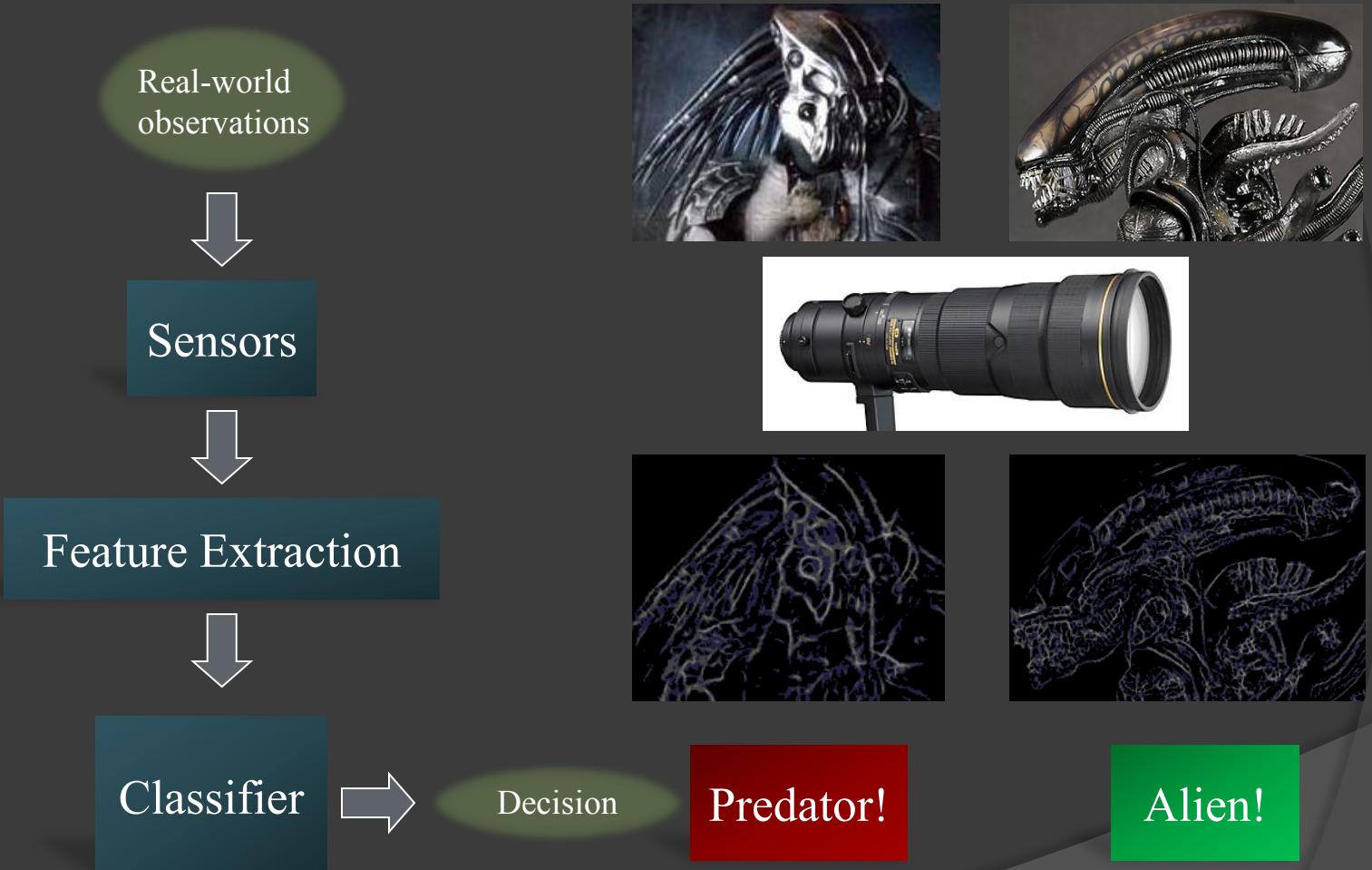


Prof. Marios Savvides

PATTERN RECOGNITION THEORY 18-794

Lecture 0: Introduction

Pattern Recognition Components



Pattern Recognition Components

Sensors

- Cameras, Shape, Temperature, Weight

Feature Extraction

- Domain Specific Knowledge
 - Gradients, DCT Coefficients, Fourier Coefficients, Phase Information etc.

Classifier

- Decision boundaries built using statistical knowledge

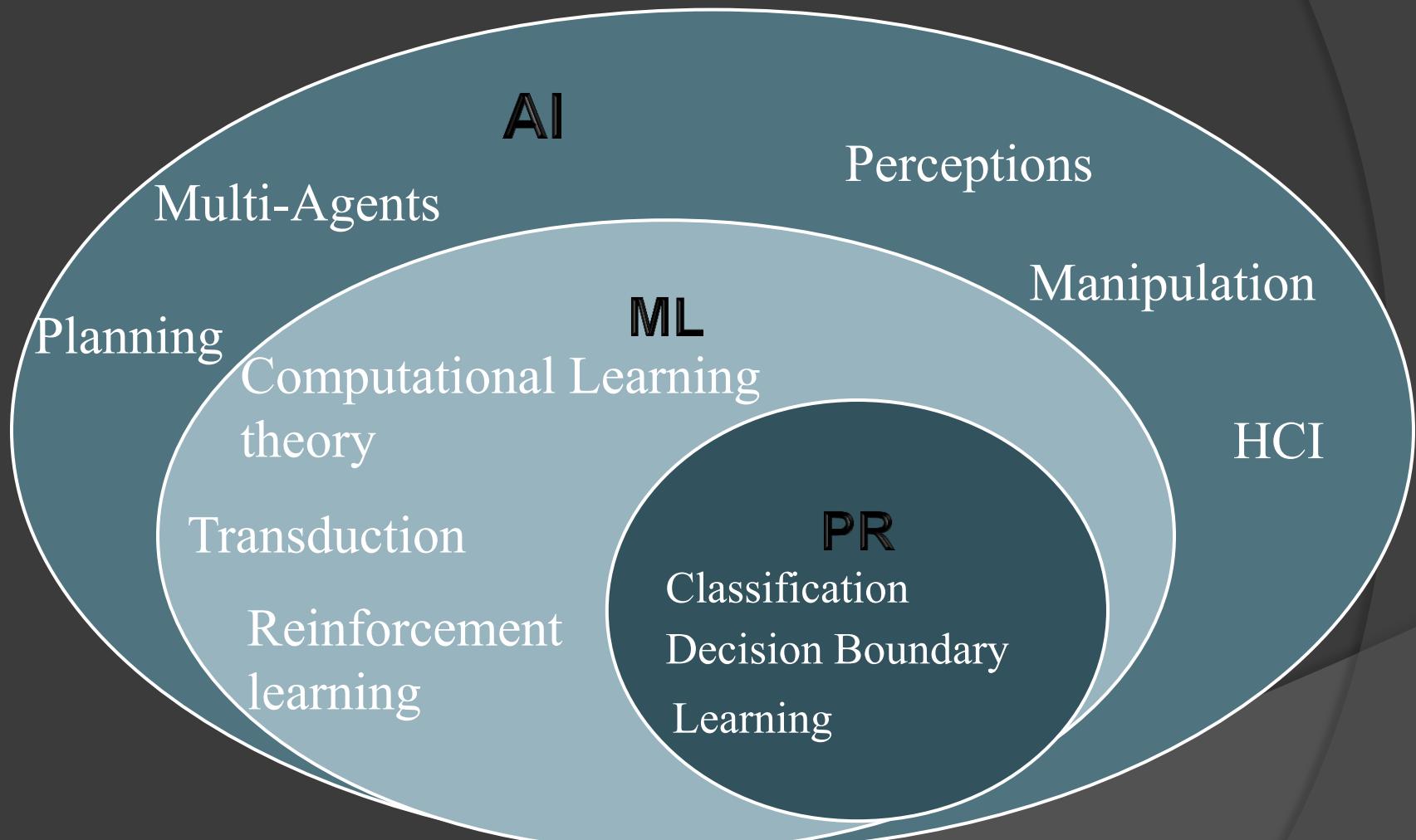
Topics to Be Covered

- Review of linear algebra and basic statistics
- Introduction to Bayesian Learning
 - Likelihood Ratio Test, Neyman-Pearson criterion, Minimax Criterion
- Parameter Estimation
 - MLE, MAP, GMM, EM
- Non-parametric methods and Clustering
 - Kernel density estimation, k-NNs
- Feature Extraction, Dimensionality Reduction
 - PCA, LDA, Linear Discriminant Functions
- Introducing sparsity in classifiers
 - Support Vector Machines
- Kernel tricks
 - Kernel PCA, Kernel LDA, Kernel SVM
- Artificial Neural Networks
 - Multilayer Perceptrons
- Correlation Filters
- Basic Image Processing Algorithms

Topics We Don't Cover

- Reinforcement Learning
- Markov Learning Process
- Decision Trees
- Graphical Models

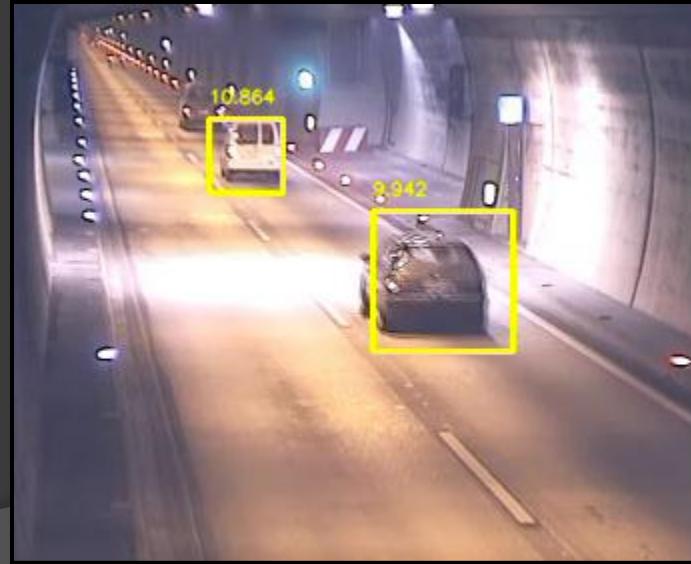
The Big Picture



A few applications...



Objection Detection: Cars



Object Detection: Pedestrians



Pedestrian Detection



Object Detection: Faces



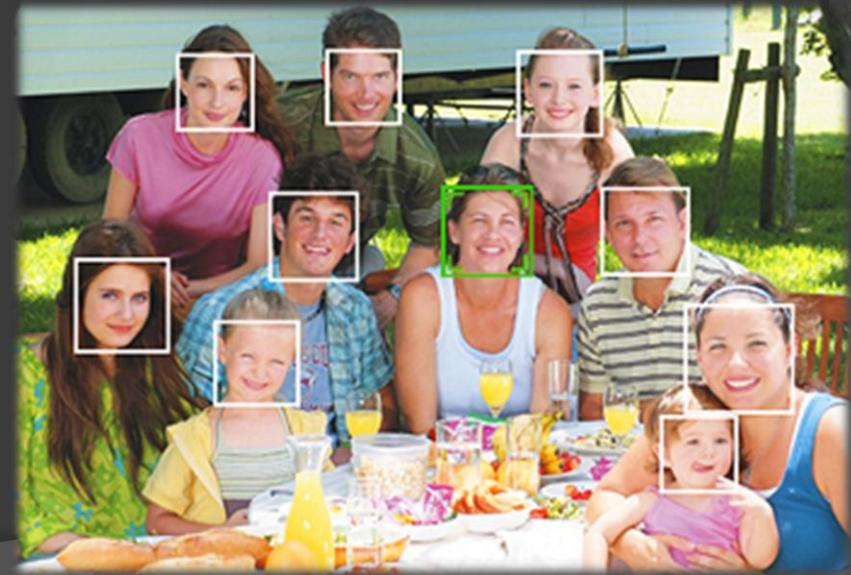
Face Detection and Alignment



Face Alignment



Face Detection: Commercial Applications



Postal Mail
90% of mail is fully-automatic
bar-coded
25 billion pieces of mail
annually in the US

Signature
Verifications
Bank checks

Consumer
Electronics
PDAs
Tablet PCs

Handwriting Recognition: OCR

Postal Mail

- 90% of mail is fully-automatic bar-coded
- 25 billion pieces of mail annually in the US

Signature Verifications

- Bank checks

Consumer Electronics

- PDAs
- Tablet PCs

Handwriting Recognition



1 9 6 8	Recognized as 1060
1 9 9 4	Recognized as 1394
1 9 9 5	Recognized as 1995
1 9 4 8	Recognized as 1940
1 9 9 0	Recognized as 1930
S C N D A	Recognized as SUNDA
S U N D A	Recognized as J?ZJM
S U N D A	Recognized as DCNDA
S U N D A	Recognized as SCAZA
S U N D A	Recognized as SCNJA
S U N D A	Recognized as SCNZA

iPhone Handwriting Recognition



Type

OR

Write!!

