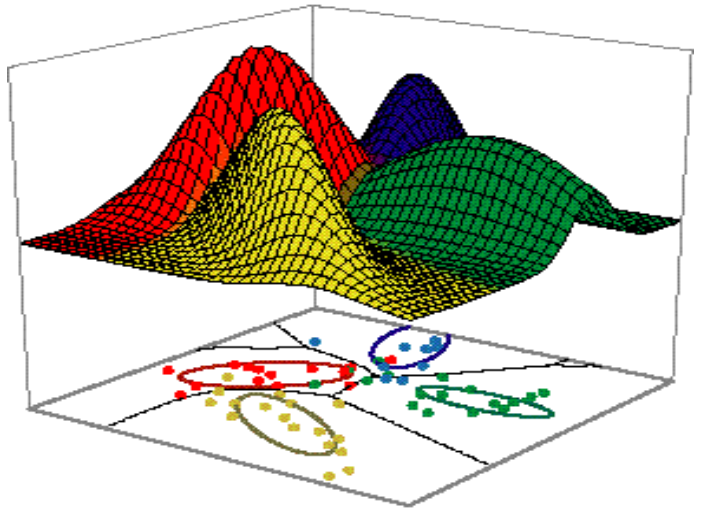
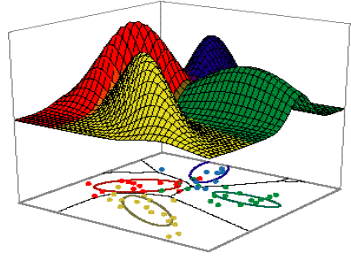
Part 8:
Linear Discriminants



Introduction
Linear Discriminant Functions and Decision Surfaces
2-Category Linearly Separable Case
Perceptron Criterion
Relaxation Procedures
MSE Procedures
Generalized Linear Discriminant Functions
Support Vector Machines

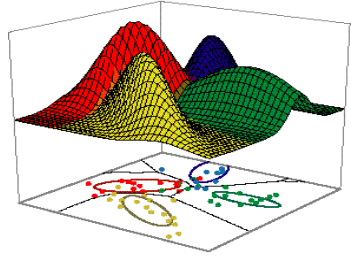
Some materials in these slides were taken from Pattern Classification (2nd ed) by R. O. Duda, P. E. Hart and D. G. Stork, John Wiley & Sons, 2000, **Chapter 5.1-8,11**
Other materials adapted from J.Tian, CS 474 course notes, Iowa State.

Introduction (5.1)



- In parameter estimation, the underlying probability densities were known (or given)
  - The training sample was used to estimate the parameters of these probability densities (ML, MAP/Bayes estimations)
- In this chapter, we only know the proper forms for the discriminant functions: similar to non-parametric techniques
  - They may not be optimal, but they are very simple to use
  - They provide us with linear classifiers

Linear discriminant functions and decision surfaces (5.2.1)



- Definition

  It is a function that is a linear combination of the components of x

  $$g(x) = w^t x + w_0 \qquad (1)$$

  where w is the weight vector and w_0 the bias

- A two-category classifier with a discriminant function of the form (1) uses the following rule:

  Decide ω_1 *if* $g(x) > 0$ and ω_2 if $g(x) < 0$

  $\Leftrightarrow$ Decide ω_1 *if* $w^t x > -w_0$ and ω_2 otherwise

  If $g(x) = 0 \Rightarrow x$ is assigned to either class

Linear discriminant functions and decision surfaces (5.2.1)

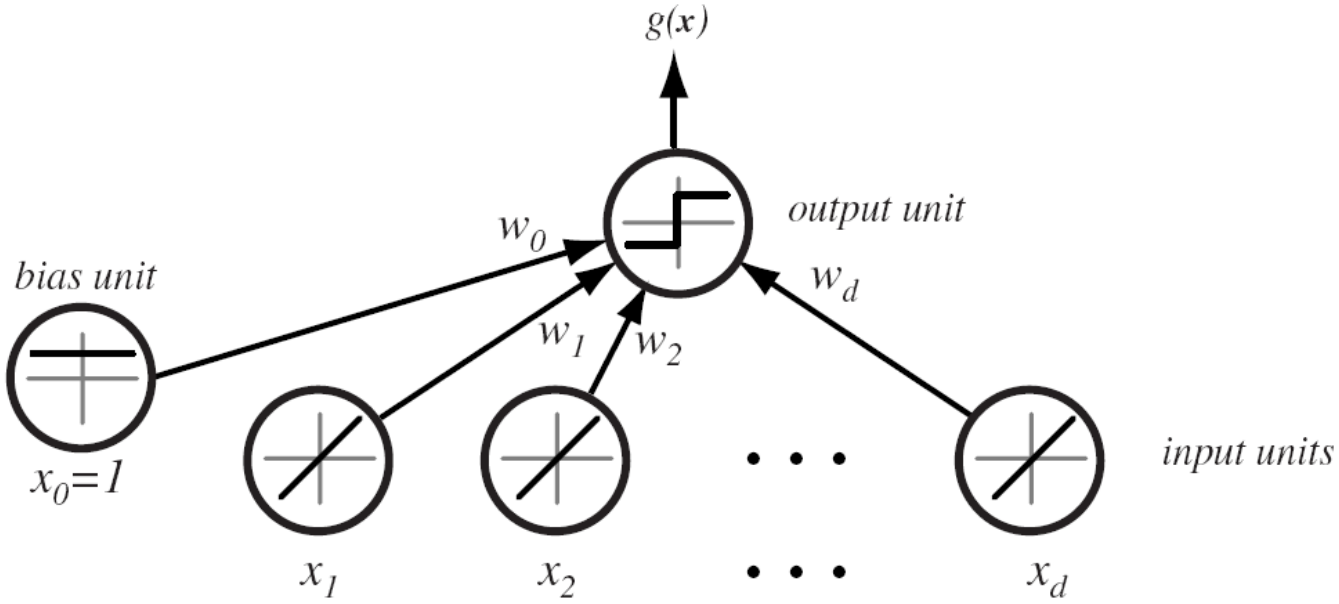
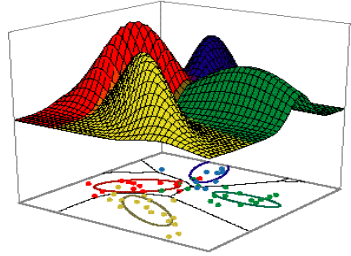


FIGURE 5.1. A simple linear classifier having d input units, each corresponding to the values of the components of an input vector. Each input feature value x_i is multiplied by its corresponding weight w_i; the effective input at the output unit is the sum all these products, $\sum w_i x_i$. We show 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 and decision surfaces (5.2.1)



- The equation $g(x) = 0$ defines the decision surface that separates points assigned to the category ω_1 from points assigned to the category ω_2
- When $g(x)$ is linear, the decision surface is a hyperplane
- Algebraic measure of the distance from x to the hyperplane (see next 2 slides)

Linear discriminant functions and decision surfaces (5.2.1)

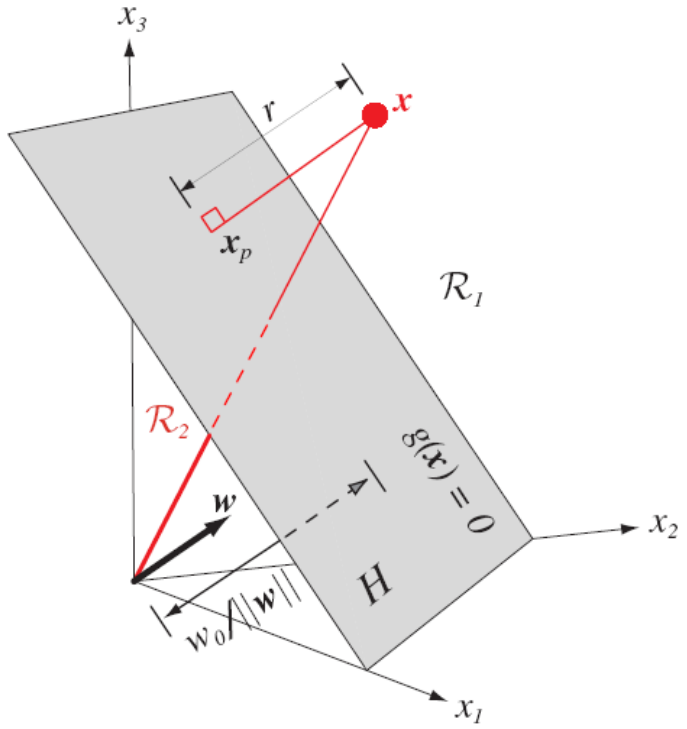
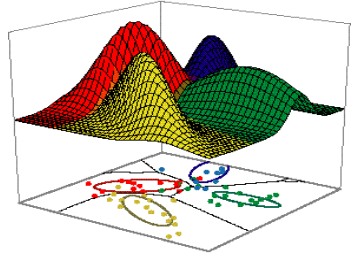


FIGURE 5.2. The linear decision boundary H, where $g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = 0$, separates the feature space into two half-spaces $\mathcal{R}_1$ (where $g(\mathbf{x}) > 0$) and $\mathcal{R}_2$ (where $g(\mathbf{x}) < 0$). From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Linear discriminant functions and decision surfaces (5.2.1)



1) $x = x_p + r\frac{w}{\|w\|}$

x_p is the normal projection of x onto H
w is colinear with ($x-x_p$) and is normal to H
r is distance from x to hyperplane H

2) $\|w\| = \sqrt{w^t w} = \sqrt{w_1^2 + w_2^2 + \ldots + w_d^2}$

Since $g(x_p)=0$, then : $g(x_p) = w^t x_p + w_0 = 0 \rightarrow w_0 = -w^t x_p$ 3)

4) $g(x) = w^t x + w_0$
$= r\|w\|$

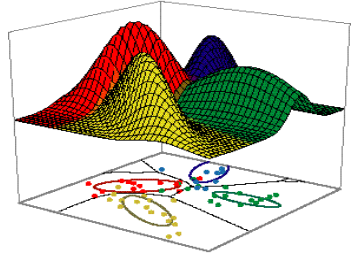
OR

5) $r = \frac{g(x)}{\|w\|}$

In particular, the distance from the origin to H is: 6) $d(0,H) = \frac{w_0}{\|w\|}$

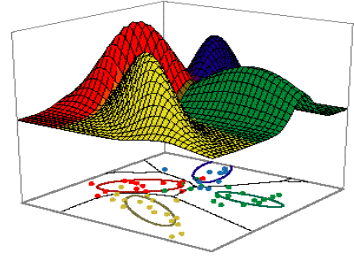
- In conclusion, a linear discriminant function divides the feature space by a hyperplane decision surface
- The magnitude of g(x) indicates how far from the decision HP x is
- The orientation of the surface is determined by the normal vector w and the location of the surface is determined by the bias w_0

Linear discriminant functions and decision surfaces – Multi-category case (5.2.2)



- There are several ways to reduce a multi-category problem into a series of 2-class problems
  - Two wrong ways (assuming c categories):
    - 1) Define c binary problems where the i^{th} classifier divides the space into "ω_i vs. not_ω_i"
    - 2) Define c(c-1)/2 linear discriminants, one for every pair of classes
  - Both of these will result in ambiguous regions.

Linear discriminant functions and decision surfaces – Multi-category case (5.2.2)



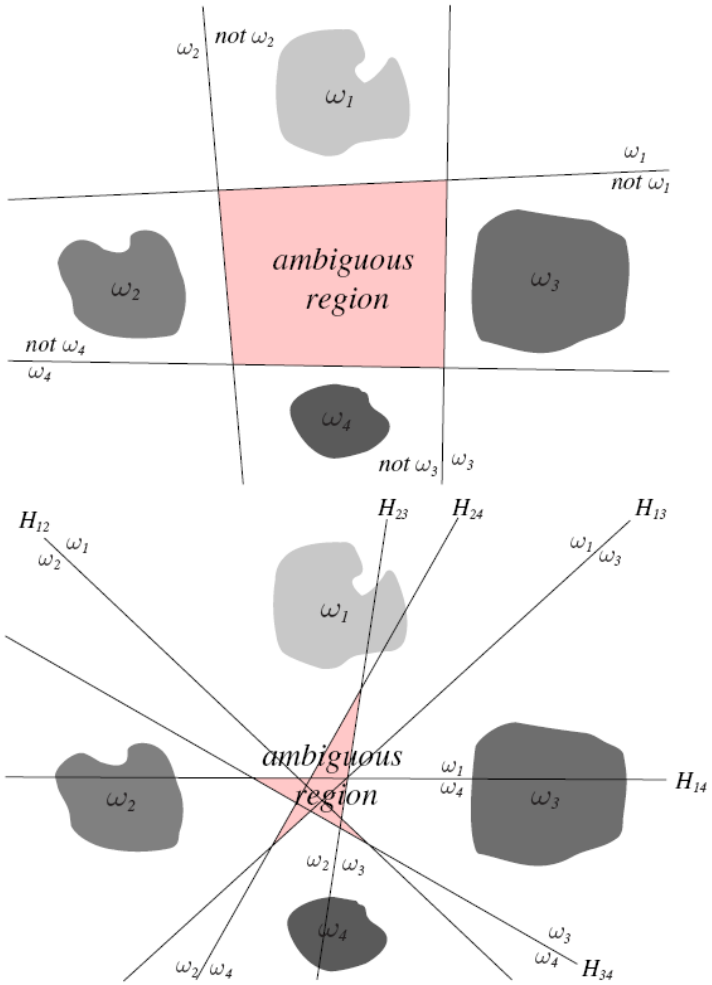
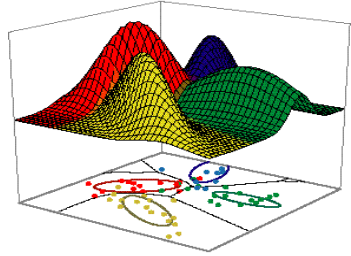


