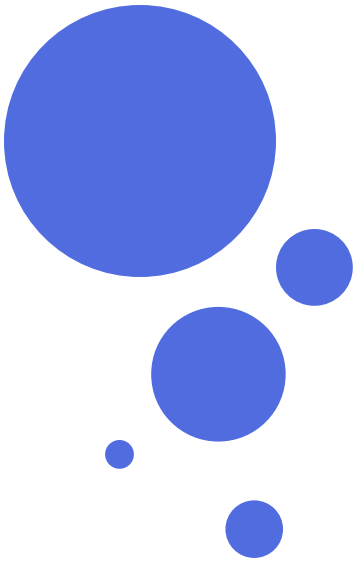
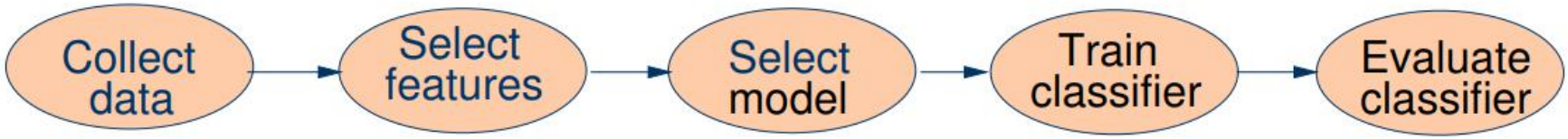
Rensselaer



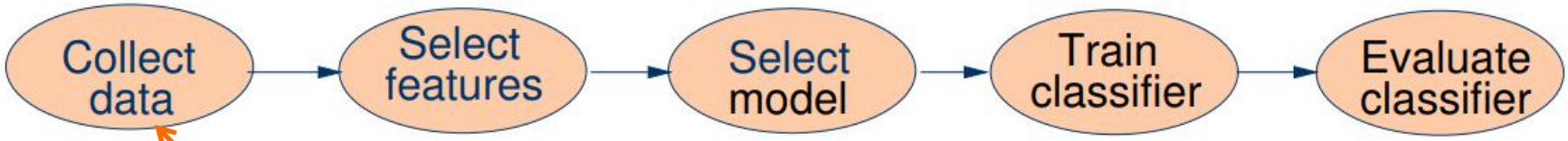
Lecture 12: Midterm Exam Review

Dr. Chengjiang Long
Computer Vision Researcher at Kitware Inc.
Adjunct Professor at RPI.
Email: **longc3@rpi.edu**

Pattern recognition design cycle



Pattern recognition design cycle



- Collecting training and testing data.
- How can we know when we have adequately large and representative set of samples?

Training/Test Split

- Randomly split dataset into two parts:
  - Training data
  - Test data
- Use training data to optimize parameters
- Evaluate error using test data

Training/Test Split

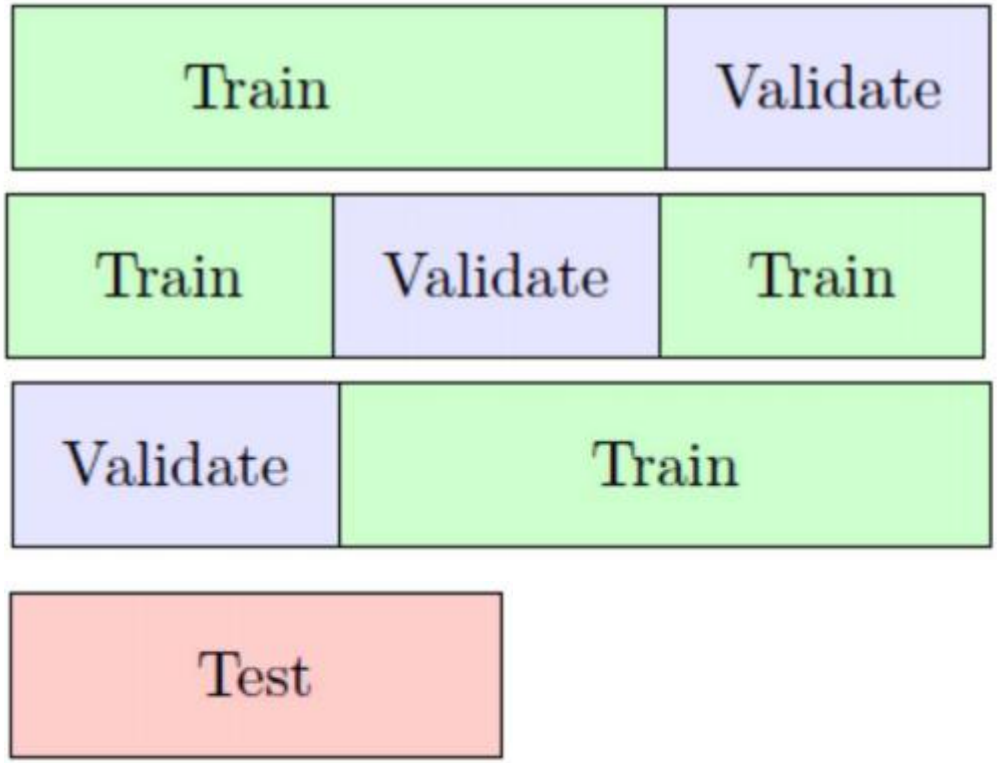
- How many points in each set?
- Very hard question
  - Too few points in training set, learned classifier is bad
  - Too few points in test set, classifier evaluation is insufficient
- Cross−validation
- Leave−one−out cross−validation

Cross-Validation

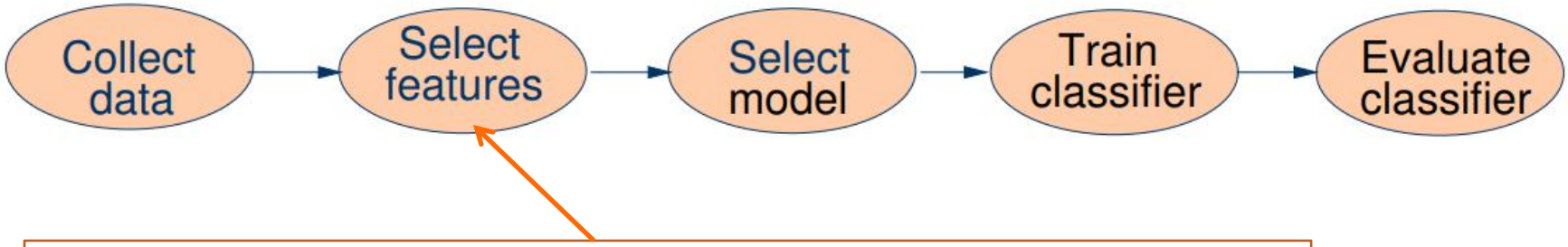
- In practice
- Available data => training and validation
- Train on the training data
- Test on the validation data
- k–fold cross validation:
  - Data randomly separated into k groups
  - Each time k
  - 1 groups used for training and one as testing

Cross Validation and Test Accuracy

- Using CV on training + validation
- Classify test data with the best parameters from CV

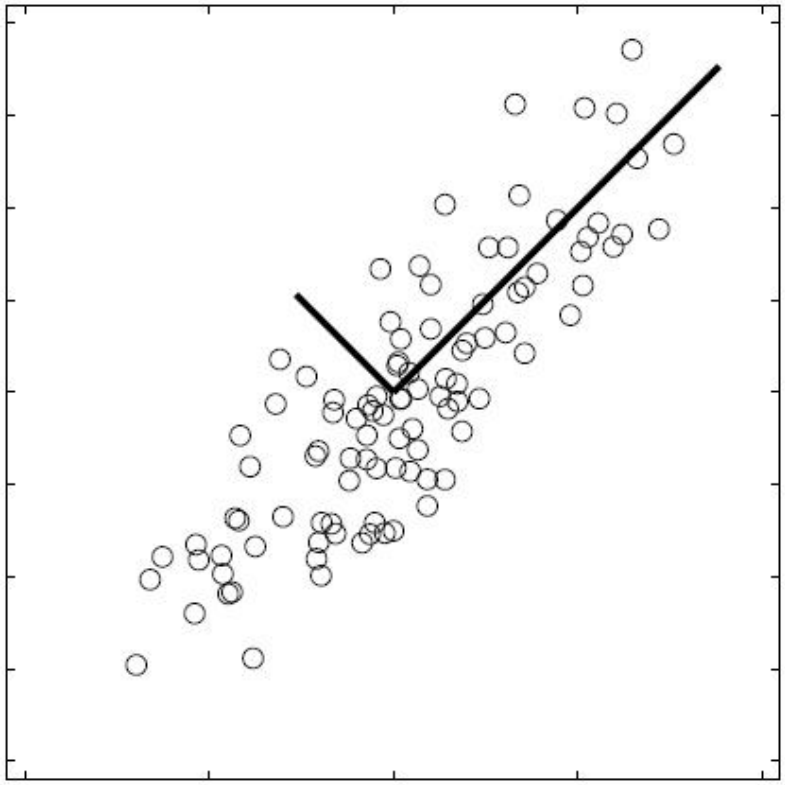


Pattern recognition design cycle



- Domain dependence and prior information.
- Computational cost and feasibility.
- Discriminative features, i.e., similar values for similar patterns, and different values for different patterns.
- Invariant features with respect to translation, rotation and scale.
- Robust features with respect to occlusion, distortion,
- deformation, and variations in environment.

PCA: Visualization



Data points are represented in a rotated orthogonal coordinate system: the origin is the mean of the data points and the axes are provided by the eigenvectors.

Computation of PCA

- In practice we compute PCA via SVD (singular value decomposition)
- Form the centered data matrix:

$$X_{p,N} = \left[(\mathbf{x}_1 - \mathbf{m}) \ldots\ldots (\mathbf{x}_N - \mathbf{m})\right]$$

- Compute its SVD:

$$X_{p,N} = U_{p,p} D_{p,p} (V_{N,p})^T$$

U and *V* are orthogonal matrices, *D* is a diagonal matrix

Computation of PCA…

- Sometimes we are given only a few high dimensional data points, *i.e.*, $p \geqslant N$
- In such cases compute the SVD of X^T:

$$X^T = V_{N,N} D_{N,N} (U_{p,N})^T$$

So we get:

$$X = U_{p,N} D_{N,N} (V_{N,N})^T$$

Then, proceed as before, choose only $d < N$ significant eigenvalues for data representation:

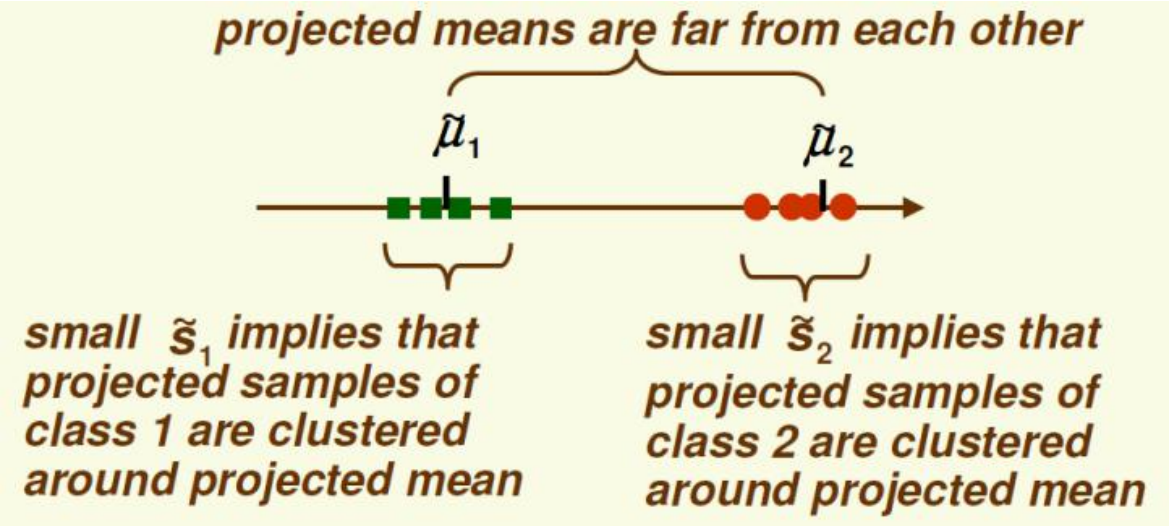
$$\tilde{\mathbf{x}}_i = \mathbf{m} + U_{p,d} (U_{p,d})^T (\mathbf{x}_i - \mathbf{m})$$

Usually we used the features with reduced dimensions to fit the classification models.

Fisher Linear Discriminant

- We need to normalize by both scatter of class 1 and scatter of class 2
- The Fisher linear discriminant is the projection on a line in the direction v which maximizes

$$J(v) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$



Fisher Linear Discriminant

- Thus our objective function can be written:

$$J(v) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2} = \frac{v^t S_B v}{v^t S_W v}$$

- Maximize J(v) by taking the derivative w.r.t. v and setting it to 0

$$\frac{d}{dv} J(v) = \frac{\left(\frac{d}{dv} v^t S_B v\right) v^t S_W v - \left(\frac{d}{dv} v^t S_W v\right) v^t S_B v}{(v^t S_W v)^2}$$

$$= \frac{(2 S_B v) v^t S_W v - (2 S_W v) v^t S_B v}{(v^t S_W v)^2} = 0$$

Fisher Linear Discriminant

Need to solve $\mathbf{v}^t \mathbf{S}_W \mathbf{v}(\mathbf{S}_B \mathbf{v}) - \mathbf{v}^t \mathbf{S}_B \mathbf{v}(\mathbf{S}_W \mathbf{v}) = 0$

$$\Rightarrow \frac{\mathbf{v}^t \mathbf{S}_W \mathbf{v}(\mathbf{S}_B \mathbf{v})}{\mathbf{v}^t \mathbf{S}_W \mathbf{v}} - \frac{\mathbf{v}^t \mathbf{S}_B \mathbf{v}(\mathbf{S}_W \mathbf{v})}{\mathbf{v}^t \mathbf{S}_W \mathbf{v}} = 0$$

$$\Rightarrow \mathbf{S}_B \mathbf{v} - \frac{\mathbf{v}^t \mathbf{S}_B \mathbf{v}(\mathbf{S}_W \mathbf{v})}{\mathbf{v}^t \mathbf{S}_W \mathbf{v}} = 0$$

$$\frac{\mathbf{v}^t \mathbf{S}_B \mathbf{v}}{\mathbf{v}^t \mathbf{S}_W \mathbf{v}} = \lambda$$

$$\Rightarrow \underbrace{\mathbf{S}_B \mathbf{v} = \lambda \mathbf{S}_W \mathbf{v}}$$

generalized eigenvalue problem

Fisher Linear Discriminant

$$S_B v = \lambda S_W v$$

- If S_W has full rank (the inverse exists), we can convert this to a standard eigenvalue problem

$$S_W^{-1} S_B v = \lambda v$$

- But $S_B x$ for any vector x, points in the same direction as $\mu_1 - \mu_2$

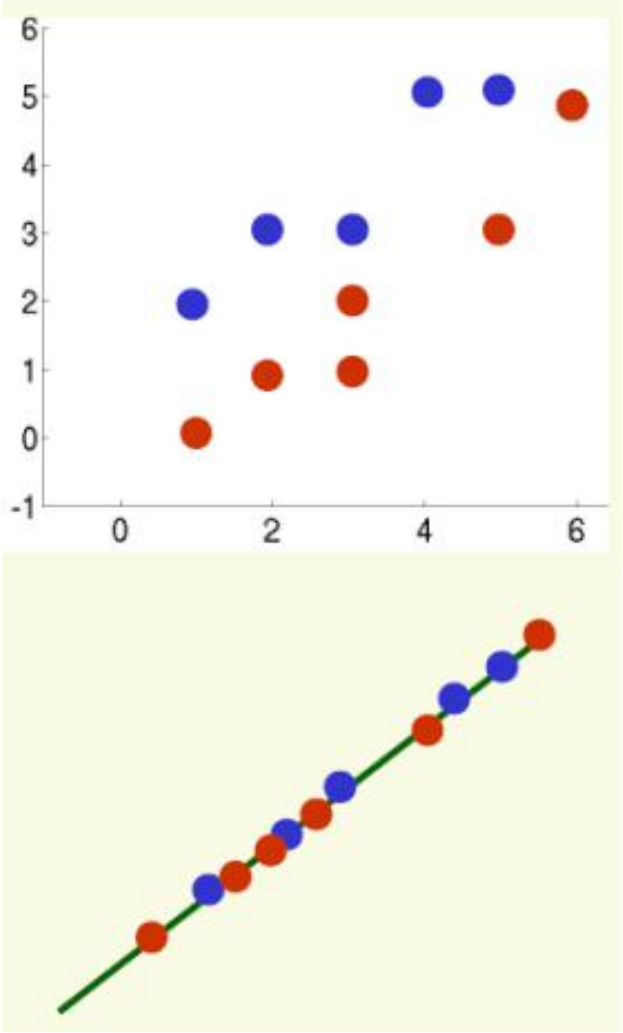
$$S_B x = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^t x = (\mu_1 - \mu_2)\underbrace{(\mu_1 - \mu_2)^t x}_{\alpha} = \alpha(\mu_1 - \mu_2)$$