Handwriting Recognition



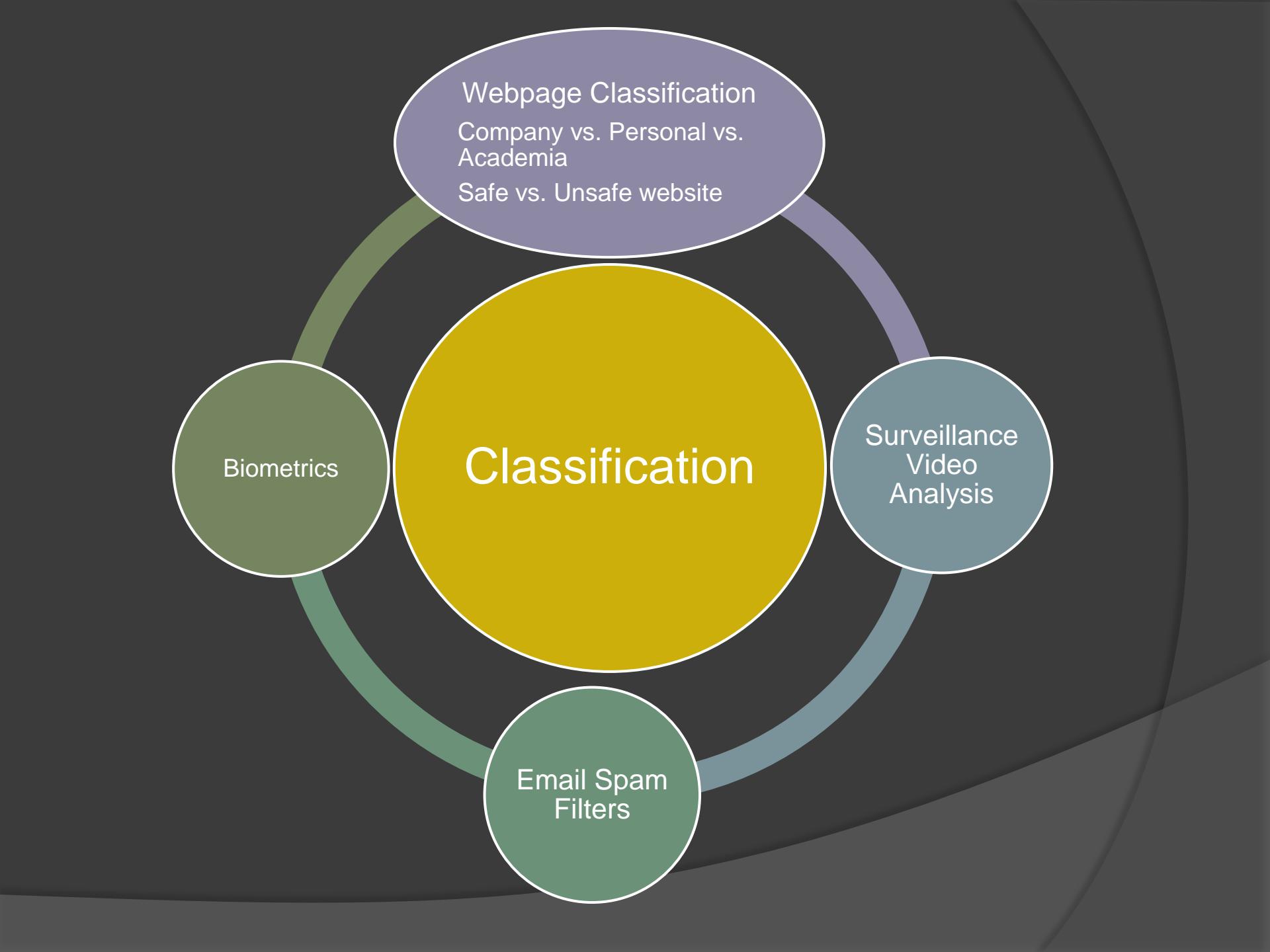
Text Recognition

Webpage Classification

- Company vs. Personal vs. Academia
- Safe vs. Unsafe website

Email Spam Filters

Surveillance Video Analysis



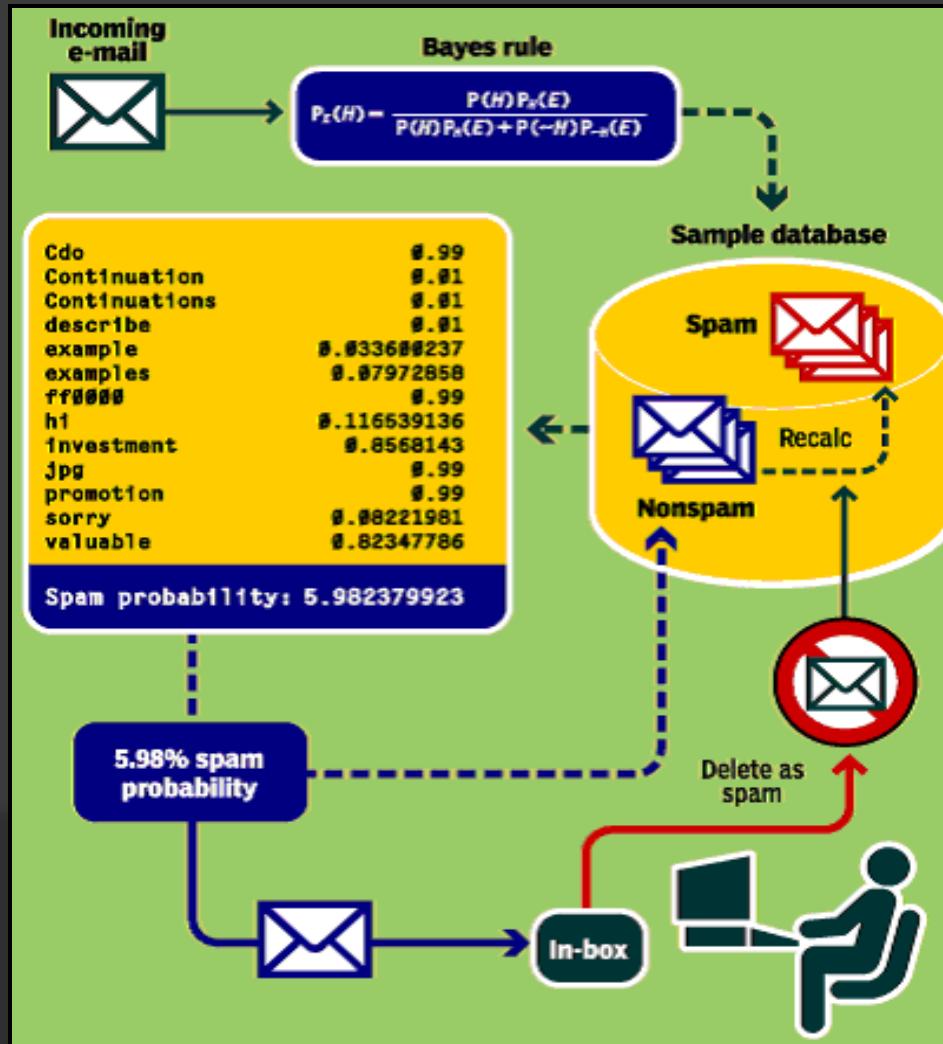
Webpage Classification
Company vs. Personal vs.
Academia
Safe vs. Unsafe website

Biometrics

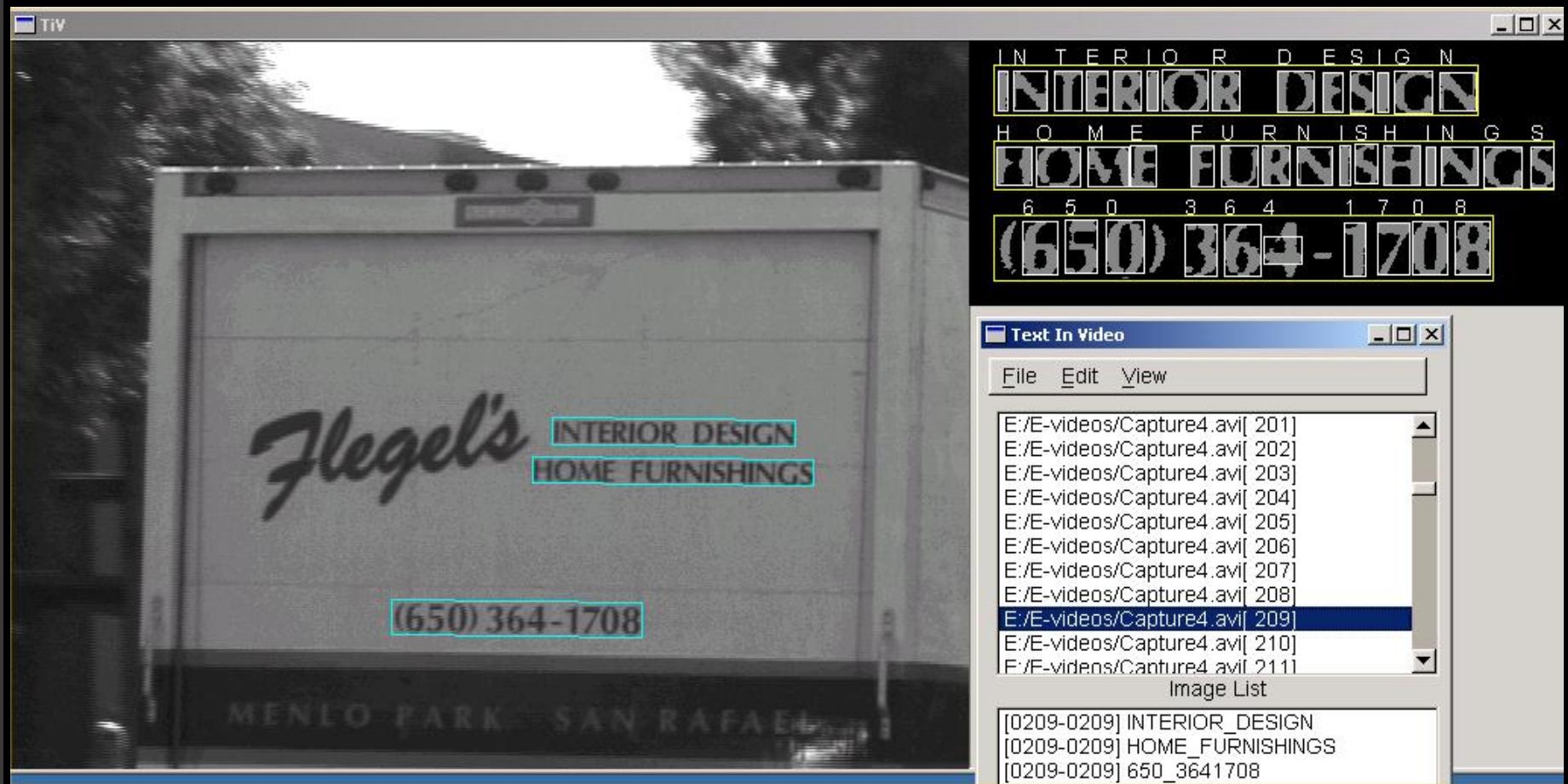
Surveillance
Video
Analysis

Email Spam
Filters

Spam Filters



Surveillance Video Analysis



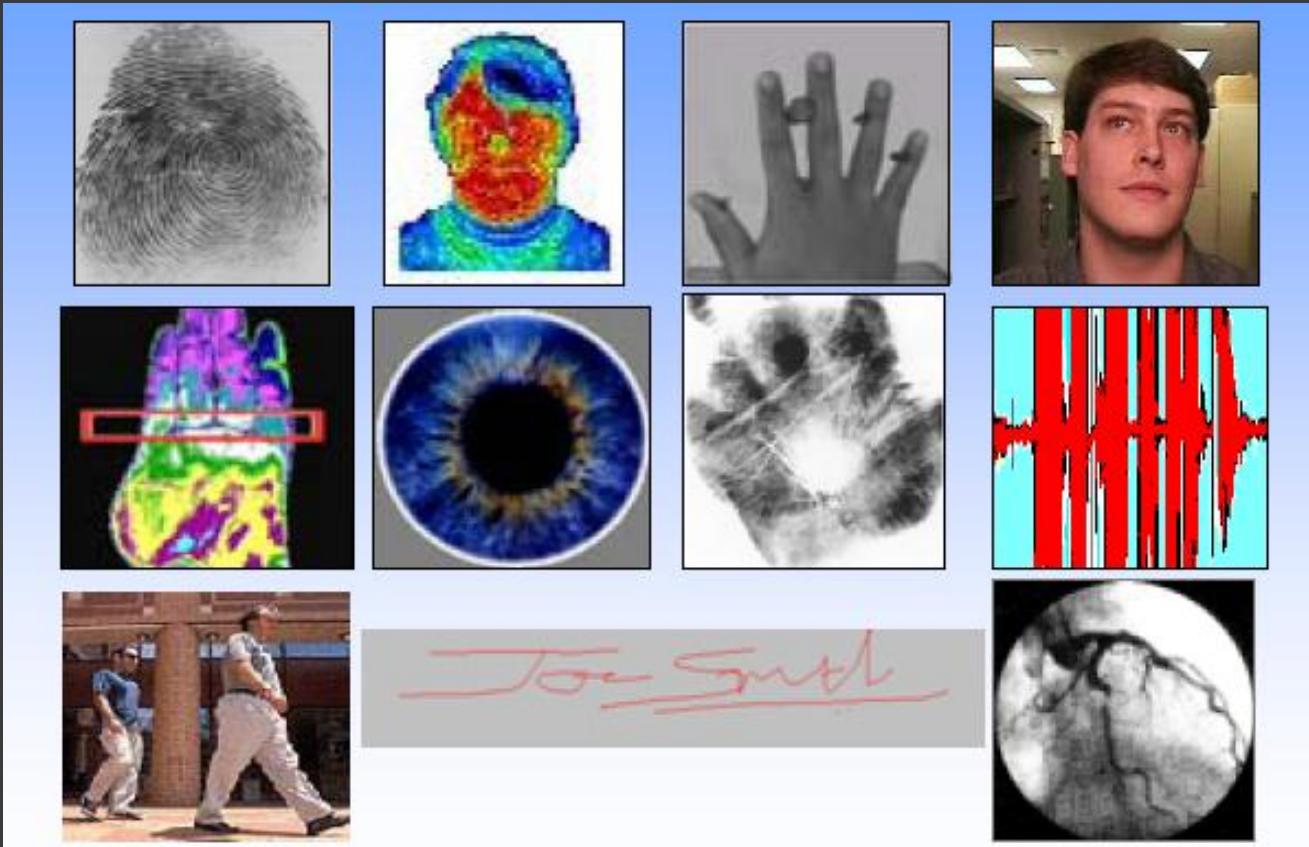
Object Classification



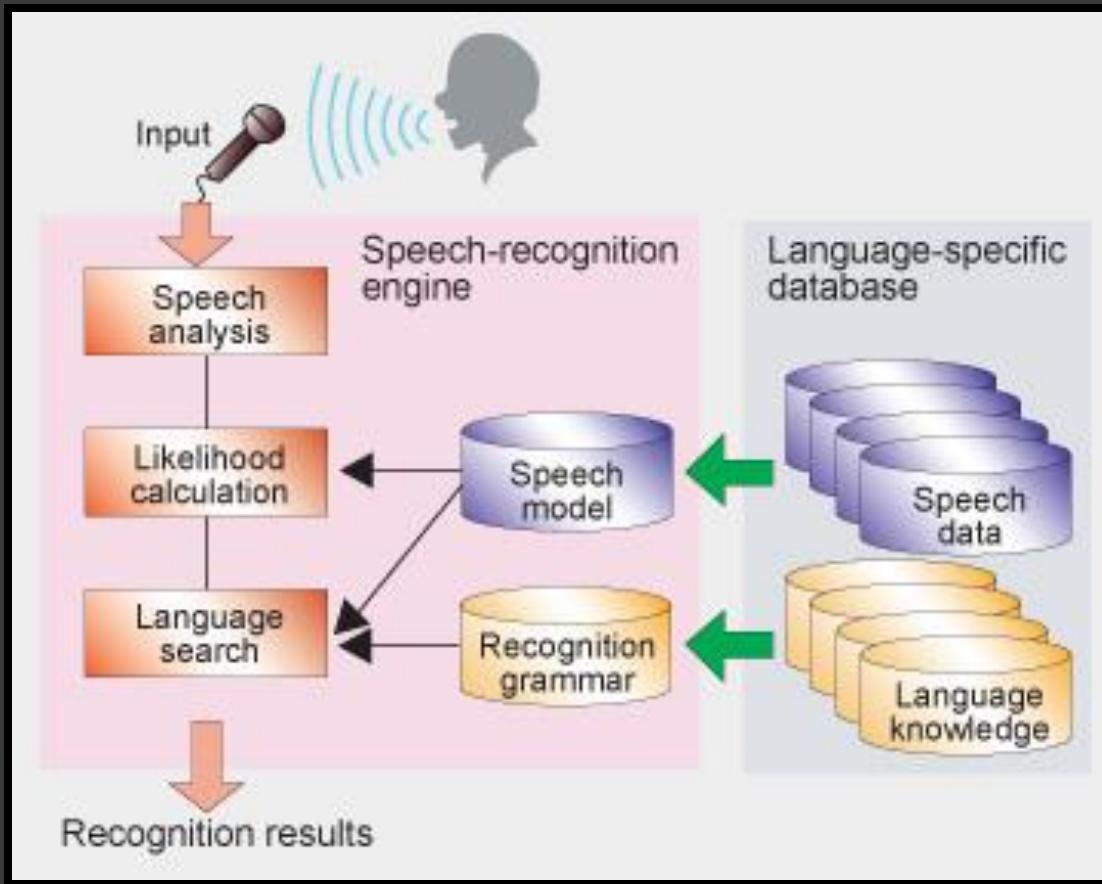
Google Goggles



Biometrics



Speech Recognition



Face Recognition



Face Recognition



Face Recognition 2



Facial Expression Recognition



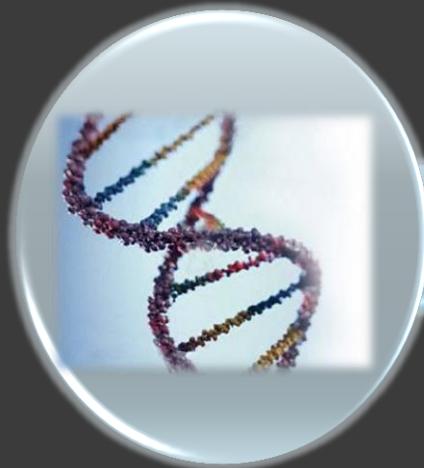
Minority Report Iris scan



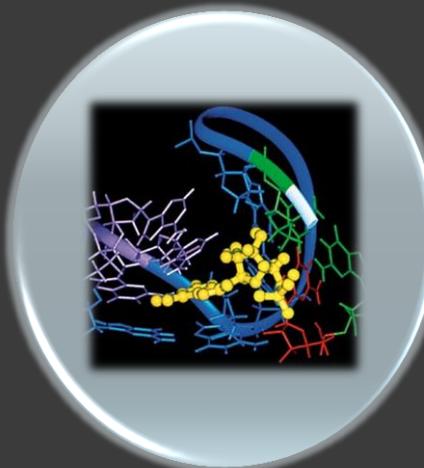
Iris Recognition



More Applications ...



