) = \vec{a}^t \vec{y}$
- Distance of $\vec{y}_k$ to H is $\frac{z_k g(\vec{y}_k)}{||a||} \geq b$
  $z_k = \pm 1$ (normalization), b is margin
- Maximize b with constrained $||a|| = \frac{1}{b} \rightarrow$ minimize $||a||$ with inequality constraints
- Kuhn–Tucker theorem, optimization with inequality constraints, generalization of Lagrange Multipliers

Maximal Margin SVM

- Maximize margin b using the Kuhn–Tucker functional $L(\vec{a}, \vec{\alpha}) = \frac{1}{2}||\vec{a}||^2 - \sum_{k=1}^{n} \alpha_k [z_k \vec{a}^t \vec{y}_k - 1]$

Maximal Margin SVM

- Maximize margin b using the Kuhn–Tucker functional
  $L(\vec{a}, \vec{\alpha}) = \frac{1}{2}||\vec{a}||^2 - \sum_{k=1}^{n} \alpha_k [z_k \vec{a}^t \vec{y}_k - 1]$
- Resulting in dual problem (quadratic optimization)
  $L(\vec{\alpha}) = \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k,j}^{n} \alpha_k \alpha_j z_k z_j \vec{y}_j^t \vec{y}_k$
  with constraints
  $\sum_{k=1}^{n} z_k \alpha_k = 0 \qquad \alpha_k \geq 0$

Maximal Margin SVM

- Maximize margin b using the Kuhn–Tucker functional
  $L(\vec{a}, \vec{\alpha}) = \frac{1}{2}||\vec{a}||^2 - \sum_{k=1}^{n} \alpha_k [z_k \vec{a}^t \vec{y}_k - 1]$
- Resulting in dual problem (quadratic optimization)
  $L(\vec{\alpha}) = \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k,j}^{n} \alpha_k \alpha_j z_k z_j \vec{y}_j^t \vec{y}_k$
  with constraints
  $\sum_{k=1}^{n} z_k \alpha_k = 0 \qquad \alpha_k \geq 0$
- Then $\vec{a}^* = \sum_{i=1}^{n} z_i \alpha_i^* \vec{y}_i$ (non–zero α_i indicates support vector)

Maximal Margin SVM

- Maximize margin b using the Kuhn–Tucker functional
  $L(\vec{a}, \vec{\alpha}) = \frac{1}{2}||\vec{a}||^2 - \sum_{k=1}^{n} \alpha_k [z_k \vec{a}^t \vec{y}_k - 1]$
- Resulting in dual problem (quadratic optimization)
  $L(\vec{\alpha}) = \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k,j}^{n} \alpha_k \alpha_j z_k z_j \vec{y}_j^t \vec{y}_k$
  with constraints
  $\sum_{k=1}^{n} z_k \alpha_k = 0 \qquad \alpha_k \geq 0$
- Then $\vec{a}^* = \sum_{i=1}^{n} z_i \alpha_i^* \vec{y}_i$ (non–zero α_i indicates support vector)
- Maximal margin $b^* = (\sum_{i=1}^{n} \alpha_i{}^*)^{-\frac{1}{2}}$

Maximal Margin Hyperplane

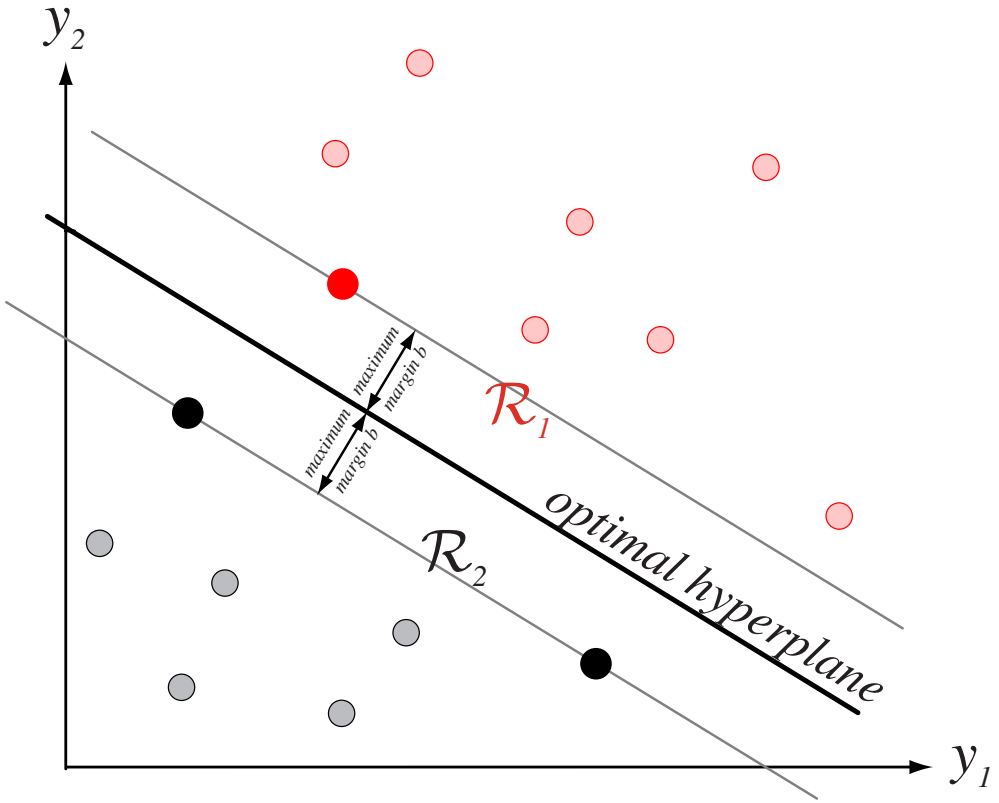


FIGURE 5.19. Training a support vector machine consists of finding the optimal hyperplane, that is, the one with the maximum distance from the nearest training patterns. The support vectors are those (nearest) patterns, a distance b from the hyperplane. The three support vectors are shown as solid dots. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Soft Margin SVM

- Maximal margin SVM is sensitive to outliers, demands linear separability for solution

Soft Margin SVM

- Maximal margin SVM is sensitive to outliers, demands linear separability for solution
- Soft Margin SVM introducing slack variables ξ
  $z_k g(\vec{y}_k) \geq b - \xi_k$ (relaxed margin)

Soft Margin SVM

- Maximal margin SVM is sensitive to outliers, demands linear separability for solution
- Soft Margin SVM introducing slack variables ξ
  $z_k g(\vec{y}_k) \geq b - \xi_k$ (relaxed margin)
- Maximize relaxed margin b with Kuhn–Tucker functional
  $L(\vec{a}, \vec{\alpha}, \vec{\xi}) = \frac{1}{2}||\vec{a}||^2 + \frac{C}{2}\sum_{k=1}^{n} \xi_i^2 - \sum_{k=1}^{n} \alpha_k [z_k \vec{a}^t \vec{y}_k - 1 + \xi_i]$

Soft Margin SVM

- Maximal margin SVM is sensitive to outliers, demands linear separability for solution
- Soft Margin SVM introducing slack variables ξ
  $z_k g(\vec{y}_k) \geq b - \xi_k$ (relaxed margin)
- Maximize relaxed margin b with Kuhn–Tucker functional
  $L(\vec{a}, \vec{\alpha}, \vec{\xi}) = \frac{1}{2}||\vec{a}||^2 + \frac{C}{2}\sum_{k=1}^{n} \xi_i^2 - \sum_{k=1}^{n} \alpha_k [z_k \vec{a}^t \vec{y}_k - 1 + \xi_i]$
- Again $\vec{a}^* = \sum_{i=1}^{n} z_i \alpha_i^* \vec{y}_i$

Soft Margin SVM

- Maximal margin SVM is sensitive to outliers, demands linear separability for solution
- Soft Margin SVM introducing slack variables ξ
  $z_k g(\vec{y}_k) \geq b - \xi_k$ (relaxed margin)
- Maximize relaxed margin b with Kuhn–Tucker functional
  $L(\vec{a}, \vec{\alpha}, \vec{\xi}) = \frac{1}{2}||\vec{a}||^2 + \frac{C}{2}\sum_{k=1}^{n} \xi_i^2 - \sum_{k=1}^{n} \alpha_k [z_k \vec{a}^t \vec{y}_k - 1 + \xi_i]$
- Again $\vec{a}^* = \sum_{i=1}^{n} z_i \alpha_i^* \vec{y}_i$
- Maximal margin $b^* = (\sum_{i=1}^{n} \alpha_i{}^* - \frac{1}{C}|\alpha_i^*|^2)^{-\frac{1}{2}}$

Soft Margin SVM

- Maximal margin SVM is sensitive to outliers, demands linear separability for solution
- Soft Margin SVM introducing slack variables ξ
  $z_k g(\vec{y}_k) \geq b - \xi_k$ (relaxed margin)
- Maximize relaxed margin b with Kuhn–Tucker functional
  $L(\vec{a}, \vec{\alpha}, \vec{\xi}) = \frac{1}{2}||\vec{a}||^2 + \frac{C}{2}\sum_{k=1}^{n} \xi_i^2 - \sum_{k=1}^{n} \alpha_k [z_k \vec{a}^t \vec{y}_k - 1 + \xi_i]$
- Again $\vec{a}^* = \sum_{i=1}^{n} z_i \alpha_i^* \vec{y}_i$
- Maximal margin $b^* = (\sum_{i=1}^{n} \alpha_i{}^* - \frac{1}{C}|\alpha_i^*|^2)^{-\frac{1}{2}}$
- Depends on parameter C!

Multilayer Neural Networks

- Real-world problems: linear discriminant often not sufficient

Multilayer Neural Networks

- Real-world problems: linear discriminant often not sufficient
- NNs also implement nonlinear mapping to higher dimension

Multilayer Neural Networks

- Real-world problems: linear discriminant often not sufficient
- NNs also implement nonlinear mapping to higher dimension
- Learning finds mapping AND linear discriminant