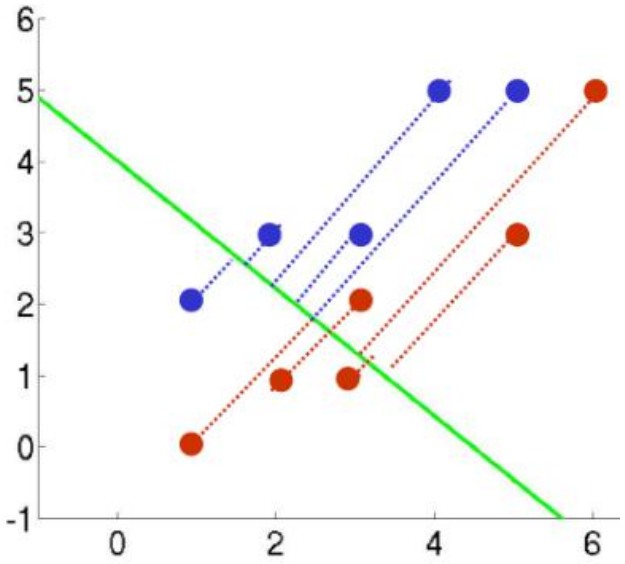
- Based on this, we can solve the eigenvalue problem directly

$$\boxed{v = S_W^{-1}(\mu_1 - \mu_2)}$$

$$S_W^{-1} S_B \underbrace{[S_W^{-1}(\mu_1 - \mu_2)]}_{v} = S_W^{-1}[\alpha(\mu_1 - \mu_2)] = \underbrace{\alpha}_{\lambda}\underbrace{[S_W^{-1}(\mu_1 - \mu_2)]}_{v}$$

Example

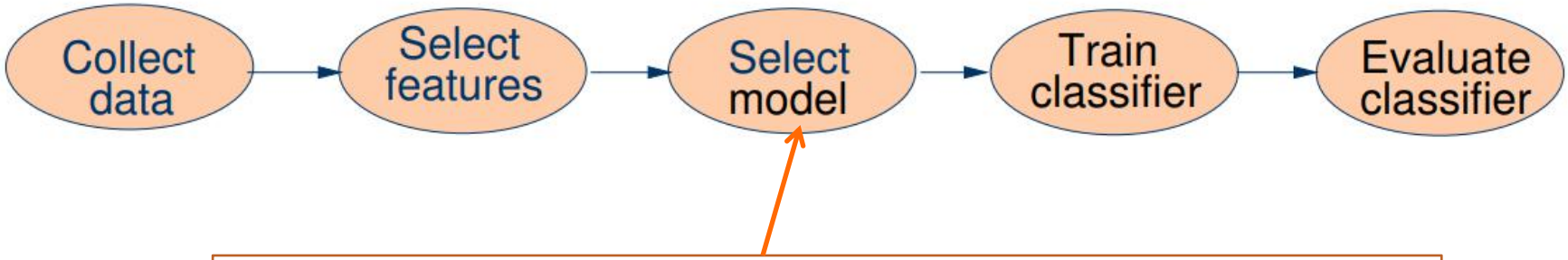




$$Y_1 = v^t c_1^t = \begin{bmatrix} -0.65 & 0.73 \end{bmatrix} \begin{bmatrix} 1 & \cdots & 5 \\ 2 & \cdots & 5 \end{bmatrix} = \begin{bmatrix} 0.81 & \cdots & 0.4 \end{bmatrix}$$

$$Y_2 = v^t c_2^t = \begin{bmatrix} -0.65 & 0.73 \end{bmatrix} \begin{bmatrix} 1 & \cdots & 6 \\ 0 & \cdots & 5 \end{bmatrix} = \begin{bmatrix} -0.65 & \cdots & -0.25 \end{bmatrix}$$

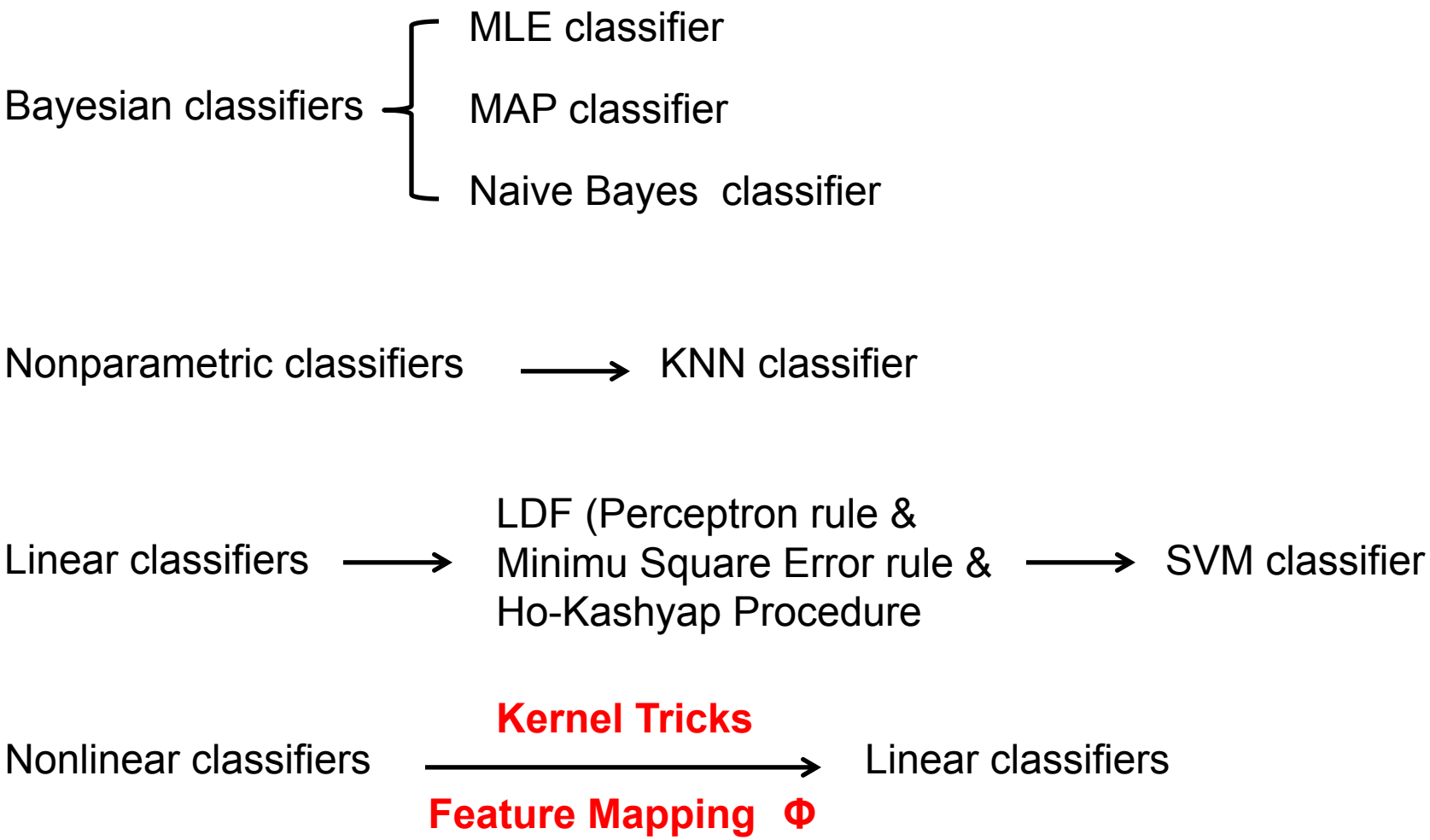
Pattern recognition design cycle



How can we know how close we are to the true model underlying the patterns?

- Domain dependence and prior information.
- Definition of design criteria.
- Parametric vs. non-parametric models.
- Handling of missing features.
- Computational complexity.
- Types of models: templates, decision-theoretic or statistical,syntactic or structural, neural, and hybrid.

The Classifiers We Have Learned So Far



Decision Rule

- Using Bayes' rule:

$$P(\omega_j / x) = \frac{p(x/\omega_j)P(\omega_j)}{p(x)} = \frac{likelihood \times prior}{evidence}$$

where

$$p(x) = \sum_{j=1}^{2} p(x/\omega_j)P(\omega_j)$$

Decide ω_1 if $P(\omega_1/x) > P(\omega_2/x)$; otherwise **decide** ω_2

or

Decide ω_1 if $p(x/\omega_1)P(\omega_1) > p(x/\omega_2)P(\omega_2)$; otherwise **decide** ω_2

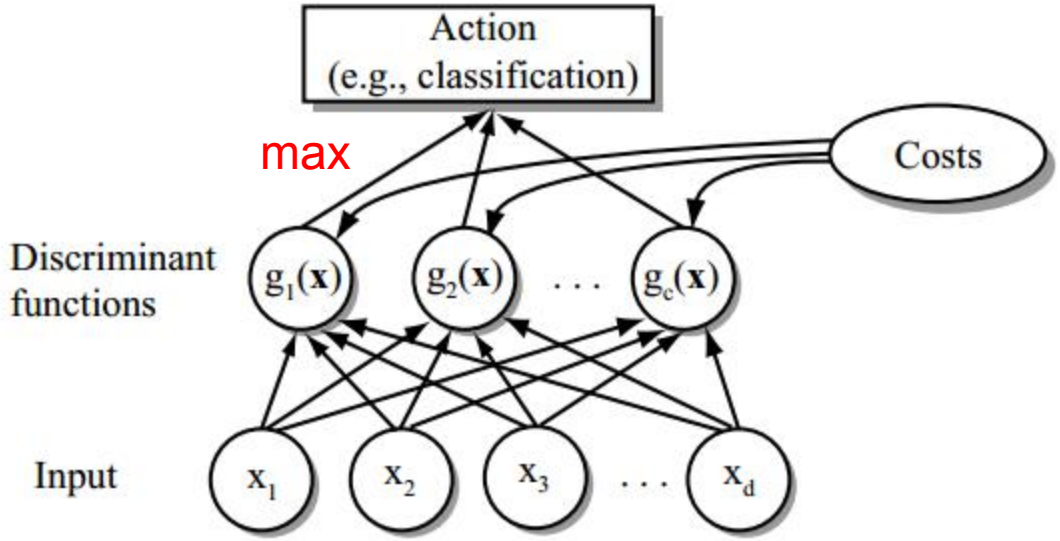
or

Decide ω_1 if $p(x/\omega_1)/p(x/\omega_2) > P(\omega_2)/P(\omega_1)$; otherwise **decide** ω_2

Discriminant Functions

- A useful way to represent a classifier is through **discriminant functions** $g_i(x)$, i = 1, . . . , c, where a feature vector x is assigned to class ω_i if

$$g_i(\mathbf{x}) > g_j(\mathbf{x}) \text{ for all } j \neq i$$



Discriminants for Bayes Classifier

- Is the choice of g_i unique?
  - Replacing $g_i(\mathbf{x})$ with $f(g_i(\mathbf{x}))$, where $f(\cdot)$ is monotonically increasing, does not change the classification results.

$g_i(x)=P(\omega_i/x)$ ⇨

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

$$g_i(\mathbf{x}) = p(\mathbf{x}/\omega_i)P(\omega_i)$$

$$g_i(\mathbf{x}) = \ln p(\mathbf{x}/\omega_i) + \ln P(\omega_i)$$

we'll use this discriminant extensively!

Case 1: Statistically Independent Features with Identical Variances

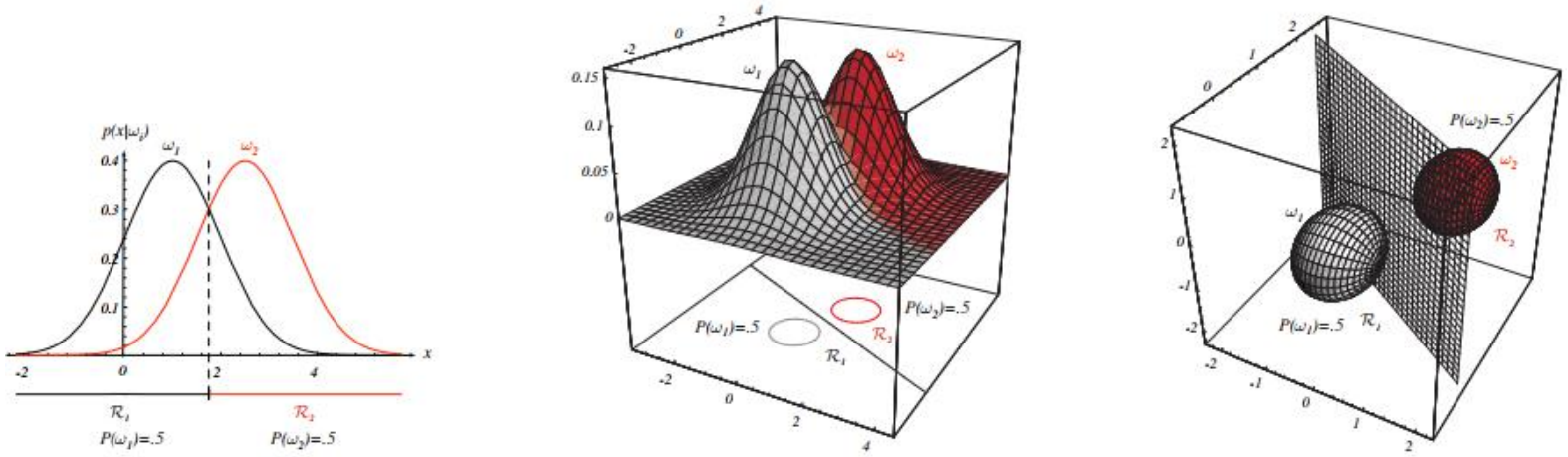
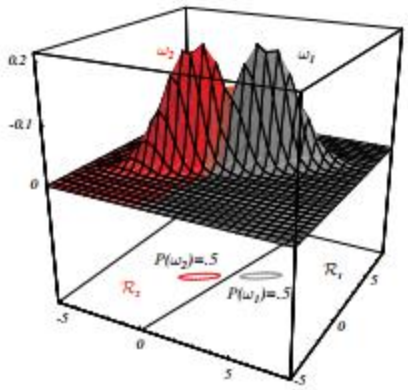
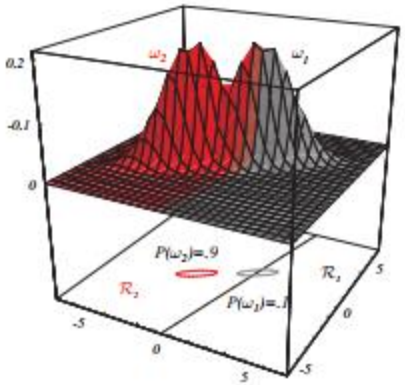