DNA Sequencing



Cancer Detection

In Conclusion...

Humans can process different patterns easily.

However, it is a challenge to teach machines to recognize patterns automatically.

In this course, we explore many popular automatic pattern recognition and decision making algorithms

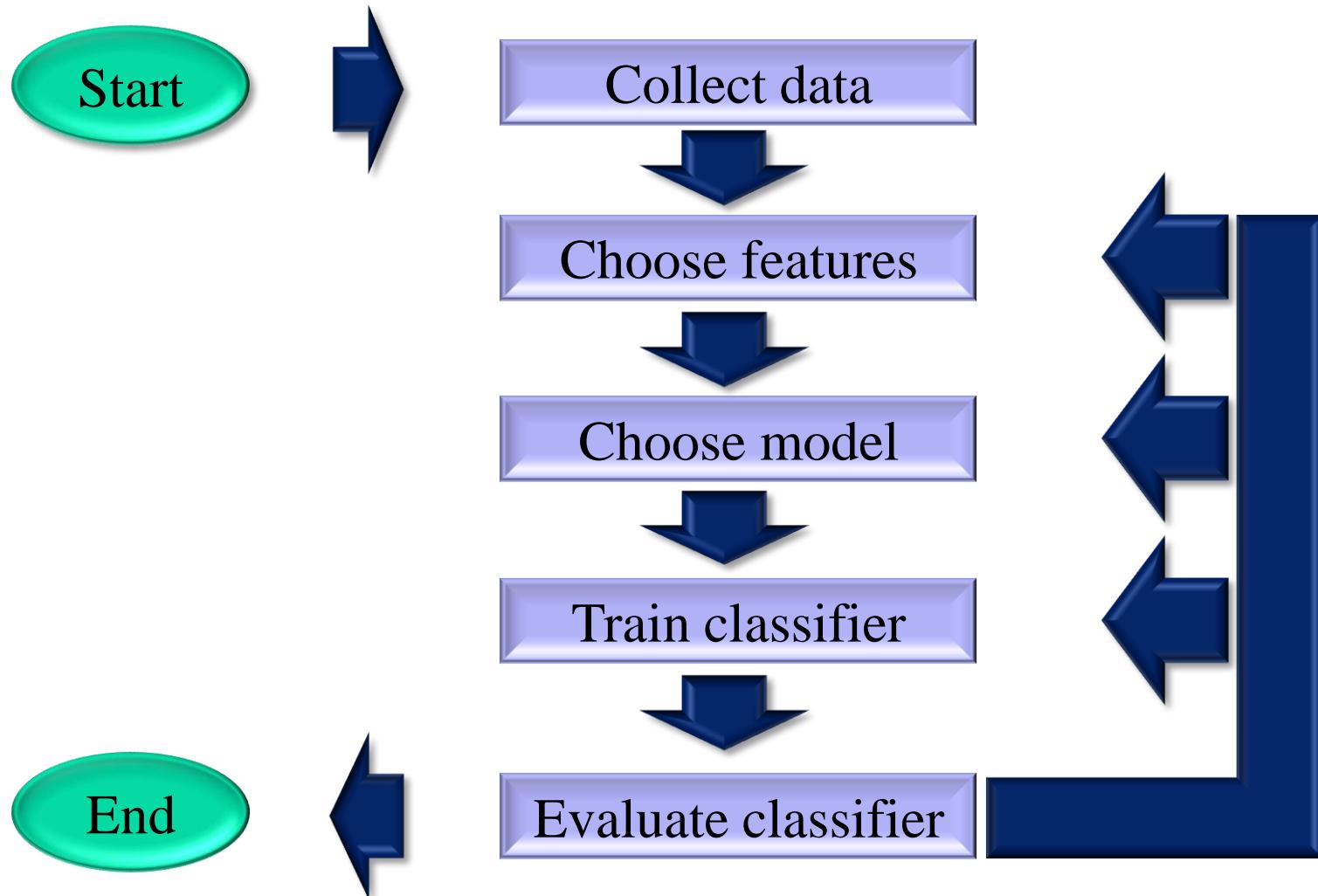
Prof. Marios Savvides

Pattern Recognition Theory

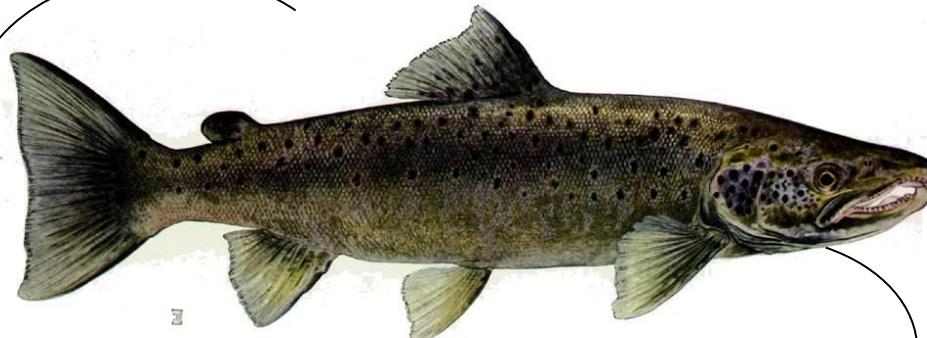
Lecture 1: Decision Theory I

All graphics from Pattern Classification, Duda, Hart and Stork, Copyright © John Wiley and Sons, 2001

Classifier Design Cycle



Our First Classification Problem ...

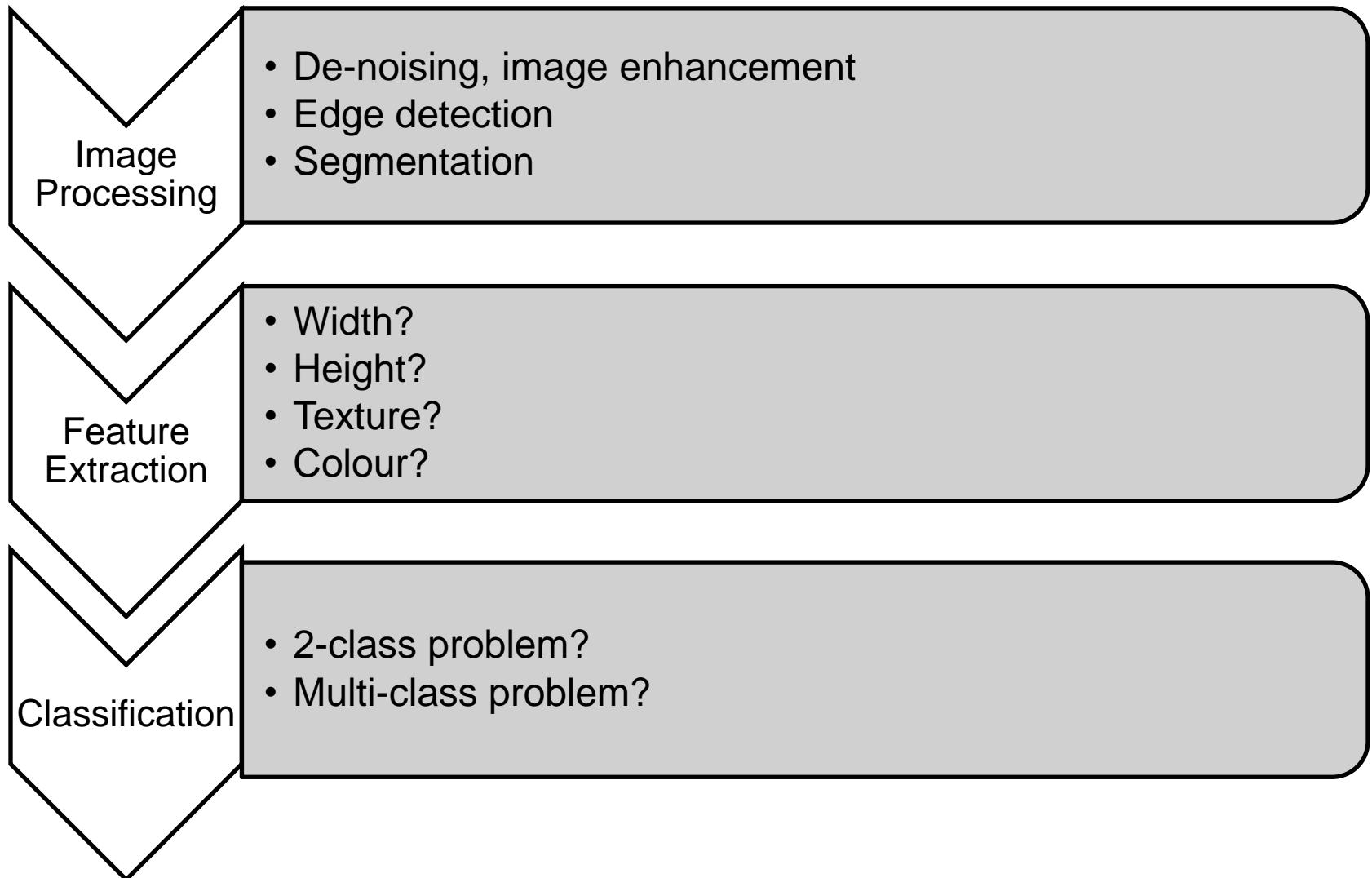


Salmon



Bass

Steps to Good Classification



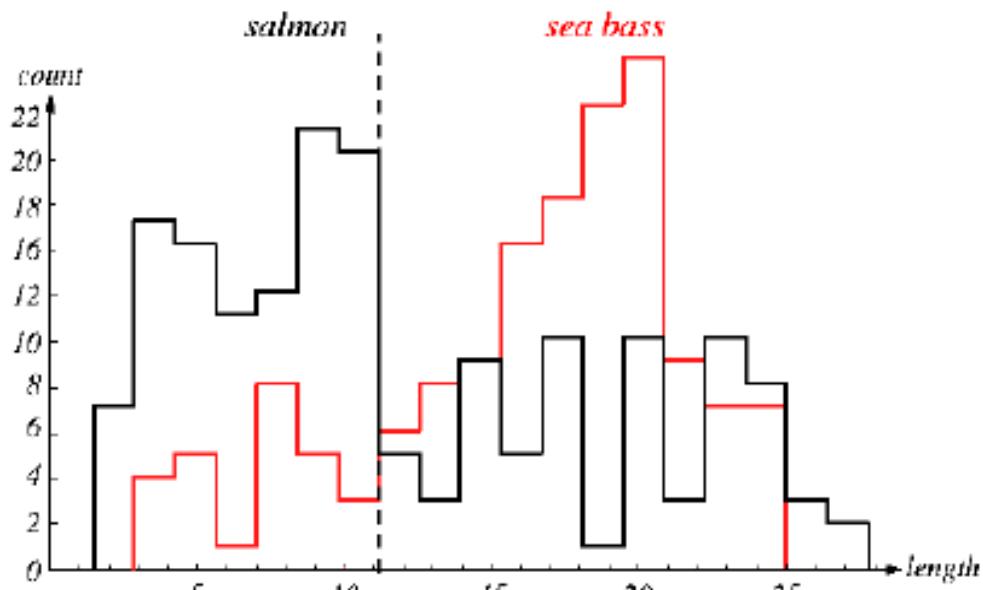
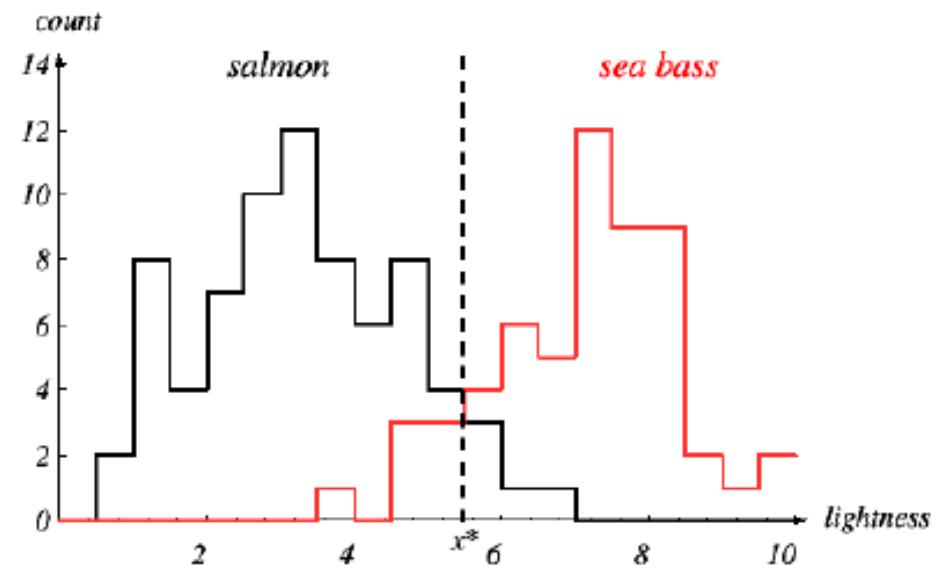
What's a Feature Space?...

- What's a feature?
 - A feature is a distinctive characteristic or quality of the object
- Combine more than one feature to get → D-dim **feature vector**
- The D-dimensional space thus defined → **feature space**

$$\mathbf{X} = \begin{bmatrix} \text{Length} \\ \text{Lightness} \\ \text{Texture} \end{bmatrix}$$

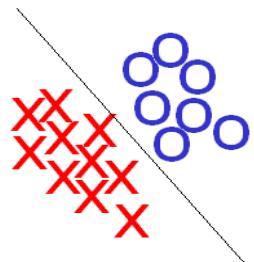
Here D = 3

How Features Help

 $x_1 = \text{length}$  $x_2 = \text{lightness}$

Properties of Features

- The quality of the feature vector is related to its ability to discriminate samples from different classes