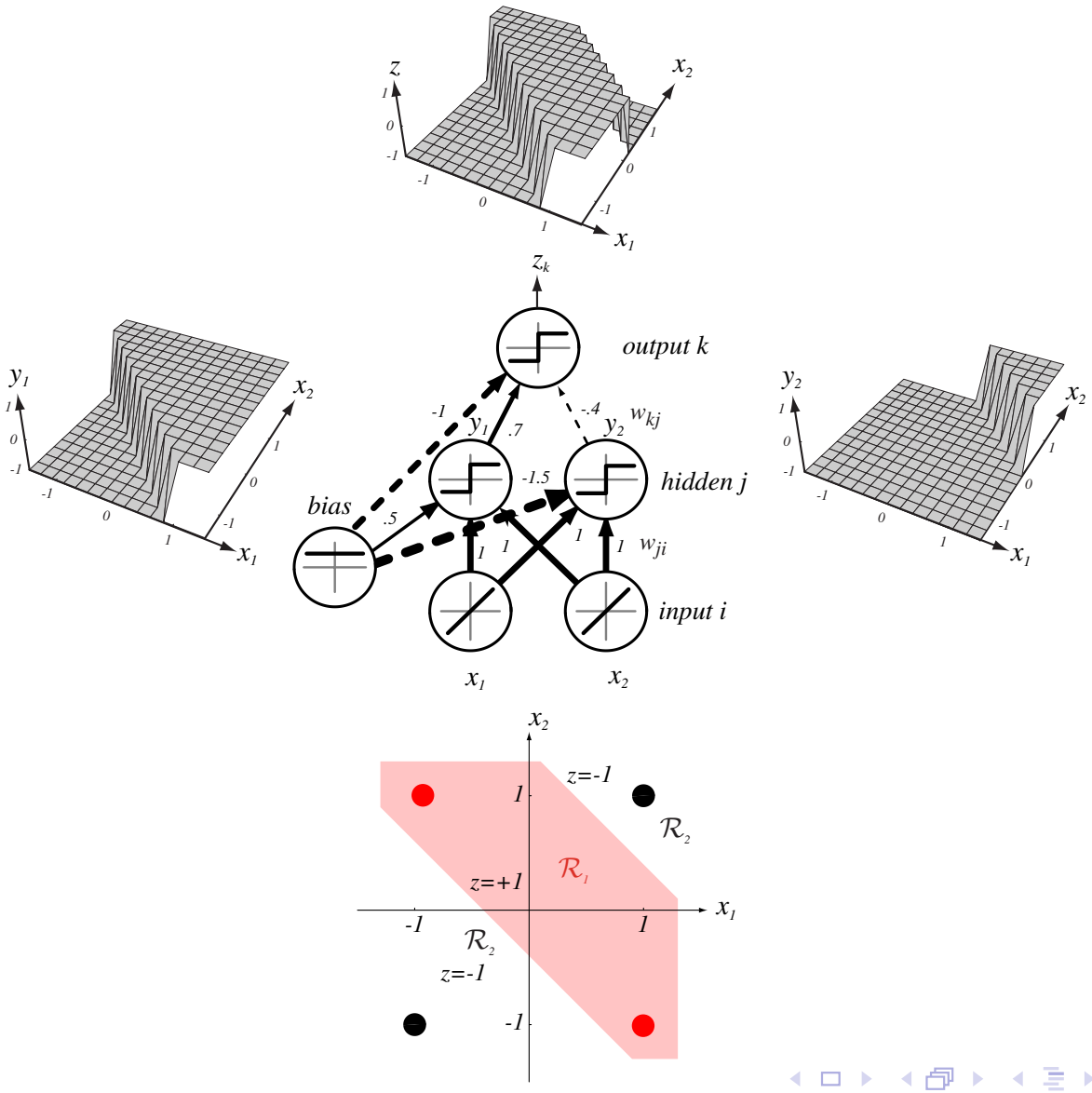
Multilayer Neural Networks

- Real-world problems: linear discriminant often not sufficient
- NNs also implement nonlinear mapping to higher dimension
- Learning finds mapping AND linear discriminant
- Error–backpropagation is least square fit to Bayes discriminant functions

Multilayer Neural Networks

- Real-world problems: linear discriminant often not sufficient
- NNs also implement nonlinear mapping to higher dimension
- Learning finds mapping AND linear discriminant
- Error–backpropagation is least square fit to Bayes discriminant functions
- NNs motivated by biology, but can be explained without it

XOR Net


z
x2
x1
y1
y2
zk
output k
hidden j
input i
bias
wkj
wji
-1
.7
-.4
-1.5
.5
1
z=-1
z=+1
R1
R2

Network Components

- Neurons and synaptic connections (weights)

Network Components

- Neurons and synaptic connections (weights)
- Net activation $net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv \vec{w}_j{}^t \vec{x}$

Network Components

- Neurons and synaptic connections (weights)
- Net activation $net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv \vec{w}_j{}^t \vec{x}$
- Neuron output $z_k = f(net_k)$, activation function

Network Components

- Neurons and synaptic connections (weights)
- Net activation $net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv \vec{w}_j{}^t \vec{x}$
- Neuron output $z_k = f(net_k)$, activation function
- Common activation function class is *sigmoid*, e.g., $f(x) = \frac{1}{1+e^{-cx}}$

Network Components

- Neurons and synaptic connections (weights)
- Net activation $net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv \vec{w}_j{}^t \vec{x}$
- Neuron output $z_k = f(net_k)$, activation function
- Common activation function class is *sigmoid*, e.g., $f(x) = \frac{1}{1+e^{-cx}}$
- Basic topologies: feed–forward and recurrent

A 2–4–1 Network

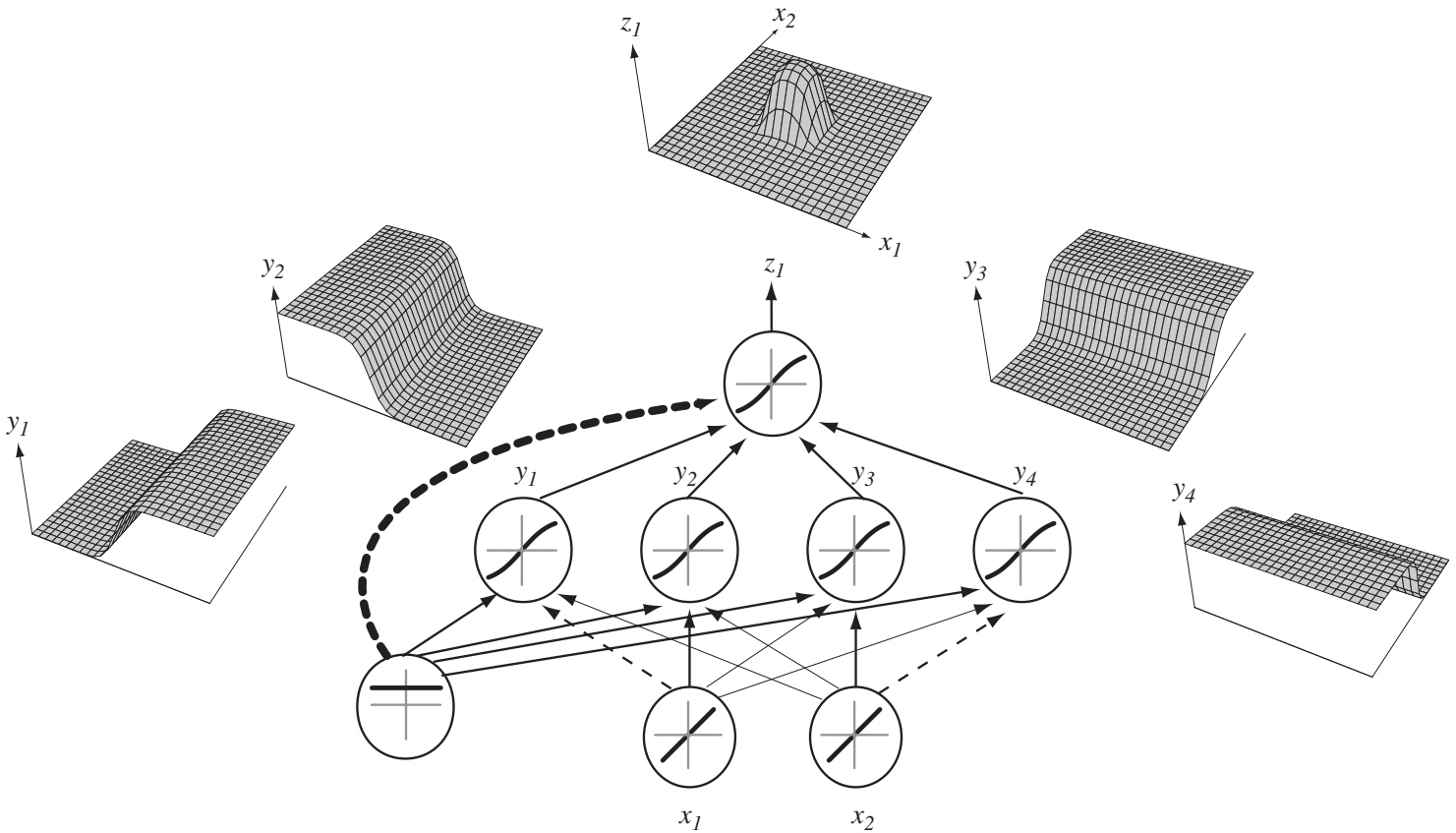


FIGURE 6.2. A 2-4-1 network (with bias) along with the response functions at different units; each hidden output unit has sigmoidal activation function $f(\cdot)$. In the case shown, the hidden unit outputs are paired in opposition thereby producing a "bump" at the output unit. Given a sufficiently large number of hidden units, any continuous function from input to output can be approximated arbitrarily well by such a network. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

NN Decision Boundaries

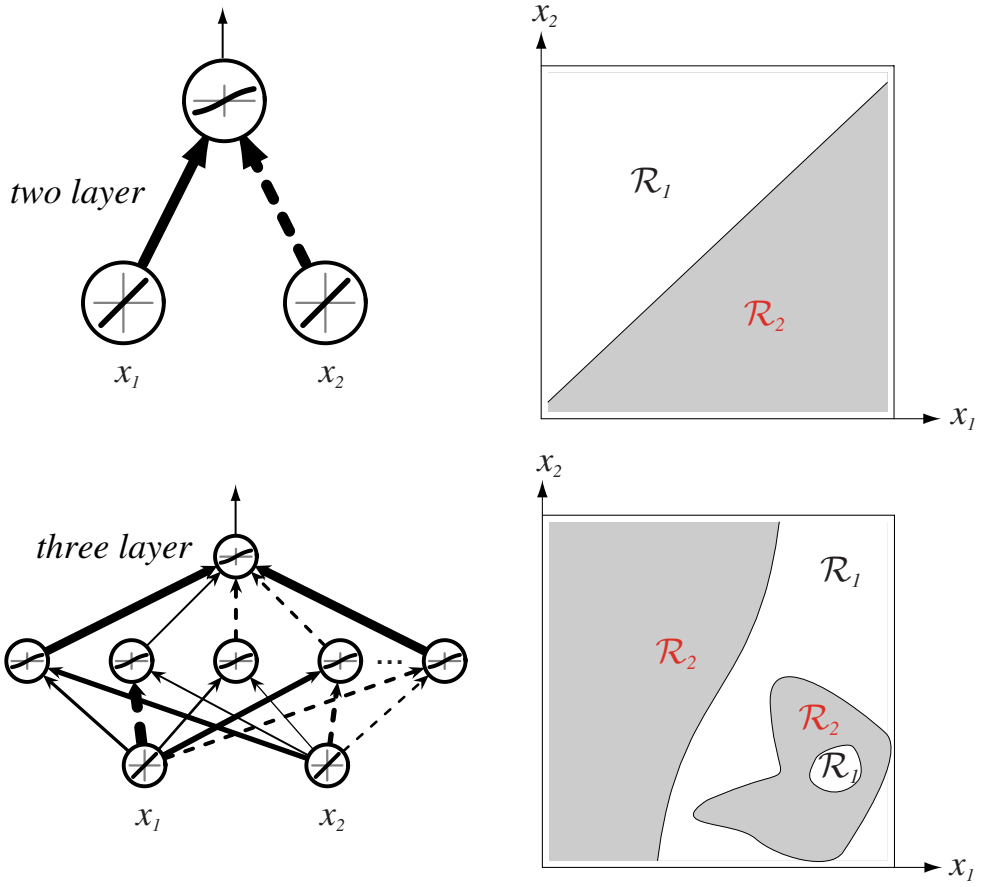


FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Network Learning

- Learning as minimization (of network error)