FIGURE 5.3. Linear decision boundaries for a four-class problem. The top figure shows ω_i/not ω_i dichotomies while the bottom figure shows ω_i/ω_j dichotomies and the corresponding decision boundaries H_{ij}. The pink regions have ambiguous category assignments. From: Richard O. Duda, Peter E. Hart,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

Linear discriminant functions and decision surfaces – Multi-category case (5.2.2)



- Instead, we define c linear discriminant functions

$$g_i(x) = w_i^t x + w_{i0} \qquad i = 1,\ldots,c$$

  and assign x to ω_i if $g_i(x) > g_j(x) \;\forall j \neq i$;
  - in case of ties, the classification is undefined
- In this case, the classifier is a "linear machine"
  - Also known as a 'winner takes all network'
- A linear machine divides the feature space into c decision regions, with $g_i(x)$ being the largest discriminant if x is in the region $\mathcal{R}_i$
- For two contiguous regions $\mathcal{R}_i$ and $\mathcal{R}_j$, the boundary that separates them is a portion of hyperplane H_{ij} defined by:

$$g_i(x) = g_j(x) \quad \Rightarrow \quad (w_i - w_j)^t x + (w_{i0} - w_{j0}) = 0$$

- $(w_i - w_j)$ is normal to H_{ij} and $d(x, H_{ij}) = \dfrac{g_i - g_j}{\|w_i - w_j\|}$

Linear discriminant functions and decision surfaces – Multi-category case (5.2.2)

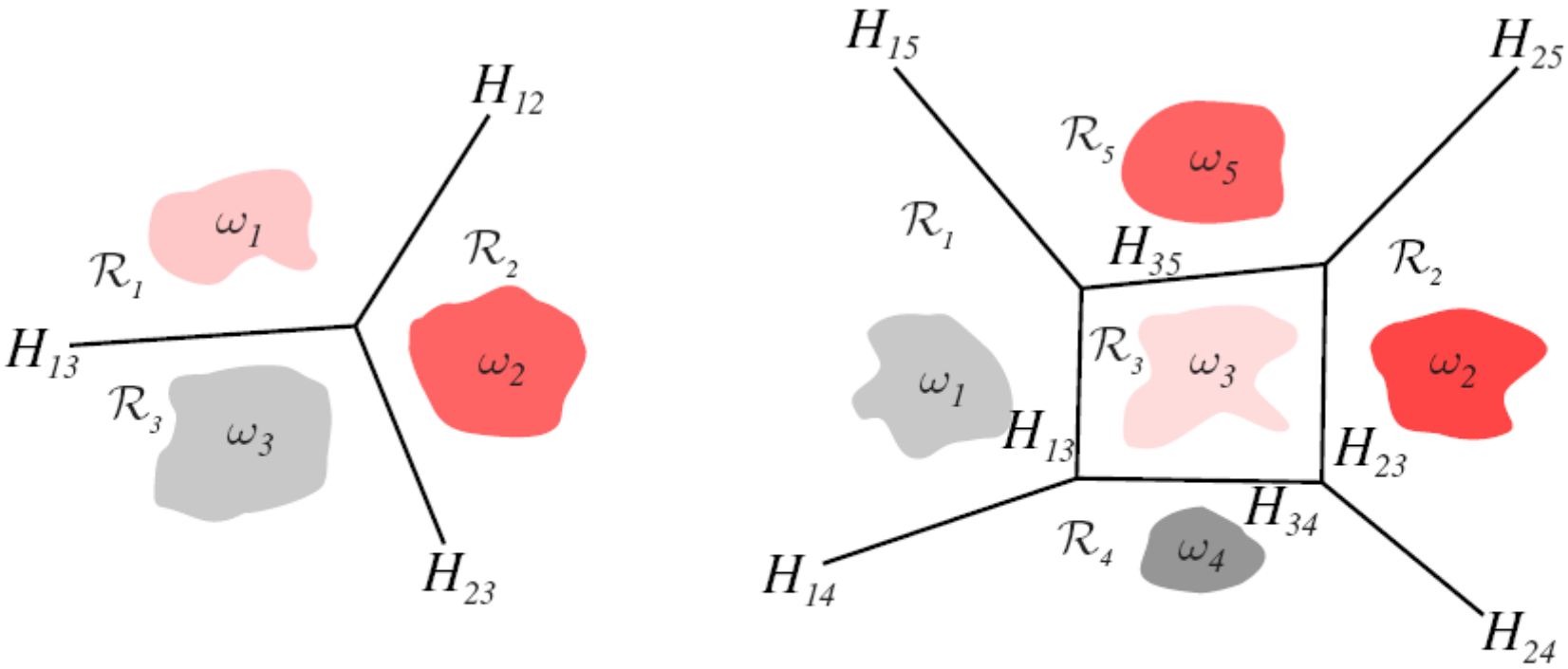
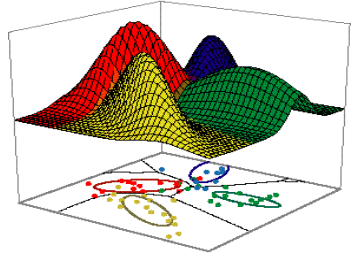


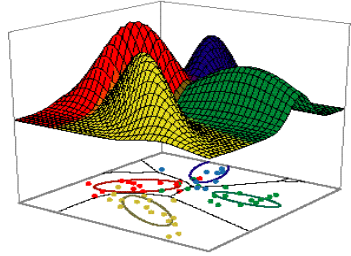
FIGURE 5.4. Decision boundaries produced by a linear machine for a three-class problem and a five-class problem. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Linear discriminant functions and decision surfaces – Multi-category case (5.2.2)



- It is easy to show that the decision regions for a linear machine are convex, this restriction limits the flexibility and accuracy of the classifier
  - If 2 points fall on the same side of the discriminant, all points between them also fall on that same side.
  - Every decision region is singly connected
    - Linear machines *tend to be* most suitable when class-conditional distributions are unimodal

Augmented Vectors (5.3)



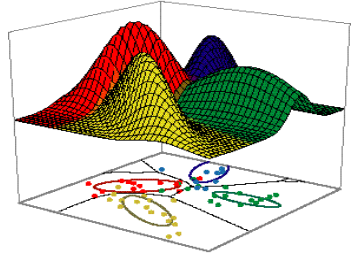
- Would like to write $g(x)=a^t y$
  - We have:

$$g(x) = w_0 + w^t x = w_0 + \sum_{i=1}^{d} w_i x_i = \sum_{i=0}^{d} w_i x_i \quad \text{if we let } x_0 = 1$$

  - Thus we create the augmented feature vector y and the augmented weight vector a:

$$y = \begin{bmatrix} 1 \\ x_1 \\ \dots \\ x_d \end{bmatrix} = \begin{bmatrix} 1 \\ x \end{bmatrix} \qquad a = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_d \end{bmatrix} = \begin{bmatrix} w_0 \\ w \end{bmatrix}$$

Augmented Vectors (5.3)



- Convenient to map from d-dim x-space to (d+1)-dim y-space
- Decision hyperplane passes through origin of y-space and is defined by $g(x)=a^t y=0$
  - (was not constrained to pass through origin in x-space)
  - Distance from y to H' decision hyperplane is ≤ distance from x to H:

$$\frac{|a^t y|}{\|a\|} = \frac{|g(x)|}{\|a\|} \leq \frac{|g(x)|}{\|w\|} \quad \text{since } \|a\| \geq \|w\|$$

- Thus, we reduce problem of finding a weight vector w and a threshold weight w_0 to the problem of finding a single weight vector a.

Augmented Vectors (5.3)

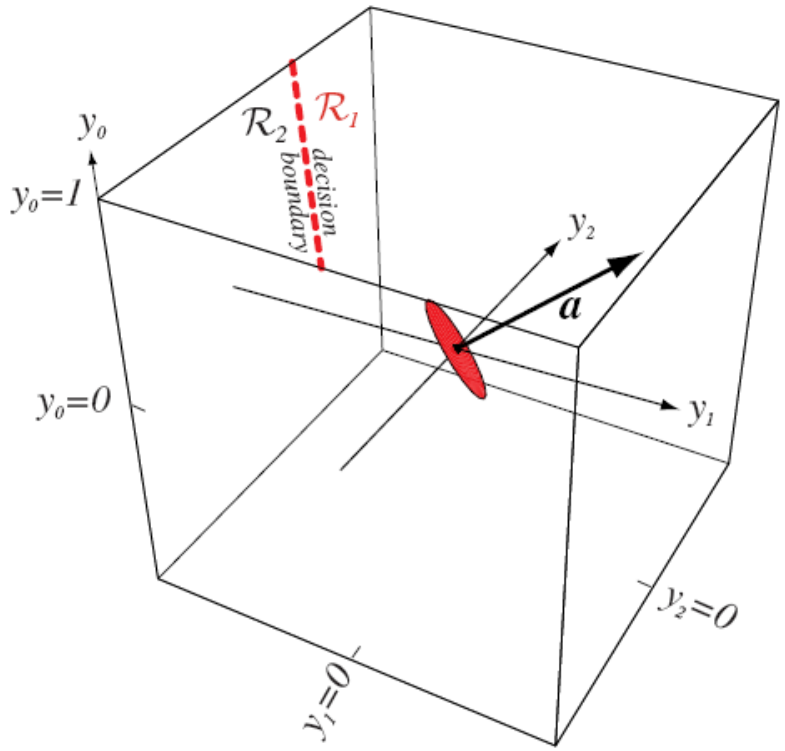
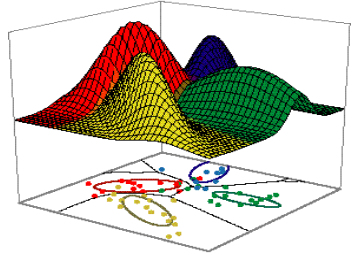
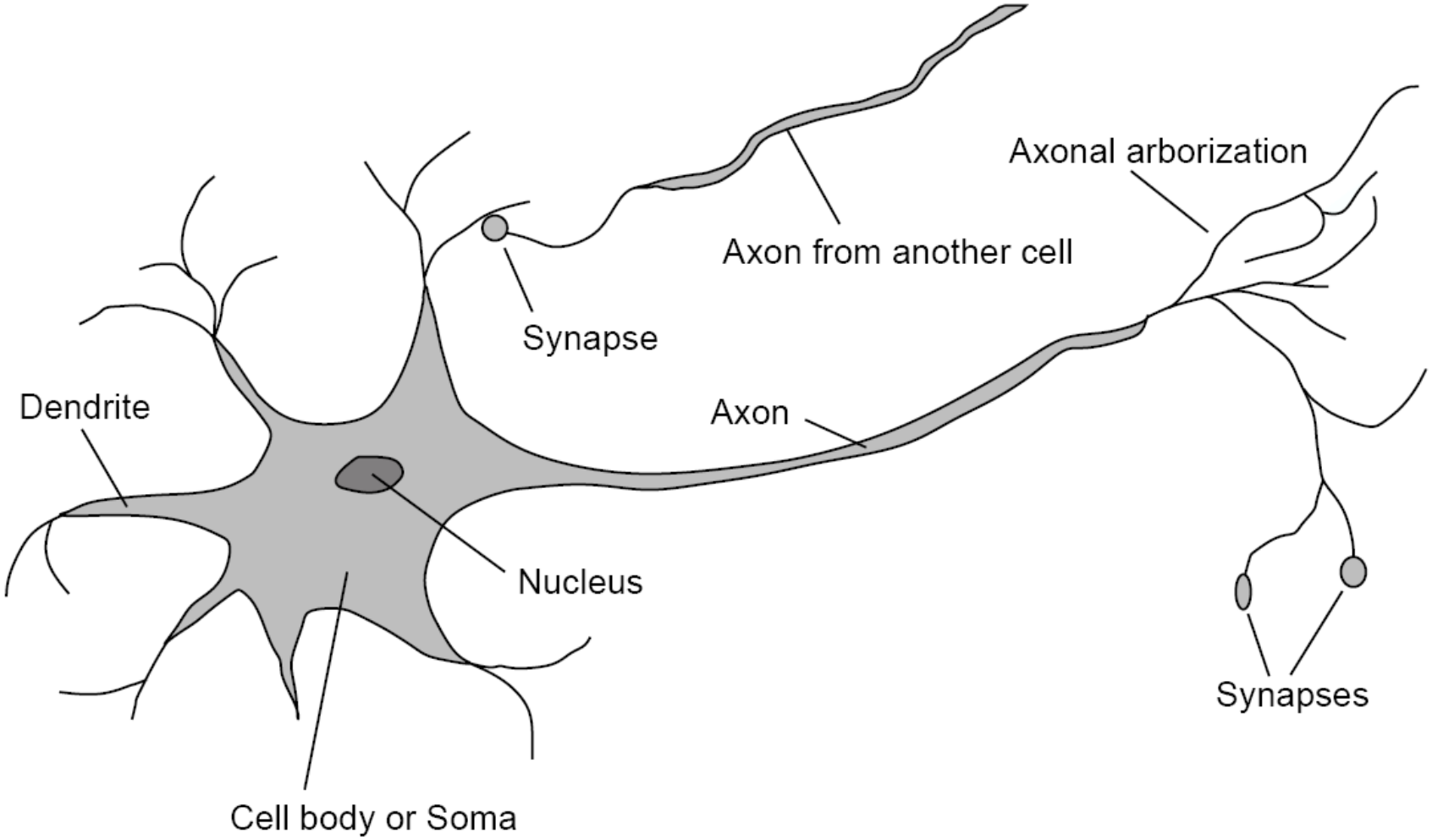