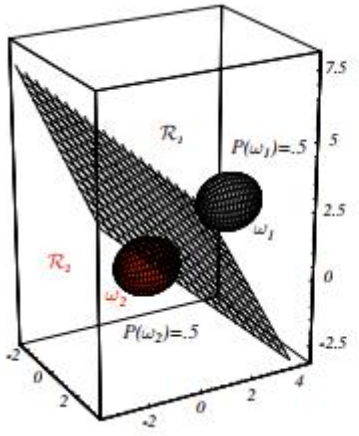
FIGURE 2.10. If the covariance matrices for two distributions are equal and proportional to the identity matrix, then the distributions are spherical in d dimensions, and the boundary is a generalized hyperplane of $d-1$ dimensions, perpendicular to the line separating the means. In these one-, two-, and three-dimensional examples, we indicate $p(\mathbf{x}|\omega_i)$ and the boundaries for the case $P(\omega_1) = P(\omega_2)$. In the three-dimensional case, the grid plane separates $\mathcal{R}_1$ from $\mathcal{R}_2$. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

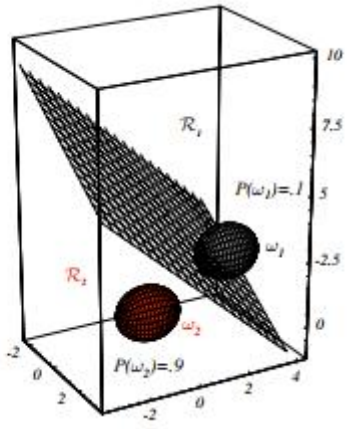
Case II: Identical Covariances $\Sigma_i = \Sigma$

- Notes on Decision Boundary
  - As for Case I, passes through point x0 lying on the line between the two class means. Again, x0 in the middle if priors identical
  - Hyperplane defined by boundary generally not orthogonal to the line between the two means

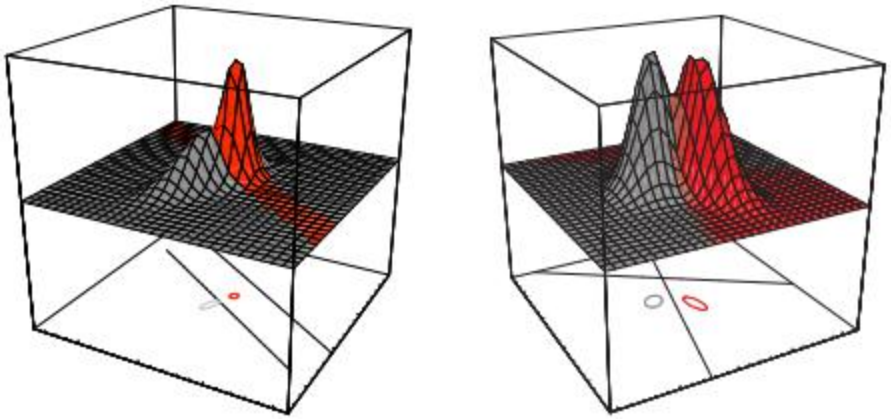




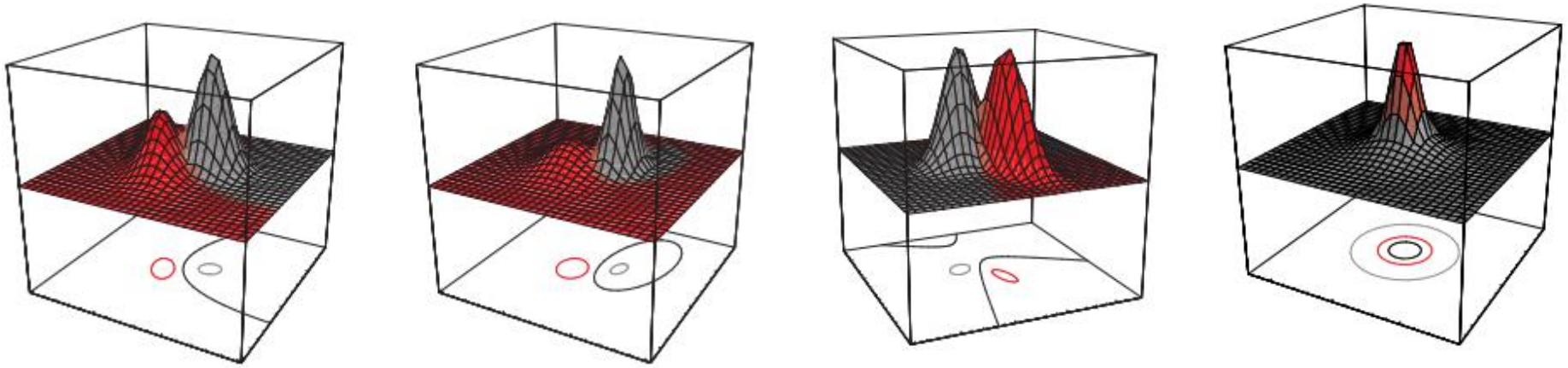




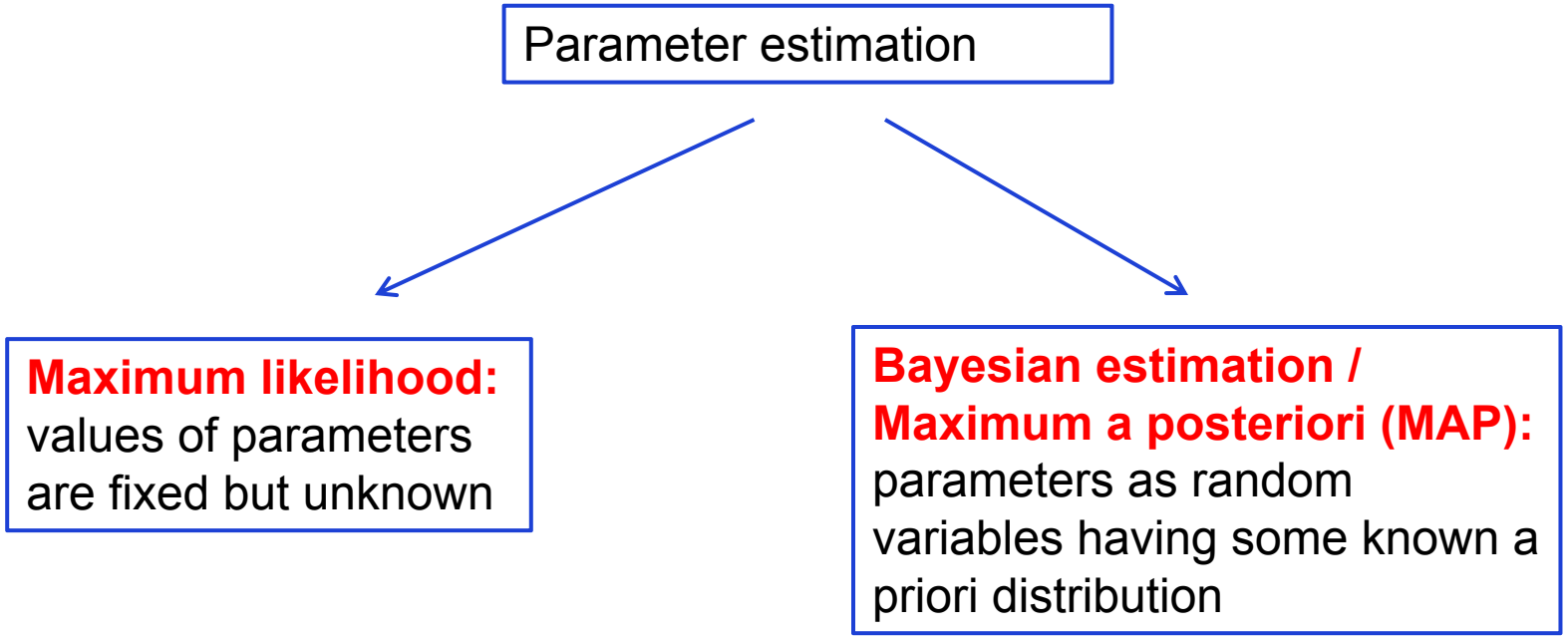
Case III: arbitrary Σ_i



Nonlinear decision boundaries



Parameter Parameter



Maximum-Likelihood Estimation

- Use set of independent samples to estimate $p(D \mid \theta)$

$$\text{Let } D = \{x_1, x_2, \ldots, x_n\} \qquad |D| = n$$

$$p(D \mid \theta) = \prod_{k=1}^{k=n} p(x_k \mid \theta)$$

- Our goal is to determine $\hat{\theta}$ (value of θ that best agrees with observed training data)

- Note if D is fixed $p(D \mid \theta)$ is not a density

Example: Gaussian case

- Assume we have c classes and

$$p(x \mid \omega_j) \sim N(\mu_j, \Sigma_j)$$
$$p(x \mid \omega_j) \equiv p(x \mid \omega_j, \theta_j)$$

- Use the information provided by the training samples to estimate $\theta = (\theta_1, \theta_2, \ldots, \theta_c)$, each θ_i is associated with each category.
- Suppose that D contains n samples, $x_1, x_2, \ldots, x_n$

$$p(D \mid \theta) = \prod_{k=1}^{k=n} p(x_k \mid \theta)$$

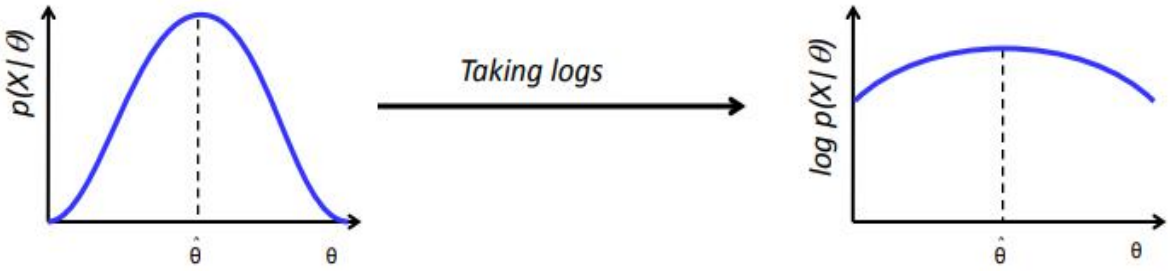
Maximum-Likelihood Estimation

$$p(D \mid \theta) = \prod_{k=1}^{k=n} p(x_k \mid \theta)$$

- $p(D|\theta)$ is called the likelihood of θ w.r.t the set of samples.
- ML estimate of θ is, by definition the value $\hat{\theta}$ that maximizes $p(D|\theta)$

"It is the value of θ that best agrees with the actually observed training sample"

$$\hat{\theta} = argmax[p(X|\theta)] = argmax[\log p(X|\theta)]$$



Optimal Estimation

- Let $\theta = (\theta_1, \theta_2, \ldots, \theta_p)^t$ and let ∇_θ be the gradient operator

$$\nabla_\theta = \left[\frac{\partial}{\partial\theta_1}, \frac{\partial}{\partial\theta_2}, \ldots, \frac{\partial}{\partial\theta_p}\right]^t$$

- We define $l(\theta)$ as the log likelihood function

$$l(\theta) = \ln p(D \mid \theta)$$

- New problem statement:
  - determine θ that maximizes the log likelihood

$$\hat{\theta} = \arg\max_{\theta} l(\theta)$$

Optimal Estimation

$$\nabla_\theta l = \sum_{k=1}^{k=n} \nabla_\theta \ln p(x_k \mid \theta)$$

$$\nabla_\theta l = 0$$

- Local or global maximum
- Local or global minimum
- Saddle point
- Boundary of parameter space

Bayesian Estimation (MAP): General Theory

- $p(x \mid D)$ computation can be applied to any situation in which the unknown density can be parameterized.
- The basic assumptions are:
  - ✓ The form of $p(x \mid \theta)$ assumed known, but the value of θ is not known exactly
  - ✓ Our knowledge about θ is assumed to be contained in a known prior density
  - ✓ The rest of our knowledge θ is contained in a set D of n random variables $x1, x2, \ldots,$ xn that follows $p(x)$

Bayesian Estimation (MAP): General Theory

- The basic problem is: **"Compute the posterior density** $p(\theta \mid D)$**" then "Derive** $p(x \mid D)$ **"**

$$p(x \mid D) = \int \underbrace{p(x \mid \theta)}_{\textit{known}} \underbrace{p(\theta \mid D)}_{\textit{unknown}} d\theta$$

- Using Bayes formula, we have:

$$p(\theta \mid D) = \frac{p(D \mid \theta) p(\theta)}{\int p(D \mid \theta) p(\theta) d\theta}$$

- And by the independence assumption:

$$p(D \mid \theta) = \prod_{k=1}^{k=n} p(x_k \mid \theta)$$

MLE vs. MAP

- Maximum Likelihood estimation (MLE)
  -- Choose value that maximizes the probability of observed data

$$\hat{\theta}_{MLE} = \arg\max_{\theta} P(D|\theta)$$

- Maximum a posteriori (MAP) estimation
  -- Choose value that is most probable given observed data and prior belief

$$\hat{\theta}_{MAP} = \arg\max_{\theta} P(\theta|D)$$
$$= \arg\max_{\theta} P(D|\theta)P(\theta)$$

When is MAP same as MLE?

Naïve Bayes Classifier (not BE)

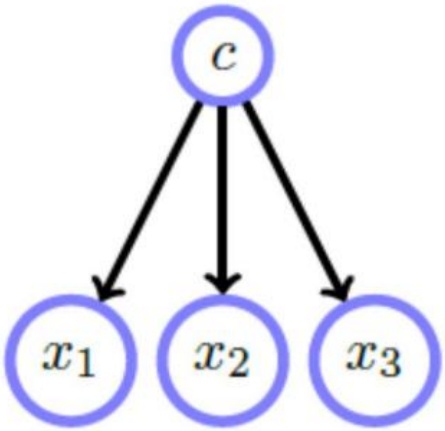
- Simple classifier that applies Bayes' rule with strong (naive) independence assumptions
- A.k.a. the 'independent feature model"

$$p(\omega_i|x_1, x_2,\ldots)= \alpha\, p(x_1|\omega_i)\, p(x_2|\omega_i)\ldots p(\omega_i)$$

- Often performs reasonably well despite simplicity

Naïve Bayes Classifier

- NB is known to produce posteriors closer to extremes (0 or 1) than true posteriors
  - Why?
- NB performs well when only small amounts of training data are available
  - Why?