Network Learning

- Learning as minimization (of network error)
- Error is a function of network parameters

Network Learning

- Learning as minimization (of network error)
- Error is a function of network parameters
- Gradient descent methods reduce error

Network Learning

- Learning as minimization (of network error)
- Error is a function of network parameters
- Gradient descent methods reduce error
- Problem with hidden layers

Network Learning

- Learning as minimization (of network error)
- Error is a function of network parameters
- Gradient descent methods reduce error
- Problem with hidden layers
- Backpropagation = Iterative Local Gradient Descent
  Werbos (1974), *Rumelhart, Hinton, Williams* (1986)

Network Learning

- Learning as minimization (of network error)
- Error is a function of network parameters
- Gradient descent methods reduce error
- Problem with hidden layers
- Backpropagation = Iterative Local Gradient Descent
  Werbos (1974), *Rumelhart, Hinton, Williams* (1986)
- Error–Backpropagation, output error is transmitted backwards as weighted error, network weights are updated **locally**

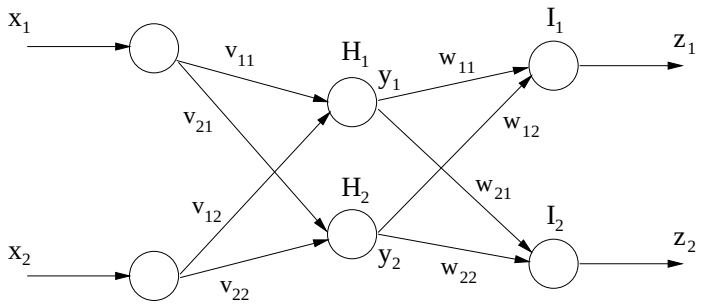
Network Learning

- Learning as minimization (of network error)
- Error is a function of network parameters
- Gradient descent methods reduce error
- Problem with hidden layers
- Backpropagation = Iterative Local Gradient Descent
  Werbos (1974), *Rumelhart, Hinton, Williams* (1986)
- Error–Backpropagation, output error is transmitted backwards as weighted error, network weights are updated **locally**
- Weight update $\Delta w_{j,i} = \eta \delta_j a_i$
  Generalized error term δ

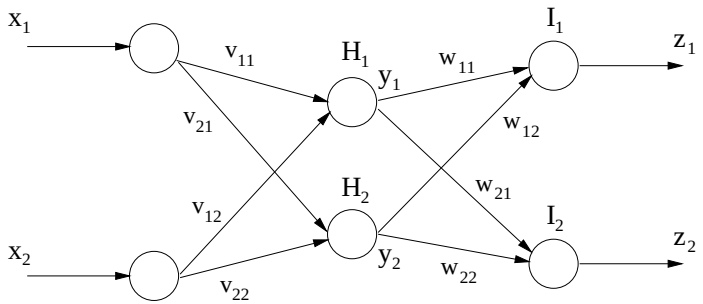
Network Learning

- Learning as minimization (of network error)
- Error is a function of network parameters
- Gradient descent methods reduce error
- Problem with hidden layers
- Backpropagation = Iterative Local Gradient Descent
  Werbos (1974), *Rumelhart, Hinton, Williams* (1986)
- Error–Backpropagation, output error is transmitted backwards as weighted error, network weights are updated **locally**
- Weight update $\Delta w_{j,i} = \eta \delta_j a_i$
  Generalized error term δ
- Common transfer functions: differentiable, nonlinear, monotonous, easily computable differentiation

Error–Backpropagation I

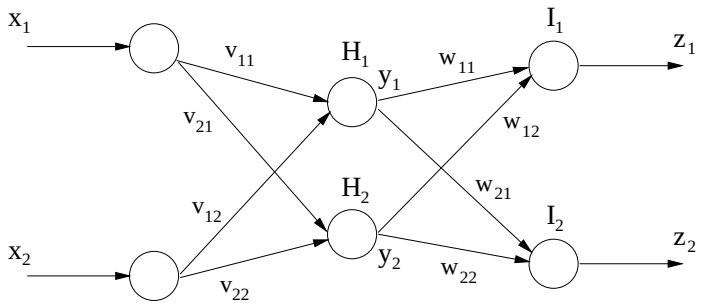


Error–Backpropagation I



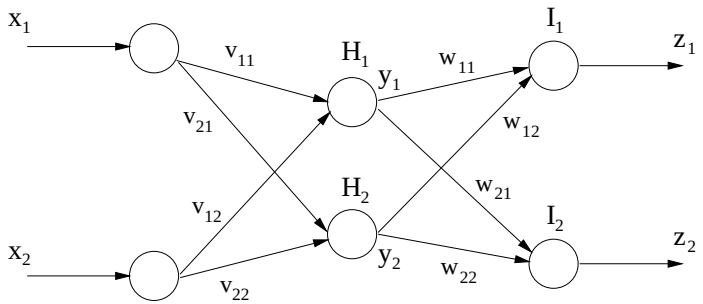
- $H_j = \sum_{i=1}^{n} v_{j,i} x_i \quad I_k = \sum_{j=1}^{h} w_{k,j} y_j$
  $y_j = f(H_j), z_k = f(I_k)$

Error–Backpropagation I



- $H_j = \sum_{i=1}^{n} v_{j,i} x_i \quad I_k = \sum_{j=1}^{h} w_{k,j} y_j$
  $y_j = f(H_j), z_k = f(I_k)$
- Error $E^{(p)} = \frac{1}{2} \sum_{k=1}^{m} (t_k^{(p)} - z_k^{(p)})^2$

Error–Backpropagation I



- $H_j = \sum_{i=1}^{n} v_{j,i} x_i \quad I_k = \sum_{j=1}^{h} w_{k,j} y_j$
  $y_j = f(H_j), z_k = f(I_k)$
- Error $E^{(p)} = \frac{1}{2} \sum_{k=1}^{m} (t_k^{(p)} - z_k^{(p)})^2$
- Output Layer: $\Delta w_{k,j} = -\eta \frac{\partial E}{\partial w_{k,j}}$
  $\frac{\partial E}{\partial w_{k,j}} = \frac{\partial E}{\partial I_k} \frac{\partial I_k}{\partial w_{k,j}} = \frac{\partial E}{\partial I_k} y_j$
  $\frac{\partial E}{\partial I_k} = \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial I_k} = -(t_k - z_k) f'(I_k)$
  $\frac{\partial E}{\partial w_{k,j}} = -(t_k - z_k) f'(I_k) y_j$ mit $\delta_k = (t_k - z_k) f'(I_k)$
  $\Delta w_{k,j} = \eta \delta_k y_j$

Error–Backpropagation II

- Hidden Layer: $\Delta v_{j,i} = -\eta \frac{\partial E}{\partial v_{j,i}}$

  $\frac{\partial E}{\partial v_{j,i}} = \frac{\partial E}{\partial H_j} \frac{\partial H_j}{\partial v_{j,i}} = \frac{\partial E}{\partial H_j} x_i$

  $\frac{\partial E}{\partial H_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial H_j} = \frac{\partial E}{\partial y_j} f'(H_j)$

  $\frac{\partial E}{\partial y_j} = -\frac{1}{2} \sum_{k=1}^{m} \frac{\partial (t_k - f(I_k))^2}{\partial y_j} = -\sum_{k=1}^{m} (t_k - z_k) f'(I_k) w_{k,j}$

  mit $\delta_j = f'(H_j) \sum_{k=1}^{m} \delta_k w_{k,j}$

  $\Delta v_{j,i} = \eta \delta_j x_i$

Error–Backpropagation II

- Hidden Layer: $\Delta v_{j,i} = -\eta \frac{\partial E}{\partial v_{j,i}}$

  $\frac{\partial E}{\partial v_{j,i}} = \frac{\partial E}{\partial H_j} \frac{\partial H_j}{\partial v_{j,i}} = \frac{\partial E}{\partial H_j} x_i$

  $\frac{\partial E}{\partial H_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial H_j} = \frac{\partial E}{\partial y_j} f'(H_j)$

  $\frac{\partial E}{\partial y_j} = -\frac{1}{2} \sum_{k=1}^{m} \frac{\partial (t_k - f(I_k))^2}{\partial y_j} = -\sum_{k=1}^{m} (t_k - z_k) f'(I_k) w_{k,j}$

  mit $\delta_j = f'(H_j) \sum_{k=1}^{m} \delta_k w_{k,j}$

  $\Delta v_{j,i} = \eta \delta_j x_i$

- Local update rules propagating error from output to input

Error–Backpropagation II

- Hidden Layer: $\Delta v_{j,i} = -\eta \frac{\partial E}{\partial v_{j,i}}$

  $\frac{\partial E}{\partial v_{j,i}} = \frac{\partial E}{\partial H_j} \frac{\partial H_j}{\partial v_{j,i}} = \frac{\partial E}{\partial H_j} x_i$

  $\frac{\partial E}{\partial H_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial H_j} = \frac{\partial E}{\partial y_j} f'(H_j)$

  $\frac{\partial E}{\partial y_j} = -\frac{1}{2} \sum_{k=1}^{m} \frac{\partial (t_k - f(I_k))^2}{\partial y_j} = -\sum_{k=1}^{m} (t_k - z_k) f'(I_k) w_{k,j}$

  mit $\delta_j = f'(H_j) \sum_{k=1}^{m} \delta_k w_{k,j}$

  $\Delta v_{j,i} = \eta \delta_j x_i$
- Local update rules propagating error from output to input
- Present all p patterns of the training set $= 1$ *Epoch* (complete training e.g., 1,000 epochs)

Error–Backpropagation II

- Hidden Layer: $\Delta v_{j,i} = -\eta \frac{\partial E}{\partial v_{j,i}}$
  $\frac{\partial E}{\partial v_{j,i}} = \frac{\partial E}{\partial H_j} \frac{\partial H_j}{\partial v_{j,i}} = \frac{\partial E}{\partial H_j} x_i$
  $\frac{\partial E}{\partial H_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial H_j} = \frac{\partial E}{\partial y_j} f'(H_j)$
  $\frac{\partial E}{\partial y_j} = -\frac{1}{2} \sum_{k=1}^{m} \frac{\partial (t_k - f(I_k))^2}{\partial y_j} = -\sum_{k=1}^{m} (t_k - z_k) f'(I_k) w_{k,j}$
  mit $\delta_j = f'(H_j) \sum_{k=1}^{m} \delta_k w_{k,j}$
  $\Delta v_{j,i} = \eta \delta_j x_i$
- Local update rules propagating error from output to input
- Present all p patterns of the training set = 1 *Epoch* (complete training e.g., 1,000 epochs)
- Batch Learning (Off–line): accumulate weight changes for all patterns, then update weights