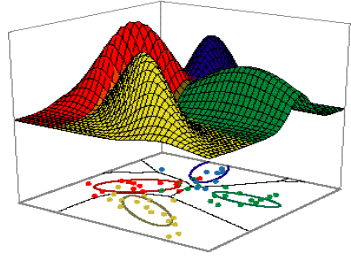
FIGURE 5.7. A three-dimensional augmented feature space $\mathbf{y}$ and augmented weight vector $\mathbf{a}$ (at the origin). The set of points for which $\mathbf{a}^t\mathbf{y} = 0$ is a plane (or more generally, a hyperplane) perpendicular to $\mathbf{a}$ and passing through the origin of $\mathbf{y}$-space, as indicated by the red disk. Such a plane need not pass through the origin of the two-dimensional feature space of the problem, as illustrated by the dashed decision boundary shown at the top of the box. Thus there exists an augmented weight vector $\mathbf{a}$ that will lead to any straight decision line in $\mathbf{x}$-space. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

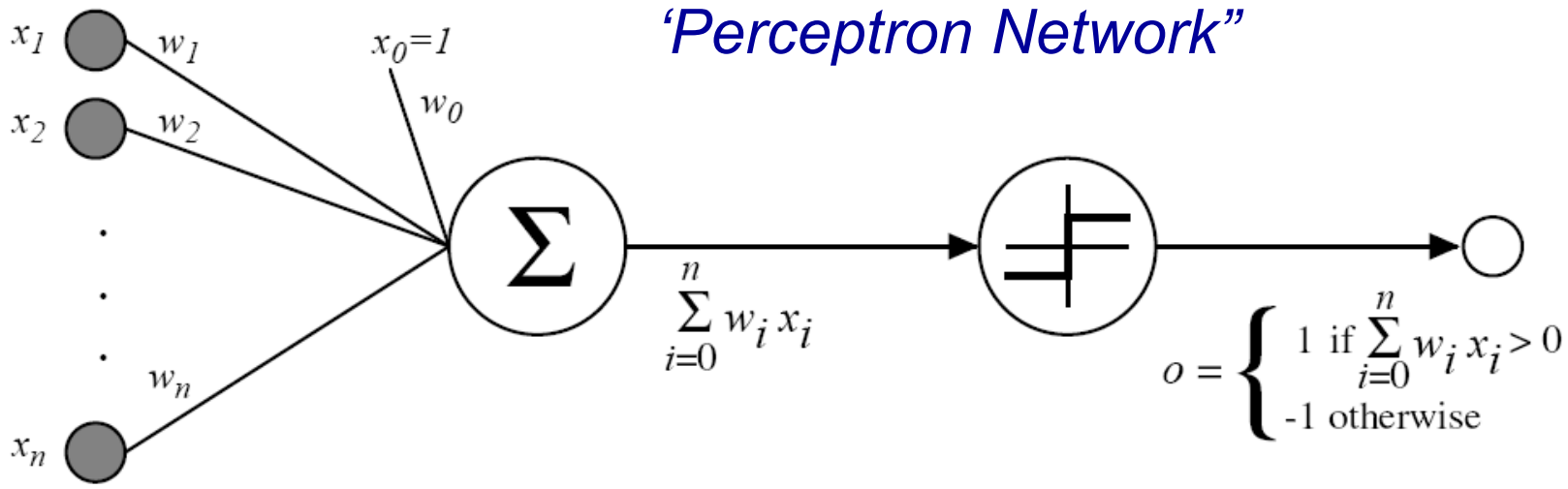
2-Category Linearly Separable Case (5.4)





2-Category Linearly Separable Case (5.4)

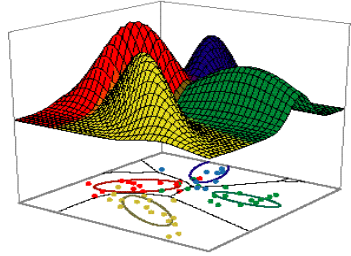




'Perceptron Network"

- Inputs $x_1,\ldots,x_n$
- Weights $w_1,\ldots, w_n$
- Threshold $T = -w_0$ (w_0 called bias)
- Fictitious input x_0 set to 1

2-Category Linearly Separable Case (5.4)



- An extremely simplified computational model of a biological neuron
  - Aka: Mcculloch-Pitts Neuron, Threshold Neuron, Threshold Logic Unit (TLU).
- Sign (read “signum”) or threshold function

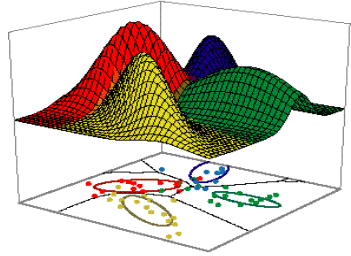
$$sgn(x) = \begin{cases} 1 & x > 0 \\ -1 & \text{otherwise} \end{cases}$$

- Output:

$$o = \text{sgn}\left(\sum_{i=0}^{n} w_i x_i\right) = \text{sgn}\left(w^t x + w_0\right) = \text{sgn}\left(a^t y\right)$$

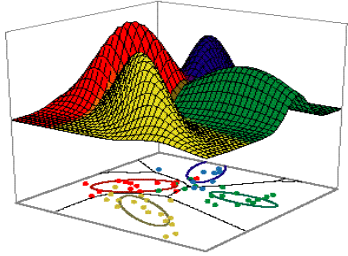
(augmented)

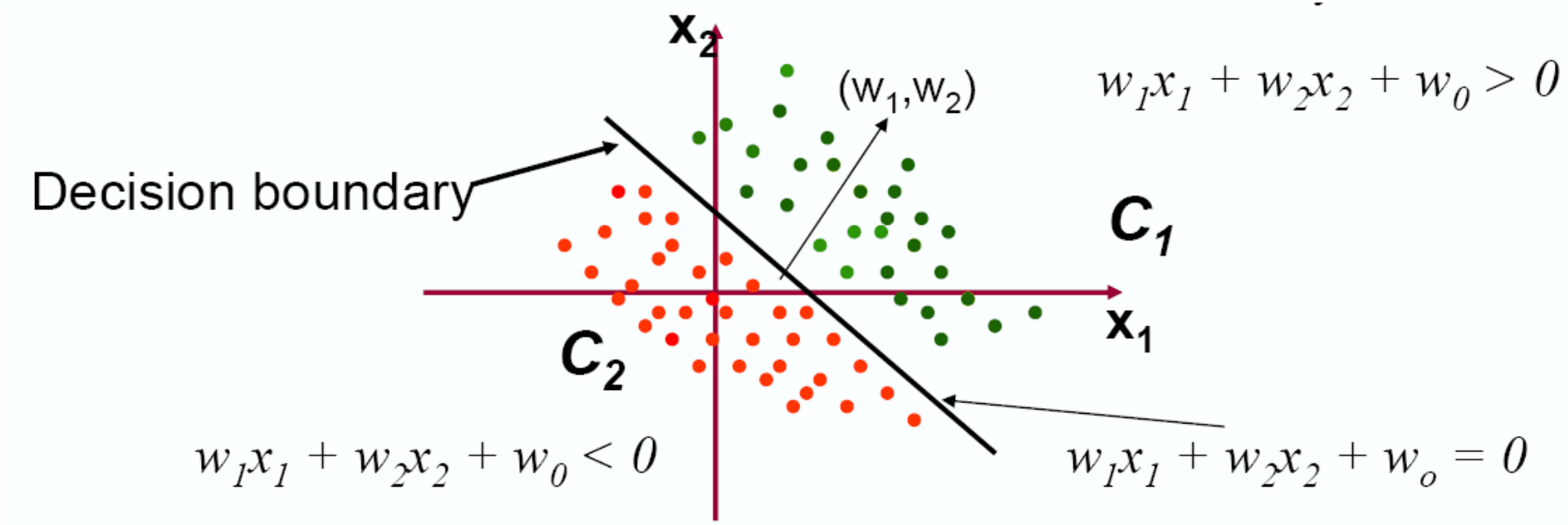
2-Category Linearly Separable Case (5.4)



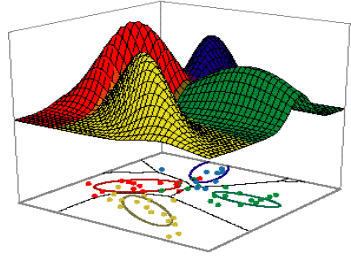
- Assume a pattern with features $\boldsymbol{x} = (\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n)$ can be in one of two categories/classes ω_1, ω_2
  - If the output of the perceptron for input pattern $\boldsymbol{x}_i$ is +1, then $\boldsymbol{x}_i$ is classified to category ω_1.
  - If the output of the perceptron for input pattern $\boldsymbol{x}_i$ is -1, then $\boldsymbol{x}_i$ is classified to category ω_2.
- The input patterns can be thought of as points in an Euclidean space – feature space
- The equation $g(x) = \sum_{i=1}^{n} w_i x_i + w_0 = \sum_{i=0}^{n} w_i x_i = a^t y = 0$ defines a decision surface that separates points assigned to the category ω_1 from points assigned to the category ω_2.

2-Category Linearly Separable Case (5.4)



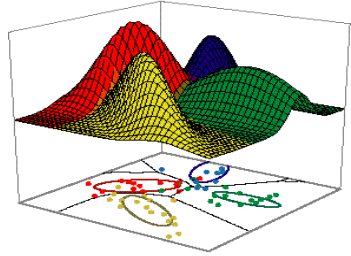


2-Category Linearly Separable Case (5.4)



- Review:
  - The perceptron outputs a 1 for instances lying on one side of the decision surface and outputs a -1 for instances lying on the other side.
  - the linear function $g(\boldsymbol{x})$ divides the feature space into two half-spaces by a hyperplane decision surface
  - The orientation of the surface is determined by the normal vector w, and the location of the surface is determined by the bias w_0.
  - The value of $g(\boldsymbol{x})$ is proportional to the signed distance from a point $\boldsymbol{x}$ to the hyperplane $g(x) = \mathrm{r}\|w\|$

2-Category Linearly Separable Case (5.4)

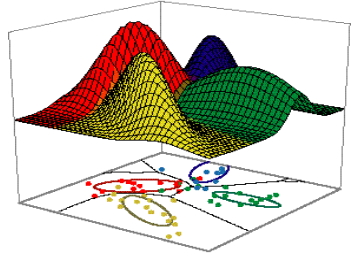


- Suppose the inputs are binary, $x_i \in \{-1,1\}$
- Then perceptron (Threshold Logic Unit) computes a Boolean function
- Example: Suppose $w_1 = w_2 = 1$, $w_0 = -1.5$.

x_1	x_2	g(x)	o
-1	-1	-3.5	-1
-1	1	-1.5	-1
1	-1	-1.5	-1
1	1	0.5	1

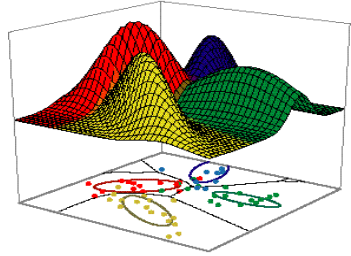
- This perceptron implements the logical AND function.

2-Category Linearly Separable Case (5.4)



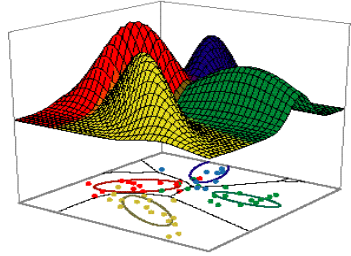
- It is possible to select weight values so that a perceptron can implement AND, OR, and NOT function.
- Any Boolean function can be expressed in terms of AND, OR, and NOT functions, e.g., in disjunctive normal form (DNF) or conjunctive normal form (CNF).
- **Theorem** A sufficiently large network of perceptrons can compute any arbitrary Boolean function.
- There exist networks of perceptrons that can, given unbounded memory, compute any computable function
- Some Boolean functions cannot be implemented by a *single* perceptron, e.g., XOR, why?
- Definition: The class of functions that can be computed by a single threshold neuron are called *threshold functions*.
  - Of the 16 2-input Boolean functions, 14 are threshold functions
  - Decreases to a negligible fraction as # inputs increases

2-Category Linearly Separable Case (5.4)



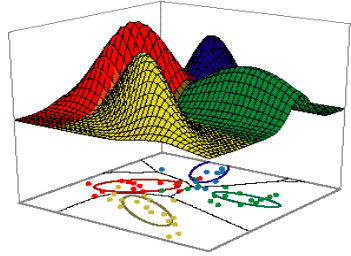
- A perceptron can compute an interesting subset of the set of functions $\{\phi : \Re^n \rightarrow \{-1, 1\}\}$.
- A natural question to ask is whether to *learn* desired (but *a-priori* unknown) threshold functions from *examples* of the desired input–output mapping.
- A *training example*: $(\boldsymbol{x}_k; t_k)$ where $t_k \in \{-1,1\}$ is the desired (target) output, that is, each input vector has been labelled as belonging to one of two classes
- A *training set* is a set of training examples $\{(\boldsymbol{x}_k; t_k)\}$.