The Classifiers We Have Learned So Far

Bayesian classifiers
- MLE classifier
- MAP classifier
- Naive Bayes classifier

Nonparametric classifiers → KNN classifier

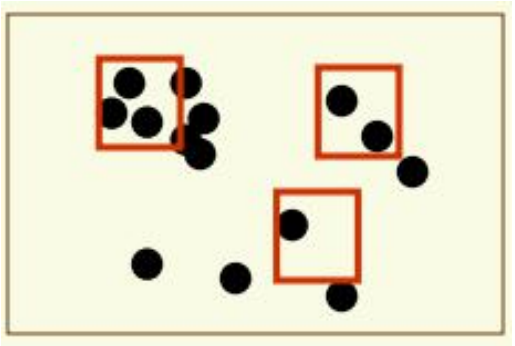
Linear classifiers → LDF (Perceptron rule & Minimu Square Error rule & Ho-Kashyap Procedure) → SVM classifier

Nonlinear classifiers → (Kernel Tricks / Feature Mapping Φ) → Linear classifiers

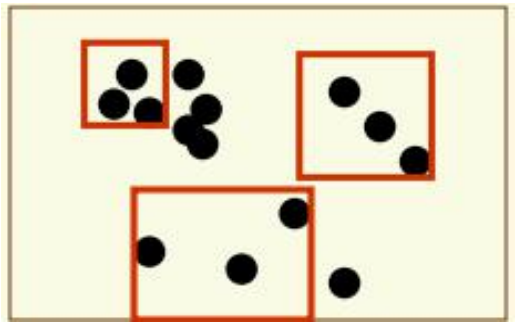
Density Estimation: Two Approaches

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

- Parzen Windows:
  - ✓ Shrink an initial region where $V_n = 1/\sqrt{n}$ and show that
    $$p_n(x) \underset{n\to\infty}{\to} p(x)$$
  - ✓ This is called "the Parzen window estimation method"

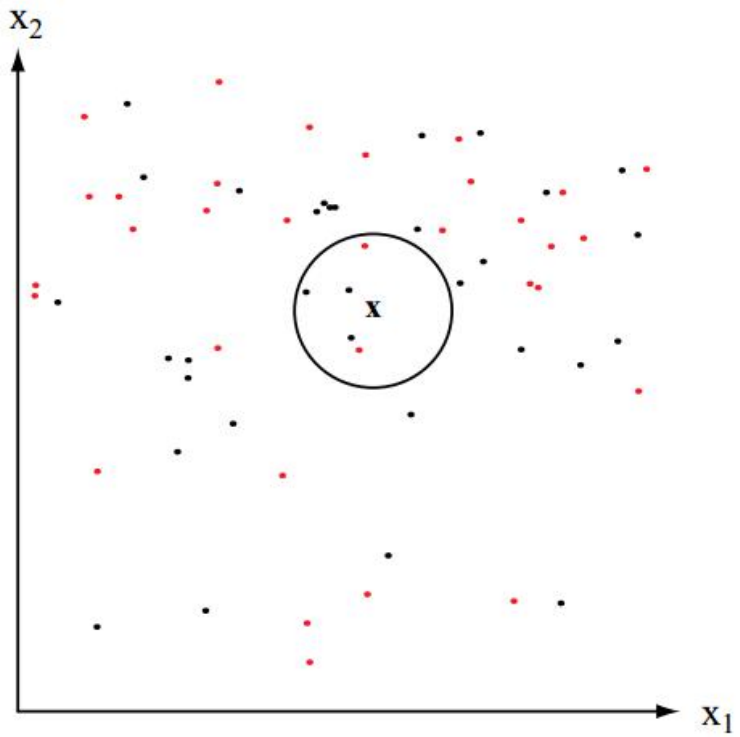


- k-Nearest Neighbors
  - ✓ Specifv *kn* as some function of n, such as $k_n = \sqrt{n}$ the volume *Vn* is grown until it encloses *kn* neighbors of x. This is called "the *kn*–nearest neighbor estimation method"



The k–Nearest-Neighbor Rule

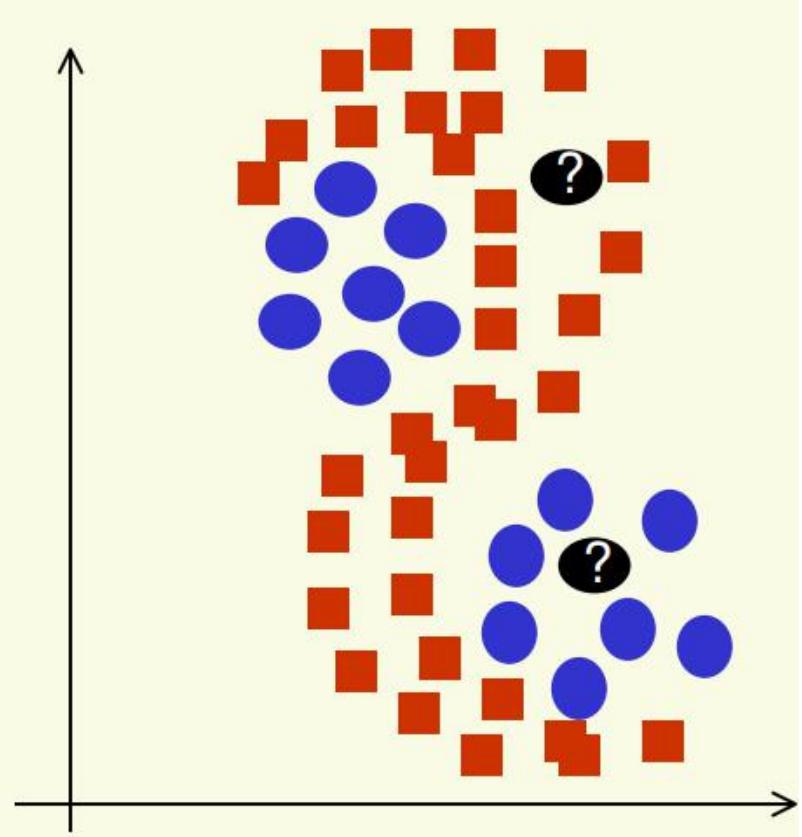
- Goal: Classify x by assigning it the label most frequently represented among the k nearest samples
- Use a voting scheme



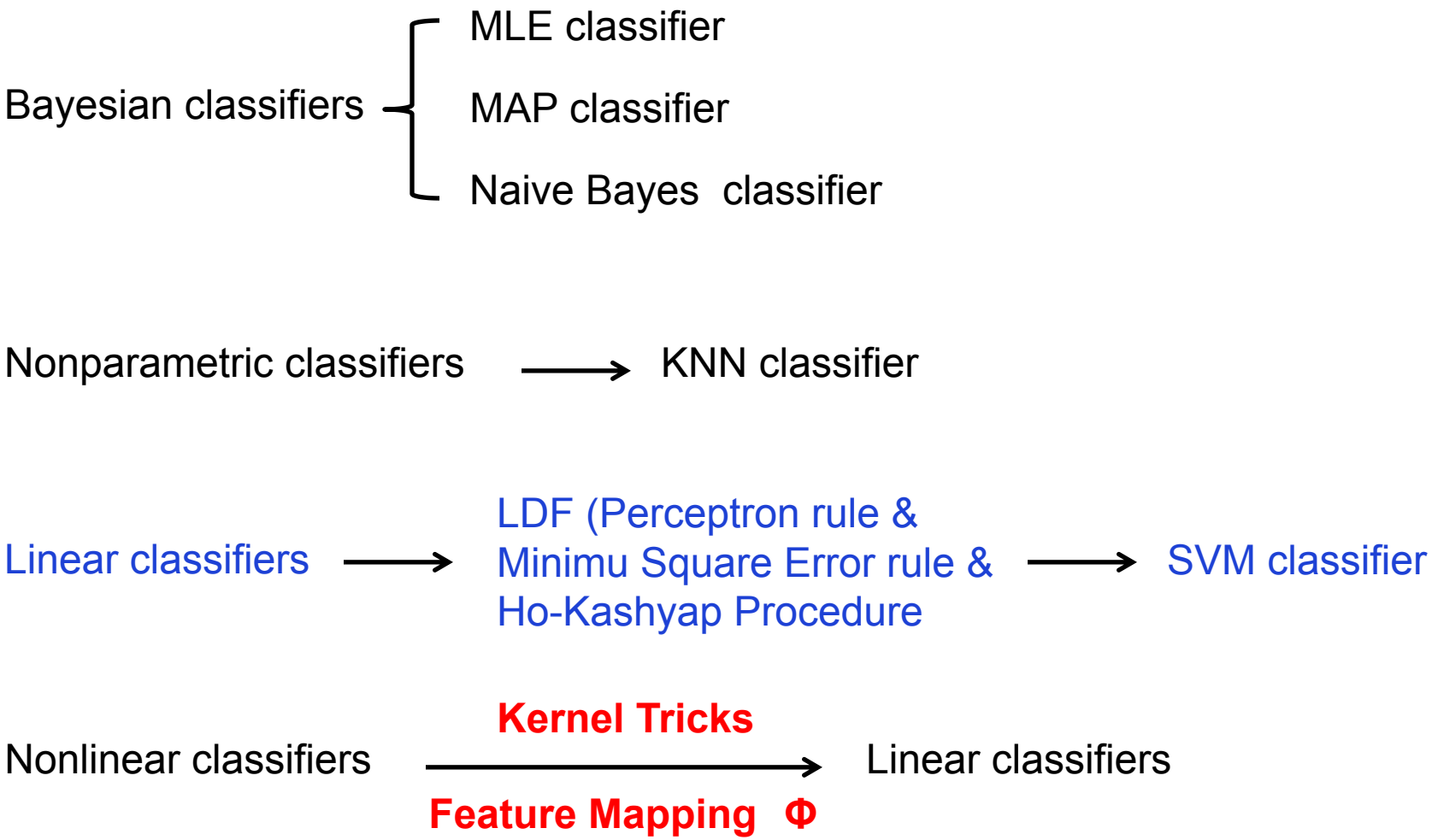
The k-nearest-neighbor query starts at the test point and grows a spherical region until it encloses k training samples, and labels the test point by a majority vote of these samples

kNN: Multi-modal Distributions

- Most parametric distributions would not work for this 2 class classification problem
- Nearest neighbors will do reasonably well, provided we have a lot of samples



The Classifiers We Have Learned So Far



Augmented feature space

- Augmented feature/parameter space

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = \sum_{i=1}^{d} w_i x_i + x_0 w_0 = \sum_{i=0}^{d} w_i x_i = \boldsymbol{\alpha}^t \mathbf{y}$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \cdots \\ w_d \end{bmatrix} \Rightarrow \boldsymbol{\alpha} = \begin{bmatrix} w_0 \\ w_1 \\ \cdots \\ w_d \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_d \end{bmatrix} \Rightarrow \mathbf{y} = \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_d \end{bmatrix}$$

Discriminant:

$$g(\mathbf{x}) = \boldsymbol{\alpha}^t \mathbf{y}$$

If	$\boldsymbol{\alpha}^t\mathbf{y}_i \geq 0$	assign $\mathbf{y}_i$ to ω_1
else if	$\boldsymbol{\alpha}^t\mathbf{y}_i < 0$	assign $\mathbf{y}_i$ to ω_2

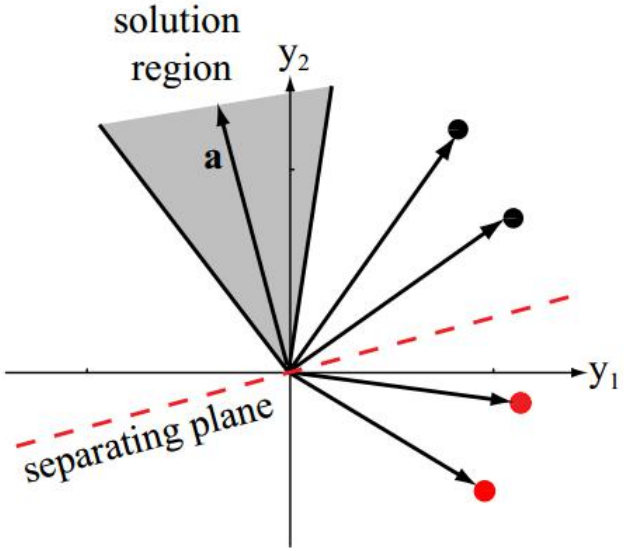
Normalization

Classification rule:

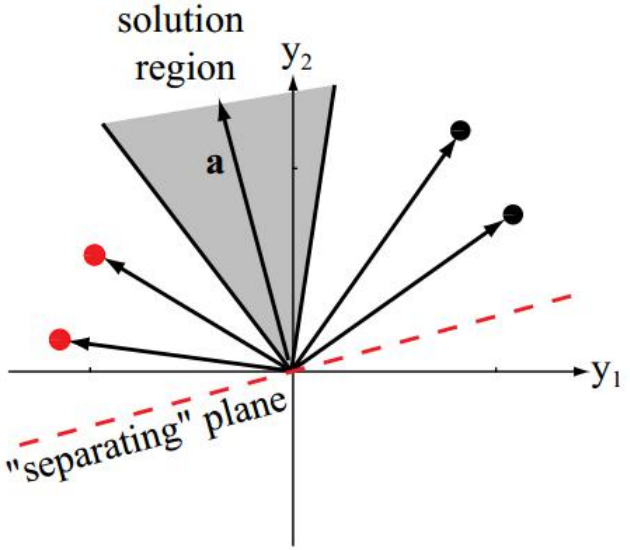
If $\alpha^t y_i > 0$ assign y_i to ω_1

else if $\alpha^t y_i < 0$ assign y_i to ω_2

- If y_i in ω_2, replace y_i by $-y_i$
- Find α such that: $\alpha^t y_i > 0$



Seek a hyperplane that separates patterns from different categories



Seek a hyperplane that puts normalized patterns on the same (positive) side

Perceptron Batch Rule

- The gradient of $J_p(\boldsymbol{\alpha})$ is:

$$J_p(\boldsymbol{\alpha}) = \sum_{\mathbf{y} \in Y(\boldsymbol{\alpha})} (-\boldsymbol{\alpha}^t \mathbf{y}) \quad \Rightarrow \quad \nabla J_p = \sum_{\mathbf{y} \in Y(\boldsymbol{\alpha})} (-\mathbf{y})$$

- It is not possible to solve analytically $\nabla J_p = 0$.

- The perceptron update rule is obtained using gradient descent:

$$\boldsymbol{\alpha}(k+1) = \boldsymbol{\alpha}(k) + \eta(k) \sum_{\mathbf{y} \in Y(\boldsymbol{\alpha})} \mathbf{y}$$

- It is called batch rule because it is based on all misclassified examples

Perceptron Single Sample Rule

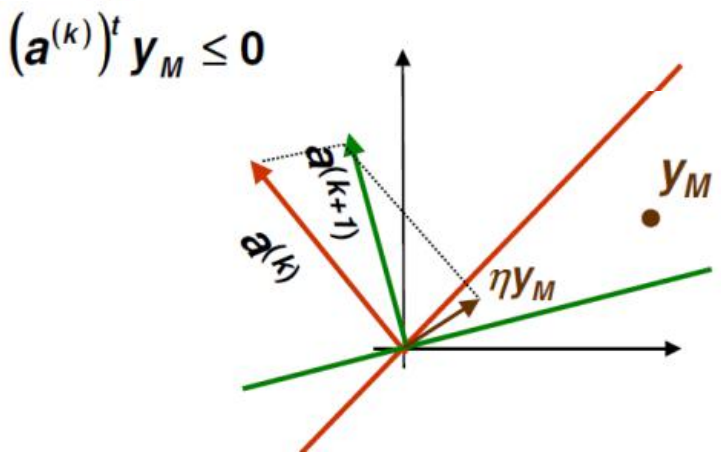
- The gradient decent single sample rule for $J_p(a)$ is:

$$a^{(k+1)} = a^{(k)} + \eta^{(k)} y_M$$

  – Note that y_M is one sample misclassified by $a^{(k)}$
  – Must have a consistent way of visiting samples

- Geometric Interpretation:

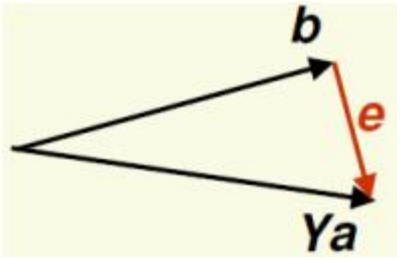
  – Note that y_M is one sample misclassified by $\left(a^{(k)}\right)^t y_M \le 0$
  – yM is on the wrong side of decision hyperplane
  – Adding ηy_M to a moves the new decision hyperplane in the right direction with respect to y_M



MSE Criterion Function

- Minimum squared error approach: find a which minimizes the length of the error vector **e**

$$\mathbf{e} = \mathbf{Ya} - \mathbf{b}$$



- Thus minimize the **minimum squared error criterion** function:

$$J_s(\mathbf{a}) = \|\mathbf{Ya} - \mathbf{b}\|^2 = \sum_{i=1}^{n} (\mathbf{a}^t \mathbf{y}_i - b_i)^2$$