Error–Backpropagation II

- Hidden Layer: $\Delta v_{j,i} = -\eta \frac{\partial E}{\partial v_{j,i}}$
  $\frac{\partial E}{\partial v_{j,i}} = \frac{\partial E}{\partial H_j} \frac{\partial H_j}{\partial v_{j,i}} = \frac{\partial E}{\partial H_j} x_i$
  $\frac{\partial E}{\partial H_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial H_j} = \frac{\partial E}{\partial y_j} f'(H_j)$
  $\frac{\partial E}{\partial y_j} = -\frac{1}{2} \sum_{k=1}^{m} \frac{\partial (t_k - f(I_k))^2}{\partial y_j} = -\sum_{k=1}^{m} (t_k - z_k) f'(I_k) w_{k,j}$
  mit $\delta_j = f'(H_j) \sum_{k=1}^{m} \delta_k w_{k,j}$
  $\Delta v_{j,i} = \eta \delta_j x_i$
- Local update rules propagating error from output to input
- Present all p patterns of the training set = 1 *Epoch* (complete training e.g., 1,000 epochs)
- Batch Learning (Off–line): accumulate weight changes for all patterns, then update weights
- On–line Learning: update weights after each pattern

Learning Curves

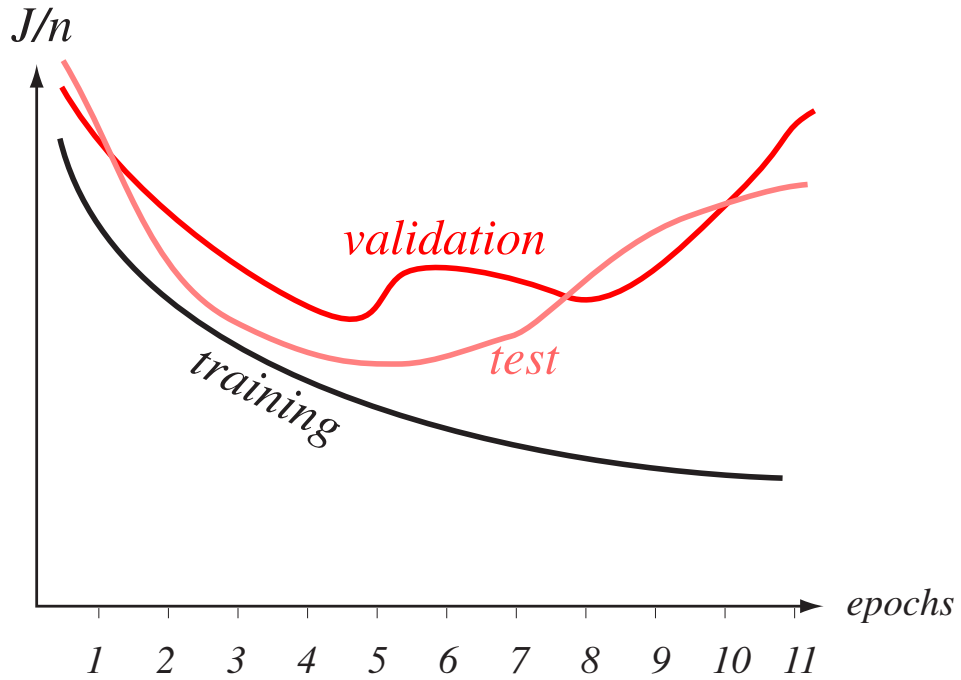
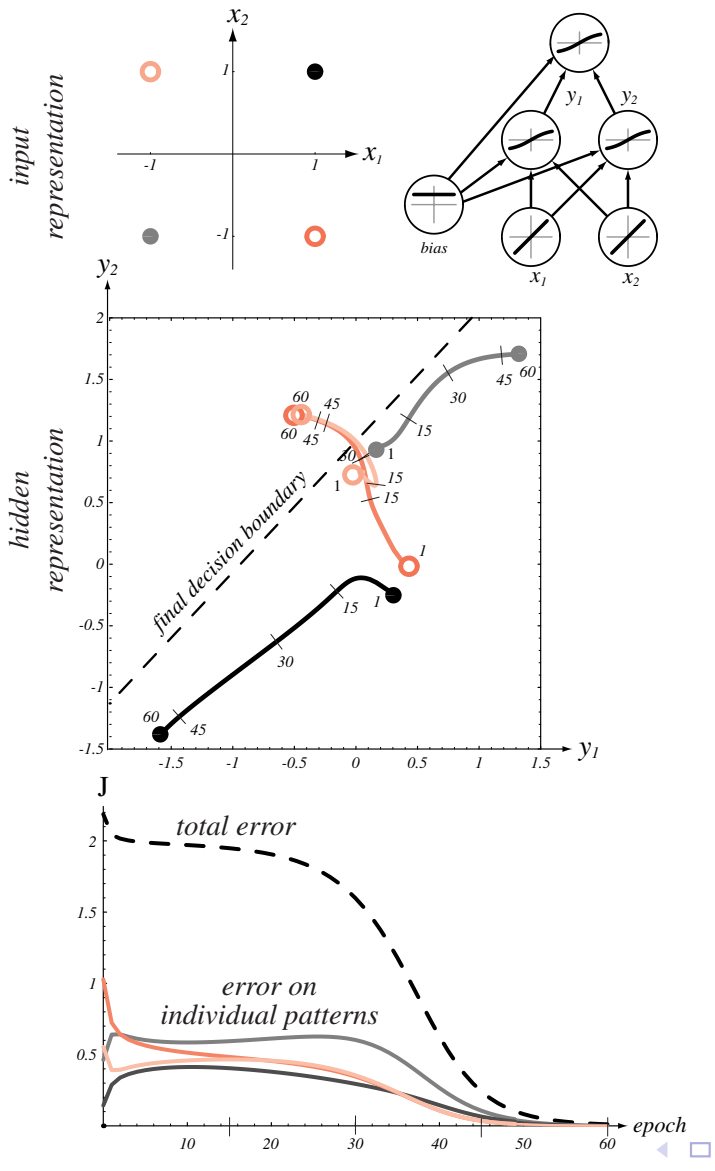


FIGURE 6.6. A learning curve shows the criterion function as a function of the amount of training, typically indicated by the number of epochs or presentations of the full training set. We plot the average error per pattern, that is, $1/n \sum_{p=1}^{n} J_p$. The validation error and the test or generalization error per pattern are virtually always higher than the training error. In some protocols, training is stopped at the first minimum of the validation set. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

XOR Learning Details



Backpropagation Variants I

- Standard Backpropagation: $\vec{w_t} = \vec{w_{t-1}} - \eta \vec{\nabla} E$

Backpropagation Variants I

- Standard Backpropagation: $\vec{w_t} = \vec{w_{t-1}} - \eta \vec{\nabla} E$
- Gradient Reuse: use $\vec{\nabla} E$ as long as error drops

Backpropagation Variants I

- Standard Backpropagation: $\vec{w_t} = \vec{w_{t-1}} - \eta \vec{\nabla} E$
- Gradient Reuse: use $\vec{\nabla} E$ as long as error drops
- BP with variable stepsize (learn rate) η

Backpropagation Variants I

- Standard Backpropagation: $\vec{w_t} = \vec{w_{t-1}} - \eta\vec{\nabla}E$
- Gradient Reuse: use $\vec{\nabla}E$ as long as error drops
- BP with variable stepsize (learn rate) η
- BP with momentum: $\Delta\vec{w_t} = -\eta\vec{\nabla}E + \alpha\Delta\vec{w_{t-1}}$

Decision Trees

- Real problems: nominal data, e.g., `car = green, red, blue`

Decision Trees

- Real problems: nominal data, e.g., `car = green, red, blue`
- Rule–based or syntactic methods

Decision Trees

- Real problems: nominal data, e.g., `car = green, red, blue`
- Rule–based or syntactic methods
- Decision tree (DT): series of questions (nodes) lead to answer at leaf (category)

Decision Trees

- Real problems: nominal data, e.g., `car = green, red, blue`
- Rule–based or syntactic methods
- Decision tree (DT): series of questions (nodes) lead to answer at leaf (category)
- DT is interpretable (decisions and categories)

Monothetic Decision Tree

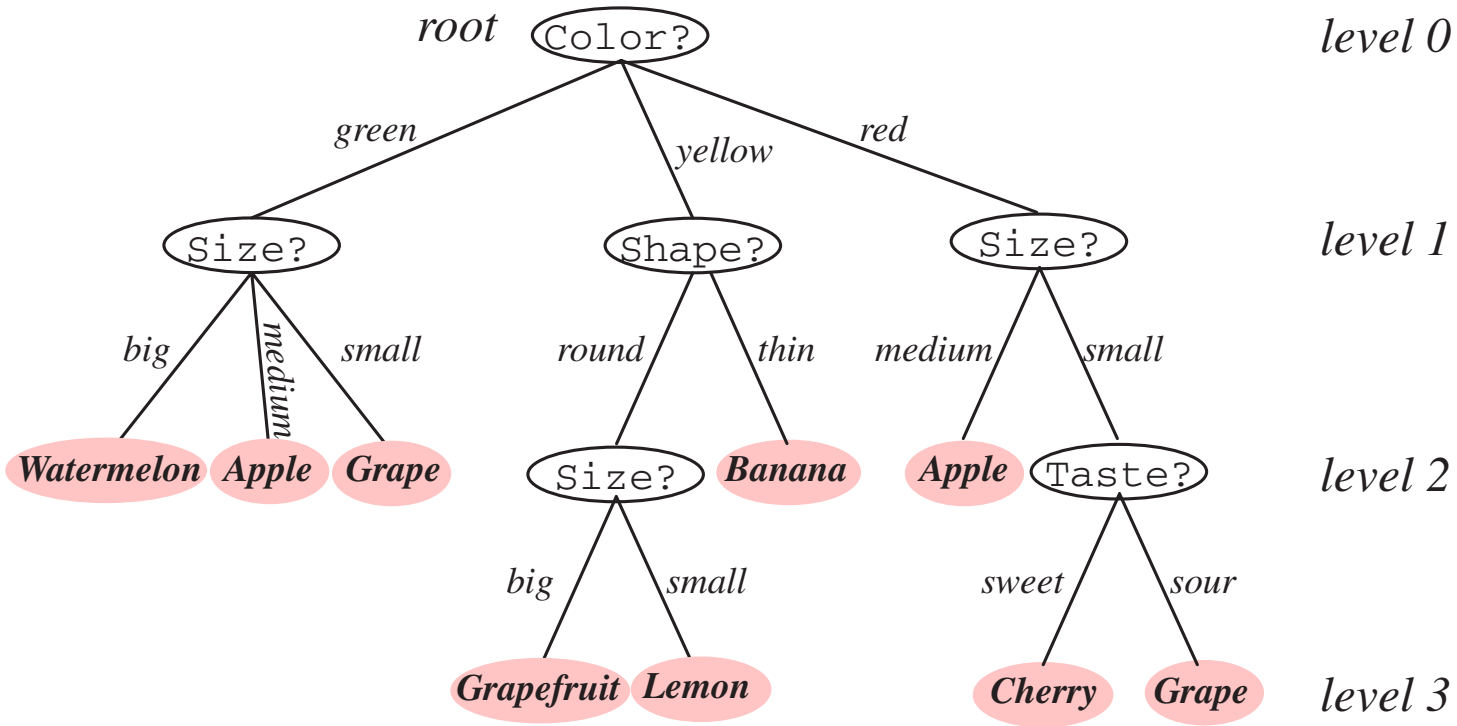


FIGURE 8.1. Classification in a basic decision tree proceeds from top to bottom. The questions asked at each node concern a particular property of the pattern, and the downward links correspond to the possible values. Successive nodes are visited until a terminal or leaf node is reached, where the category label is read. Note that the same question, `Size?`, appears in different places in the tree and that different questions can have different numbers of branches. Moreover, different leaf nodes, shown in pink, can be labeled by the same category (e.g., **Apple**). From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*.