2-Category Linearly Separable Case (5.4)



- Learning task: given a training set, find weights $\boldsymbol{w}$ such that the perceptron can correctly classify all training examples, i.e., $\boldsymbol{w}\bullet\boldsymbol{x}_k+w_0 = \mathbf{a}^t\mathbf{y} > 0$ if $t_k = 1$ and $\boldsymbol{w}\bullet\boldsymbol{x}_k+w_0 = \mathbf{a}^t\mathbf{y} < 0$ if $t_k = -1$.
  - This is possible only if the training set is linearly separable.
- *Linearly separable* data sets: if all the points can be classified correctly by a linear hyperplanar decision boundary.
- For linearly separable data sets, there exist weight and threshold values such that a perceptron can achieve perfect classification.
  - For augmented vectors, we say that there exist a weight vector **a** such that the perceptron can achieve perfect classification →Called a *solution vector*

2-Category Linearly Separable Case (5.4)



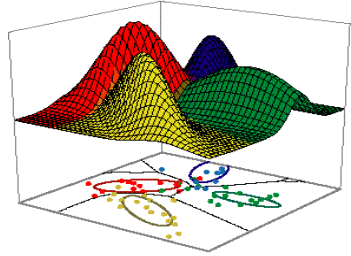
- A sample $\boldsymbol{y}_i$ is classified correctly if:

  $\boldsymbol{a}^t\boldsymbol{y}_i > 0$ and $\boldsymbol{y}_i$ is labelled ω_1, or

  $\boldsymbol{a}^t\boldsymbol{y}_i < 0$ and $\boldsymbol{y}_i$ is labelled ω_2

- *Normalization*: replace all samples labelled ω_2 by their negatives.
  - This simplifies the treatment, and we can forget the labels
- GOAL: Look for a weight vector $\boldsymbol{a}$ such that $\boldsymbol{a}^t\boldsymbol{y}_i > 0$ for all of the (normalized) samples.

2-Category Linearly Separable Case (5.4)



- The solution vector – if it exists – is not unique.

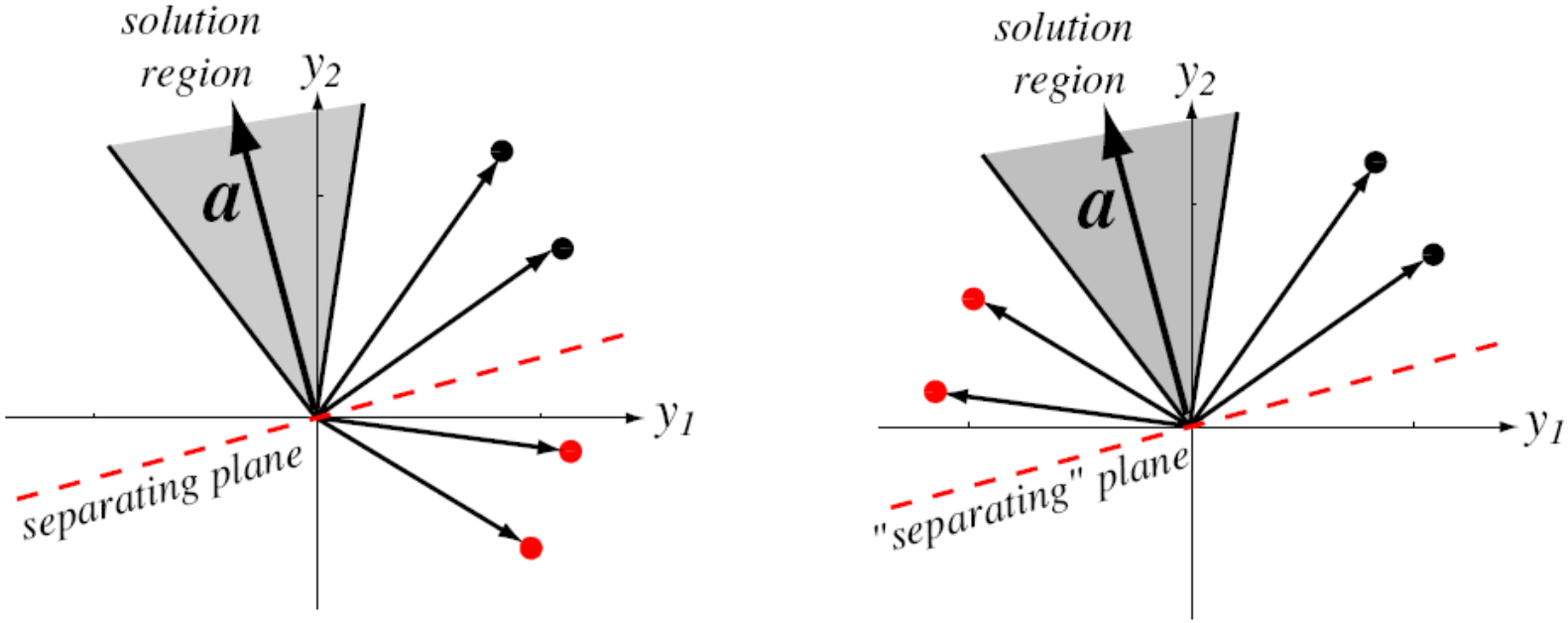
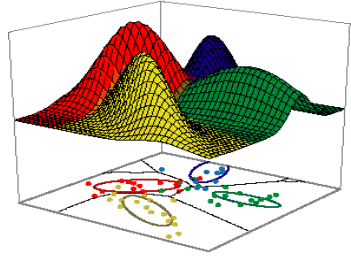


FIGURE 5.8. Four training samples (black for ω_1, red for ω_2) and the solution region in feature space. The figure on the left shows the raw data; the solution vectors leads to a plane that separates the patterns from the two categories. In the figure on the right, the red points have been "normalized"—that 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.

2-Category Linearly Separable Case (5.4)



- A weight vector $\boldsymbol{a}$ is a point in the *weight space*
- Each sample $\boldsymbol{y}_i$ places a constraint on the possible location of a solution vector
  - The equation $\boldsymbol{a}^t\boldsymbol{y}_i = 0$ defines a hyperplane through the origin of weight space having $\boldsymbol{y}_i$ as a normal vector.
  - The solution vector must lie on the correct side of every hyperplane.
  - Can impose a **margin**, b, and solve $\boldsymbol{a}^t\boldsymbol{y}_i \geq b > 0$

2-Category Linearly Separable Case (5.4)

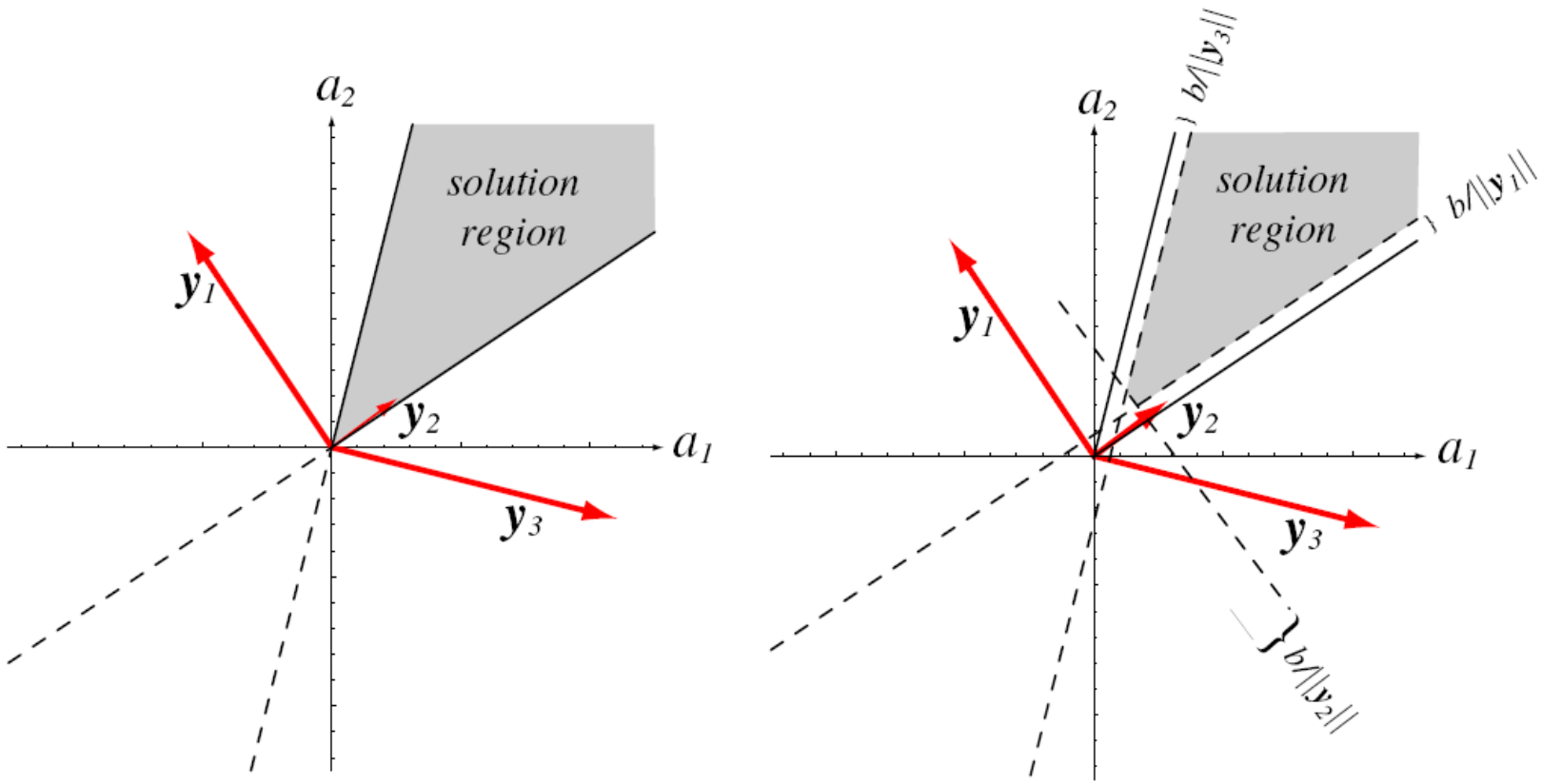
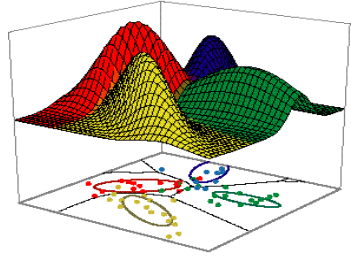


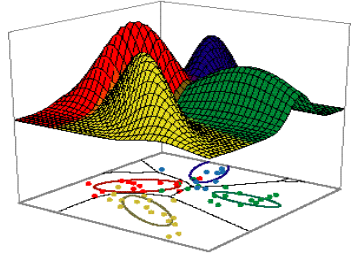
FIGURE 5.9. The effect of the margin on the solution region. At the left is the case of no margin ($b = 0$) equivalent to a case such as shown at the left in Fig. 5.8. At the right is the case $b > 0$, shrinking the solution region by margins $b/\|\mathbf{y}_i\|$. From: Richard O. Duda, Peter E. Hart,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

2-Category Linearly Separable Case (5.4)



- Gradient Descent
  - Define criterion function J(**a**) that is minimized if **a** is a solution vector.
  - Will use *gradient descent* on J(**a**) to find a solution to the set of linear inequalities $\boldsymbol{a}^t\boldsymbol{y}_i > 0$
    - Start with arbitrarily chosen initial weight vector **a**(1).
    - Compute gradient vector ∇J(**a**(1))
    - Next value **a**(2) is obtained by moving some distance from **a**(1) in direction of steepest descent
      - i.e. along negative gradient.
      - Ends when local minimum is reached.

2-Category Linearly Separable Case (5.4)

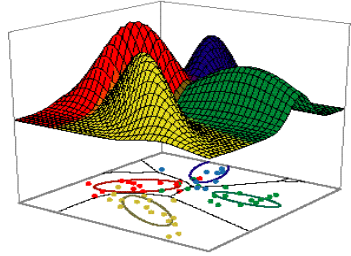


- Gradient Descent
  - Gradient of J(**w**) w.r.t. **w**:

$$\nabla J[\vec{w}] \equiv \left[\frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}, \cdots \frac{\partial J}{\partial w_d}\right]^t$$

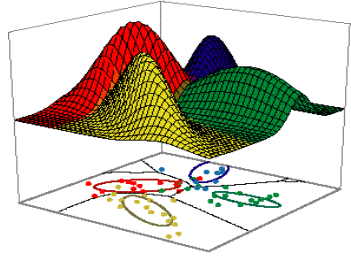
  - When interpreted as a vector in weight space, the gradient specifies the direction that produces the steepest increase in *J*.

2-Category Linearly Separable Case (5.4)



- Gradient Descent
  - In general: $a(k+1) = a(k) - \eta(k) \nabla J(a(k))$
    - Where η is a positive *learning rate* that sets the step size.
    - Typically continue iterations until $|\eta(k) \nabla J(a(k))| < \theta$
      - θ is a stopping threshold.
- Questions:
  - What criterion function $J(a)$ to use?
  - What learning rate $\eta(k)$ to use?
    - Can be constant/fixed or decreasing with k.
    - The choice of the learning rate $\eta(k) > 0$:
      - If $\eta(k)$ is too small, convergence is needlessly slow
      - If $\eta(k)$ is too large, the process will overshoot and may oscillate or diverge.
      - One common practice is to gradually reduce the value of $\eta(k)$.