- Unlike the perceptron criterion function, we can optimize the minimum squared error criterion function analytically by setting the gradient to 0

Computing the Gradient

$$J_s(a) = \|Ya - b\|^2 = \sum_{i=1}^{n} (a^t y_i - b_i)^2$$

$$\nabla J_s(a) = \begin{bmatrix} \frac{\partial J_s}{\partial a_0} \\ \vdots \\ \frac{\partial J_s}{\partial a_d} \end{bmatrix} = \frac{dJ_s}{da} = \sum_{i=1}^{n} \frac{d}{da}(a^t y_i - b_i)^2$$

$$= \sum_{i=1}^{n} 2(a^t y_i - b_i)\frac{d}{da}(a^t y_i - b_i)$$

$$= \sum_{i=1}^{n} 2(a^t y_i - b_i) y_i$$

$$= 2Y^t(Ya - b)$$

Pseudo-Inverse Solution

$$\nabla J_s(a) = 2Y^t(Ya - b)$$

- Setting the gradient to 0:

$$2Y^t(Ya - b) = 0 \Rightarrow Y^tYa = Y^tb$$

- The matrix Y^tY is square (it has d $+1$ rows and columns) and it is often non−singular
- If Y^tY is non−singular, its inverse exists and we can solve for a uniquely:

$$a = \boxed{(Y^tY)^{-1}Y^t}\, b$$

pseudo inverse of Y

$$((Y^tY)^{-1}Y^t)Y = (Y^tY)^{-1}(Y^tY) = I$$

Ho-Kashyap Procedure

$$J_{HK}(a,b) = \|Ya - b\|^2$$

- As usual, take partial derivatives w.r.t. a and b

$$\nabla_a J_{HK} = 2Y^t(Ya - b) = 0$$
$$\nabla_b J_{HK} = -2(Ya - b) = 0$$

- Use modified gradient descent procedure to find a minimum of $J_{HK}(a,b)$

- Alternate the two steps below until convergence:
  ① Fix b and minimize $J_{HK}(a,b)$ with respect to a
  ② Fix a and minimize $J_{HK}(a,b)$ with respect to b

LDF Summary

- Perceptron procedures
  - Find a separating hyperplane in the linearly separable case,
  - Do not converge in the non–separable case
  - Can force convergence by using a decreasing learning rate, but are not guaranteed a reasonable stopping point
- MSE procedures
  - Converge in separable and not separable case
  - May not find separating hyperplane even if classes are linearly separable
  - Use pseudoinverse if $\mathbf{Y}^t\mathbf{Y}$ is not singular and not too large
  - Use gradient descent (Widrow–Hoff procedure) otherwise
- Ho–Kashyap procedures
  - always converge
  - find separating hyperplane in the linearly separable case
  - more costly

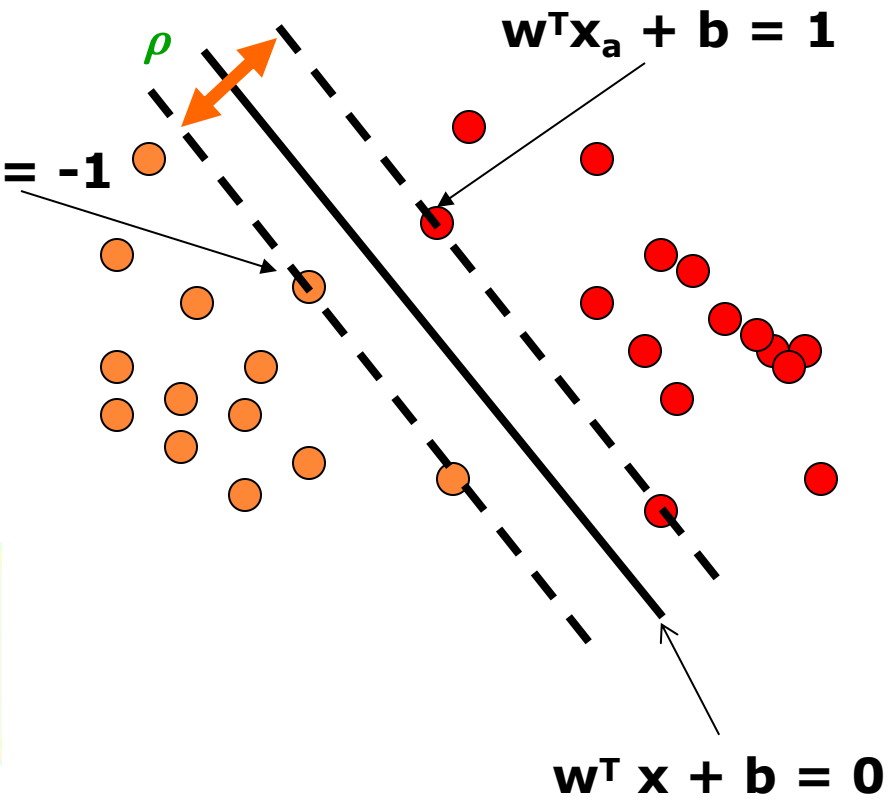
Linear Support Vector Machine (SVM)

- Maximize margin

$\rho = ||x_a - x_b|| = 2/||w||$

- Prime problem

$$\text{minimize } J(w) = \frac{1}{2}\|w\|^2$$
$$\text{constrained to } z_i(w^t x_i + w_o) \geq 1 \quad \forall i$$



SVM solution: Lagrange multipliers

- We can then swap 'max' and 'min':

$$\min_{\mathbf{w}} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{n} \max_{\alpha_i>0} \alpha_i(1 - z_i(\mathbf{w}^t\mathbf{x}_i + w_0)) \right\}$$

$$= \min_{\mathbf{w}} \max_{\alpha_i>0} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{n} \alpha_i(1 - z_i(\mathbf{w}^t\mathbf{x}_i + w_0)) \right\}$$

$$= \max_{\alpha_i>0} \min_{\mathbf{w}} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{n} \alpha_i(1 - z_i(\mathbf{w}^t\mathbf{x}_i + w_0)) \right\}$$

- We can find the optimal w as a function of $\{\alpha_i\}$ by setting the derivatives to zero:

$$\frac{\partial}{\partial \mathbf{w}} J(\mathbf{w}, \alpha) = \mathbf{w} - \sum_{i=1}^{n} \alpha_i z_i \mathbf{x}_i = \mathbf{0}$$

$$\frac{\partial}{\partial w_0} J(\mathbf{w}, \alpha) = -\sum_{i=1}^{n} \alpha_i z_i = 0$$

SVM: Optimal Hyperplane

- Use Kuhn-Tucker Theorem (KTT) condition to convert our problem to:

$$\text{maximize} \quad L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j z_i z_j \mathbf{x}_i^t \mathbf{x}_j$$

$$\text{constrained to} \quad \alpha_i \geq 0 \quad \forall i \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i z_i = 0$$

- $a = \{a_1, \ldots, a_n\}$ are new variables, one for each sample
- Optimized by quadratic programming

SVM: Optimal Hyperplane

- After finding the optimal $a = \{a_1, \ldots, a_n\}$
- Final discriminant function:

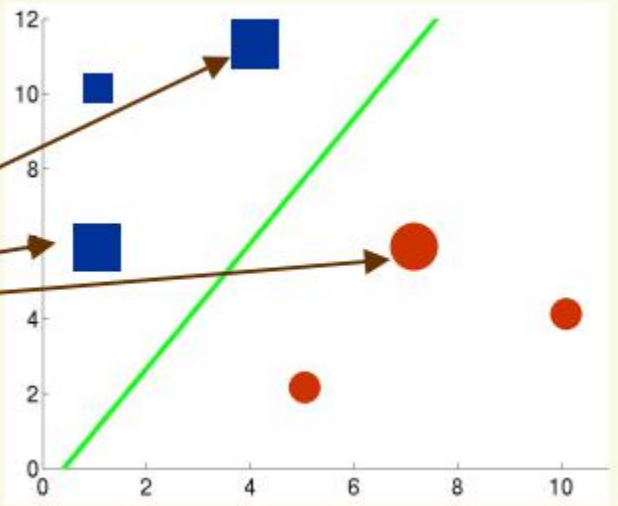
$$g(x) = \left(\sum_{x_i \in S} \alpha_i z_i x_i \right)^t x + w_0$$

$$\alpha_i \left[z_i \left(w^t x_i + w_0 \right) - 1 \right] = 0 \quad \Rightarrow \quad w_0 = \frac{1}{z_i} - w^t x_i$$

- where S is the set of **support vectors**

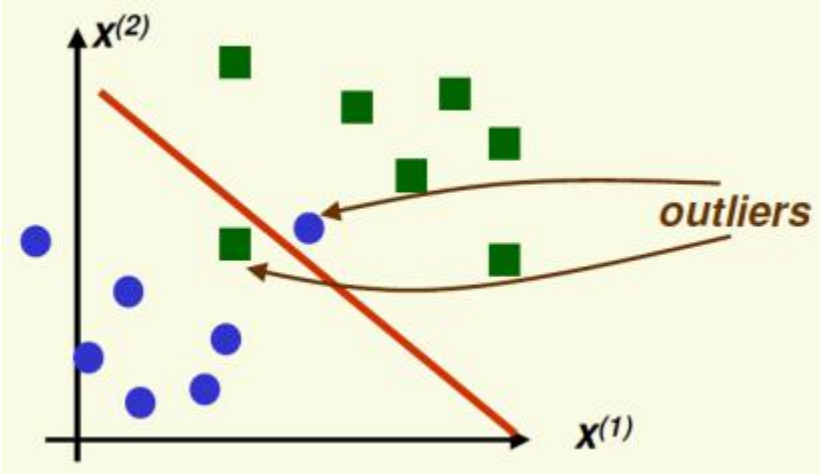
$$\alpha = \begin{bmatrix} 0.036 \\ 0 \\ 0.039 \\ 0 \\ 0.076 \\ 0 \end{bmatrix}$$

support vectors



SVM: Non-Separable Case

- Data are most likely to be not linearly separable, but linear classifier may still be appropriate



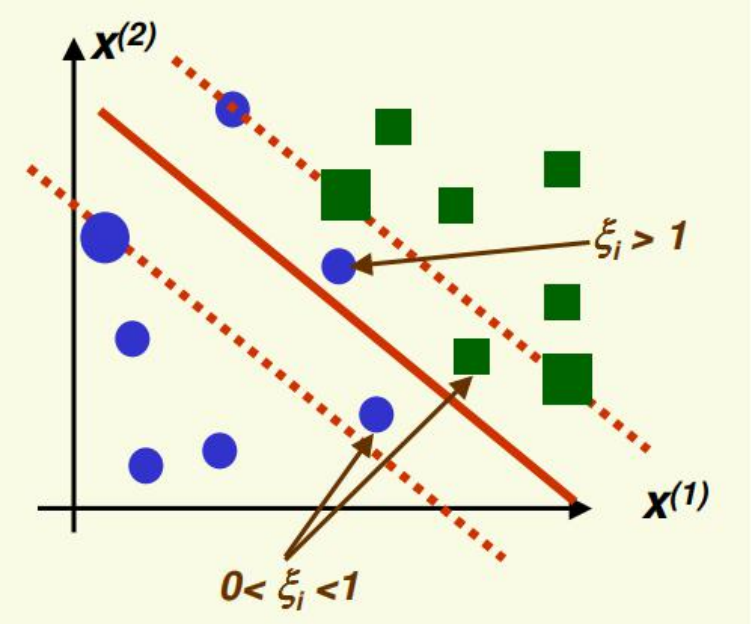
- Can apply SVM in non linearly separable case
- Data should be "almost" linearly separable for good performance

SVM: Non-Separable Case

- Use slack variables $\xi_1, \ldots, \xi_n$ (one for each sample)
- Change constraints:

$$z_i(w^t x_i + w_0) \geq 1 \quad \forall i \quad \Rightarrow \quad z_i(w^t x_i + w_0) \geq 1 - \xi_i \quad \forall i$$

- ξ_i is a measure of deviation from the ideal for x_i
  - $\xi_i \geq 1$: x_i is on the wrong side of the separating hyperplane
  - $0 < \xi_i < 1$: x_i is on the right side of separating hyperplane but within the region of maximum margin
  - $\xi_i < 0$: is the ideal case for x_i



SVM: Non-Separable Case

- Unfortunately this minimization problem is NP–hard due to the discontinuity of I(ξ_i)
- Instead, we minimize

$$J(w,\xi_1,\ldots,\xi_n)=\frac{1}{2}\|w\|^2+\beta\sum_{i=1}^{n}\xi_i$$

a measure of # of misclassified examples

subject to

$$\begin{cases} z_i\left(w^t x_i + w_0\right)\geq 1-\xi_i & \forall i \\ \xi_i \geq 0 & \forall i \end{cases}$$

SVM: Non-Separable Case

- Use Kuhn–Tucker Theorem (KTT) conditon to convert to:

$$\text{maximize} \quad L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j z_i z_j x_i^t x_j$$

$$\text{constrained to} \quad 0 \le \alpha_i \le \beta \quad \forall i \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i z_i = 0$$

- w and w_0 is computed using:

$$w = \sum_{i=1}^{n} \alpha_i z_i x_i \quad \Rightarrow \quad \alpha_i \left[z_i \left(w^t x_i + w_0 \right) - 1 \right] = 0$$

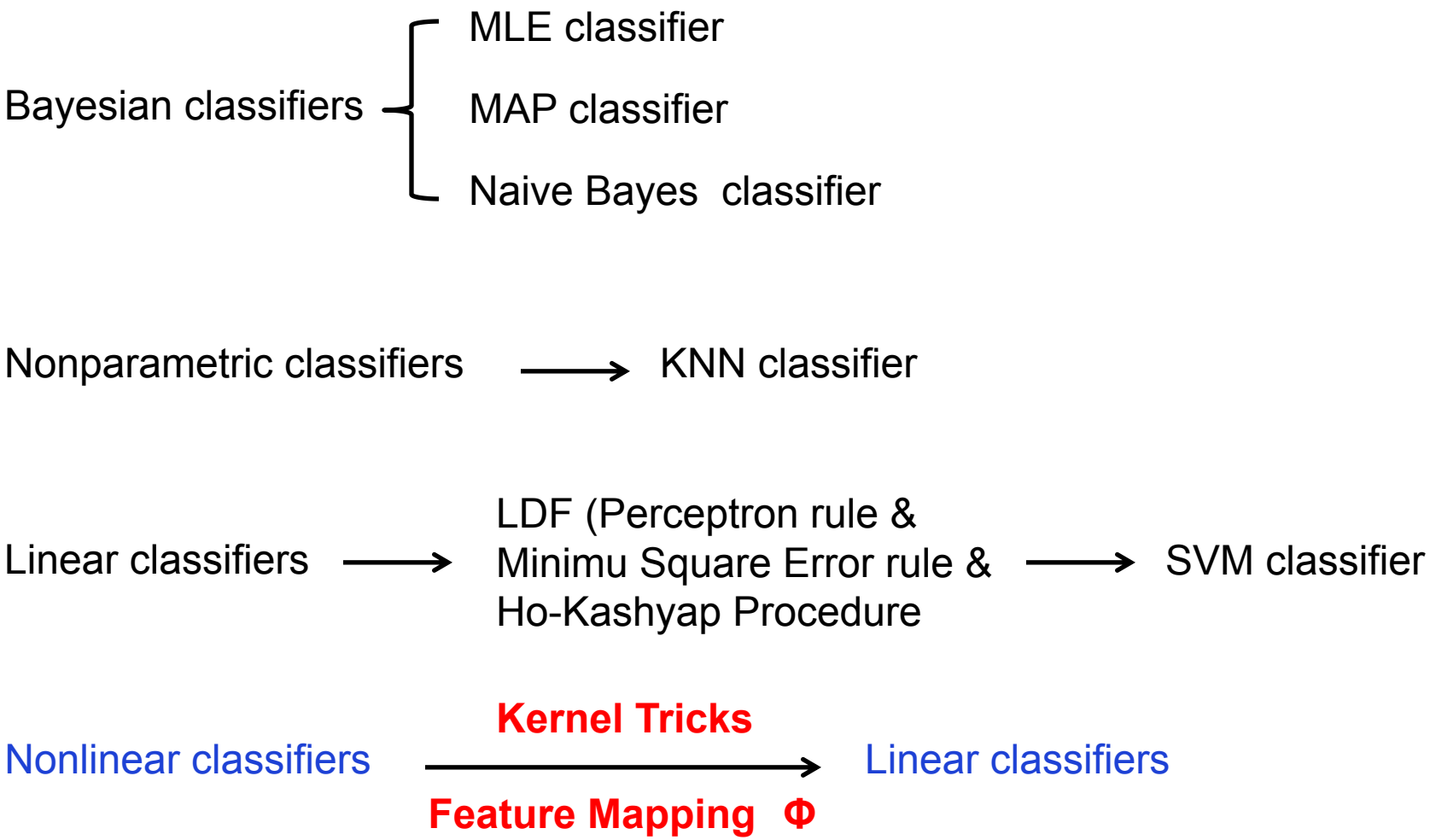
- Remember that:

$$g(x) = \left(\sum_{x_i \in S} \alpha_i z_i x_i \right)^t x + w_0$$

What about multi-class SVMs?