CART

- Goal: construct pure nodes (ideally, all leaf nodes are pure)

CART

- Goal: construct pure nodes (ideally, all leaf nodes are pure)
- A pure leaf node resembles only patterns of single category

CART

- Goal: construct pure nodes (ideally, all leaf nodes are pure)
- A pure leaf node resembles only patterns of single category
- Design Issues

CART

- Goal: construct pure nodes (ideally, all leaf nodes are pure)
- A pure leaf node resembles only patterns of single category
- Design Issues
  - Branching factor = splits?

CART

- Goal: construct pure nodes (ideally, all leaf nodes are pure)
- A pure leaf node resembles only patterns of single category
- Design Issues
  - Branching factor = splits?
  - Which query (property) at which node?

CART

- Goal: construct pure nodes (ideally, all leaf nodes are pure)
- A pure leaf node resembles only patterns of single category
- Design Issues
  - Branching factor = splits?
  - Which query (property) at which node?
  - Termination (leaf node)?

CART

- Goal: construct pure nodes (ideally, all leaf nodes are pure)
- A pure leaf node resembles only patterns of single category
- Design Issues
  - Branching factor = splits?
  - Which query (property) at which node?
  - Termination (leaf node)?
  - Pruning (simplification)?

CART

- Goal: construct pure nodes (ideally, all leaf nodes are pure)
- A pure leaf node resembles only patterns of single category
- Design Issues
  - Branching factor = splits?
  - Which query (property) at which node?
  - Termination (leaf node)?
  - Pruning (simplification)?
  - Missing data?

Monothetic Decision Boundaries

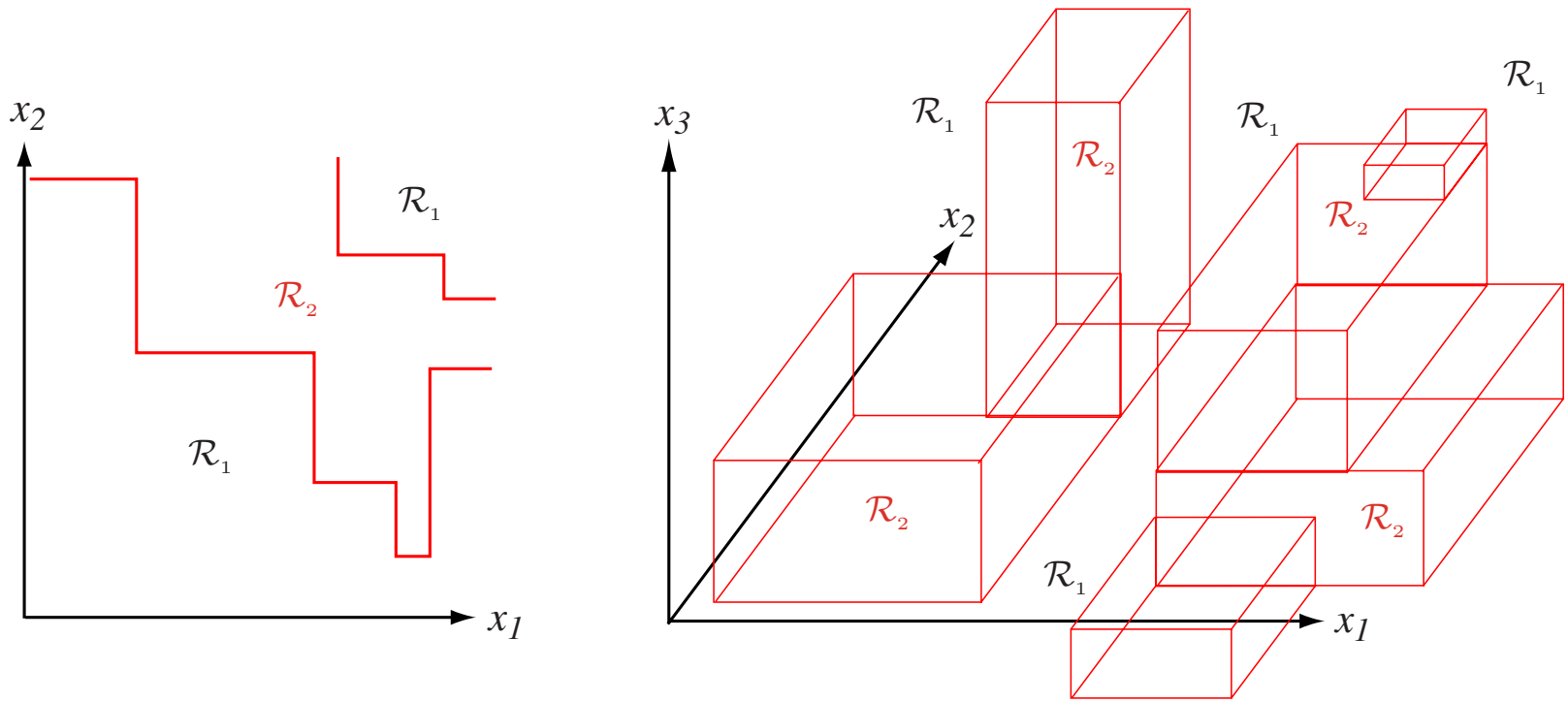


FIGURE 8.3. Monothetic decision trees create decision boundaries with portions perpendicular to the feature axes. The decision regions are marked $\mathcal{R}_1$ and $\mathcal{R}_2$ in these two-dimensional and three-dimensional two-category examples. With a sufficiently large tree, any decision boundary can be approximated arbitrarily well in this way. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Entropy Impurity

- Each non-binary tree can be transformed to binary tree

Entropy Impurity

- Each non-binary tree can be transformed to binary tree
- Monothetic (single feature node) and polythetic (multiple features node) trees

Entropy Impurity

- Each non-binary tree can be transformed to binary tree
- Monothetic (single feature node) and polythetic (multiple features node) trees
- Any query at a node should gain maximal purity (or minimal impurity)

Entropy Impurity

- Each non-binary tree can be transformed to binary tree
- Monothetic (single feature node) and polythetic (multiple features node) trees
- Any query at a node should gain maximal purity (or minimal impurity)
- Entropy impurity of a node N with class "probabilities" P $i(N) = -\sum_j P(\omega_j) ld P(\omega_j)$

Entropy Impurity

- Each non-binary tree can be transformed to binary tree
- Monothetic (single feature node) and polythetic (multiple features node) trees
- Any query at a node should gain maximal purity (or minimal impurity)
- Entropy impurity of a node N with class "probabilities" P $i(N) = -\sum_j P(\omega_j) ld P(\omega_j)$
- $i(N) = 0 \rightarrow$ pure node

Other Impurity Measures

- *Gini* impurity (generalization of variance impurity)
  $i(N) = \sum_{i \neq j} P(\omega_i)P(\omega_j) = \frac{1}{2}[1 - \sum_j P^2(\omega_j)]$

Other Impurity Measures

- *Gini* impurity (generalization of variance impurity)
  $i(N) = \sum_{i \neq j} P(\omega_i)P(\omega_j) = \frac{1}{2}[1 - \sum_j P^2(\omega_j)]$
- Expected error rate at N (if pattern is selected from distribution at N)

Other Impurity Measures

- *Gini* impurity (generalization of variance impurity)
  $i(N) = \sum_{i \neq j} P(\omega_i)P(\omega_j) = \frac{1}{2}[1 - \sum_j P^2(\omega_j)]$
- Expected error rate at N (if pattern is selected from distribution at N)
- Misclassification impurity (discontinous derivative may cause problems)
  $i(N) = 1 - \max_j P(\omega_j)$

Other Impurity Measures

- *Gini* impurity (generalization of variance impurity)
  $i(N) = \sum_{i \neq j} P(\omega_i)P(\omega_j) = \frac{1}{2}[1 - \sum_j P^2(\omega_j)]$
- Expected error rate at N (if pattern is selected from distribution at N)
- Misclassification impurity (discontinous derivative may cause problems)
  $i(N) = 1 - \max_j P(\omega_j)$
- Minimal probability of a misclassified pattern at N

Greedy Query Search

- Select query with largest impurity decrease from N to N_L (left child) and N_R (right child)
  $\Delta i(N) = i(N) - P_L i(N_L) - (1 - P_L) i(N_R)$

Greedy Query Search

- Select query with largest impurity decrease from N to N_L (left child) and N_R (right child)
  $\Delta i(N) = i(N) - P_L i(N_L) - (1 - P_L) i(N_R)$
- Nominal features (exhaustive search), continous features (gradient descent)

Greedy Query Search

- Select query with largest impurity decrease from N to N_L (left child) and N_R (right child)
  $\Delta i(N) = i(N) - P_L i(N_L) - (1 - P_L) i(N_R)$
- Nominal features (exhaustive search), continous features (gradient descent)
- Specific choice of impurity measure is uncritical, more important are stop splitting and pruning methods

Greedy Query Search

- Select query with largest impurity decrease from N to N_L (left child) and N_R (right child)
  $\Delta i(N) = i(N) - P_L i(N_L) - (1 - P_L) i(N_R)$
- Nominal features (exhaustive search), continous features (gradient descent)
- Specific choice of impurity measure is uncritical, more important are stop splitting and pruning methods
- Multiway splits ($B > 2$), simple impurity decrease favors large splits, scaling of impurity decrease, Gain Ratio Impurity
  $\Delta i'(N, B) = \frac{\Delta i(N,B)}{-\sum_k P_k ld P_k}$

Stop Splitting Methods

- Naive Stop: each leaf node has impurity 0 (perfect overfitting), may degenerate to a look–up table (a leaf node for each pattern)

Stop Splitting Methods

- Naive Stop: each leaf node has impurity 0 (perfect overfitting), may degenerate to a look–up table (a leaf node for each pattern)
- Measure split performance with a separate validation set (minimal error on validation set)

Stop Splitting Methods

- Naive Stop: each leaf node has impurity 0 (perfect overfitting), may degenerate to a look–up table (a leaf node for each pattern)
- Measure split performance with a separate validation set (minimal error on validation set)
- Impurity threshold $\Delta i(N) \leq \beta$, unbalanced trees, choice of β?