Perceptron Criterion (5.5)



- What criterion function J(a) to use?
  - Most obvious choice would be number of misclassified points
    - (minimizing this would lead to solution vector)
    - Piecewise constant so poor choice for gradient descent
  - Better choice is *perceptron criterion*:

$$J_p(a) = \sum_{y \in \psi} \left(-a^t y\right)$$

    - Where ψ is the set of all misclassified **y**

Perceptron Criterion (5.5)

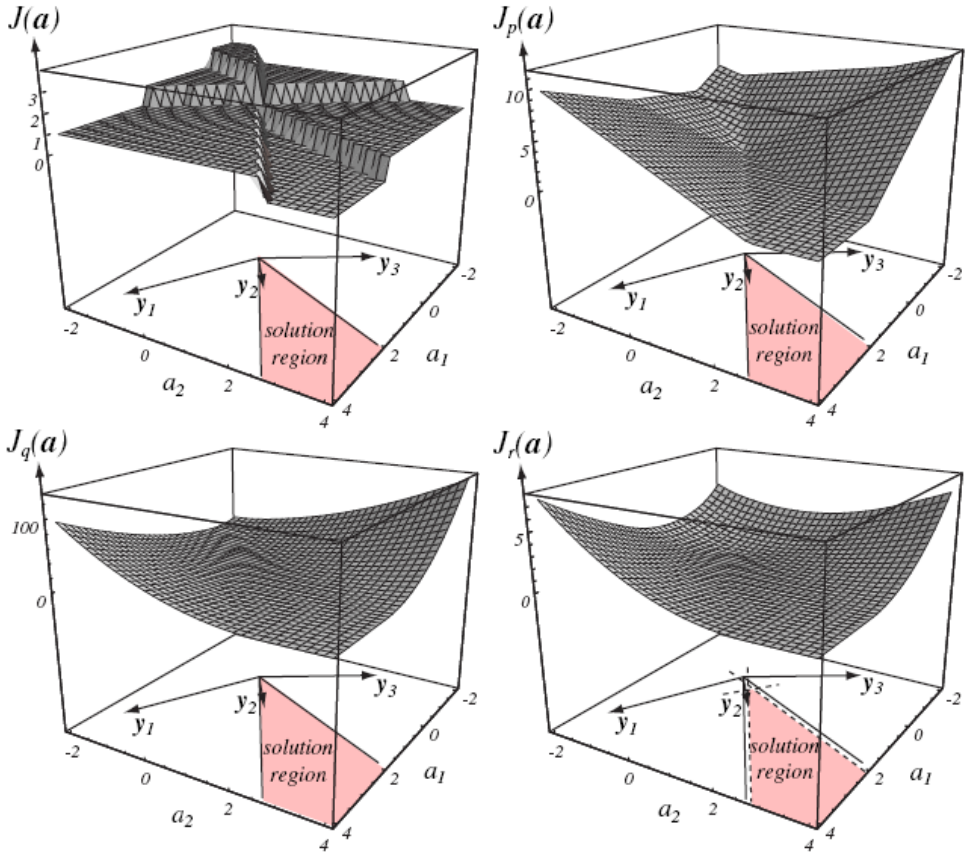
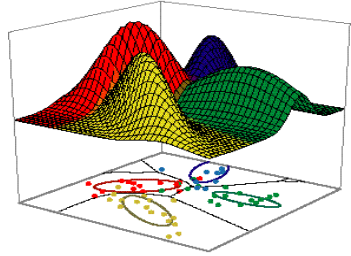


FIGURE 5.11. Four learning criteria as a function of weights in a linear classifier. At the upper left is the total number of patterns misclassified, which is piecewise constant and hence unacceptable for gradient descent procedures. At the upper right is the Perceptron criterion (Eq. 16), which is piecewise linear and acceptable for gradient descent. The lower left is squared error (Eq. 32), which has nice analytic properties and is useful even when the patterns are not linearly separable. The lower right is the square error with margin (Eq. 33). A designer may adjust the margin b in order to force the solution vector to lie toward the middle of the $b = 0$ solution region in hopes of improving generalization of the resulting classifier. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Perceptron Criterion (5.5)



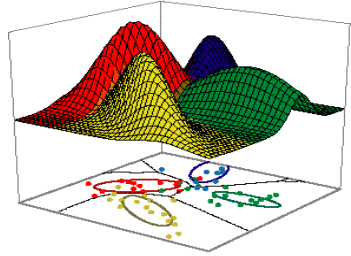
- Perceptron criterion:

$$J_p(a) = \sum_{y \in \psi} \left(-a^t y\right)$$

  - Where ψ is the set of all misclassified **y**
  - Since $\mathbf{a}^t\mathbf{y} \leq 0$ for misclassified **y**, $J_p(\mathbf{a})$ is ≥ 0
  - $J_p(\mathbf{a}) = 0$ if **a** is a solution vector (or on decision boundary)
  - Geometrically, $J_p(\mathbf{a})$ is proportional to sum of dist from misclassified samples to the decision boundary.
- Gradient:
  - Since j^{th} component of gradient of J_p is $\partial J_p/\partial a_j = y_j$, then:

$$\nabla J_p = \sum_{y \in \psi} \left(-y\right)$$

Perceptron Criterion (5.5)



- Batch Perceptron algorithm:

  1) initialize **a**, $\eta()$, criterion θ, k=0

  2) do: k=k+1

  3) **a**=**a**+$\eta(k)\Sigma_{y\in\psi}$**y**

  4) until: $|\eta(k)\Sigma_{y\in\psi}\mathbf{y}| < \theta$

  - The next weight vector is obtained by adding some multiple of the sum of the misclassified samples to the present weight vector.

Perceptron Criterion (5.5)

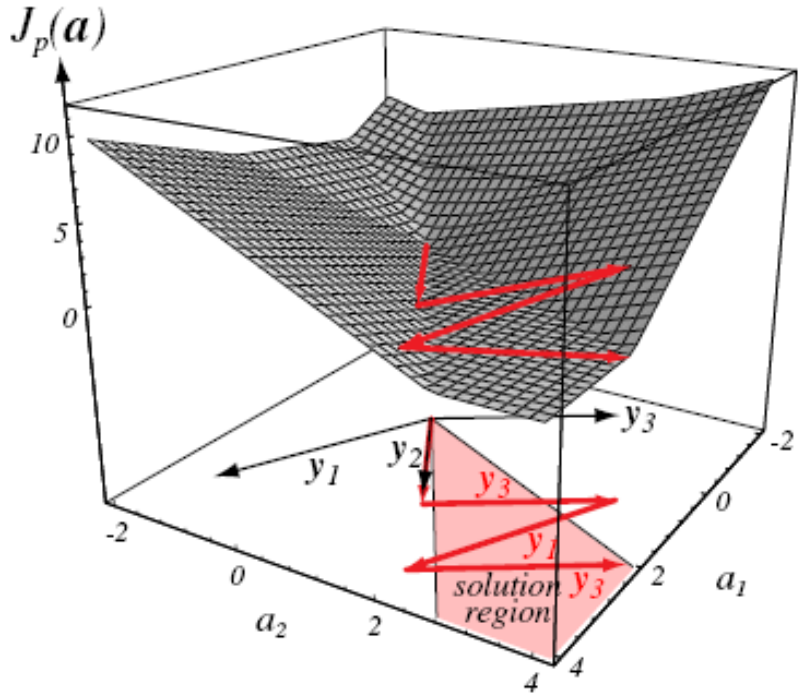
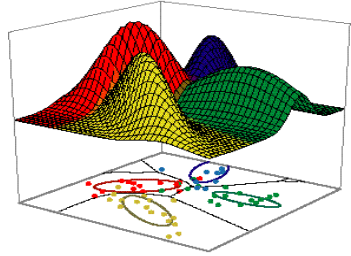


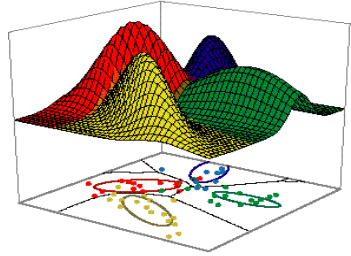
FIGURE 5.12. The Perceptron criterion, $J_p(\mathbf{a})$, is plotted as a function of the weights a_1 and a_2 for a three-pattern problem. The weight vector begins at $\mathbf{0}$, and the algorithm sequentially adds to it vectors equal to the "normalized" misclassified patterns themselves. In the example shown, this sequence is $\mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_1, \mathbf{y}_3$, at which time the vector lies in the solution region and iteration terminates. Note that the second update (by $\mathbf{y}_3$) takes the candidate vector *farther* from the solution region than after the first update (cf. Theorem 5.1). From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Perceptron Criterion (5.5)



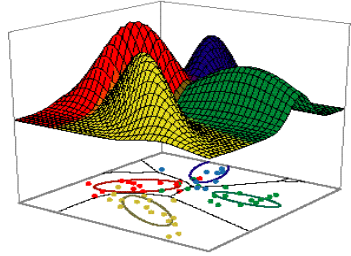
- Perceptron Convergence Theorem
  - The sequence of weight vectors given by the Perceptron algorithm will terminate at a solution vector after a finite number of updates if a solution vector exists, that is, if training samples are linearly separable.
    - <proof omitted>
  - Perceptron Convergence Theorem holds for any finite learning rate $\eta > 0$
  - The bound is not useful in detemining when to stop the algorithm in practice because it depends on an unknown solution vector
  - The convergence theorem offers no guarantees when the training set is not linearly separable

Perceptron Criterion (5.5)



- Variants on perceptron algorithm
  - Fixed-increment single-sample perceptron update rule
    - $\mathbf{a}(k+1) = \mathbf{a}(k) + \eta\mathbf{y}$, if $\mathbf{a}^t(k)\mathbf{y} \leq 0$
  - Variable-Increment perceptron update rule
    - $\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)\mathbf{y}$, if $\mathbf{a}^t(k)\mathbf{y} \leq 0$
  - Variable-Increment Perceptron with margin
    - $\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)\mathbf{y}$, if $\mathbf{a}^t(k)\mathbf{y} \leq b$
  - These converge when $\eta(k)$ satisfies certain conditions: e.g., it is positive constant; it is bounded positive; it decreases as $1/k$.

Relaxation Procedures (5.6)



- Consider minimizing other criterion functions…
  - Aim for continuous function, avoid boundaries, etc.

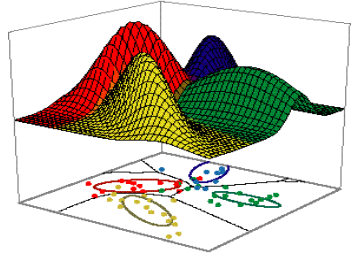
$$J_r(a)=\frac{1}{2}\sum_{y\in\Psi}\frac{\left(a^t y-b\right)^2}{\|y\|^2} \qquad \nabla J_r(a)=\sum_{y\in\Psi}\frac{a^t y-b}{\|y\|^2}y$$

- Leads to single-sample relaxation rule with margin:

$$a(k+1)=a(k)+\eta\frac{b-a^t(k)y}{\|y\|^2}y, \qquad \text{if } a^t(k)y\le b$$

- Geometrical interpretation in weight space:
  - If a weight vector **a** incorrectly classifies a sample **y**, we need to change **a** so that it moves across the hyperplane $\mathbf{a}^t(k)\mathbf{y}-b=0$
  - The quickest way to move a point across a hyperplane is along the normal to that hyperplane.
    - → move in direction of **y**

Relaxation Procedures (5.6)

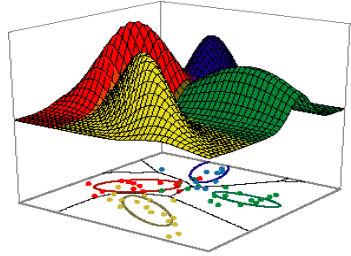


- Geometrical interpretation in weight space:
  - $\mathbf{y}/\|\mathbf{y}\|$ is the unit normal vector for the hyperplane $\boldsymbol{a}^t(k)\boldsymbol{y}-b = 0$
    - Ensures that 'longer' training vectors don't unduly affect result
  - Distance from $\boldsymbol{a}(k)$ to the hyperplane $\boldsymbol{a}^t(k)\boldsymbol{y}-b = 0$ is

$$r(k) = \frac{b - a^t(k)y}{\|y\|^2}$$

  - $\eta = 1$, $\boldsymbol{a}$ is moved exactly to the hyperplane, absolute correction rule
  - $0 < \eta < 2$ for convergence, *fractional correction rule*

Relaxation Procedures (5.6)