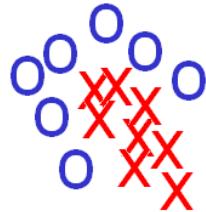


Good Features – Linearly Separable

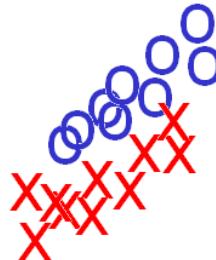


Bad Features – Not Discriminating

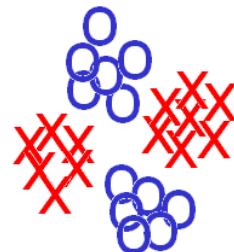
- Other Properties



Non-linearly Separable



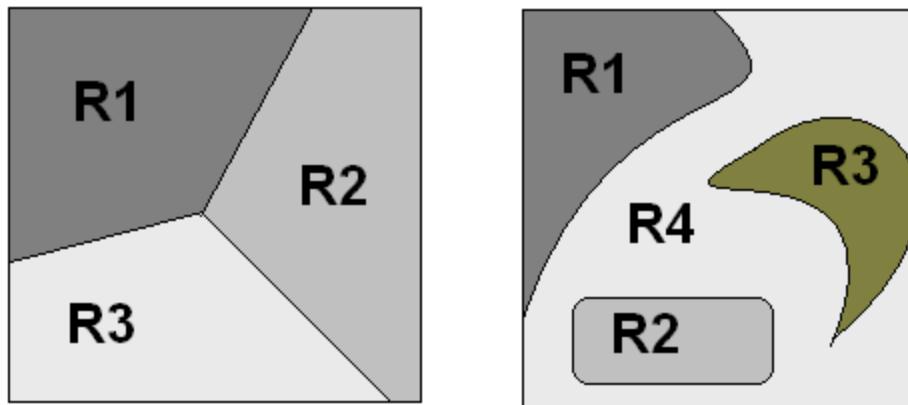
Positively Correlated



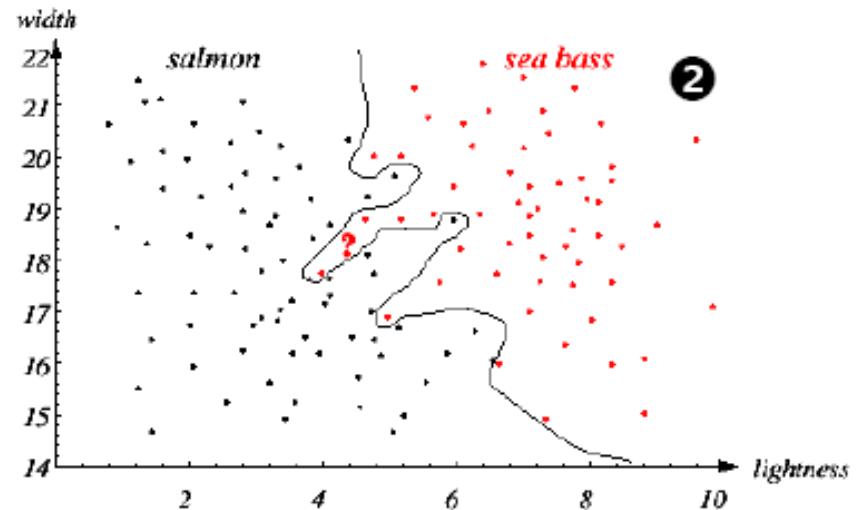
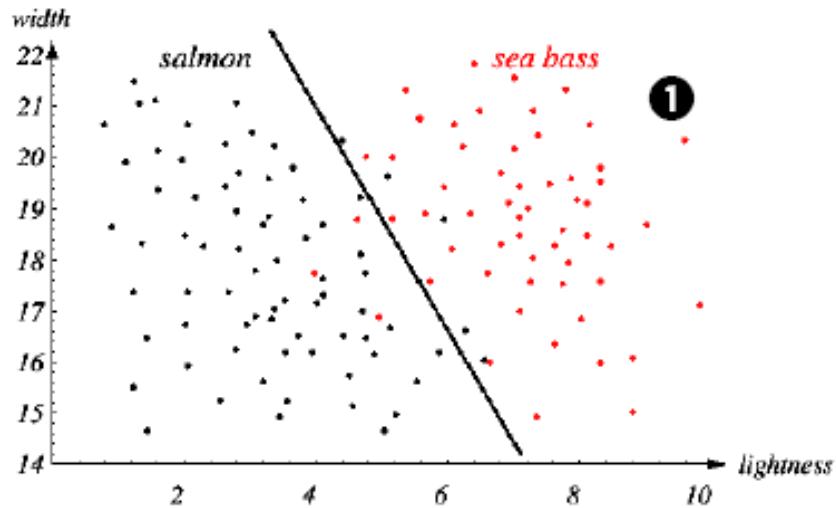
Multi-modal

Classifiers

- A classifier is designed to partition the feature space into class-corresponding **decision regions**
 - What criterion do I have to minimize?
 - What cost function to use?
 - Are all errors equals?
- Borders between the decision regions are called **decision boundaries**

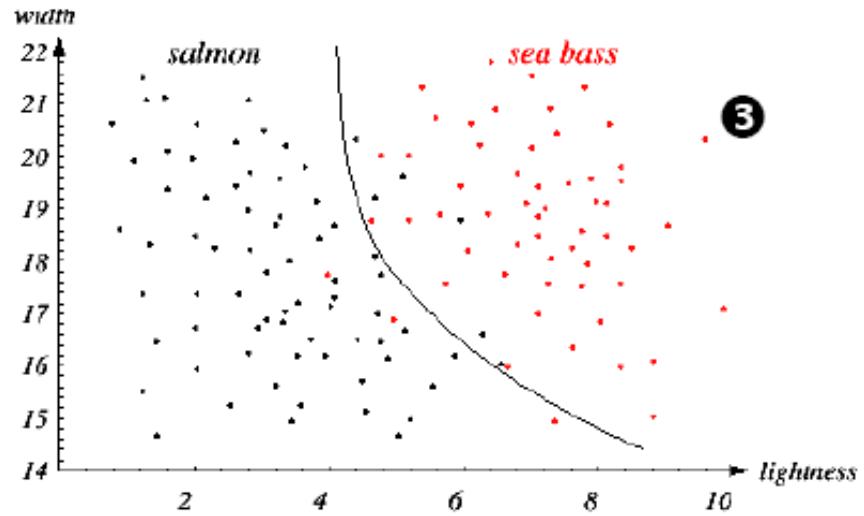


Decision Boundaries



Which decision boundary?

- 1: Linear boundary with significant training error
- 2: Nonlinear complex boundary with zero training error
- 3: Simple non-linear boundary with small training error

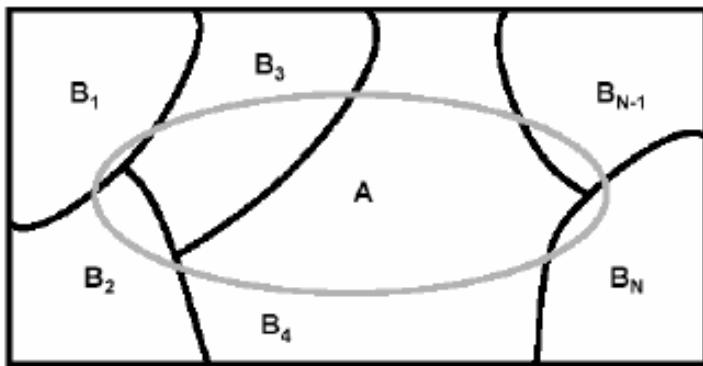


Our First Classifier

- Minimum error classifier
 - Goal is to design a classifier to partition the decision space to minimize the number of errors (misclassifications)
 - During classifier design, we assume that the underlying probabilistic structure (class dependence) is known
 - i.e. the probability $f(\text{feature vector } \mathbf{x} \text{ given class } \omega)$ is known $\rightarrow f(\mathbf{x} | \omega)$

A Short Detour: Bayes Rule

Given that event 'A' has occurred, what is the probability that one of the event 'B's occur?



Rev. Thomas Bayes
(1702-1761)

$$P(B_i | A) = \frac{P(A \cap B_i)}{P(A)} = \frac{P(A | B_i)P(B_i)}{\sum_j P(A | B_j)P(B_j)}$$

A Few Definitions...

Likelihood: The conditional probability of observing a feature value of x given that the correct class is ω_i

Posterior Probability: The conditional probability of correct class being ω_i given that feature value x has been observed

Prior Probability: The probability of class ω_i
 $P(\omega_1) + P(\omega_2) + \dots + P(\omega_c) = 1$

$$P(\omega_i | x) = \frac{P(x \cap \omega_i)}{P(x)} = \frac{P(x | \omega_i)P(\omega_i)}{\sum_{k=1}^c P(x | \omega_k)P(\omega_k)}$$

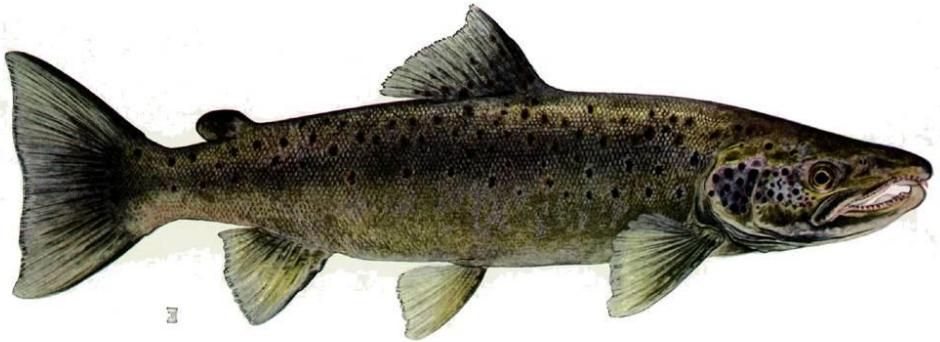
Evidence : The total probability of observing the feature value of x

Posterior Probability For Classification

$$P(\omega_i | x) = \frac{P(x \cap \omega_i)}{P(x)} = \frac{P(x | \omega_i)P(\omega_i)}{\sum_{k=1}^c P(x | \omega_k)P(\omega_k)}$$

- Bayes Classifiers decide on the class that has the **largest posterior probability**
- It can be shown that Bayes classifiers are statistically the best classifiers one can construct i.e. they are minimum error classifiers (optimal)

And Now, Back to our Fish...

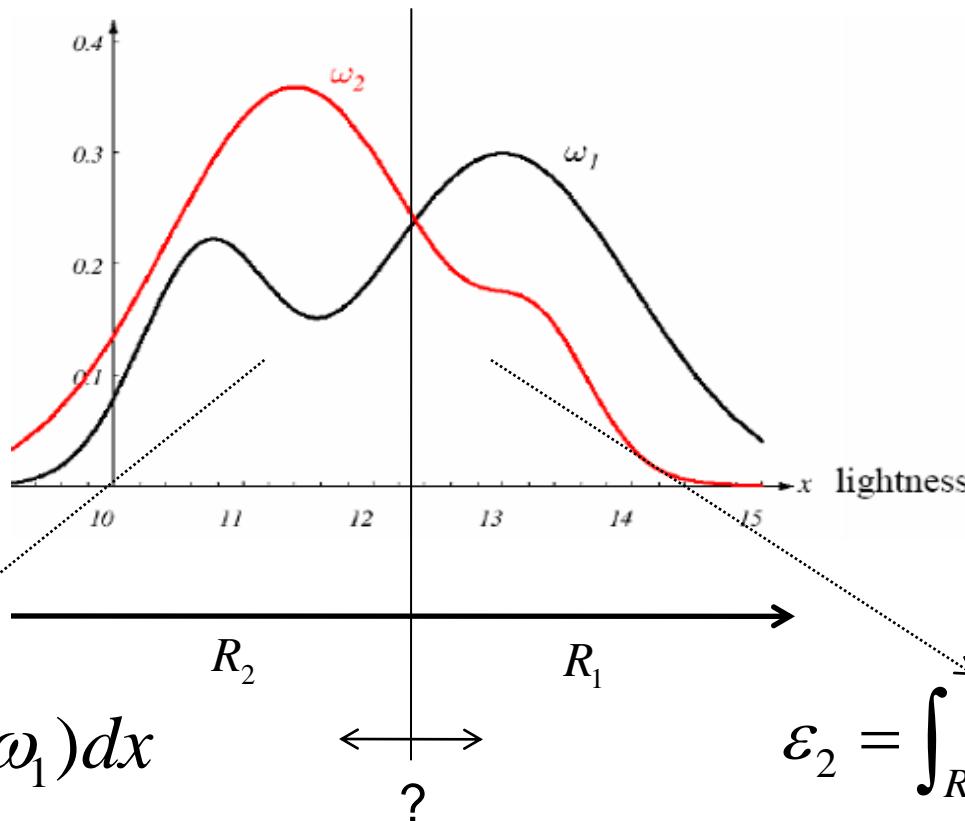


- We will build a **Bayes classifier** i.e. a minimum error classifier, to distinguish salmon samples from bass samples
- For this classifier, consider only one of the **features** i.e. the **lightness** of the fish

Minimum Probability of Error

$\varepsilon = P(error \mid class)$ probability of assigning x to the wrong class ω

$P_e = P(\omega_1)\varepsilon_1 + P(\omega_2)\varepsilon_2$ total probability of error



Where to put the threshold?

Minimum Error Classifier

We want to minimize P_e

$$P_e = P(\omega_1)\varepsilon_1 + P(\omega_2)\varepsilon_2$$

$$= P(\omega_1) \int_{R_2} f(x | \omega_1) dx + P(\omega_2) \int_{R_1} f(x | \omega_2) dx$$

$$= P(\omega_1) \{1 - \int_{R_1} f(x | \omega_1) dx\} + P(\omega_2) \int_{R_1} f(x | \omega_2) dx$$

$$= P(\omega_1) + \{P(\omega_2) \int_{R_1} f(x | \omega_2) dx - P(\omega_1) \int_{R_1} f(x | \omega_1) dx\}$$

$$= P(\omega_1) + \{\int_{R_1} P(\omega_2)f(x | \omega_2) - P(\omega_1)f(x | \omega_1)\} dx$$

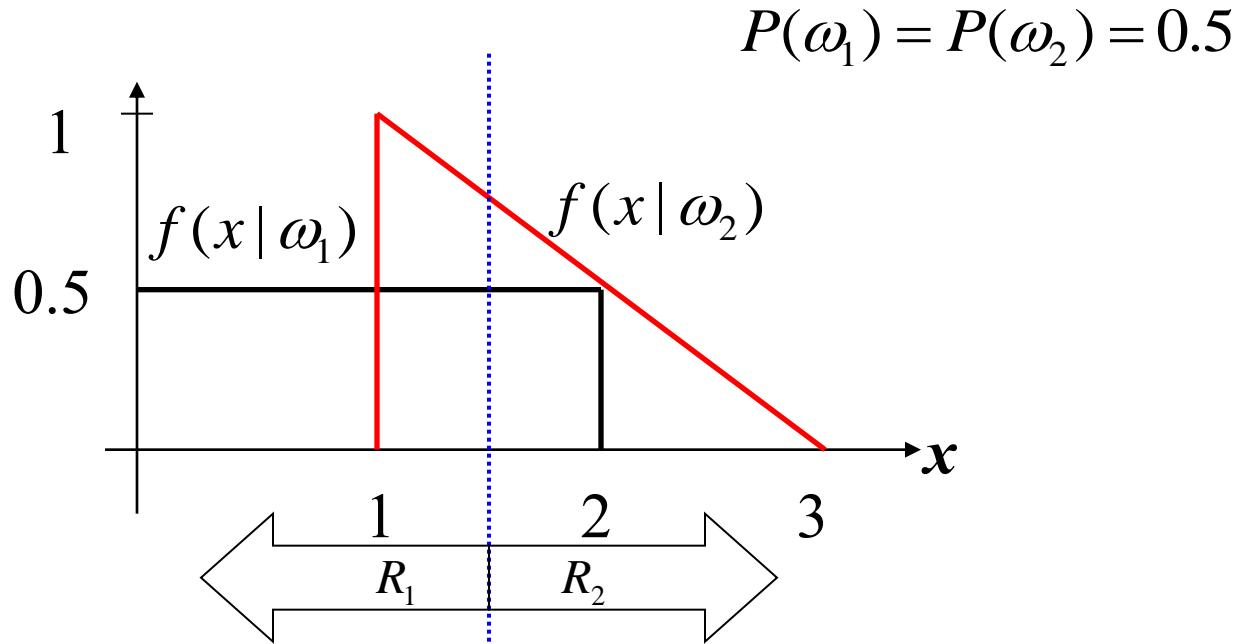
as small as possible – this difference is negative within R_1

i.e., if $P(\omega_2)f(x | \omega_2) - P(\omega_1)f(x | \omega_1) < 0 \rightarrow \omega_1$

Minimum Error Classifier

$$\left\{ P(\omega_1)f(x|\omega_1) - P(\omega_2)f(x|\omega_2) \right\} \begin{matrix} \omega_1 \\ > \\ \omega_2 \end{matrix} 0$$

Example : 2-Class Classifier Error

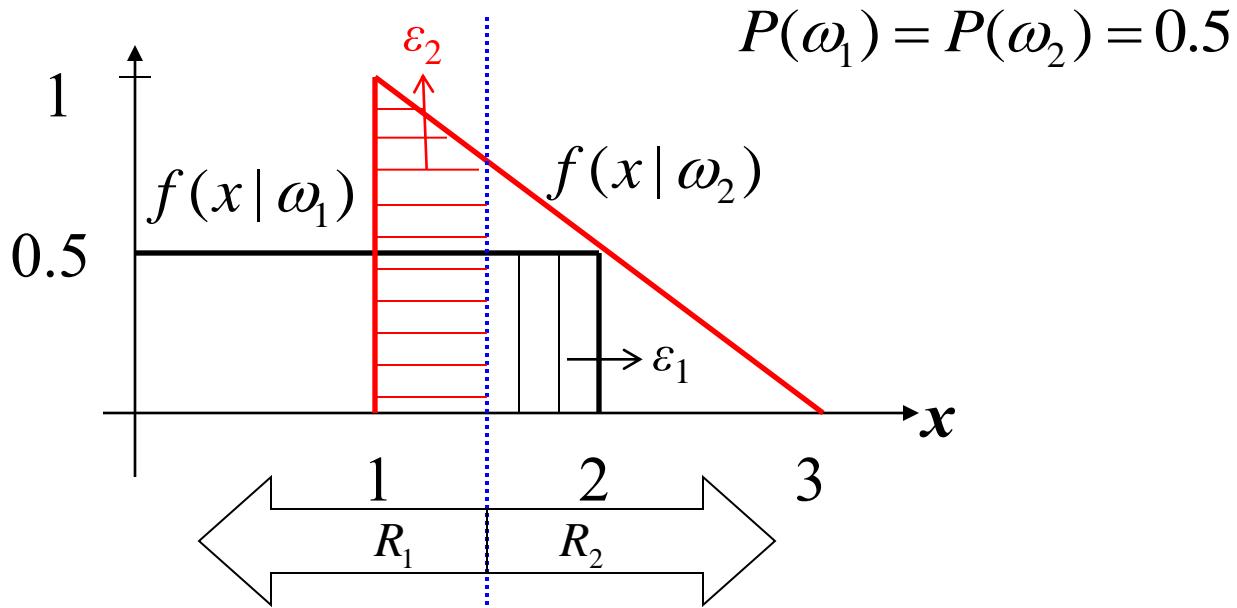


The Classifier assigns:

$x \leq 1.5$ to class 1

$x > 1.5$ to class 2

Example : 2-Class Classifier Error



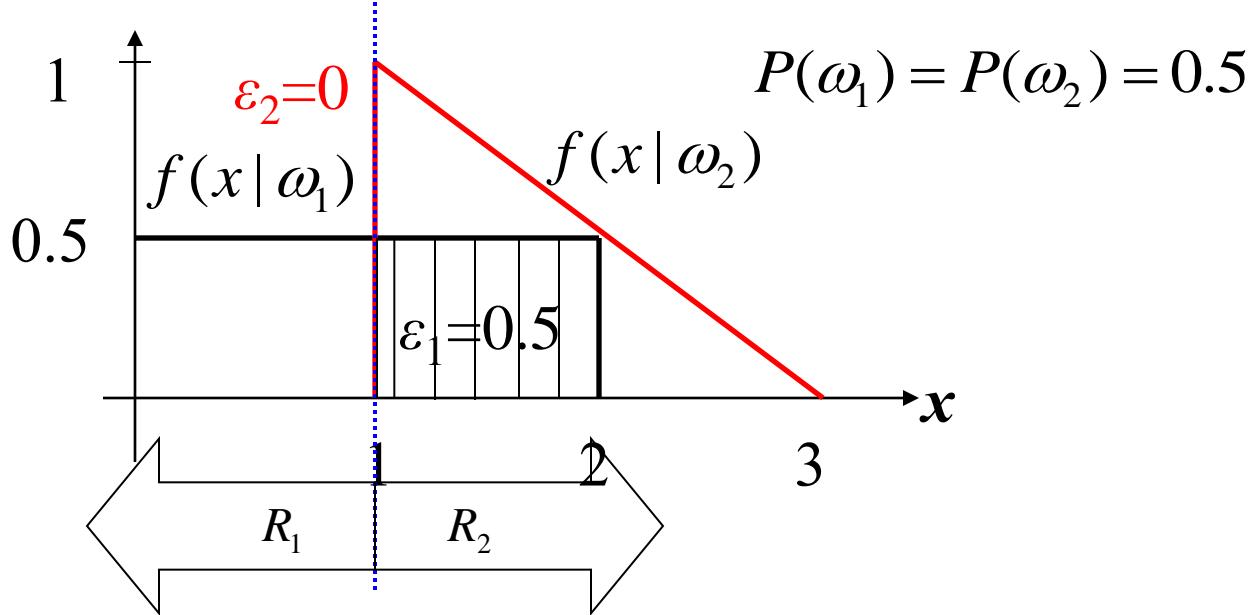
$$\varepsilon_1 = \int_{R_2} f(x | \omega_1) dx = \int_{1.5}^3 f(x | \omega_1) dx = \int_{1.5}^3 (1/2) dx = 1/4$$

$$\varepsilon_2 = \int_{R_1} f(x | \omega_2) dx = \int_0^{1.5} f(x | \omega_2) dx = \int_1^{1.5} \left(1 - \frac{1}{2}(x-1)\right) dx = \frac{3}{4} - \frac{5}{16} = \frac{7}{16}$$

$$P_e = P(\omega_1)\varepsilon_1 + P(\omega_2)\varepsilon_2 = (0.5)\frac{1}{4} + (0.5)\frac{7}{16} = 0.343$$

Is this optimal ?

Example : 2-Class Classifier Error



Minimum Error Classifier will give a smaller error!!

$$P_e = P(\omega_1)\varepsilon_1 + P(\omega_2)\varepsilon_2 = (0.5)(0.5) + 0 = 0.25$$

Likelihood Ratio

$$\{P(\omega_1)f(x|\omega_1) - P(\omega_2)f(x|\omega_2)\} \stackrel{\omega_1}{\gtrless} \stackrel{\omega_2}{\gtrless} 0$$

$$l(x) = \frac{f(x|\omega_1)}{f(x|\omega_2)} \stackrel{\omega_1}{\gtrless} \stackrel{\omega_2}{\gtrless} \frac{P(\omega_2)}{P(\omega_1)} = T$$

Likelihood ratio

Ratio of a priori probabilities

Log

monotonically increasing both sides
without affecting decision rule

Log Likelihood ratio

$$\ln(l(x)) = \ln\left(\frac{f(x|\omega_1)}{f(x|\omega_2)}\right) \stackrel{\omega_1}{\gtrless} \stackrel{\omega_2}{\gtrless} \ln\left(\frac{P(\omega_2)}{P(\omega_1)}\right) = \ln(T)$$

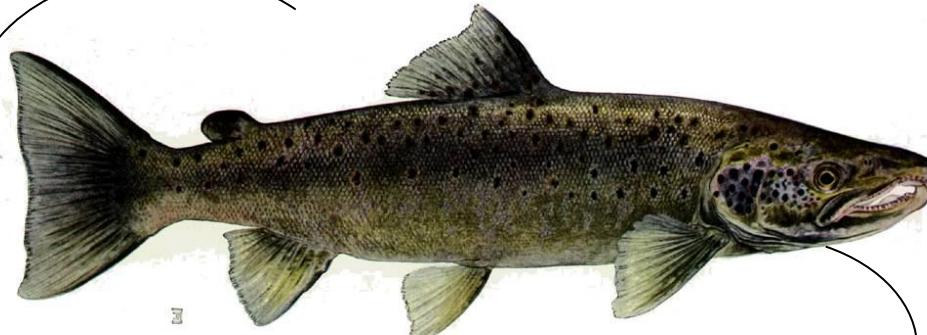
Prof. Marios Savvides

Pattern Recognition Theory

Lecture 2: Decision Theory II

All graphics from Pattern Classification, Duda, Hart and Stork, Copyright © John Wiley and Sons, 2001

Our First Classification Problem ...

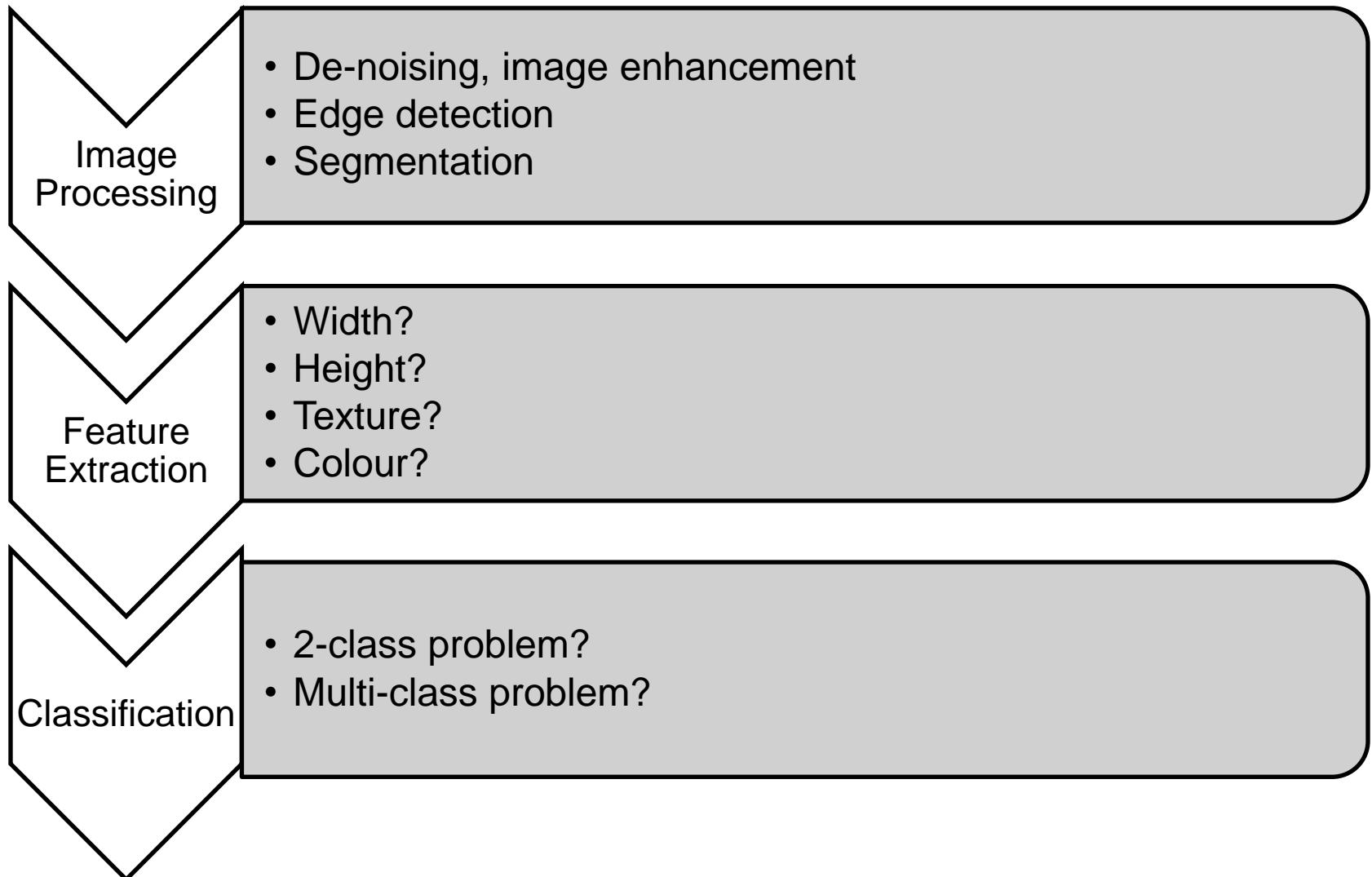


Salmon



Bass

Steps to Good Classification



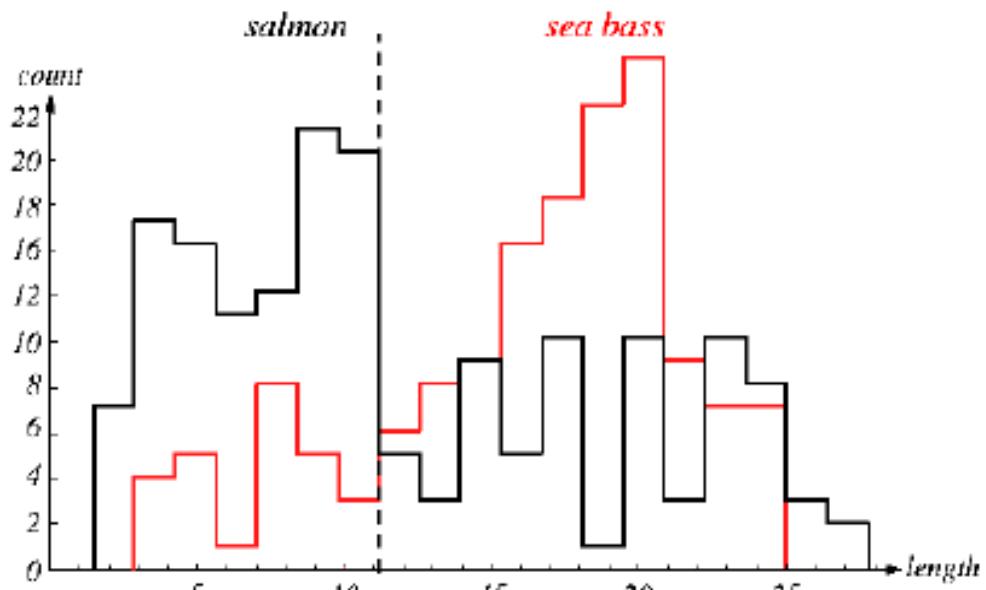
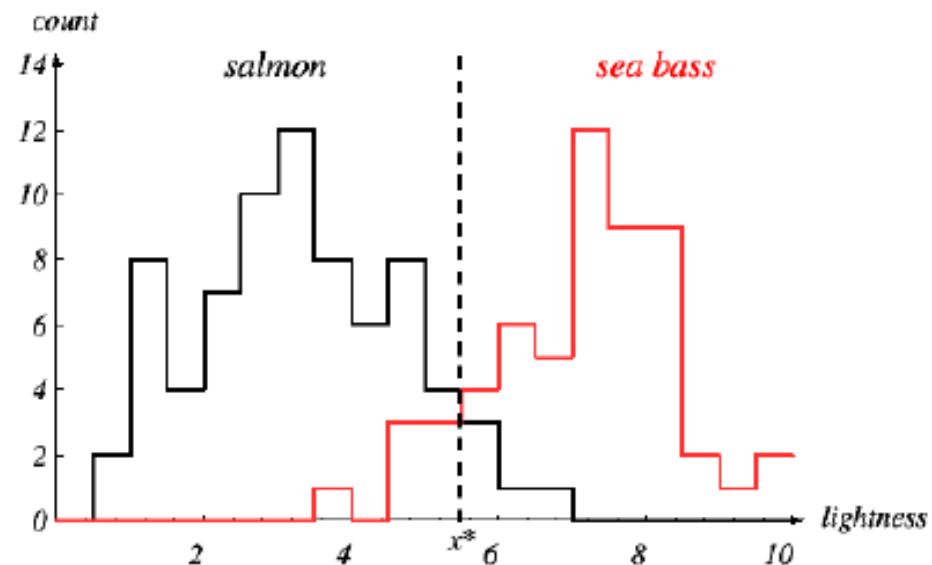
What's a Feature Space?...