Stop Splitting Methods

- Naive Stop: each leaf node has impurity 0 (perfect overfitting), may degenerate to a look–up table (a leaf node for each pattern)
- Measure split performance with a separate validation set (minimal error on validation set)
- Impurity threshold $\Delta i(N) \leq \beta$, unbalanced trees, choice of β?
- Pattern threshold: stop when a node represents a certain (small) number (percentage) of patterns

Stop Splitting Methods

- Naive Stop: each leaf node has impurity 0 (perfect overfitting), may degenerate to a look–up table (a leaf node for each pattern)
- Measure split performance with a separate validation set (minimal error on validation set)
- Impurity threshold $\Delta i(N) \leq \beta$, unbalanced trees, choice of β?
- Pattern threshold: stop when a node represents a certain (small) number (percentage) of patterns
- Minimum Description Length (regularization reduces complexity)
  $J(DT) = \alpha \# N + \sum_{LN} i(LN)$ $\quad$ (LN = leaf nodes)

Stop Splitting Methods

- Naive Stop: each leaf node has impurity 0 (perfect overfitting), may degenerate to a look–up table (a leaf node for each pattern)
- Measure split performance with a separate validation set (minimal error on validation set)
- Impurity threshold $\Delta i(N) \leq \beta$, unbalanced trees, choice of β?
- Pattern threshold: stop when a node represents a certain (small) number (percentage) of patterns
- Minimum Description Length (regularization reduces complexity)
  $J(DT) = \alpha \# N + \sum_{LN} i(LN)$ $\quad$ (LN = leaf nodes)
- Statistical significance of impurity reduction (distribution of Δi)

Pruning

- Stop Splitting: insufficient look–ahead (horizon effect) due to greedy search

Pruning

- Stop Splitting: insufficient look–ahead (horizon effect) due to greedy search
- Pruning: merge nodes, starts at leaf nodes, but any node is possible

Pruning

- Stop Splitting: insufficient look–ahead (horizon effect) due to greedy search
- Pruning: merge nodes, starts at leaf nodes, but any node is possible
- Uses complete data set, huge cost with large data sets

Pruning

- Stop Splitting: insufficient look–ahead (horizon effect) due to greedy search
- Pruning: merge nodes, starts at leaf nodes, but any node is possible
- Uses complete data set, huge cost with large data sets
- Rule pruning: construct and simplify rules (conjunctions) for each leaf

Pruning

- Stop Splitting: insufficient look–ahead (horizon effect) due to greedy search
- Pruning: merge nodes, starts at leaf nodes, but any node is possible
- Uses complete data set, huge cost with large data sets
- Rule pruning: construct and simplify rules (conjunctions) for each leaf
- Context pruning: prune specific rules for specific patterns

Pruning

- Stop Splitting: insufficient look–ahead (horizon effect) due to greedy search
- Pruning: merge nodes, starts at leaf nodes, but any node is possible
- Uses complete data set, huge cost with large data sets
- Rule pruning: construct and simplify rules (conjunctions) for each leaf
- Context pruning: prune specific rules for specific patterns
- Improved interpretability

Feature Extraction

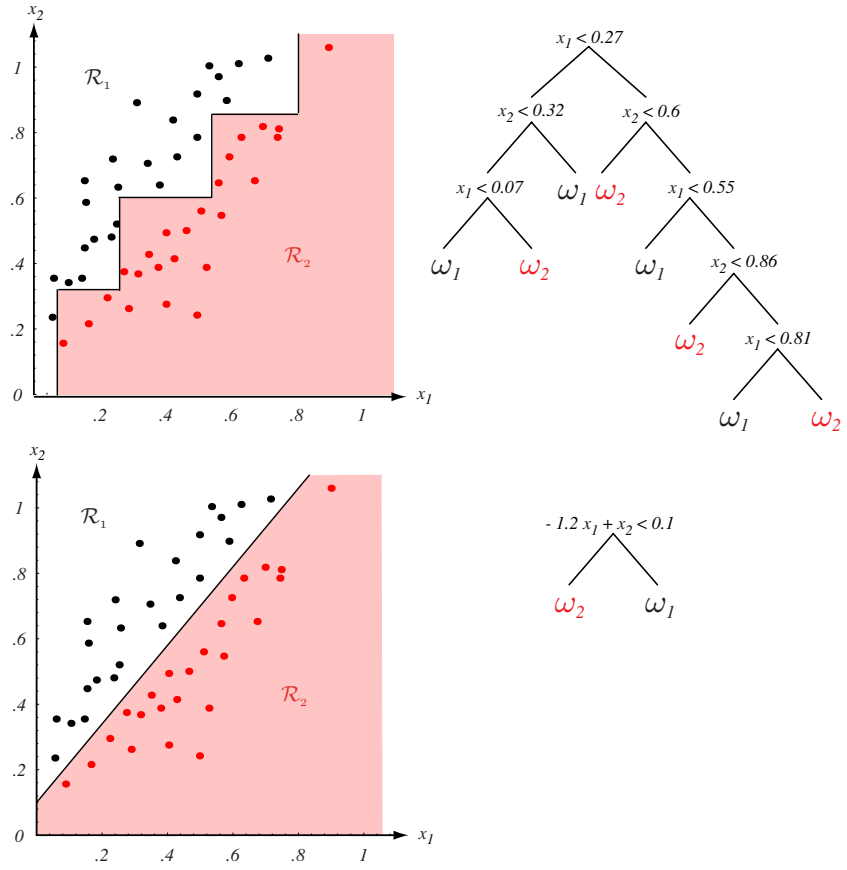


FIGURE 8.5. If the class of node decisions does not match the form of the training data, a very complicated decision tree will result, as shown at the top. Here decisions are parallel to the axes while in fact the data is better split by boundaries along another direction. If, however, "proper" decision forms are used (here, linear combinations of the features), the tree can be quite simple, as shown at the bottom. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Potential Improvements

- At each node train a linear classifier, arbitrary linear decision boundaries

Potential Improvements

- At each node train a linear classifier, arbitrary linear decision boundaries
- Long training, (again) fast recall

Potential Improvements

- At each node train a linear classifier, arbitrary linear decision boundaries
- Long training, (again) fast recall
- Integrate priors and/or costs by weights

Potential Improvements

- At each node train a linear classifier, arbitrary linear decision boundaries
- Long training, (again) fast recall
- Integrate priors and/or costs by weights
- Weighted *Gini* Impurity with cost λ_{ij}
  $i(N) = \sum_{ij} \lambda_{ij} P(\omega_i) P(\omega_j)$

Multivariate Decision Trees

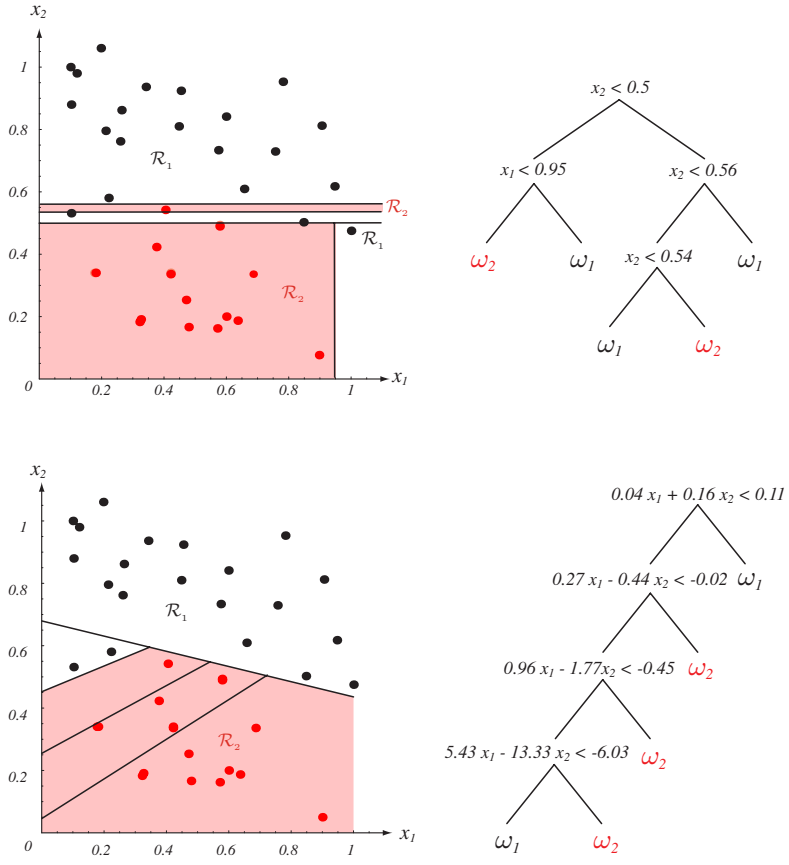


FIGURE 8.6. One form of multivariate tree employs general linear decisions at each node, giving splits along arbitrary directions in the feature space. In virtually all interesting cases the training data are not linearly separable, and thus the LMS algorithm is more useful than methods that require the data to be linearly separable, even though the LMS need not yield a minimum in classification error (Chapter 5). The tree at the bottom can be simplified by methods outlined in Section 8.4.2. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Missing Attributes

- Naive approach: use only non-deficient patterns

Missing Attributes

- Naive approach: use only non-deficient patterns
- Better: use only non–deficient attributes

Missing Attributes

- Naive approach: use only non-deficient patterns
- Better: use only non–deficient attributes
- Works with training, but how to classify a deficient pattern?

Missing Attributes

- Naive approach: use only non-deficient patterns
- Better: use only non–deficient attributes
- Works with training, but how to classify a deficient pattern?
- Surrogate splits: find alternative splits using different features having maximal predictive association (correlation)

Missing Attributes

- Naive approach: use only non-deficient patterns
- Better: use only non–deficient attributes
- Works with training, but how to classify a deficient pattern?
- Surrogate splits: find alternative splits using different features having maximal predictive association (correlation)
- Virtual values, e.g., mean value of non–deficient feature values

ID3

- ID3 stems from third interactive dichotomizer

ID3

- ID3 stems from third interactive dichotomizer
- Nominal features (real are binned)

ID3

- ID3 stems from third interactive dichotomizer
- Nominal features (real are binned)
- Branch factor is number of attributes

ID3

- ID3 stems from third interactive dichotomizer
- Nominal features (real are binned)
- Branch factor is number of attributes
- Train until all nodes pure or no more features

ID3

- ID3 stems from third interactive dichotomizer
- Nominal features (real are binned)
- Branch factor is number of attributes
- Train until all nodes pure or no more features
- Results in tree depth = number of features

ID3

- ID3 stems from third interactive dichotomizer
- Nominal features (real are binned)
- Branch factor is number of attributes
- Train until all nodes pure or no more features
- Results in tree depth = number of features
- No pruning

C4.5

- Refinement of ID3

C4.5

- Refinement of ID3
- $B > 2$ with nominal features, $B = 2$ with real features

C4.5

- Refinement of ID3
- $B > 2$ with nominal features, $B = 2$ with real features
- Pruning based on statistical significance of splits

C4.5

- Refinement of ID3
- $B > 2$ with nominal features, $B = 2$ with real features
- Pruning based on statistical significance of splits
- Missing features: sample all subtrees of missing feature using training data

C4.5

- Refinement of ID3
- $B > 2$ with nominal features, $B = 2$ with real features
- Pruning based on statistical significance of splits
- Missing features: sample all subtrees of missing feature using training data
- Additional rule pruning, can prune any node (see Figure 8.6)

Stochastic Search

- Analytical methods problematic in high dimensions or with complex models

Stochastic Search

- Analytical methods problematic in high dimensions or with complex models
- Large number of local optima makes gradient descent very costly

Stochastic Search

- Analytical methods problematic in high dimensions or with complex models
- Large number of local optima makes gradient descent very costly
- Stochastic methods try to localize promising search regions

Stochastic Search

- Analytical methods problematic in high dimensions or with complex models
- Large number of local optima makes gradient descent very costly
- Stochastic methods try to localize promising search regions
- Pure random search is often not sufficient