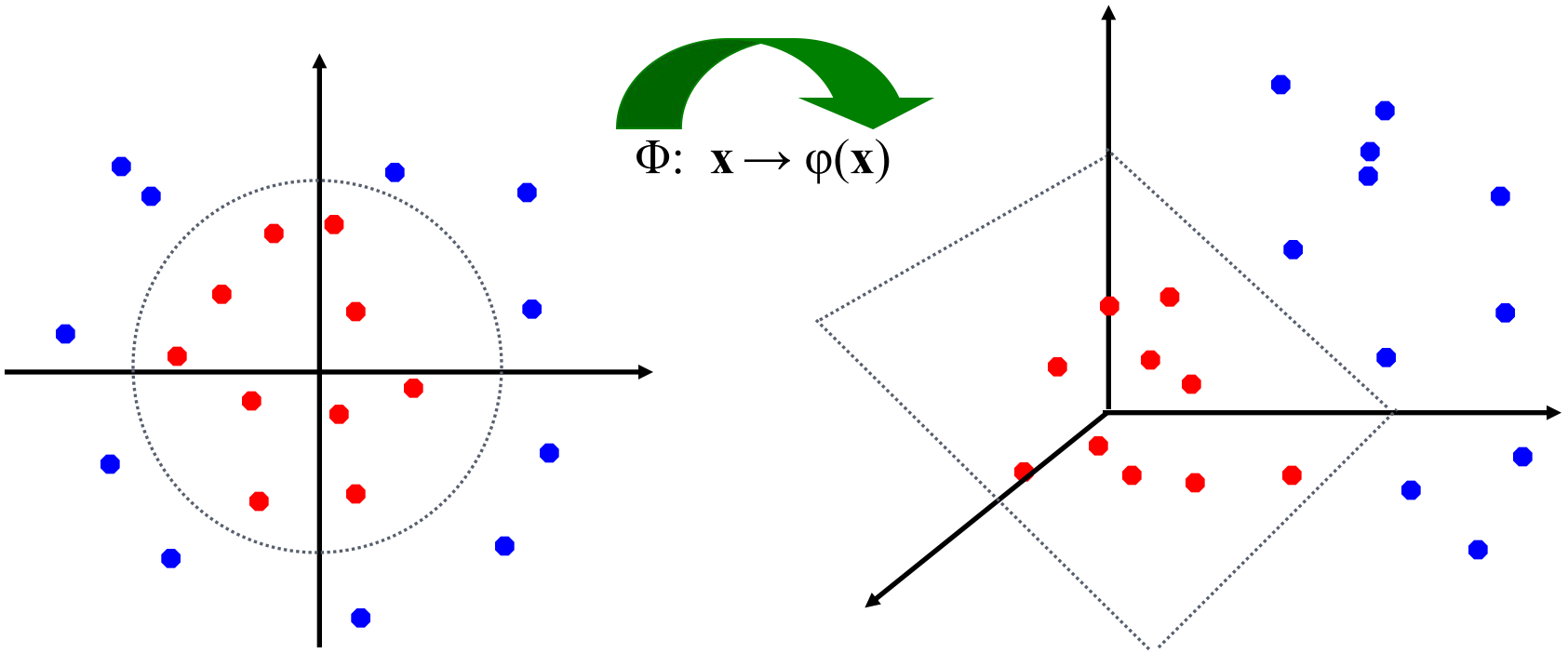
- Unfortunately, there is no "definitive" multi−class SVM formulation
- In practice, we have to obtain a multi−class SVM by combining multiple two/ class SVMs
- One vs. others
  - Traning: learn an SVM for each class vs. the others
  - Testing: apply each SVM to test example and assign to it the class of the SVM that returns the highest decision value
- One vs. one
  - Training: learn an SVM for each pair of classes
  - Testing: each learned SVM "votes" for a class to assign to the test example

The Classifiers We Have Learned So Far



Non-linear SVMs: Feature spaces

- General idea: the original feature space can always be mapped to some higher dimensional feature space where the training set is separable:



Kernels

- SVM optimization:

Maximize $$L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j z_i z_j x_i^t x_j$$

- Note this optimization depends on samples x_i only through the dot product $x_i^t x_j$

- If we lift x_i to high dimension using $\phi(\mathbf{x_i})$, we need to compute high dimensional product $\phi(\mathbf{x_i})^T \phi(\mathbf{x_j})$

$$L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j z_i z_j \underbrace{\varphi(x_i)^t \varphi(x_j)}_{K(x_i, x_j)}$$

- Idea: find kernel function K(xi,xj) s.t. $K(x_i, x_j) = \varphi(x_i)^t \varphi(x_j)$

Kernel Trick

- Then we only need to compute $K(x_i, x_j)$ instead of $\phi(\mathbf{x_i})^T \phi(\mathbf{x_j})$

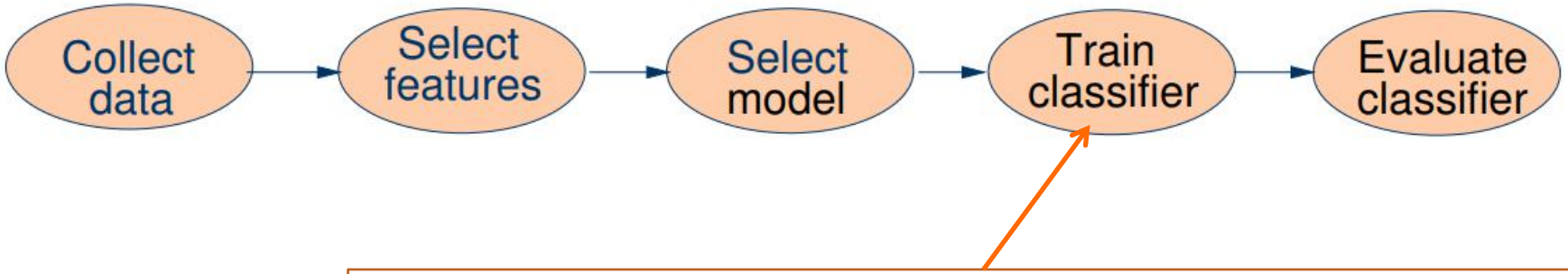
- "**kernel trick**": do not need to perform operations in high dimensional space explicitly

Choice of Kernel

- How to choose kernel function $K(x_i, x_j)$?
  - $K(x_i, x_j)$ should correspond to $\phi(\mathbf{x_i})^T \phi(\mathbf{x_j})$ in a higher dimensional space
  - Mercer's condition tells us which kernel function can be expressed as dot product of two vectors
  - If K and K' are kernels aK+bK' is a kernel

- The mappings $\phi(x_i)$ never have to be computed!!

Pattern recognition design cycle



How can we learn the rule from data?

- **Supervised learning**: a teacher provides a category label or cost for each pattern in the training set.
- **Unsupervised learning**: the system forms clusters or natural groupings of the input patterns.
- **Reinforcement learning**: no desired category is given but the teacher provides feedback to the system such as the decision is right or wrong.

Learning Algorithms

- To design a learning algorithm, we face the following problems:
  ① Whether to stop?
  ② In what direction to proceed?
  ③ How long a step to take?

  Is the criterion satisfactory?

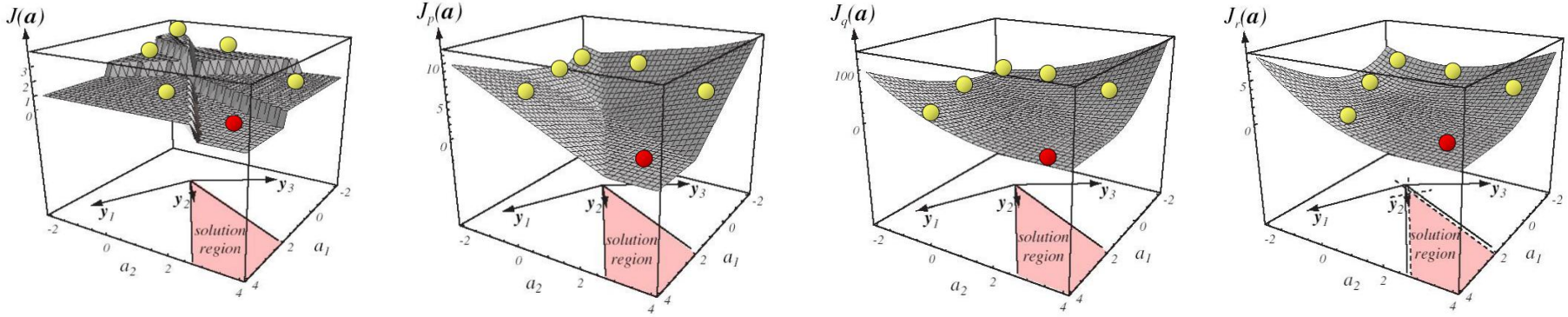
Criterion Function

- To facilitate learning, we usually define a scalar *criterion function*.
- It usually represents the *penalty* or *cost* of a solution.
- Our goal is to *minimize* its value, i.e., *Function optimization*.

$$J_p(\mathbf{a}) = \sum_{\mathbf{y} \in \mathcal{Y}} (-\mathbf{a}^T \mathbf{y})$$

$$J_q(\mathbf{a}) = \sum_{\mathbf{y} \in \mathcal{Y}} (\mathbf{a}^T \mathbf{y})^2$$

$$J_r(\mathbf{a}) = \sum_{\mathbf{y} \in \mathcal{Y}} \frac{(\mathbf{a}^T \mathbf{y} - b)^2}{\| \mathbf{y} \|^2}$$



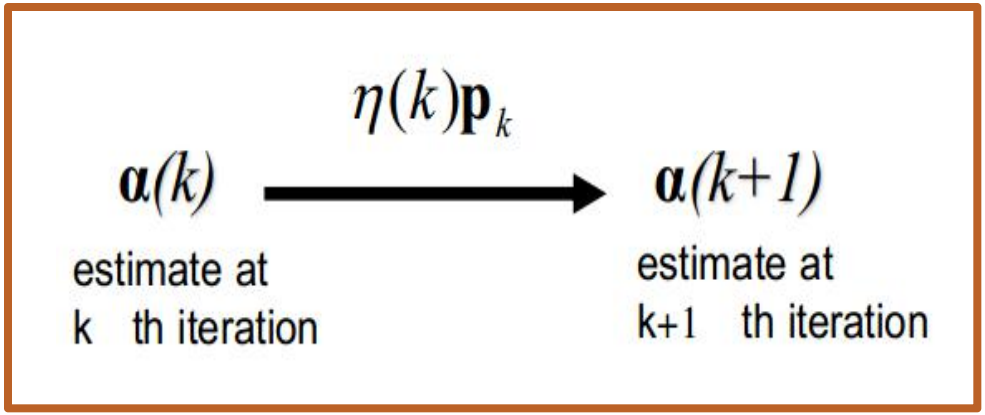
Learning Using Iterative Optimization

- Minimize an error function $J(\boldsymbol{\alpha})$ (e.g., classification error) with respect to $\boldsymbol{\alpha}$:

- Minimize $J(\boldsymbol{\alpha})$ iteratively: $\boldsymbol{\alpha}(k+1) = \boldsymbol{\alpha}(k) + \eta(k)\mathbf{p}_k$

learning rate

search direction



How should we choose $\mathbf{p}_k$?

Choosing p_k using Gradient Descent

$$\mathbf{p}_k = -\nabla J(\boldsymbol{\alpha}(k))$$

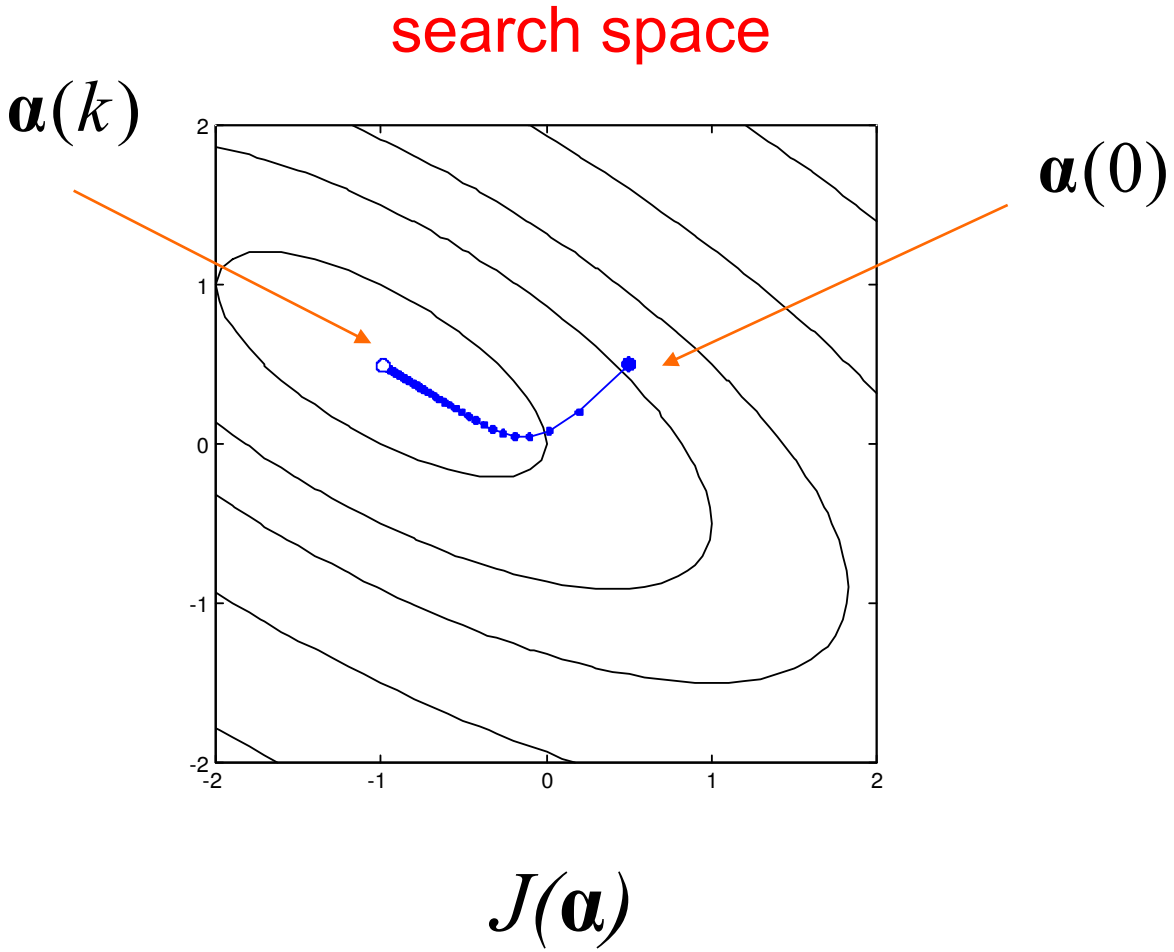
$$\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k)\nabla J(\mathbf{a}(k))$$

Algorithm 1 (Basic gradient descent)

```
1 begin initialize  a, criterion θ, η(·), k = 0
2    do k ← k + 1
3        a ← a − η(k)∇J(a)
4    until η(k)∇J(a) < θ
5 return a
6 end
```

(a = α)

Gradient Decent (cont'd)



Gradient Decent (cont'd)

- How to choose the learning rate $\eta(k)$?