- What's a feature?
 - A feature is a distinctive characteristic or quality of the object
- Combine more than one feature to get → D-dim **feature vector**
- The D-dimensional space thus defined → **feature space**

$$\mathbf{X} = \begin{bmatrix} \text{Length} \\ \text{Lightness} \\ \text{Texture} \end{bmatrix}$$

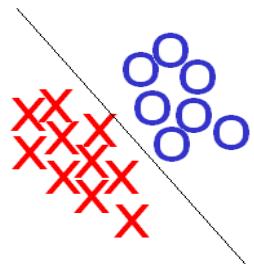
Here D = 3

How Features Help

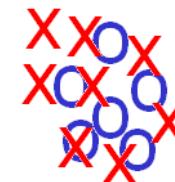
 $x_1 = \text{length}$  $x_2 = \text{lightness}$

Properties of Features

- The quality of the feature vector is related to its ability to discriminate samples from different classes

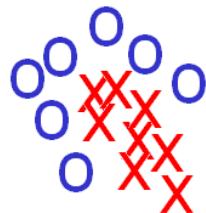


Good Features – Linearly Separable

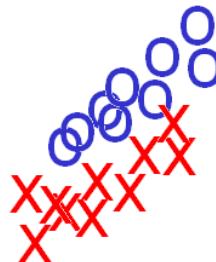


Bad Features – Not Discriminating

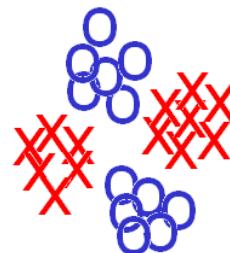
- Other Properties



Non-linearly Separable



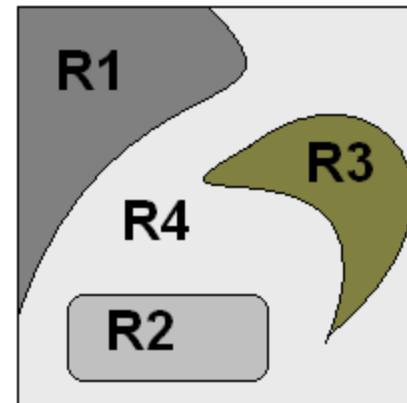
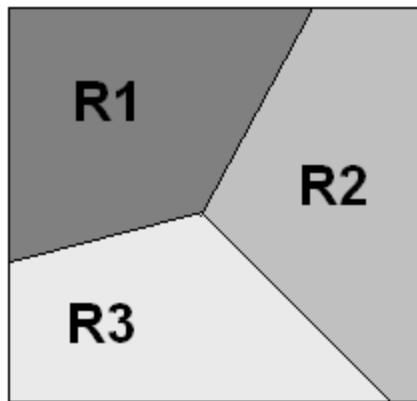
Positively Correlated



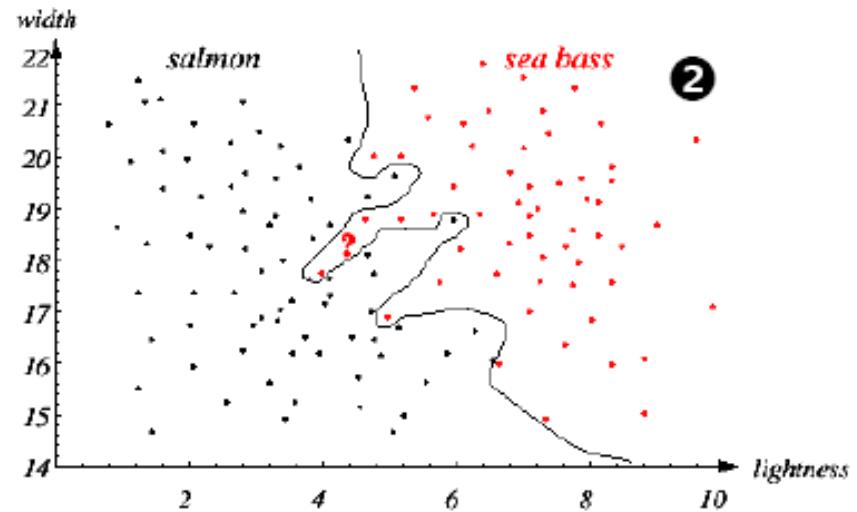
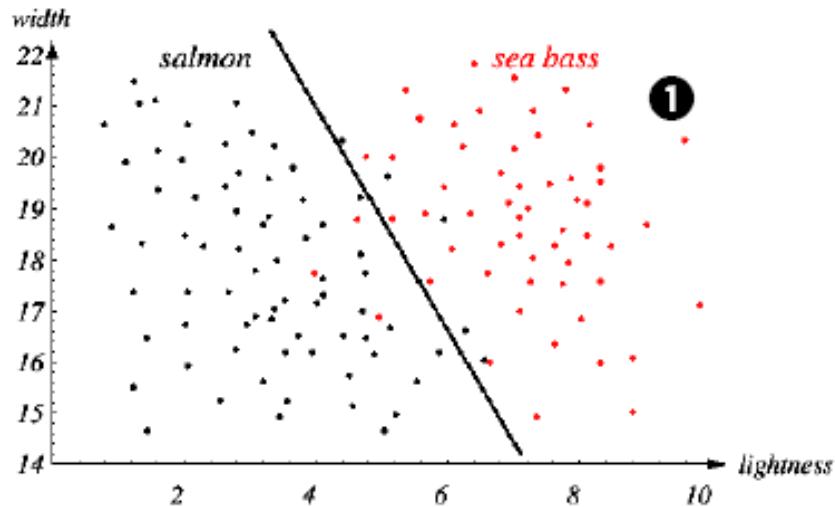
Multi-modal

Classifiers

- A classifier is designed to partition the feature space into class-corresponding **decision regions**
 - What criterion do I have to minimize?
 - What cost function to use?
 - Are all errors equals?
- Borders between the decision regions are called **decision boundaries**

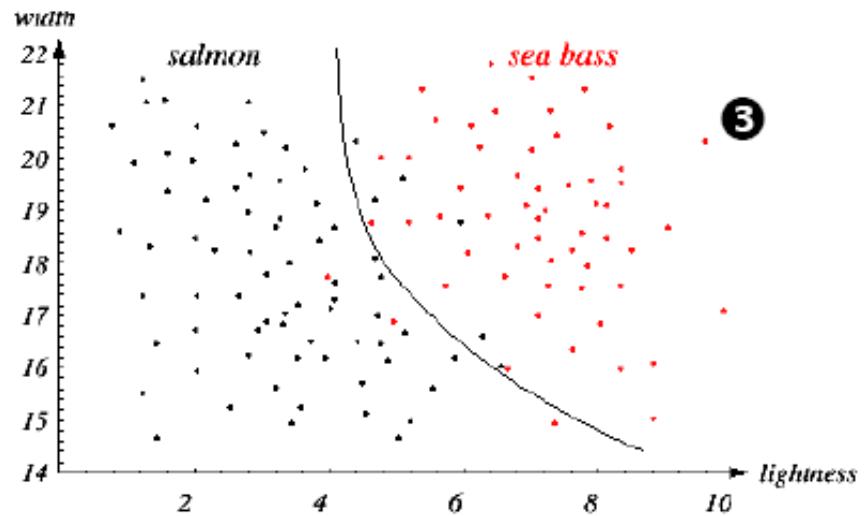


Decision Boundaries



Which decision boundary?

- 1: Linear boundary with significant training error
- 2: Nonlinear complex boundary with zero training error
- 3: Simple non-linear boundary with small training error

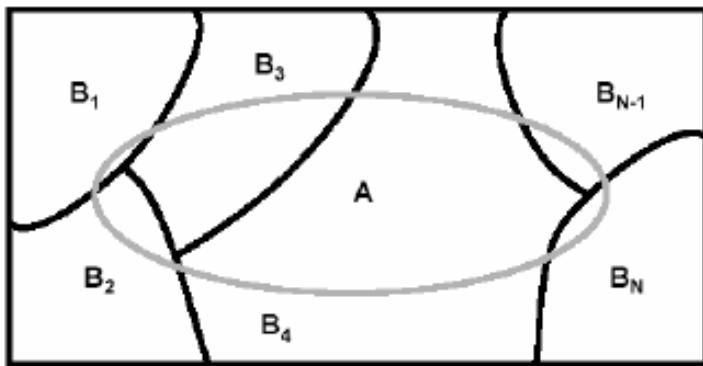


Our First Classifier

- Minimum error classifier
 - Goal is to design a classifier to partition the decision space to minimize the number of errors (misclassifications)
 - During classifier design, we assume that the underlying probabilistic structure (class dependence) is known
 - i.e. the probability $f(\text{feature vector } \mathbf{x} \text{ given class } \omega)$ is known $\rightarrow f(\mathbf{x} | \omega)$

A Short Detour: Bayes Rule

Given that event 'A' has occurred, what is the probability that one of the event 'B's occur?



Rev. Thomas Bayes
(1702-1761)

$$P(B_i | A) = \frac{P(A \cap B_i)}{P(A)} = \frac{P(A | B_i)P(B_i)}{\sum_j P(A | B_j)P(B_j)}$$

A Few Definitions...

Likelihood: The conditional probability of observing a feature value of x given that the correct class is ω_i

Posterior Probability: The conditional probability of correct class being ω_i given that feature value x has been observed

Prior Probability: The probability of class ω_i
 $P(\omega_1) + P(\omega_2) + \dots + P(\omega_c) = 1$

$$P(\omega_i | x) = \frac{P(x \cap \omega_i)}{P(x)} = \frac{P(x | \omega_i)P(\omega_i)}{\sum_{k=1}^c P(x | \omega_k)P(\omega_k)}$$

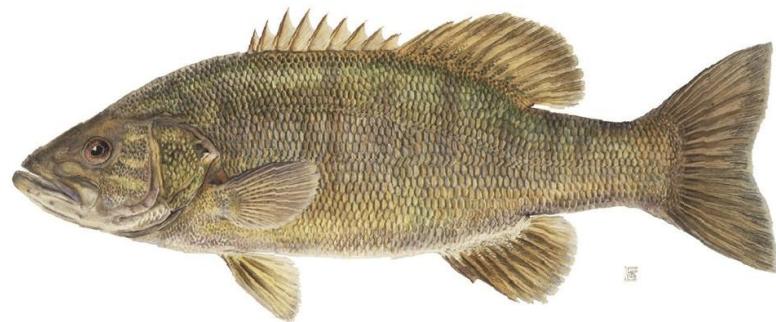
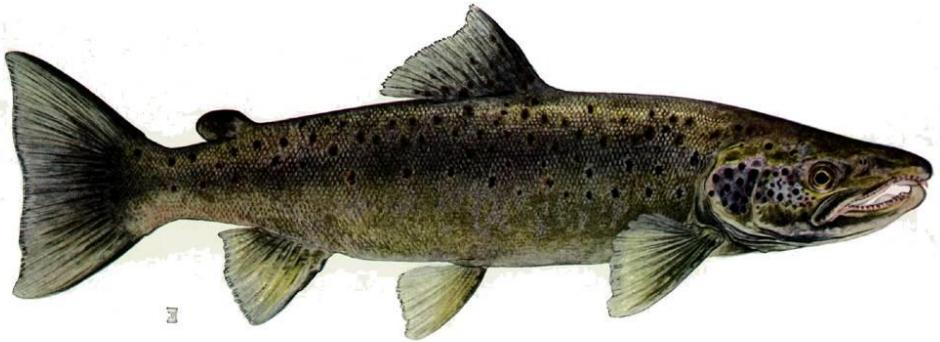
Evidence : The total probability of observing the feature value of x

Posterior Probability For Classification

$$P(\omega_i | x) = \frac{P(x \cap \omega_i)}{P(x)} = \frac{P(x | \omega_i)P(\omega_i)}{\sum_{k=1}^c P(x | \omega_k)P(\omega_k)}$$

- Bayes Classifiers decide on the class that has the **largest posterior probability**
- It can be shown that Bayes classifiers are statistically the best classifiers one can construct i.e. they are minimum error classifiers (optimal)

And Now, Back to our Fish...

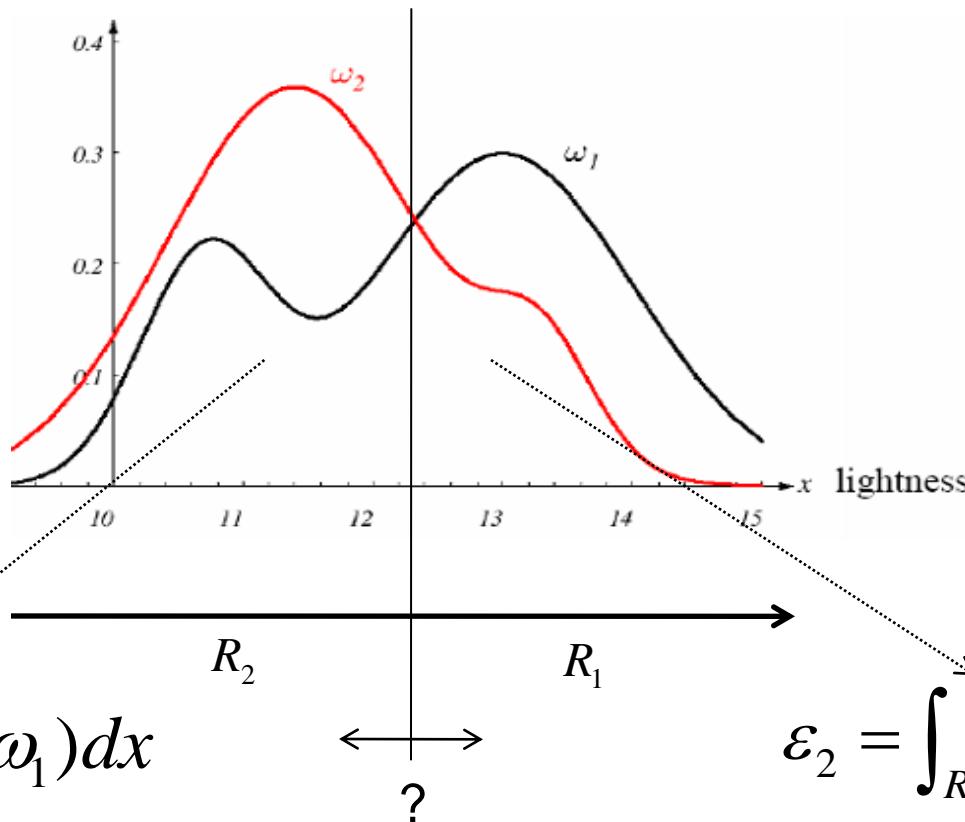


- We will build a **Bayes classifier** i.e. a minimum error classifier, to distinguish salmon samples from bass samples
- For this classifier, consider only one of the **features** i.e. the **lightness** of the fish

Minimum Probability of Error

$\varepsilon = P(error \mid class)$ probability of assigning x to the wrong class ω

$P_e = P(\omega_1)\varepsilon_1 + P(\omega_2)\varepsilon_2$ total probability of error



Where to put the threshold?

Minimum Error Classifier

We want to minimize P_e

$$\begin{aligned}
 P_e &= P(\omega_1)\varepsilon_1 + P(\omega_2)\varepsilon_2 \\
 &= P(\omega_1)\int_{R_2} f(x|\omega_1)dx + P(\omega_2)\int_{R_1} f(x|\omega_2)dx \\
 &= P(\omega_1)\{1 - \int_{R_1} f(x|\omega_1)dx\} + P(\omega_2)\int_{R_1} f(x|\omega_2)dx \\
 &= P(\omega_1) + \{P(\omega_2)\int_{R_1} f(x|\omega_2)dx - P(\omega_1)\int_{R_1} f(x|\omega_1)dx\} \\
 &= P(\omega_1) + \{\int_{R_1} P(\omega_2)f(x|\omega_2) - P(\omega_1)f(x|\omega_1)\}dx
 \end{aligned}$$

as small as possible – this difference is negative within R_1

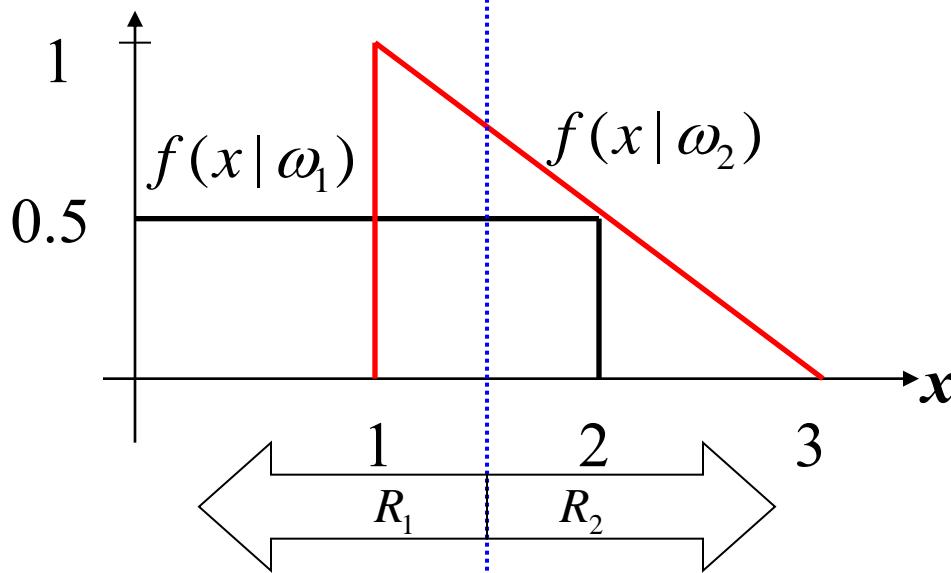
i.e., if $P(\omega_2)f(x|\omega_2) - P(\omega_1)f(x|\omega_1) < 0 \rightarrow \omega_1$