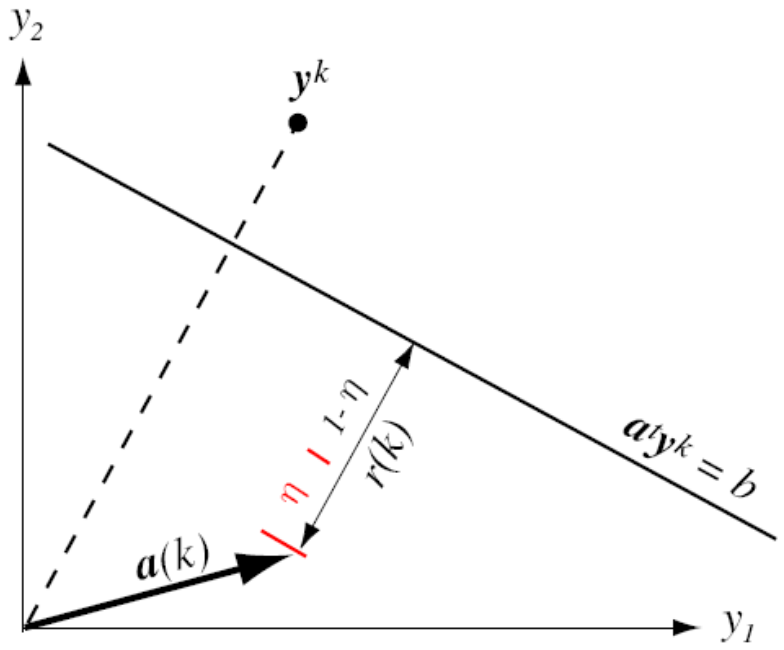
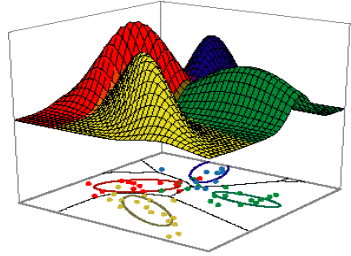


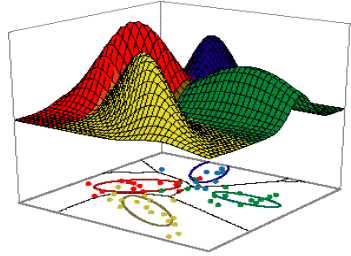
FIGURE 5.14. In each step of a basic relaxation algorithm, the weight vector is moved a proportion η of the way toward the hyperplane defined by $\mathbf{a}^t\mathbf{y}^k = b$. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Nonseparable Behaviour (5.7)



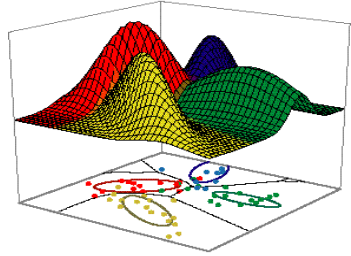
- Error-correcting procedures such as Perceptron modify weight vector only when an error occurs
  - Relentless search for error-free solution only makes sense for datasets expected to be (nearly) linearly separable.
- Any training set of less than 2d points is likely to be linearly separable.
  - Need many times this amount to overdetermine classifier (generalization)
  - However, a large training set will likely be nonseparable.
- How do these algorithms behave for nonseparable datasets?
  - If the training set is not linearly separable, the perceptron algorithms will never terminate.
  - If we arbitrarily stop the process there is no guarantee that the weight vector found will work well → consider using average of last few iterations.
- A number of variants have been proposed with a view to giving good performance on problems not linearly separable while still ensuring convergence when the problem is linearly separable.
  - e.g., the value of η decreases, e.g. $\eta(k) = \eta(1)/k$ or decrese η as performance increases.

Minimum Squared Error Procedures (5.7)



- AKA Least mean squares (LMS)
- Has good compromise performance on both separable and nonseparable problems but sacrifices ability to obtain a separating vector for separable cases
- Criterion function involves all samples
  - Not just misclassified samples
- Instead of $\mathbf{a}^t\mathbf{y}_i>0$, try to make $\mathbf{a}^t\mathbf{y}_i=b_i$ for some b_i
- Have changed from finding a solution to set of linear inequalities, to finding solution to a set of linear equations
  - Can apply standard MSE procedures.

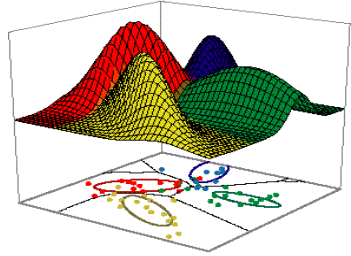
Minimum Squared Error Procedures (5.8)



- Introduce n×d matrix **Y** where rows are vectors $\mathbf{y}_i$, **b** is vector of b_i values.
- Find **a** such that **Ya=b**
- Y often singular (more rows than columns) so cannot simply find $\mathbf{a}=\mathbf{Y}^{-1}\mathbf{b}$
- Instead define error **e=Ya-b**
  - Minimize length of **e** → $J_s(a) = \|Ya - b\|^2 = \sum_{i=1}^{n} \left(a^t y_i - b_i\right)^2$ *MSE Criterion*
- End up with $a = Y^{\tau} b$
  - Y^{τ} is pseudoinverse. Recall: $M^{\tau} = [M^T M]^{-1} M^T$

$$M^{\tau} M = I$$

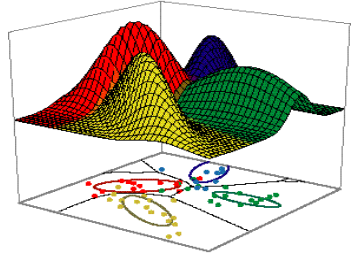
Minimum Squared Error Procedures (5.7)



- Setting $\mathbf{b}=\mathbf{1}_n$ leads to Bayes discriminant function for infinite training data
- Setting $b = \begin{bmatrix} \frac{n}{n_1} 1_1 \\ \frac{n}{n_2} 1_2 \end{bmatrix}$ n_1 *samples from* ω_1 / n_2 *samples from* ω_2
  - Leads to Fisher's Linear Discriminant (5.8.2)
- Note that gradient descent can be used instead of pseudoinverse.
  - Widrow-Hoff (5.8.4) or Ho-Kashyap* (5.9) procedures
    - Use in either batch mode (update weights after full pass through training data) or single-sample (online) learning.

*Can find separating vector if it exists, and behaves well on nonseparable.

Minimum Squared Error Procedures (5.7)



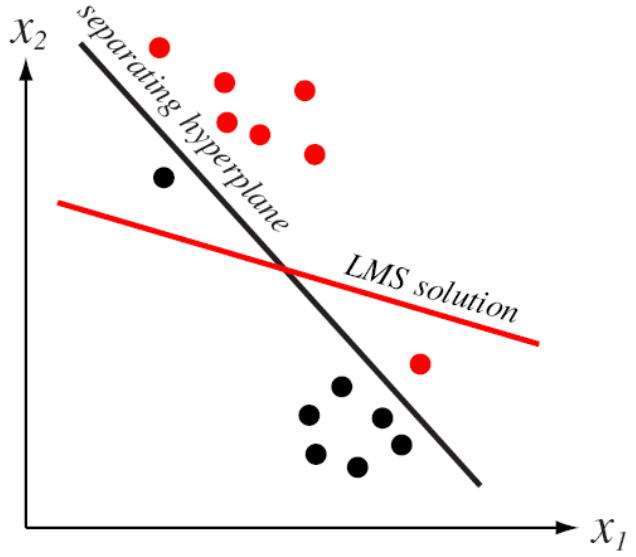
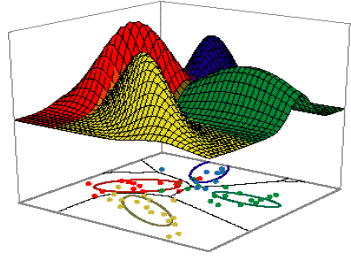


FIGURE 5.17. The LMS algorithm need not converge to a separating hyperplane, even if one exists. Because the LMS solution minimizes the sum of the squares of the distances of the training points to the hyperplane, for this example the plane is rotated clockwise compared to a separating hyperplane. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

LMS = Widrow-Hoff = gradient descent applied to MSE criterion, rather than pseudoinverse



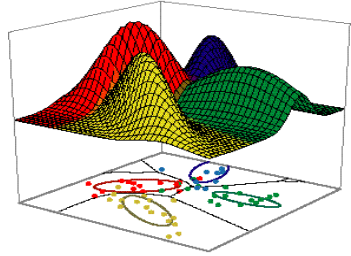
Generalized Linear Discriminant Functions (5.3)

- Decision boundaries which separate between classes may not always be linear
- The complexity of the boundaries may sometimes require the use of highly non-linear surfaces
- A popular approach to generalize the concept of linear decision functions is to consider a generalized decision function as:

$$g(x) = w_1 f_1(x) + w_2 f_2(x) + \ldots + w_N f_N(x) + w_0 \qquad (1)$$

where $f_i(x)$, $1 \leq i \leq N$ are scalar functions of the pattern x, $x \in R^d$ *(Euclidean Space)*

Generalized Linear Discriminant Functions (5.3)



- By adding the dummy function $f_0(x) = 1$ we get:

$$g(x) = \sum_{i=0}^{N} w_i f_i(x) = w^T y$$

$$where\ \mathrm{w} = (\mathrm{w}_0, \mathrm{w}_1, w_2, ..., w_N)^T \text{ and } \mathrm{y} = (\mathrm{f}_0(x), \mathrm{f}_1(x), f_2(x), ..., f_N(x))^T$$

- This latter representation of *g(x)* implies that any decision function defined by equation (1) can be treated as linear in the *(N+1)* dimensional y-space *(N+1 > d)*

- *g(x)* is linear in its weights, but maintains its nonlinear characteristics in R^d *x-space*

Generalized Linear Discriminant Functions (5.3)

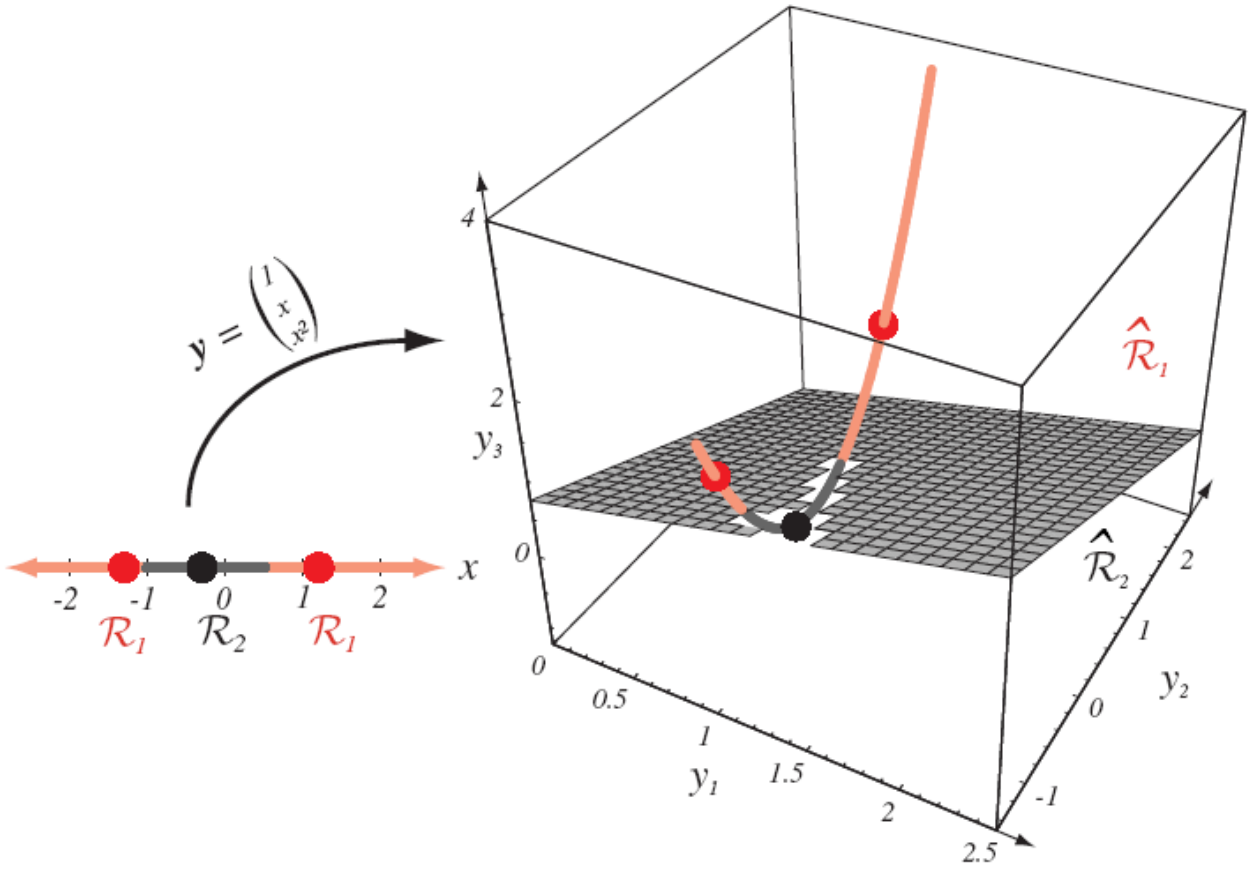


FIGURE 5.5. The mapping $\mathbf{y} = (1, x, x^2)^t$ takes a line and transforms it to a parabola in three dimensions. A plane splits the resulting $\mathbf{y}$-space into regions corresponding to two categories, and this in turn gives a nonsimply connected decision region in the one-dimensional x-space.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

Generalized Linear Discriminant Functions (5.3)