Taylor series approximation (a = α)

$$J(\mathbf{a}) \simeq J(\mathbf{a}(k)) + \nabla J^t(\mathbf{a} - \mathbf{a}(k)) + \frac{1}{2}(\mathbf{a} - \mathbf{a}(k))^t \mathbf{H} (\mathbf{a} - \mathbf{a}(k))$$

Hessian (2nd derivatives)

Setting a=a(k+1) and using $\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k)\nabla J(\mathbf{a}(k))$

$$J(\mathbf{a}(k+1)) \simeq J(\mathbf{a}(k)) - \eta(k)\|\nabla J\|^2 + \frac{1}{2}\eta^2(k)\nabla J^t \mathbf{H} \nabla J$$

$$\eta(k) = \frac{\|\nabla J\|^2}{\nabla J^t \mathbf{H} \nabla J}$$

optimum learning rate

Choosing pk using Newton's Method

$$\mathbf{p}_k = -H^{-1}\nabla J(\boldsymbol{\alpha}(k))$$

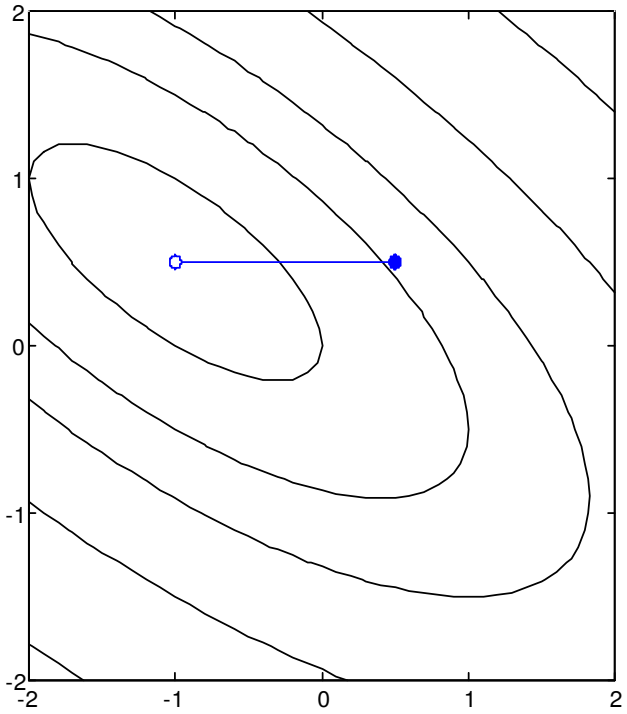
$$\mathbf{a}(k+1) = \mathbf{a}(k) - \mathbf{H}^{-1}\nabla J$$ requires inverting H

Algorithm 2 (Newton descent)

```
1 begin initialize  a, criterion θ
2         do
3             a ← a − H^{-1} ∇J(a)        (a = α)
4         until H^{-1} ∇J(a) < θ
5     return a
6 end
```

Newton's Method (cont'd)

$J(\boldsymbol{\alpha})$

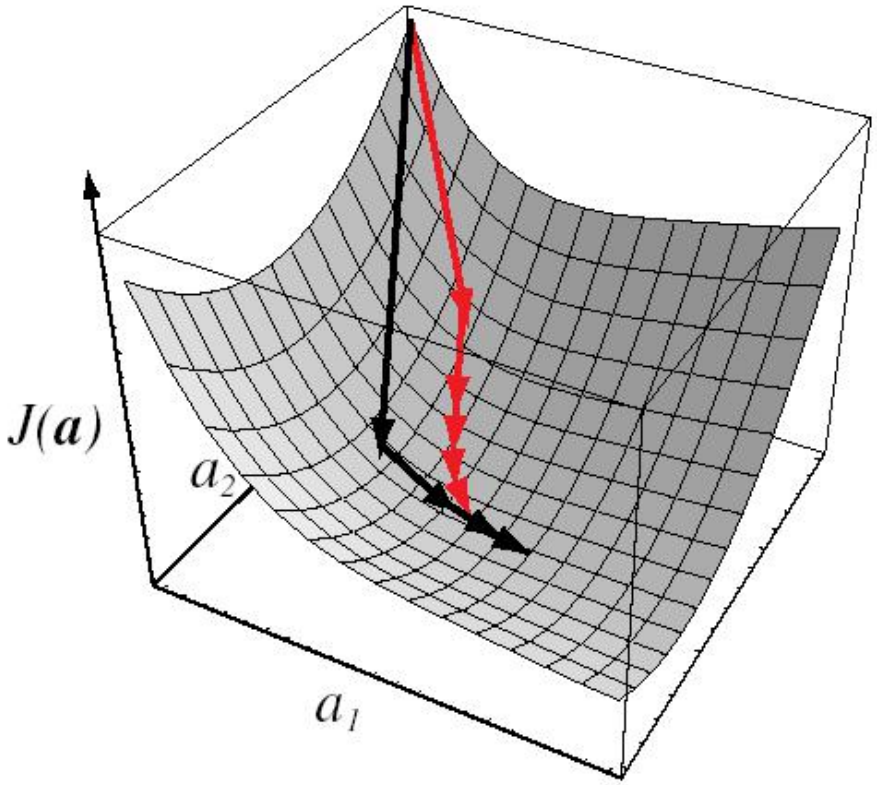


If $J(\boldsymbol{\alpha})$ is **quadratic**, Newton's method converges in **one step**!

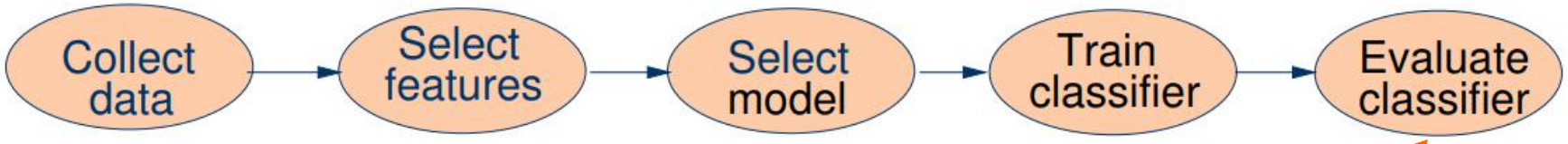
Gradient decent vs Newton's method

Newton's method

Gradient Decent



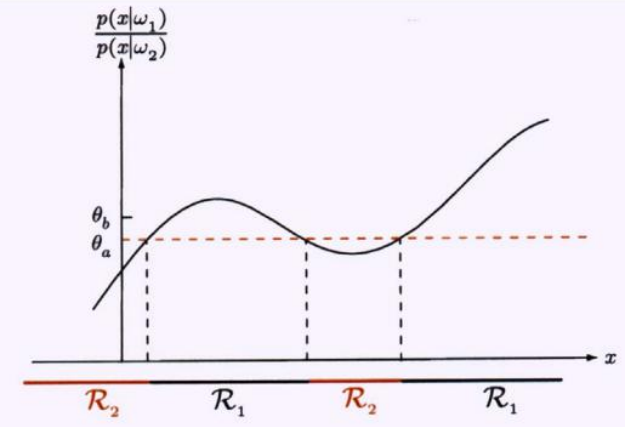
Pattern recognition design cycle



- How can we estimate the performance with training samples?
- How can we predict the performance with future data?
- Problems of overfitting and generalization.

Receiver Operating Characteristic (ROC) Curve

- Every classifier typically employs some kind of a threshold.

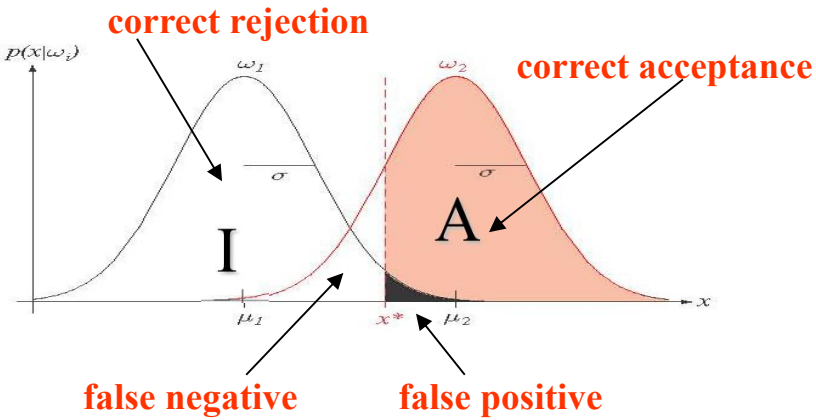


$$\theta_a = P(\omega_2)/P(\omega_1)$$

$$\theta_b = \frac{P(\omega_2)(\lambda_{12}-\lambda_{22})}{P(\omega_1)(\lambda_{21}-\lambda_{11})}$$

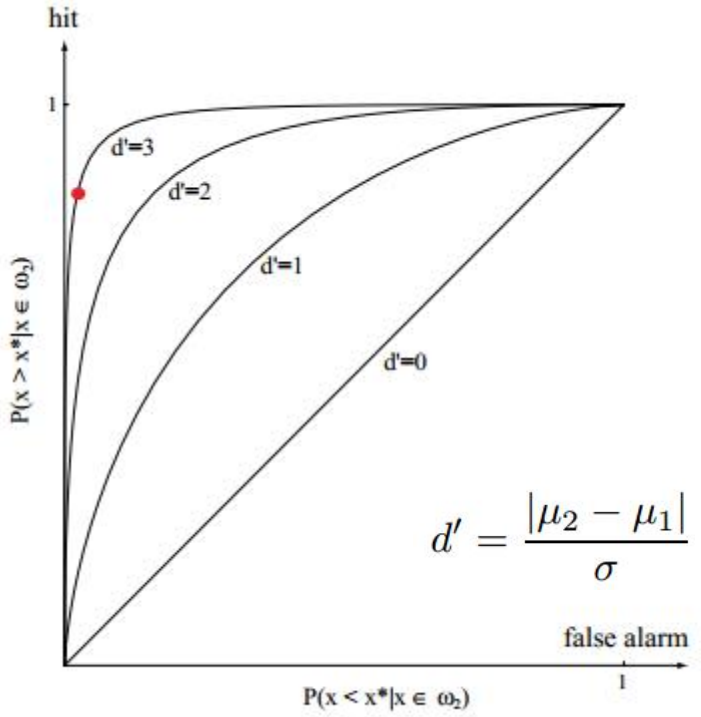
- Changing the threshold will affect the performance of the classifier.
- ROC curves allow us to evaluate the performance of a classifier using different thresholds.

ROC Curve



$$TPR = TP/P = TP/(TP + FN)$$

$$FPR = FP/N = FP/(FP + TN)$$



FPR: False Positive Rate (X-axis)
TRR: True Postive Rate (Y-axis)

Overfitting

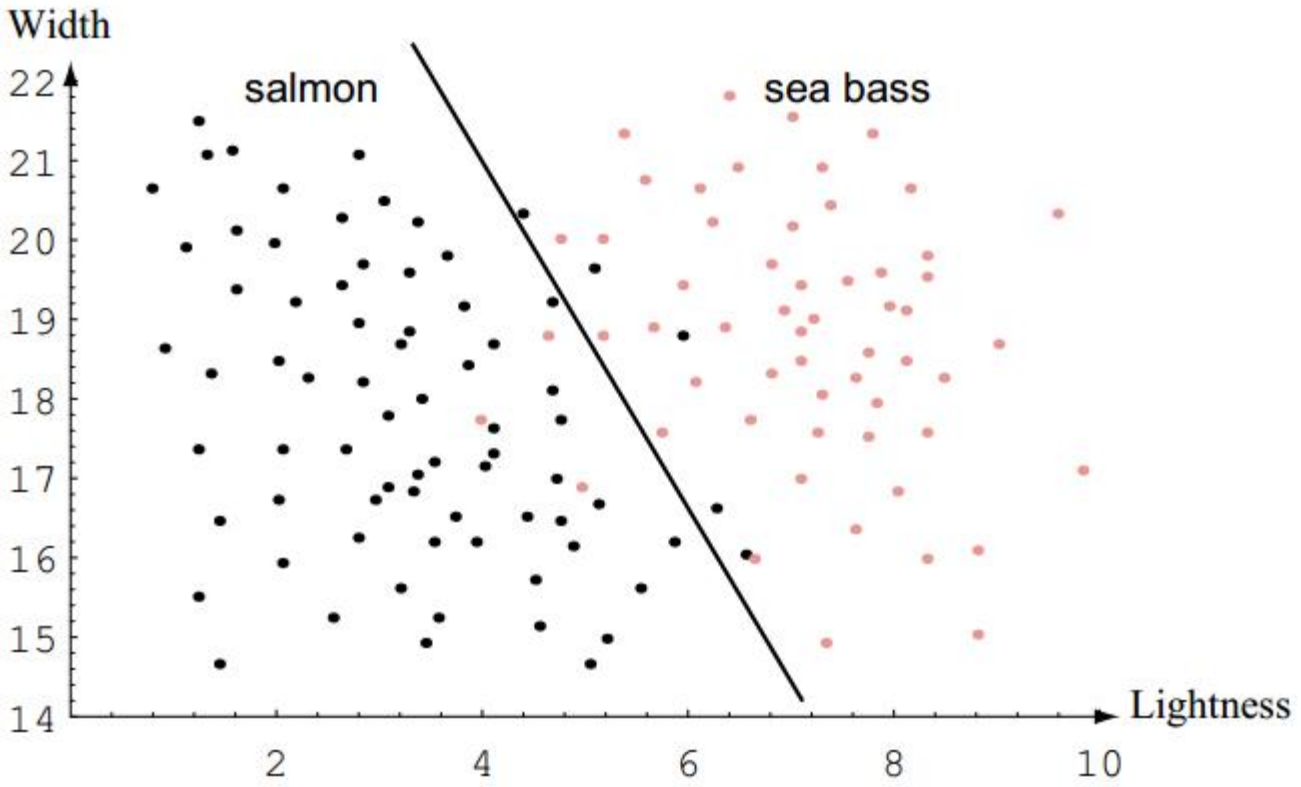
- Prediction error: probability of test pattern not in class with max posterior (true)
- Training error: probability of test pattern not in class with max posterior (estimated)
- Classifier optimized w.r.t. training error
  - Training error: optimistically biased estimate of prediction error