Minimum Error Classifier

$$\left\{ P(\omega_1)f(x|\omega_1) - P(\omega_2)f(x|\omega_2) \right\} \begin{matrix} \omega_1 \\ > \\ \omega_2 \end{matrix} 0$$

Example : 2-Class Classifier Error

$$P(\omega_1) = P(\omega_2) = 0.5$$

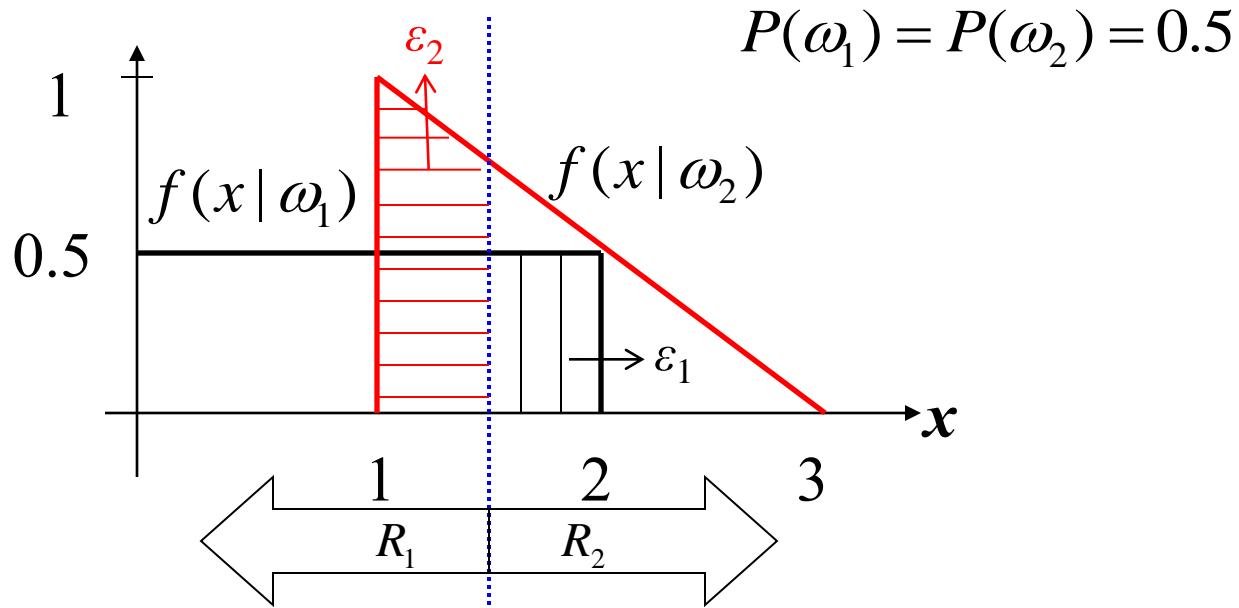


The Classifier assigns:

$x \leq 1.5$ to class 1

$x > 1.5$ to class 2

Example : 2-Class Classifier Error



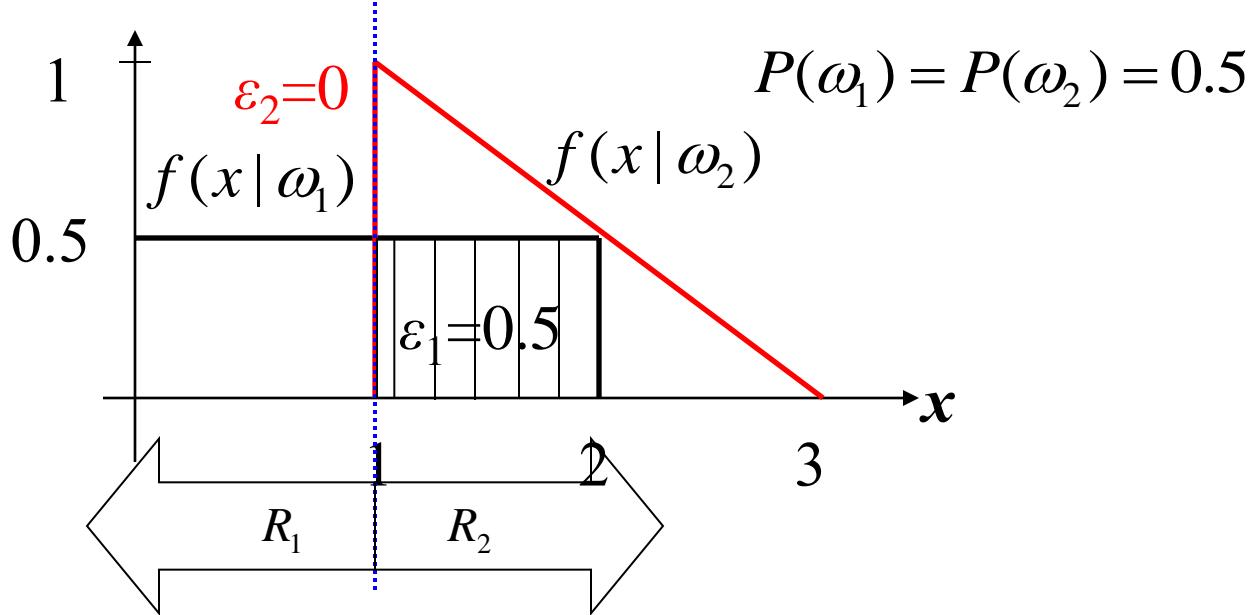
$$\varepsilon_1 = \int_{R_2} f(x | \omega_1) dx = \int_{1.5}^3 f(x | \omega_1) dx = \int_{1.5}^3 (1/2) dx = 1/4$$

$$\varepsilon_2 = \int_{R_1} f(x | \omega_2) dx = \int_0^{1.5} f(x | \omega_2) dx = \int_1^{1.5} \left(1 - \frac{1}{2}(x-1)\right) dx = \frac{3}{4} - \frac{5}{16} = \frac{7}{16}$$

$$P_e = P(\omega_1)\varepsilon_1 + P(\omega_2)\varepsilon_2 = (0.5)\frac{1}{4} + (0.5)\frac{7}{16} = 0.343$$

Is this optimal ?

Example : 2-Class Classifier Error



Minimum Error Classifier will give a smaller error!!

$$P_e = P(\omega_1)\varepsilon_1 + P(\omega_2)\varepsilon_2 = (0.5)(0.5) + 0 = 0.25$$

Likelihood Ratio

$$\{P(\omega_1)f(x|\omega_1) - P(\omega_2)f(x|\omega_2)\} \stackrel{\omega_1}{\gtrless} \stackrel{\omega_2}{\gtrless} 0$$

$$l(x) = \frac{f(x|\omega_1)}{f(x|\omega_2)} \stackrel{\omega_1}{\gtrless} \stackrel{\omega_2}{\gtrless} \frac{P(\omega_2)}{P(\omega_1)} = T$$

Likelihood ratio

Ratio of a priori probabilities

Log

monotonically increasing both sides
without affecting decision rule

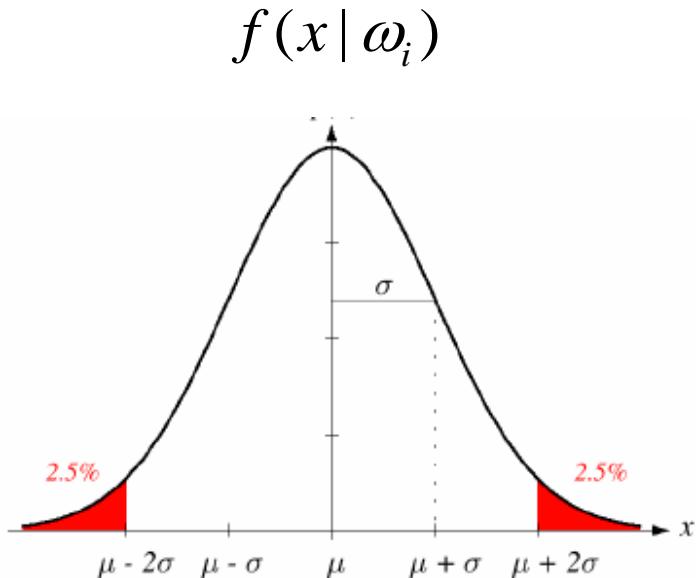
Log Likelihood ratio

$$\ln(l(x)) = \ln\left(\frac{f(x|\omega_1)}{f(x|\omega_2)}\right) \stackrel{\omega_1}{\gtrless} \stackrel{\omega_2}{\gtrless} \ln\left(\frac{P(\omega_2)}{P(\omega_1)}\right) = \ln(T)$$

Suppose fish lightness had a Gaussian Distribution

- Likelihood $f(x | \omega_i)$ are now assumed to be Gaussians with mean μ_i and variance σ_i^2

$$f(x | \omega_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{1}{2}\left(\frac{x - \mu_i}{\sigma_i}\right)^2\right)$$



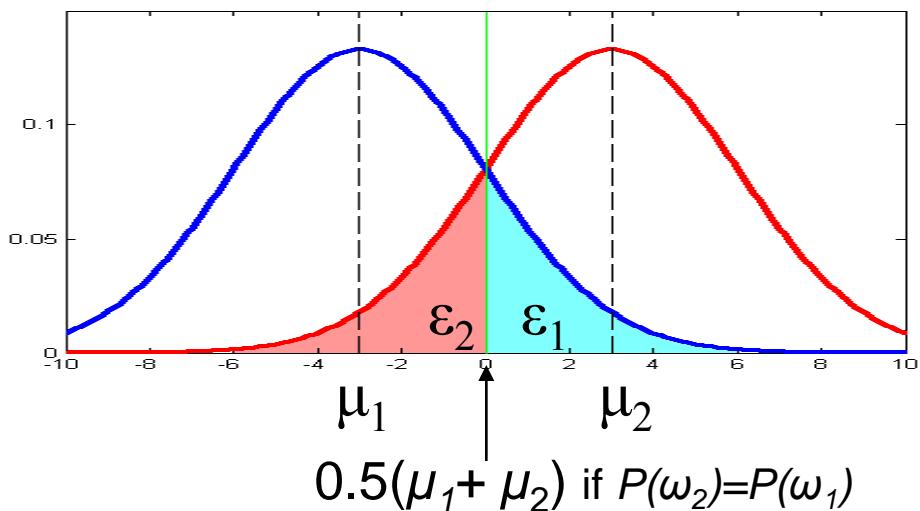
Log Likelihood ratio

$$\begin{aligned} \ln(l(x)) &= \ln \left[\frac{\frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{1}{2}\left(\frac{x - \mu_1}{\sigma_1}\right)^2\right)}{\frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{1}{2}\left(\frac{x - \mu_2}{\sigma_2}\right)^2\right)} \right] \\ &= \ln\left(\frac{\sigma_2}{\sigma_1}\right) + \frac{1}{2}\left(\frac{x - \mu_2}{\sigma_2}\right)^2 - \frac{1}{2}\left(\frac{x - \mu_1}{\sigma_1}\right)^2 \end{aligned}$$

Much easier to find the decision boundary

Gaussian – Linear Classifier

CASE : $\sigma_1 = \sigma_2$



$$\begin{aligned} & \ln\left(\frac{\sigma_2}{\sigma_1}\right) + \frac{1}{2}\left(\frac{x-\mu_2}{\sigma_2}\right)^2 - \frac{1}{2}\left(\frac{x-\mu_1}{\sigma_1}\right)^2 \\ &= \frac{1}{2}\left[\left(\frac{x-\mu_2}{\sigma}\right)^2 - \frac{1}{2}\left(\frac{x-\mu_1}{\sigma}\right)^2\right] \\ &= \frac{1}{\sigma^2}\left[x(\mu_1 - \mu_2) + \left(\frac{\mu_2^2 - \mu_1^2}{2}\right)\right] \end{aligned}$$

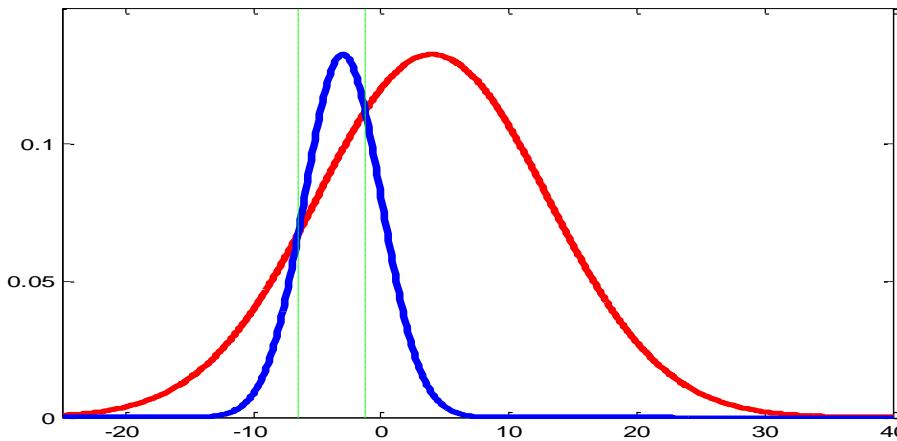
$$x(\mu_1 - \mu_2) + \frac{1}{2}(\mu_2^2 - \mu_1^2) \gtrless_{\omega_1, \omega_2} \sigma^2 \ln\left(\frac{P(\omega_2)}{P(\omega_1)}\right)$$

If equal priors, then the threshold value for $x = 0.5(\mu_1 + \mu_2)$

$$P_e = 0.5(\varepsilon_1 + \varepsilon_2) = \varepsilon_1 = \varepsilon_2$$

Gaussian – Quadratic Classifier

CASE : $\sigma_1 \neq \sigma_2$



$$\ln\left(\frac{\sigma_2}{\sigma_1}\right) + \frac{1}{2} \left(\frac{x - \mu_2}{\sigma_2} \right)^2 - \frac{1}{2} \left(\frac{x - \mu_1}{\sigma_1} \right)^2 \stackrel{\omega_1}{<} \stackrel{\omega_2}{>} \ln\left(\frac{P(\omega_2)}{P(\omega_1)}\right)$$

$$ax^2 + bx + c \geq 0$$

$$(x - x_1)(x - x_2) \geq 0$$

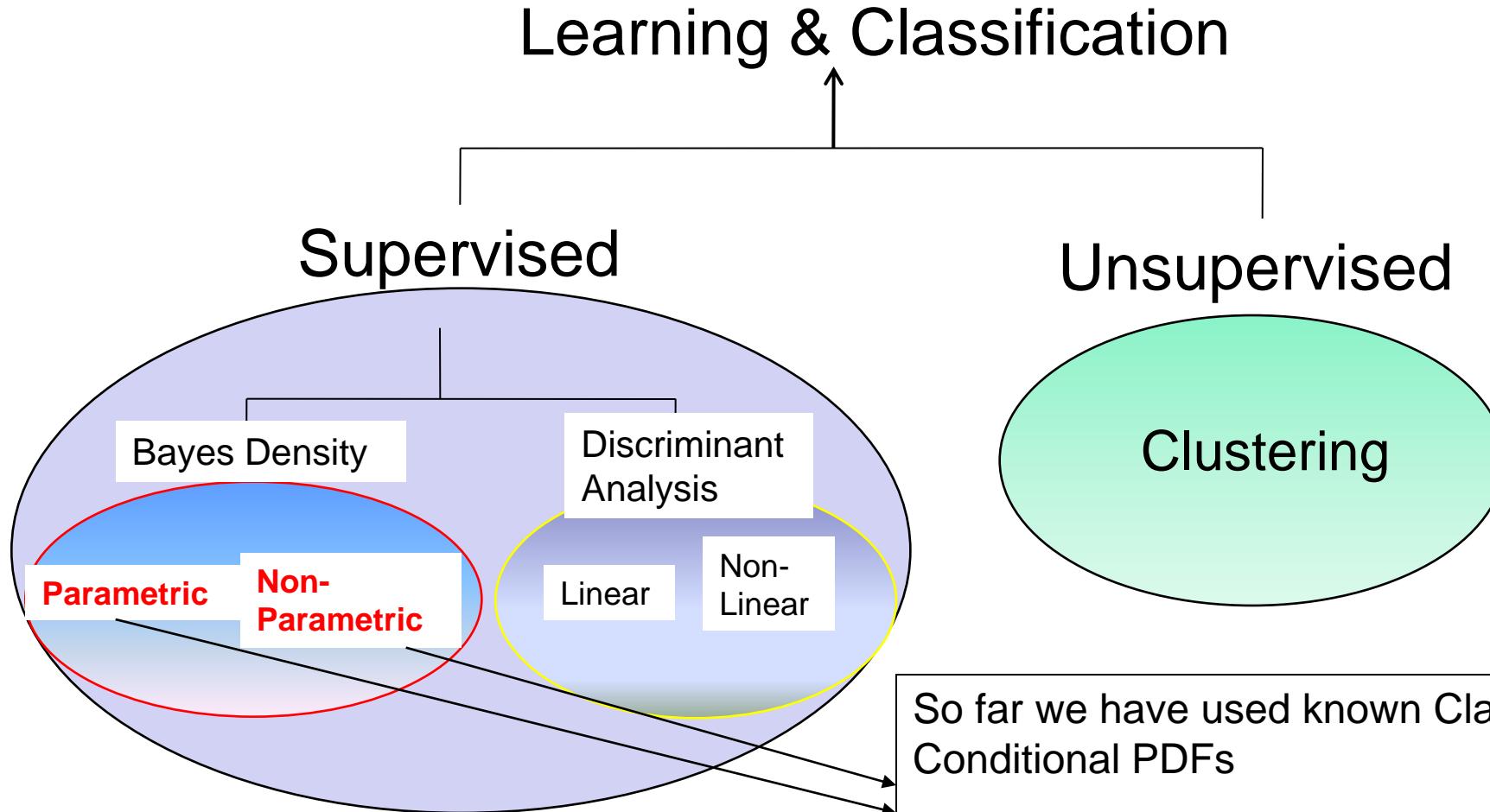
Prof. Marios Savvides

Pattern Recognition Theory

Lecture 5 : Parametric & Non-Parametric Density Estimation

All graphics from Pattern Classification, Duda, Hart and Stork, Copyright © John Wiley and Sons, 2001