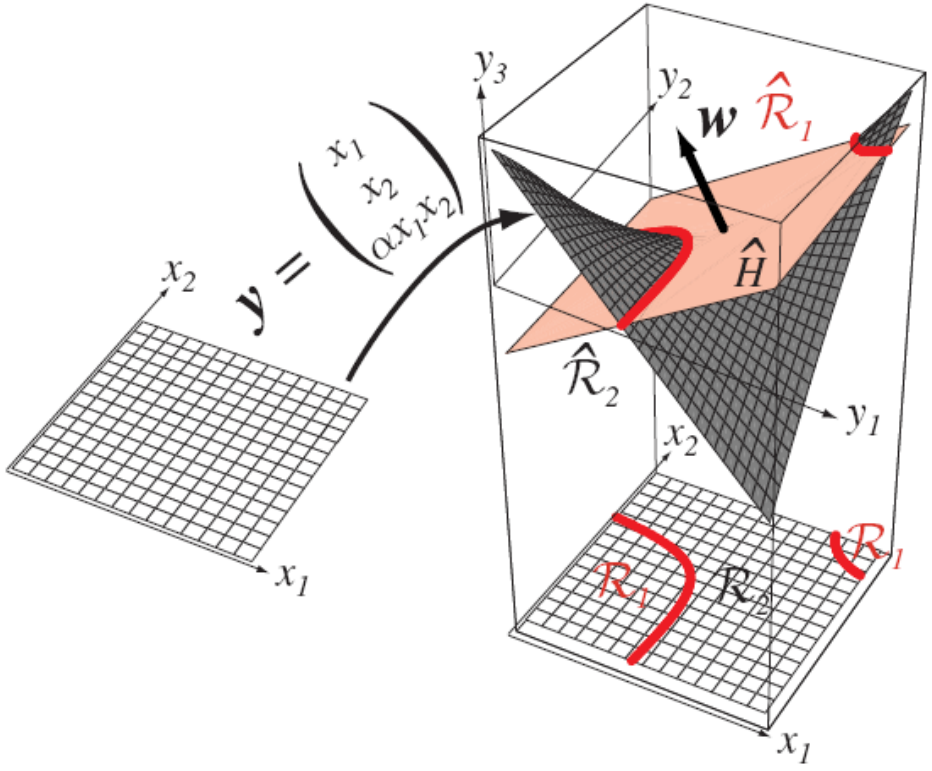
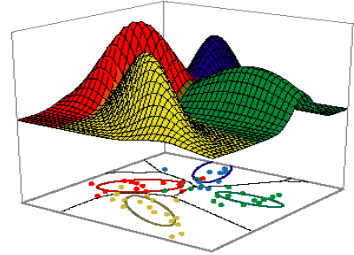
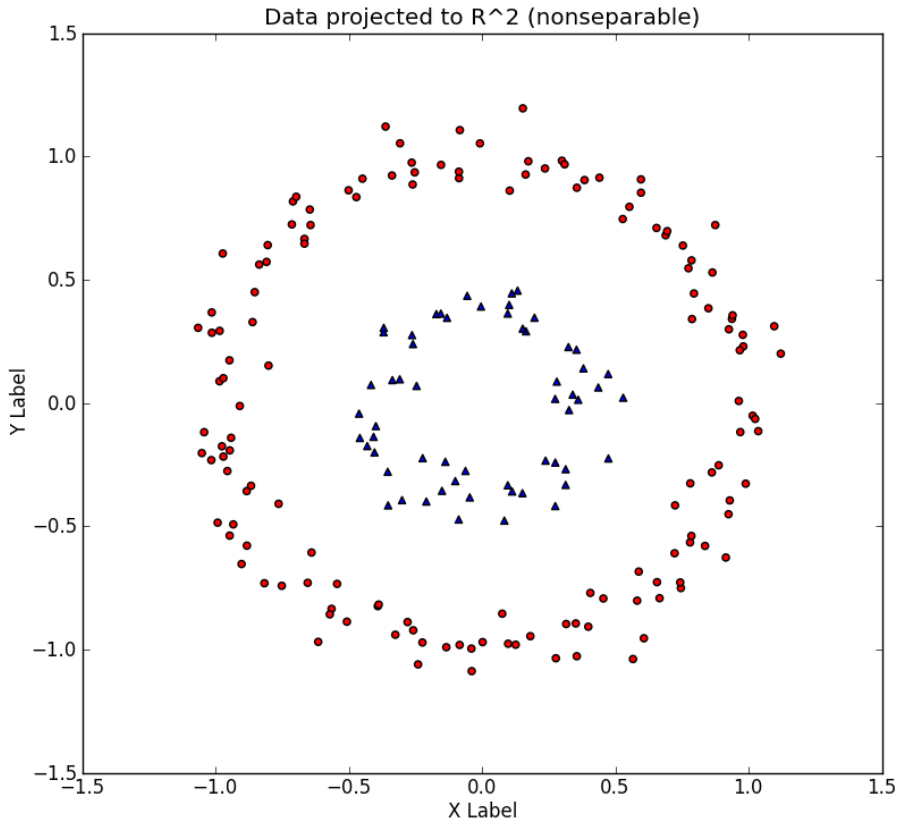
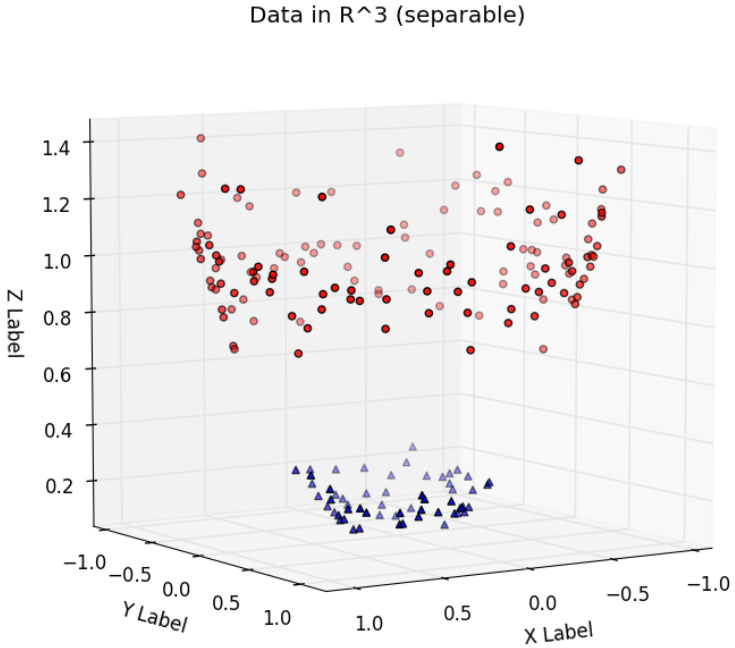


FIGURE 5.6. The two-dimensional input space $\mathbf{x}$ is mapped through a polynomial function f to $\mathbf{y}$. Here the mapping is $y_1 = x_1$, $y_2 = x_2$ and $y_3 \propto x_1 x_2$. A linear discriminant in this transformed space is a hyperplane, which cuts the surface. Points to the positive side of the hyperplane $\hat{H}$ correspond to category ω_1, and those beneath it correspond to category ω_2. Here, in terms of the $\mathbf{x}$ 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.

Another example



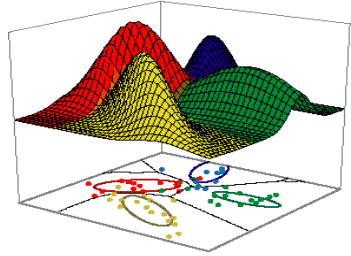


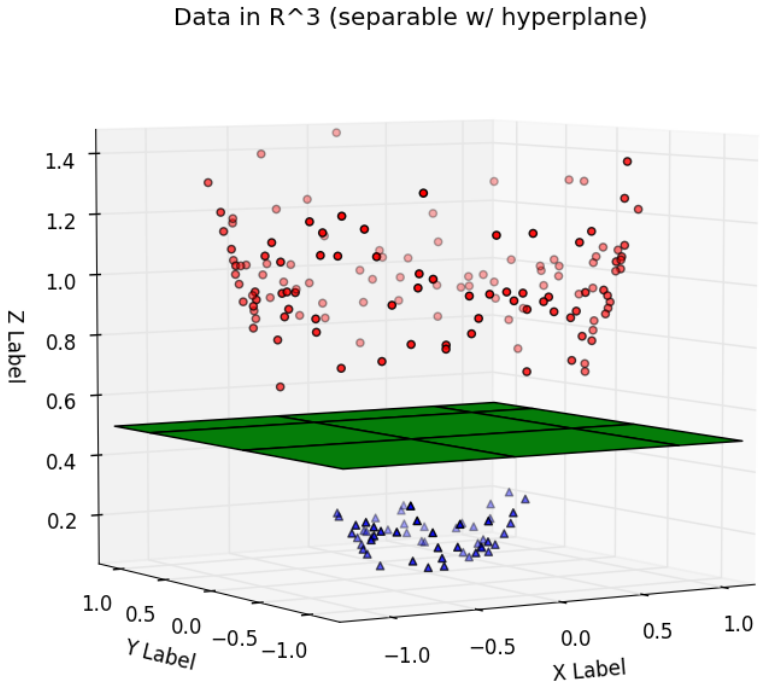


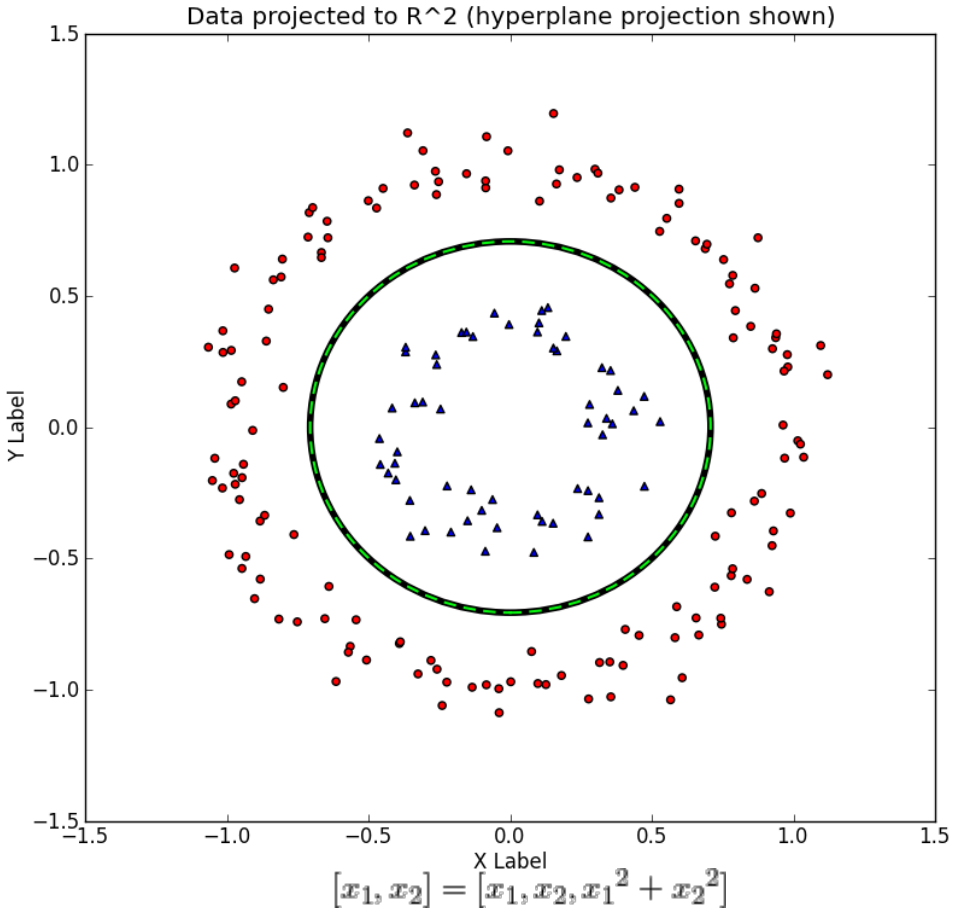
$$[x_1, x_2] = [x_1, x_2, {x_1}^2 + {x_2}^2]$$

From http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html

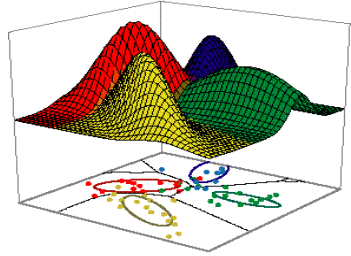
Another example continued







From http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html



Generalized Linear Discriminant Functions (5.3)

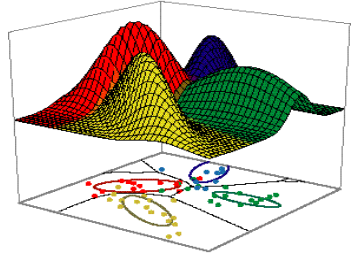
- The most commonly used generalized decision function is *g(x)* for which $f_i(x)$ $(1 \le i \le N)$ are polynomials
  - Typically, quadratic.

- Quadratic decision functions for a 2-dimensional feature space

$$g(x) = w_1 x_1^2 + w_2 x_1 x_2 + w_3 x_2^2 + w_4 x_1 + w_5 x_2 + w_6$$

$$here: \mathrm{w} = (\mathrm{w}_1, w_2, ..., w_6)^T \text{ and } \mathrm{y} = (\mathrm{x}_1^2, x_1 x_2, x_2^2, x_1, x_2, 1)^T$$

Generalized Linear Discriminant Functions



- For patterns $x \in R^d$, the most general quadratic decision function is given by:

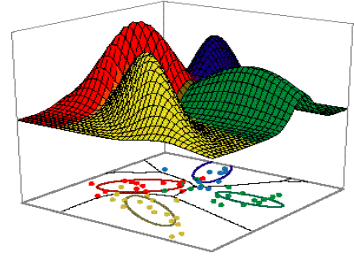
$$g(x) = \sum_{i=1}^{d} w_{ii} x_i^2 + \sum_{i=1}^{d-1} \sum_{j=i+1}^{d} w_{ij} x_i x_j + \sum_{i=1}^{d} w_i x_i + w_{d+1} \qquad (2)$$

The number of terms at the right-hand side is:

$$l = N + 1 = d + \frac{d(d-1)}{2} + d + 1 = \frac{(d+1)(d+2)}{2}$$

This is the total number of weights which are the free parameters of the problem

- If for example d = 3, the vectors w and y are 10-dimensional
- If for example d = 10, the vectors w and y are 65-dimensional
  - Curse of dimensionality… Watch for over-fitting!