Stochastic Search

- Analytical methods problematic in high dimensions or with complex models
- Large number of local optima makes gradient descent very costly
- Stochastic methods try to localize promising search regions
- Pure random search is often not sufficient
- *Simulated Annealing* and *Boltzmann Learning* motivated by statistical mechanics

Stochastic Search

- Analytical methods problematic in high dimensions or with complex models
- Large number of local optima makes gradient descent very costly
- Stochastic methods try to localize promising search regions
- Pure random search is often not sufficient
- *Simulated Annealing* and *Boltzmann Learning* motivated by statistical mechanics
- *Evolutionary Computation* motivated by evolutionary principles from biology

Energy Minimization

- Example: minimizing (model) energy in a (Hopfield) network

Energy Minimization

- Example: minimizing (model) energy in a (Hopfield) network
- Energy $E = -\frac{1}{2} \sum_{i,j=1}^{N} w_{ij} s_i s_j \quad s_i = \pm 1$

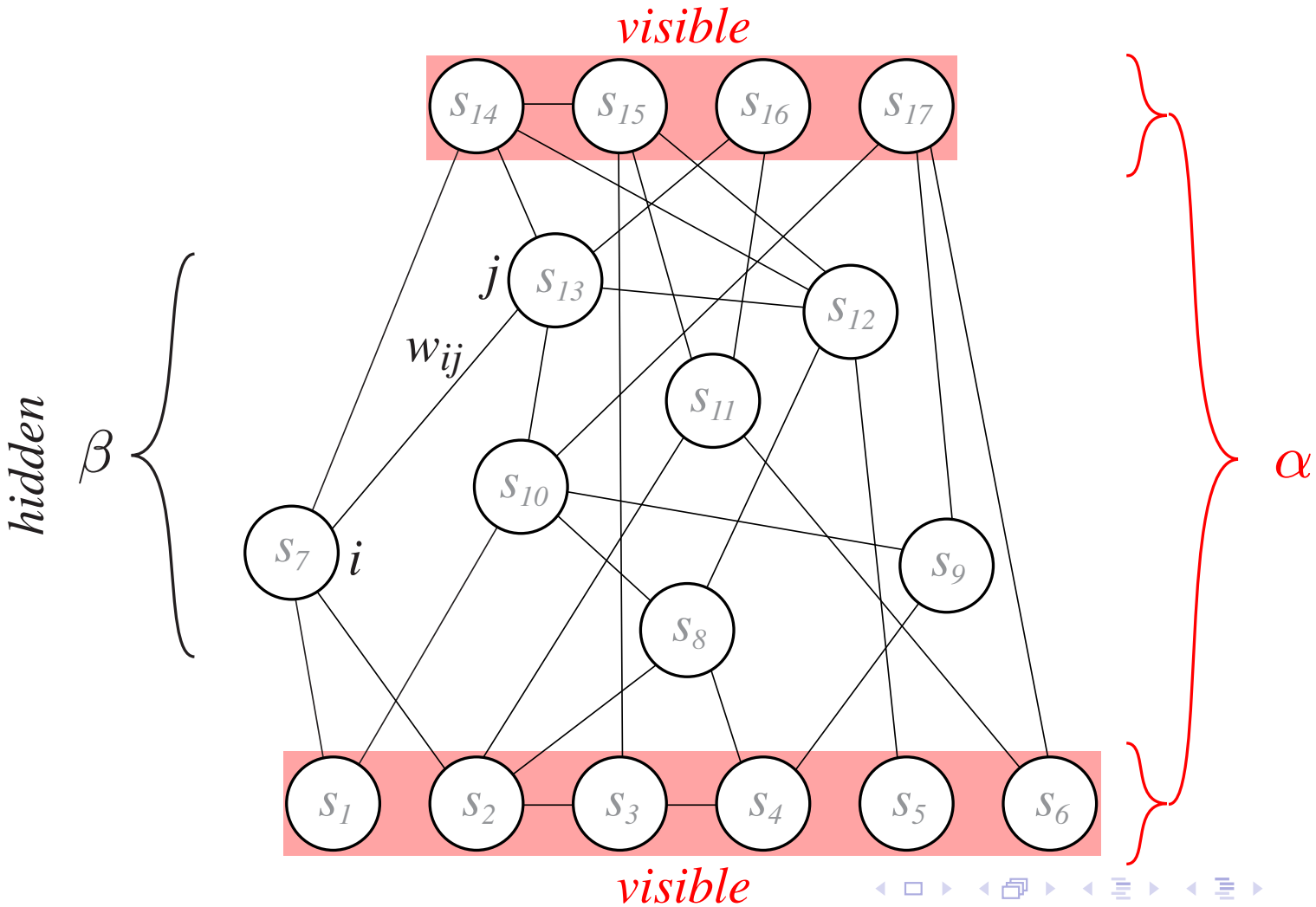
Energy Minimization

- Example: minimizing (model) energy in a (Hopfield) network
- Energy $E = -\frac{1}{2} \sum_{i,j=1}^{N} w_{ij} s_i s_j \quad s_i = \pm 1$
- Minimize energy of spin–glass model

Energy Minimization

- Example: minimizing (model) energy in a (Hopfield) network
- Energy $E = -\frac{1}{2}\sum_{i,j=1}^{N} w_{ij} s_i s_j \quad s_i = \pm 1$
- Minimize energy of spin–glass model
- Probability of energy state, *Boltzmann* factor
  $P(\gamma) = \frac{e^{\frac{-E_\gamma}{T}}}{Z(T)}$

Recurrent Net



Energy Landscape

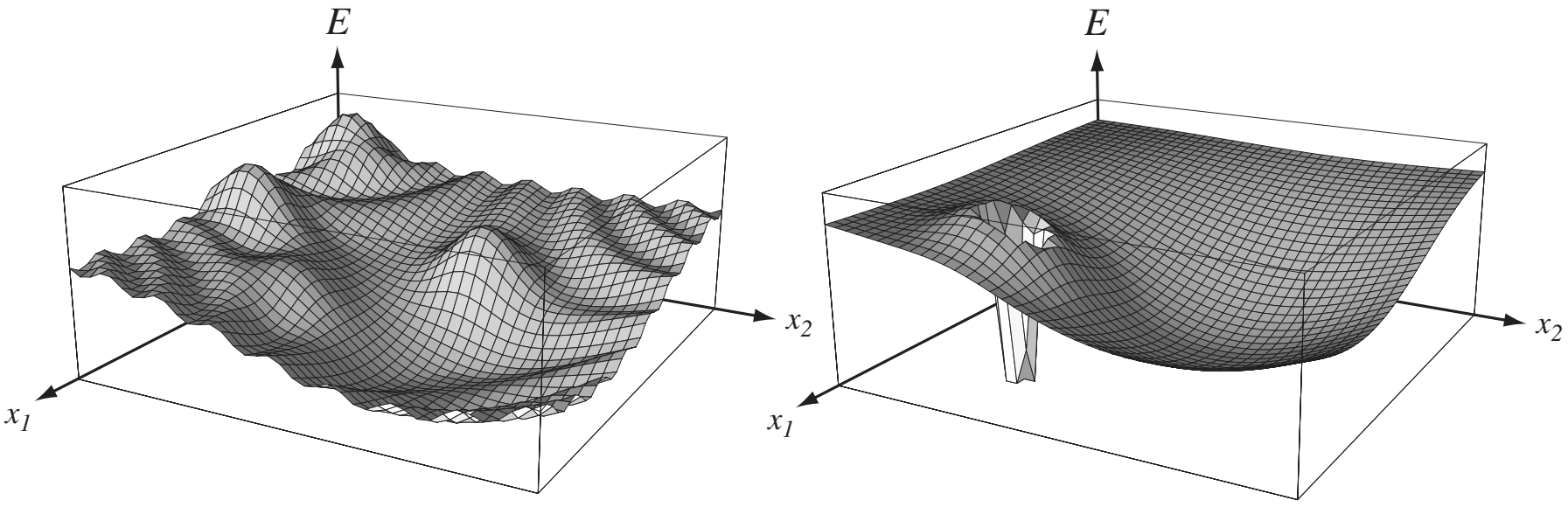


FIGURE 7.2. The energy function or energy "landscape" on the left is meant to suggest the types of optimization problems addressed by simulated annealing. The method uses randomness, governed by a control parameter or "temperature" T to avoid getting stuck in local energy minima and thus to find the global minimum, like a small ball rolling in the landscape as it is shaken. The pathological "golf course" landscape at the right is, generally speaking, not amenable to solution via simulated annealing because the region of lowest energy is so small and is surrounded by energetically unfavorable configurations. The configuration spaces of the problems we shall address are discrete and are more accurately displayed in Fig. 7.6. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Simulated Annealing Basics

- Stochastic search for state of lower energy

Simulated Annealing Basics

- Stochastic search for state of lower energy
- Basic idea: occasionally go to higher energy to possibly escape local minima

Simulated Annealing Basics

- Stochastic search for state of lower energy
- Basic idea: occasionally go to higher energy to possibly escape local minima
- After random change of parameter s_i
  $\Delta E_{ab} = E_b - E_a$
  accept E_b, if $E_b < E_a$ or
  accept E_a with $P = e^{\frac{-\Delta E_{ab}}{T}}$