Overfitting

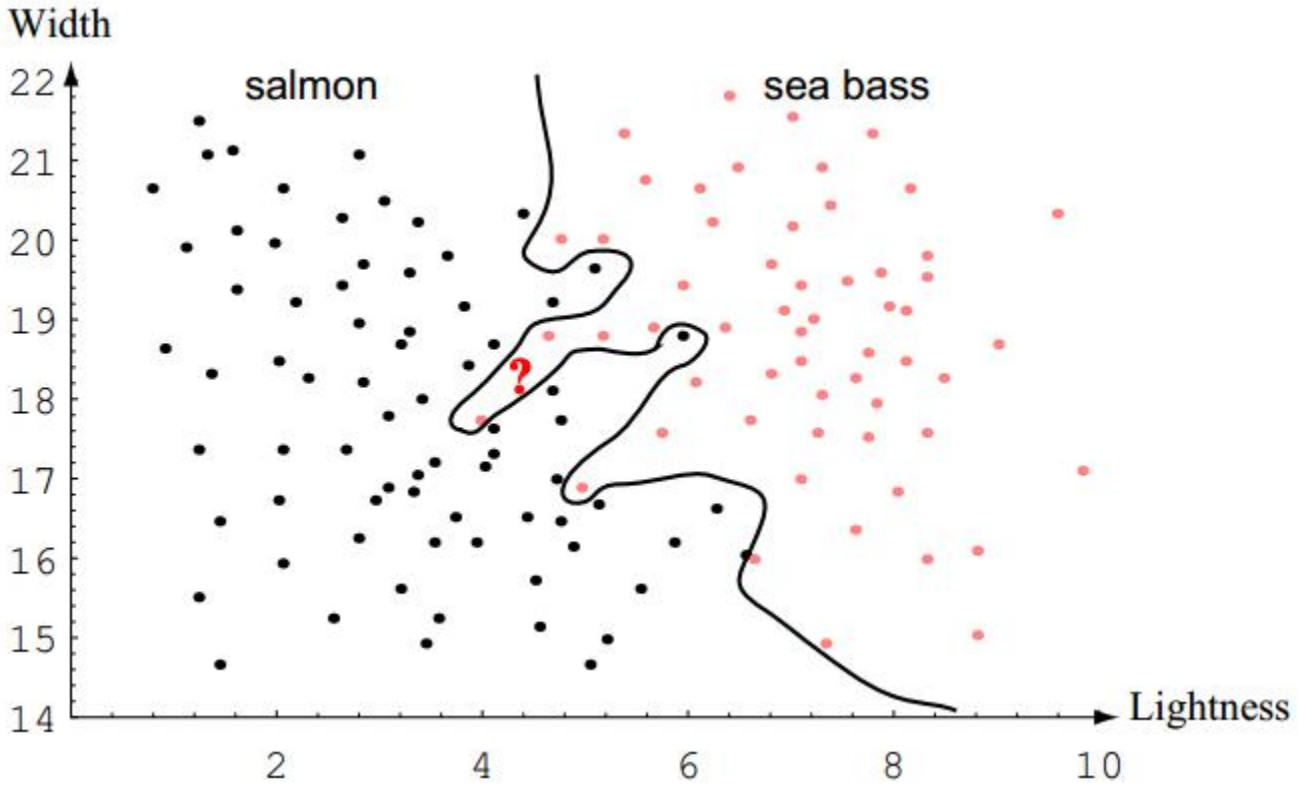
- Overfitting: a learning algorithm overfits the training data if it outputs a solution w when another solution w' exists such that:

$$\text{error}_{\text{train}}(w) < \text{error}_{\text{train}}(w')$$
$$\text{AND}$$
$$\text{error}_{\text{true}}(w') < \text{error}_{\text{true}}(w)$$

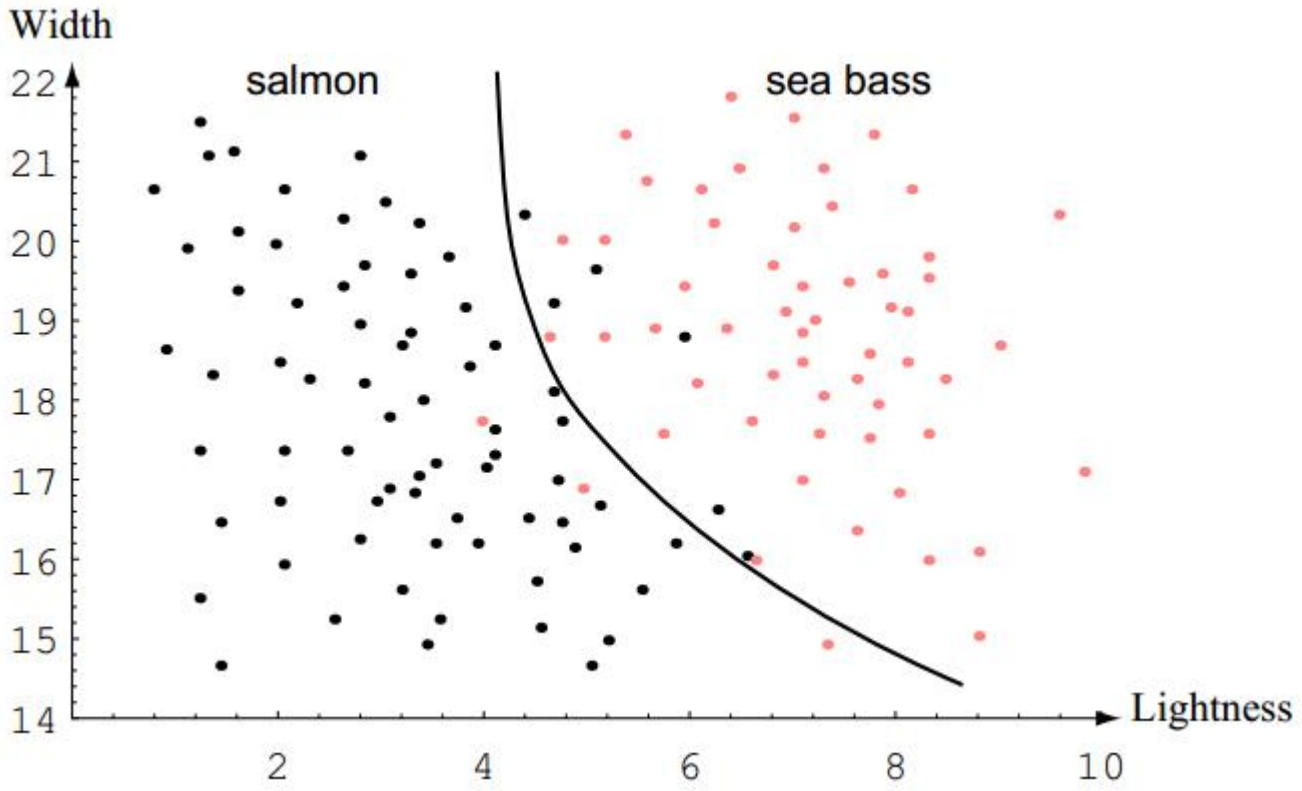
Example: Fish Classifier



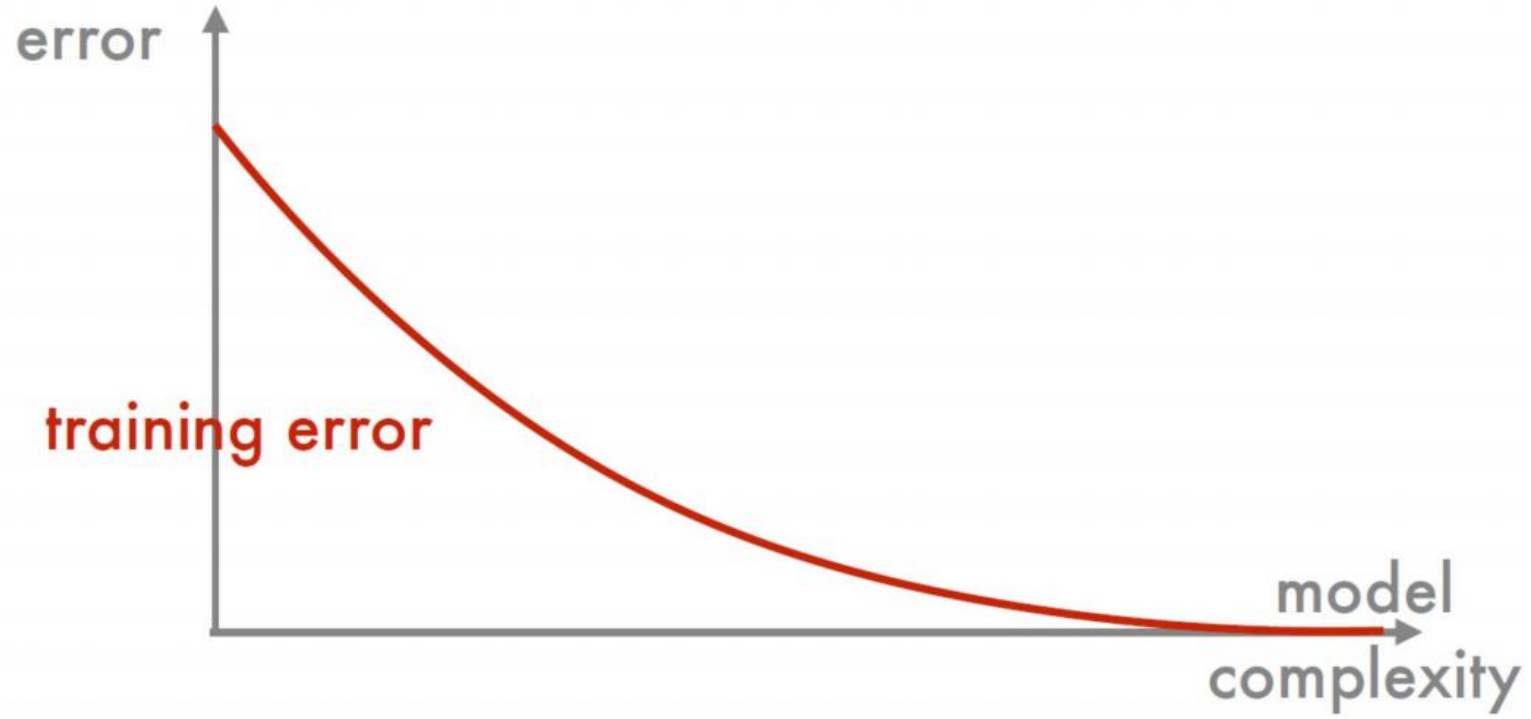
Minimum Training Error



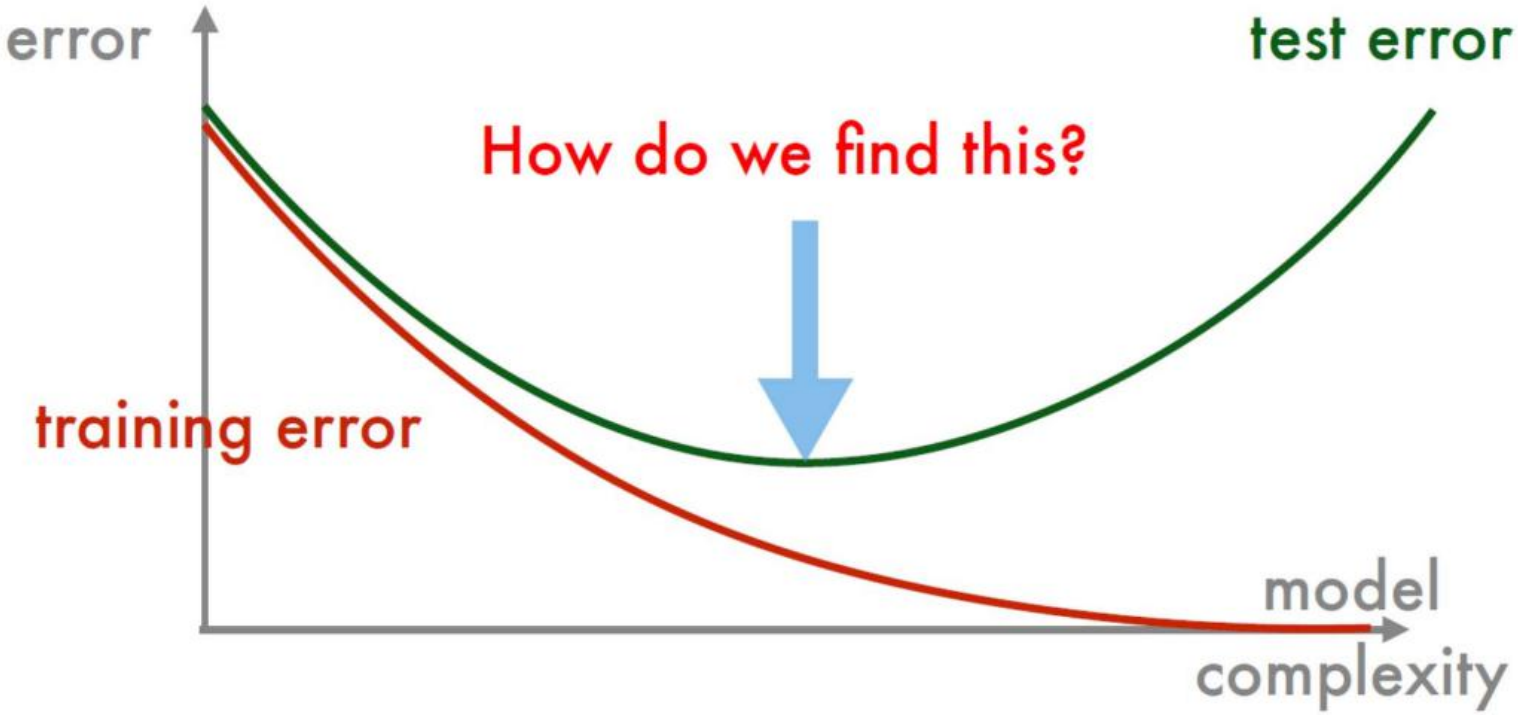
Final Decision Boundary



Typical Behavior

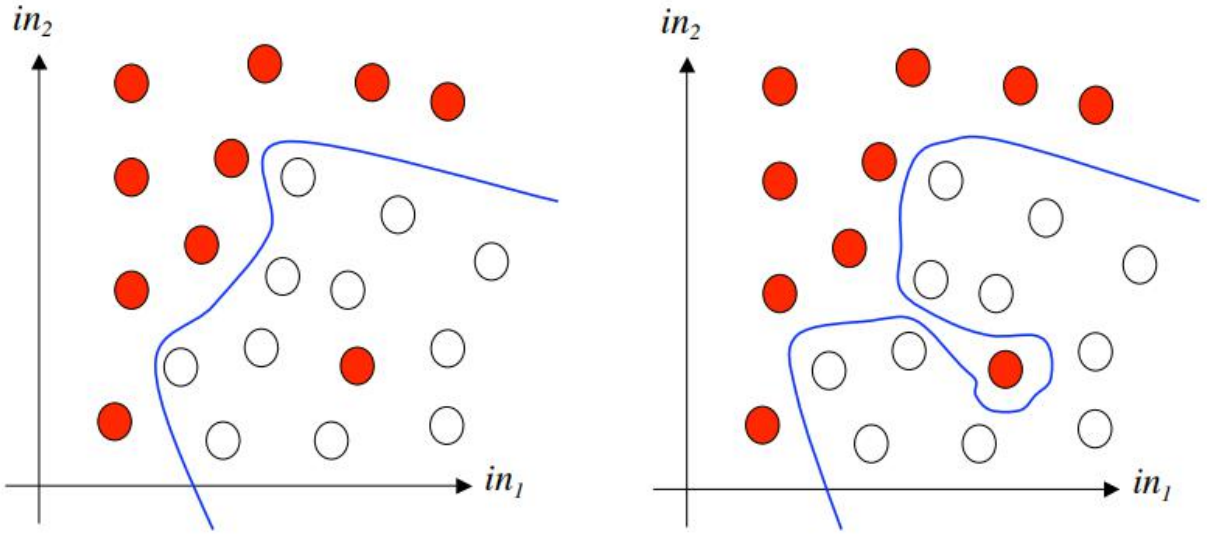


Typical Behavior



Typical Behavior

- The aim is to get a classification model to generalize to classify new inputs appropriately.
- If the training data is known to contain noise, we don't necessarily want the training data to be classified totally accurately, because that is likely to reduce the generalization ability.



Q & A