Generalized Linear Discriminant Functions

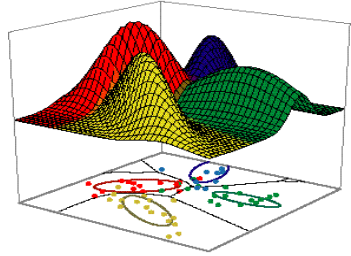
- The commonly used quadratic decision function can be represented as the general n-dimensional quadratic surface:

$$g(x) = x^T Ax + x^T b + c$$

where the matrix $A = (a_{ij})$, the vector $b = (b_1, b_2, \ldots, b_n)^T$ and c, depends on the weights w_{ii}, w_{ij}, w_i of equation (2)

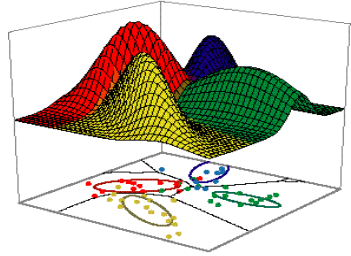
- In conclusion: it is only the matrix A which determines the shape and characteristics of the decision function
  - If A is positive definite then the decision function is a hyperellipsoid with axes in the directions of the eigenvectors of A
  - In particular: if $A = I_n$ (Identity), the decision function is simply the n-dimensional hypersphere
  - If A is negative definite, the decision function describes a hyperhyperboloid

Support Vector Machines (5.11)



- Uses concepts from training linear machines with margins and preprocessing data to represent patterns in a high dimension.
- With suitable nonlinear mapping function to a sufficiently high dimension, any dataset becomes linearly separable
- Simultaneously minimize the empirical classification error and maximize the geometric margin
  - Known as **maximum margin classifiers**.

Support Vector Machines (5.11)

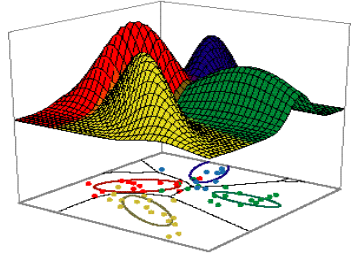


- Let $g(y)=a^t y$
  - where $y_i=\varphi(x_i)$
    - choice of nonlinear mapping function φ discussed later
- Let $z_k=\pm 1$ denote class of sample k
- A separating plane ensures

$$z_k g(\mathbf{y}_k) \geq 0,\ k=1,\ldots,n$$

- Goal of SVM is to find the separating hyperplane with the largest margin
- Distance from transformed sample **y** to decision hyperplane is $r = \frac{|g(y)|}{\|a\|}$

Support Vector Machines (5.11)

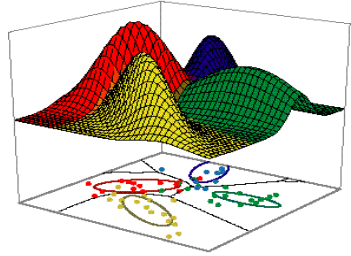


- Assuming a positive margin b exists, then:

$$\frac{z_k g(y)}{\|a\|} \geq b, \quad k = 1,\ldots,n \qquad (\text{eq } 106)$$

- Goal is to find weight vector **a** to maximize b
  - Constrain b||a|| = 1 to avoid arbitrary scaling of a.
  - Therefore, maximizing b also minimizes $||a||^2$ *Objective1*
- Support vectors are those transformed training points for which eq 106 is an equality
  - i.e. those points which are closest to the boundary
    - Distance to decision boundary is exactly 'b'
  - These are hardest points to classify, and are also the most informative points.

Support Vector Machines (5.11)



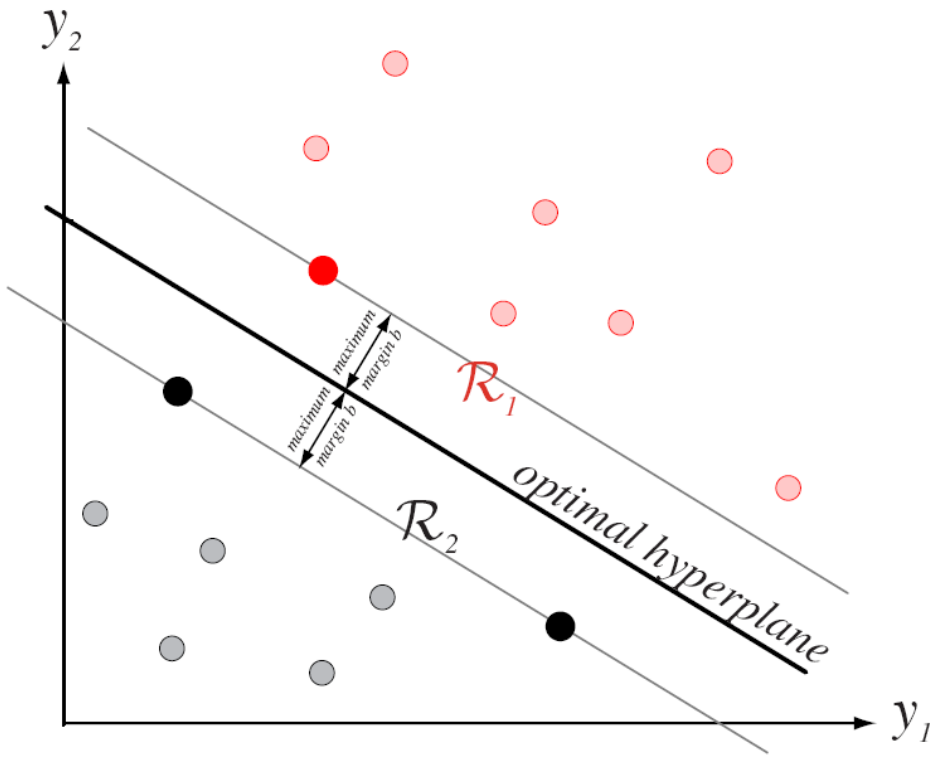
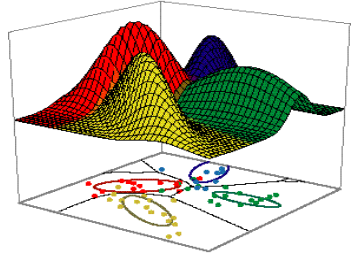


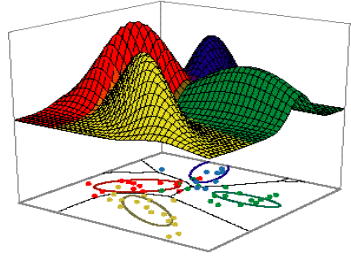
FIGURE 5.19. Training a support vector machine consists of finding the optimal hyperplane, that is, the one with the maximum distance from the nearest training patterns. The support vectors are those (nearest) patterns, a distance b from the hyperplane. The three support vectors are shown as solid dots.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

Support Vector Machines (5.11)



- Could apply modified Perceptron learning rule
  - Instead of modifying weight vector by *any* misclassified point, always choose worst-classified y_k
    - i.e. point on wrong side of decision boundary and furthest from decision boundary
  - Unfortunately this is too costly to use on large problems since requires searching all training data for worst point.
  - Before moving on to practical training algorithm, first analyze expected error rate of this simple learning rule…

Support Vector Machines (5.11)

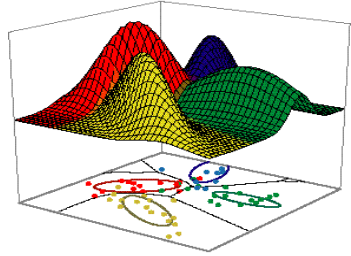


- Could apply modified Perceptron learning rule
  - For such a classifier with N_s support vectors, the expected generalization error is bounded by:

$$\varepsilon_n[error] \leq \frac{\varepsilon_n[N_s]}{n}$$

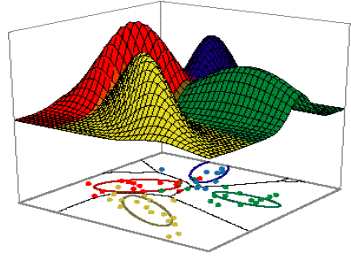
  - Related to leave-one-out bound.
    - If trained on n-1 points and tested on n^{th} point, it will be misclassified iff it is a support vector
- Motivates choosing a transformation, φ, that will result in few support vectors and hence low expected generalization error

Training Support Vector Machines (5.11)



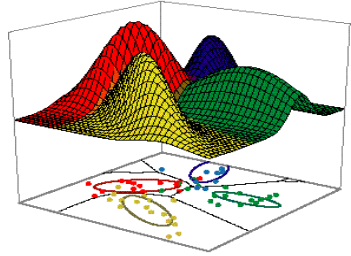
- Optimize weight vector, **a**, by using Lagrange multipliers to simultaneously optimize weight vector to
  - 1) minimize magnitude of ||**a**||
  - 2) under the constraints that data are correctly classified/separated
    - Can include slack variables to permit misclassification.
      - Wider margin → better generalization. Trade bias for variance…
    - C "hardness" parameter weights these errors (more/less penalty)
  - Solve using quadratic programming or other approaches
- Complexity of SVM classifier is determined by N_s rather than dimensionality of mapped space
  - Reduced chance of overfitting…

Training Support Vector Machines (5.11)



- If data are not linearly separable
  - Can select nonlinear transformation, φ, to higher-dimensional space where data become separable
- Choice often motivated by prior knowledge about domain *(or guess)*
  - Dimensionality of mapped space may be can be arbitrarily high (constrained by computational complexity)
- Important realization:
  - For both training and classifying, we only need to be able to compute the dot product between pairs of samples in the new space (not the mapped vectors themselves)
- Kernel function $k(x_i, x_k) = <\varphi(x_i), \varphi(x_k)>$ (dot-product in new space; measures similarity)
  - Linear: $k(x_i, x_k) = (x_i \cdot x_k)$ *(don't map to higher order space)*
  - Polynomial: $k(x_i, x_k) = (x_i \cdot x_k + 1)^d$
  - Gaussian: $k(x_i, x_k) = e^{\left(-\frac{\|x_i - x_k\|^2}{2\sigma^2}\right)}$

Training Support Vector Machines (5.11)



- The kernel trick:
  - Mapping all data to higher dimensional space then fitting a linear discriminant is too expensive
  - Turns out, we only actually need to compute dot-products between points in new space
  - Define kernel function that measures dot product (*form of similarity measure*) between two points in new space, without explicitly mapping all data

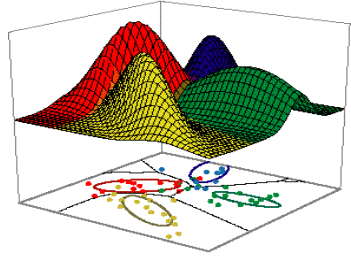
From: http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html

For $x_i, x_i \in R^N, K(x_i, x_i) = \langle \emptyset(x_i), \emptyset(x_j) \rangle_M$

Where $\langle °, ° \rangle_M$ is an inner product of R^M, M > N,

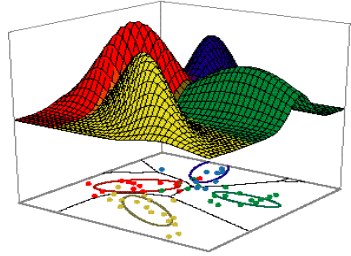
and $\emptyset(x_i)$ transforms x_i to R^M $(\emptyset: R^N \rightarrow R^M)$

SVM Resources



- Great MIT Lecture: https://www.youtube.com/watch?v=_PwhiWxHK8o
  - Derives SVM from first principles and tells the story of how Vapnik invented SVM
  - Lagrange multipliers allow us to find a max/min of an equation, under constraints
  - 37min mark shows that the optimization of the weights (w) depends only on dot products between pairs of inputs ($x_i \cdot x_j$)
  - 38min mark shows that the decision rule for an unknown input, *u*, also depends only on dot products between ($u \cdot x_i$)
    - This is all before any discussion of using kernels or changing feature spaces…
  - 41min mark shows how the optimization (training) fails to converge for linearly inseparable data

SVM Resources



- Tutorial: https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f
  - Final equations illustrate how only dot products between samples are required:

- Solution of the dual problem gives us:

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \hat{\alpha}_i y_i \mathbf{x}_i$$

- The decision boundary:

$$\hat{\mathbf{w}}^T \mathbf{x} + w_0 = \sum_{i \in SV} \hat{\alpha}_i y_i (\mathbf{x}_i^T \mathbf{x}) + w_0$$

- The decision:

$$\hat{y} = \text{sign}\left[\sum_{i \in SV} \hat{\alpha}_i y_i (\mathbf{x}_i^T \mathbf{x}) + w_0 \right]$$

- Mapping to a feature space, we have the decision:

$$\hat{y} = \text{sign}\left[\sum_{i \in SV} \hat{\alpha}_i y_i (\phi(\mathbf{x}) \cdot \phi(\mathbf{x}_i)) + w_0 \right]$$