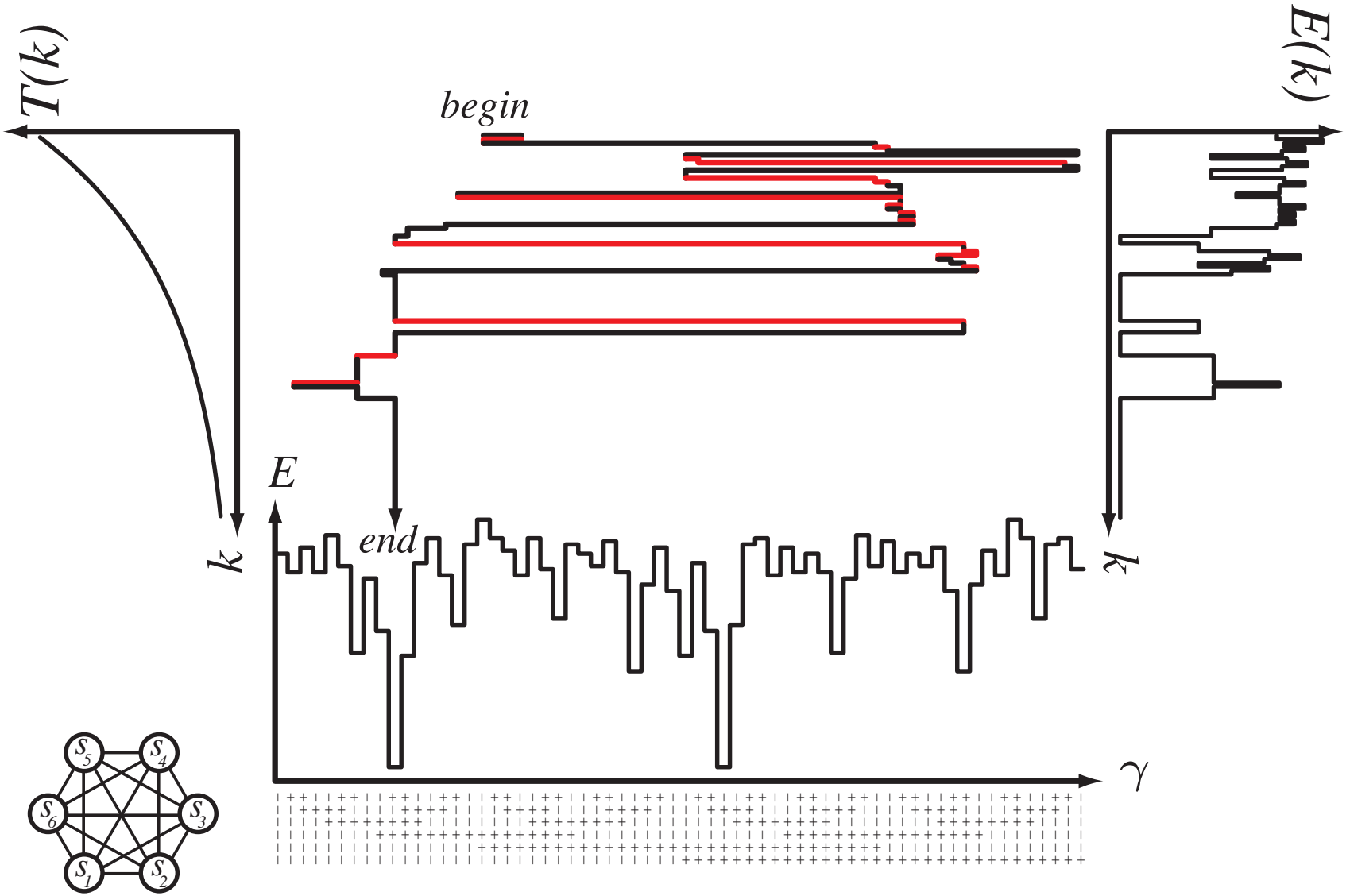
Simulated Annealing Basics

- Stochastic search for state of lower energy
- Basic idea: occasionally go to higher energy to possibly escape local minima
- After random change of parameter s_i
  $\Delta E_{ab} = E_b - E_a$
  accept E_b, if $E_b < E_a$ or
  accept E_a with $P = e^{\frac{-\Delta E_{ab}}{T}}$
- Annealing Schedule, e.g., $T(k+1) = cT(k) \quad 0 < c < 1$
  typically $0.8 < c < 0.99$

Simulated Annealing Basics

- Stochastic search for state of lower energy
- Basic idea: occasionally go to higher energy to possibly escape local minima
- After random change of parameter s_i
  $\Delta E_{ab} = E_b - E_a$
  accept E_b, if $E_b < E_a$ or
  accept E_a with $P = e^{\frac{-\Delta E_{ab}}{T}}$
- Annealing Schedule, e.g., $T(k+1) = cT(k) \quad 0 < c < 1$
  typically $0.8 < c < 0.99$
- High initial temperature, large c and large k_{max} (number of iterations) leads to good results (but also computational cost)

Simulated Annealing Experiment



Empirical Energy States

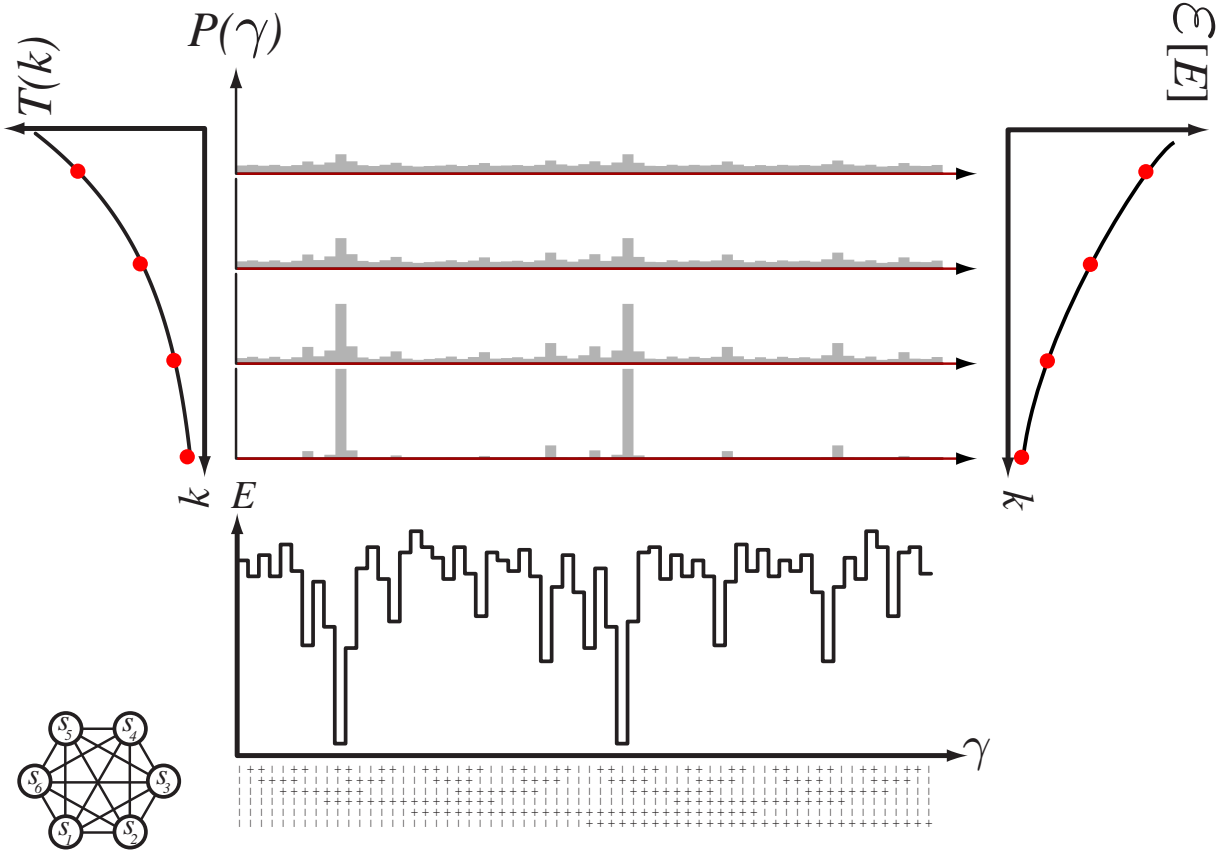


FIGURE 7.4. An estimate of the probability $P(\gamma)$ of being in a configuration denoted by γ is shown for four temperatures during a slow anneal. (These estimates, based on a large number of runs, are nearly the theoretical values e^{-E_γ}/T.) Early, at high T, each configuration is roughly equal in probability while late, at low T, the probability is strongly concentrated at the global minima. The expected value of the energy, $\mathcal{E}[E]$ (i.e., averaged at temperature T), decreases gradually during the anneal. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Project Teams

- 2 students form a team

Project Teams

- 2 students form a team
- Implementation of a pattern classification method

Project Teams

- 2 students form a team
- Implementation of a pattern classification method
- Use existing (free) software (e.g., WEKA)

Project Teams

- 2 students form a team
- Implementation of a pattern classification method
- Use existing (free) software (e.g., WEKA)
- Data import, pre–processing, results (graphics, tables)

Project Teams

- 2 students form a team
- Implementation of a pattern classification method
- Use existing (free) software (e.g., WEKA)
- Data import, pre–processing, results (graphics, tables)
- Methods must be understood, parameters!

Project Teams

- 2 students form a team
- Implementation of a pattern classification method
- Use existing (free) software (e.g., WEKA)
- Data import, pre–processing, results (graphics, tables)
- Methods must be understood, parameters!
- Project report (February 10, 2014)

Project Topics

- k-NN Classifier (different metrics)
  Kauba, *Mayer*

Project Topics

- k-NN Classifier (different metrics)
  Kauba, *Mayer*
- Artificial Neural Networks (Boone)
  Reissig, *DiStolfo*

Project Topics

- k-NN Classifier (different metrics)
  Kauba, *Mayer*
- Artificial Neural Networks (Boone)
  Reissig, *DiStolfo*
- Support Vector Machine
  Linortner, N.N.

Project Topics

- k-NN Classifier (different metrics)
  Kauba, *Mayer*
- Artificial Neural Networks (Boone)
  Reissig, *DiStolfo*
- Support Vector Machine
  Linortner, N.N.
- Decision Tree (C4.5)

Project Topics

- k-NN Classifier (different metrics)
  Kauba, *Mayer*
- Artificial Neural Networks (Boone)
  Reissig, *DiStolfo*
- Support Vector Machine
  Linortner, N.N.
- Decision Tree (C4.5)
- Simulated Annealing (meta)

Project Topics

- k-NN Classifier (different metrics)
  Kauba, *Mayer*
- Artificial Neural Networks (Boone)
  Reissig, *DiStolfo*
- Support Vector Machine
  Linortner, N.N.
- Decision Tree (C4.5)
- Simulated Annealing (meta)
- Genetic Algorithm (meta, JEvolution)
  Auracher, *Herzog*, *Kirchgasser*

Project Topics

- k-NN Classifier (different metrics)
  Kauba, *Mayer*
- Artificial Neural Networks (Boone)
  Reissig, *DiStolfo*
- Support Vector Machine
  Linortner, N.N.
- Decision Tree (C4.5)
- Simulated Annealing (meta)
- Genetic Algorithm (meta, JEvolution)
  Auracher, *Herzog*, *Kirchgasser*
- Genetic Programming (optional, JEvolution)

Project Data Sets

- Data Sets

Project Data Sets

- Data Sets
  - UCI Machine Learning Archive

Project Data Sets

- Data Sets
  - UCI Machine Learning Archive
  - `http://www.ics.uci.edu/~mlearn/MLRepository.html`

Project Data Sets

- Data Sets
  - UCI Machine Learning Archive
  - `http://www.ics.uci.edu/~mlearn/MLRepository.html`
  - **Ionosphere**: Radar Signals

Project Data Sets

- Data Sets
  - UCI Machine Learning Archive
  - `http://www.ics.uci.edu/~mlearn/MLRepository.html`
  - **Ionosphere**: Radar Signals
  - **Semeion Handwritten Digit**: Digit Recognition

Project Data Sets

- Data Sets
  - UCI Machine Learning Archive
  - `http://www.ics.uci.edu/~mlearn/MLRepository.html`
  - **Ionosphere**: Radar Signals
  - **Semeion Handwritten Digit**: Digit Recognition
  - **Wine Quality**: Wine Critic

Project Data Sets

- Data Sets
  - UCI Machine Learning Archive
  - `http://www.ics.uci.edu/~mlearn/MLRepository.html`
  - **Ionosphere**: Radar Signals
  - **Semeion Handwritten Digit**: Digit Recognition
  - **Wine Quality**: Wine Critic
- Leave–one–out validation (common partitioning)

Project Data Sets

- Data Sets
  - UCI Machine Learning Archive
  - `http://www.ics.uci.edu/~mlearn/MLRepository.html`
  - **Ionosphere**: Radar Signals
  - **Semeion Handwritten Digit**: Digit Recognition
  - **Wine Quality**: Wine Critic
- Leave–one–out validation (common partitioning)
- Confusion matrix