Overview Of Pattern Recognition



So far we have used known Class Conditional PDFs

This lecture will focus on how to determine PDFs using **Parametric & Non-Parametric Forms**

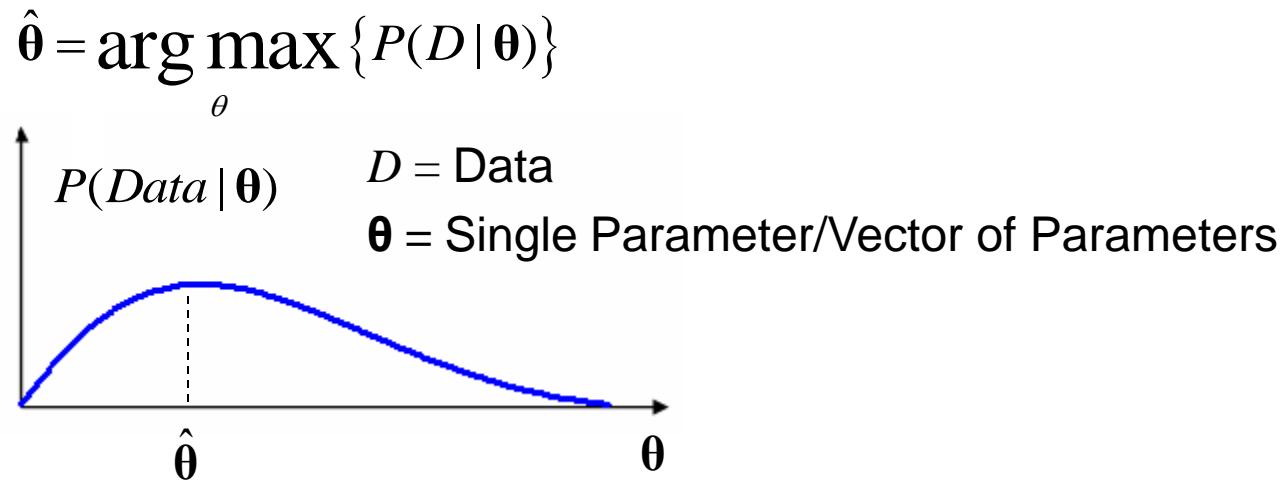
Overview

- Previous lectures have shown how to develop classifiers when the underlying statistical structure is known
- **Parametric Estimation**
 - This method assumes a **particular form** of a PDF (e.g. Gaussian) is known so that we only need to determine the **parameters** (e.g. Mean & Variance)
 - Maximum Likelihood Estimation (MLE)
 - Maximum A Posteriori (Bayesian) Estimation (MAPE)
- **Non-Parametric Density Estimation**
 - This method **does not assume ANY knowledge** about the density
 - Parzen Windows
 - Kernel Density Estimation
 - K-Nearest Neighbor Rule

ML Estimation (MLE)

- **Maximum Likelihood Estimation**

- Assume $P(\mathbf{x}|\omega)$ has a known parametric form uniquely determined by the parameter vector θ
- The parameters are assumed to be **FIXED** (i.e. **NON RANDOM**) but unknown
- Suppose we have a dataset D with the samples in D having been drawn **independently** according to the probability law $P(\mathbf{x}|\omega)$
- The MLE is the value of θ that best explains the data and **once we know this value, we know $P(\mathbf{x}|\omega)$**



“Choose the value of θ that is the most likely to give rise to the data we observe”

MLE

$D = \{x_1, x_2, \dots, x_N\}$ **N independent observations**

$$P(D | \theta) = P(x_1, x_2, \dots, x_N | \theta) = \prod_{k=1}^N P(x_k | \theta)$$

↓
**The likelihood of observing
a particular pattern (random variable)**

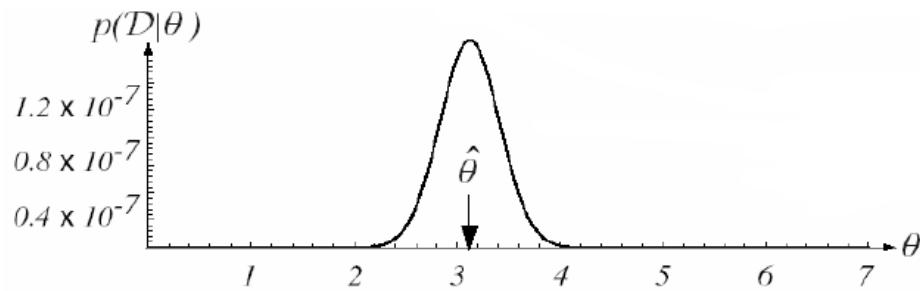
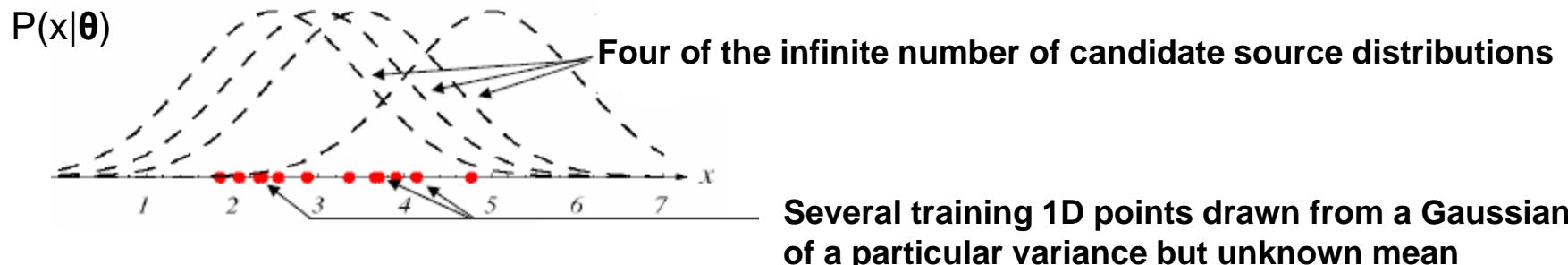
$$\hat{\theta} = \arg \max_{\theta} \{P(Data | \theta)\}$$

“Choose the value of θ that is the most likely to give rise to the data we observe”

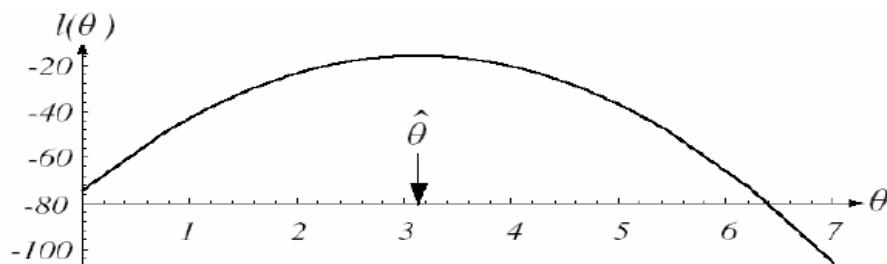
MLE contd..

- It is convenient to work with the **log of the likelihood**

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



The likelihood $P(\text{Data}|\theta)$ as a function of the mean
If we had a very large number of training points
this likelihood would be very narrow



The log of the likelihood ($l(\theta)$) is maximized at the same theta that maximizes the likelihood since log is a **monotonically increasing** function

How To Solve For The ML Estimate?

- Let $\boldsymbol{\theta}$ be the p-component parameter vector $\boldsymbol{\theta} = [\theta_1, \theta_2, \dots, \theta_p]^T$
- Let this be the gradient operator $\nabla_{\boldsymbol{\theta}} = \left[\frac{\partial}{\partial \theta_1}, \frac{\partial}{\partial \theta_2}, \dots, \frac{\partial}{\partial \theta_p} \right]^T$
- We have $P(D | \boldsymbol{\theta}) = \prod_{k=1}^n P(x_k | \boldsymbol{\theta})$
- We define $l(\boldsymbol{\theta})$ the log-likelihood of the function

$$l(\boldsymbol{\theta}) = \log(P(D | \boldsymbol{\theta})) = \sum_{k=1}^n \log(P(x_k | \boldsymbol{\theta}))$$

- And

$$\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \log(P(D | \boldsymbol{\theta})) = \sum_{k=1}^n \nabla_{\boldsymbol{\theta}} \log(P(x_k | \boldsymbol{\theta}))$$

- A set of necessary condition for the ML estimate can be obtained from the set of p equations:

$$\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta}) = \mathbf{0}$$

MLE Example: Univariate Gaussian

- Now assume **neither the mean nor the covariance** matrix are known
- First consider univariate case:

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 = \mu \\ \theta_2 = \sigma^2 \end{bmatrix} \quad P(D | \boldsymbol{\theta}) = \prod_{k=1}^n P(x_k | \boldsymbol{\theta}) \quad \log P(x_k | \boldsymbol{\theta}) = -\frac{1}{2} \log(2\pi\theta_2) - \frac{1}{2\theta_2}(x_k - \theta_1)^2$$

- Its derivative is:

$$\nabla_{\boldsymbol{\theta}} l = \nabla_{\boldsymbol{\theta}} \log(P(x_k | \boldsymbol{\theta})) = \begin{bmatrix} \frac{1}{\theta_2}(x_k - \theta_1) \\ -\frac{1}{2\theta_2} + \frac{(x_k - \theta_1)^2}{2\theta_2^2} \end{bmatrix} = 0$$

- Setting it to zero leads to:

$$\sum_{k=1}^n \frac{1}{\theta_2}(x_k - \theta_1) = 0 \quad \text{and} \quad -\sum_{k=1}^n \frac{1}{\theta_2} + \sum_{k=1}^n \frac{(x_k - \theta_1)^2}{\theta_2^2} = 0$$

- Rearranging:

$$\hat{\mu} = \frac{1}{n} \sum_{k=1}^n x_k$$

Sample Mean

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{k=1}^n (x_k - \hat{\mu})^2$$

ML Estimate

Sample Variance

MLE Example: Multivariate Gaussian

$$P(\mathbf{x}|\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left[-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^t \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right]$$

$$l(\theta) = \log(P(D | \Theta)) = \sum_{k=1}^n \log(P(\mathbf{x}_k | \Theta))$$

Consider **only the mean is unknown**:

$$\log P(\mathbf{x}_k | \boldsymbol{\mu}) = -\frac{1}{2} \log((2\pi)^d |\Sigma|) - \frac{1}{2} (\mathbf{x}_k - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_k - \boldsymbol{\mu})$$

Derivative of log likelihood must be set to 0 to obtain the MLE

$$\nabla_{\boldsymbol{\mu}} \log(P(D | \boldsymbol{\mu})) = \sum_{k=1}^n \Sigma^{-1} (\mathbf{x}_k - \boldsymbol{\mu}) = \mathbf{0}$$

The ML estimate must satisfy:

$$\hat{\boldsymbol{\mu}} = \frac{1}{n} \sum_{k=1}^n \mathbf{x}_k$$

Sample Mean -> ML Estimate

MLE Example: Multivariate Gaussian

- Neither the mean nor the covariance matrix are known

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 = \boldsymbol{\mu} \\ \theta_2 = \Sigma \end{bmatrix} \quad \log P(\mathbf{x}_k | \boldsymbol{\theta}) = -\frac{1}{2} \log \left((2\pi)^d |\Sigma| \right) - \frac{1}{2} (\mathbf{x}_k - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_k - \boldsymbol{\mu})$$

- Derivative of log likelihood is:

$$\nabla_{\boldsymbol{\theta}} l = \nabla_{\boldsymbol{\theta}} \log(P(\mathbf{x}_k | \boldsymbol{\theta})) = \begin{bmatrix} \Sigma^{-1} (\mathbf{x}_k - \boldsymbol{\mu}) \\ ? \end{bmatrix}$$

- How to take the gradient of a determinant of a matrix?

$$\begin{aligned} P(\mathbf{x} | \Sigma) &= \prod_{k=1}^n P(\mathbf{x}_k | \Sigma) = \prod_{k=1}^n \left\{ \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp \left(-\frac{1}{2} (\mathbf{x}_k - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_k - \boldsymbol{\mu}) \right) \right\} \\ &= \frac{1}{[(2\pi)^d |\Sigma|]^{n/2}} \exp \left(-\frac{1}{2} \sum_{k=1}^n (\mathbf{x}_k - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_k - \boldsymbol{\mu}) \right) \end{aligned}$$

Need to take gradient with respect to Σ

MLE Example: Multivariate Gaussian

$$\begin{aligned}
 P(\mathbf{x} | \Sigma) &= \prod_{k=1}^n P(\mathbf{x}_k | \Sigma) = \prod_{k=1}^n \left\{ \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{1}{2} (\mathbf{x}_k - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_k - \boldsymbol{\mu})\right) \right\} \\
 &= \frac{1}{[(2\pi)^d |\Sigma|]^{n/2}} \exp\left(-\frac{1}{2} \sum_{k=1}^n (\mathbf{x}_k - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_k - \boldsymbol{\mu})\right)
 \end{aligned}$$

Scalar



$$\mathbf{b}^T \mathbf{B} \mathbf{b} = \text{trace}(\mathbf{b}^T \mathbf{B} \mathbf{b}) = \text{trace}(\mathbf{B} \mathbf{b} \mathbf{b}^T)$$

$$\text{trace}(\mathbf{A} + \mathbf{B}) = \text{trace}(\mathbf{A}) + \text{trace}(\mathbf{B})$$

$$\text{trace}(\mathbf{C}(\mathbf{A} + \mathbf{B})) = \text{trace}(\mathbf{CA}) + \text{trace}(\mathbf{CB})$$

$$\sum_{k=1}^n (\mathbf{x}_k - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_k - \boldsymbol{\mu})$$

$$= (\mathbf{x}_1 - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_1 - \boldsymbol{\mu}) + \dots + (\mathbf{x}_N - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_N - \boldsymbol{\mu})$$

$$= \text{trace}\left(\Sigma^{-1} (\mathbf{x}_1 - \boldsymbol{\mu})(\mathbf{x}_1 - \boldsymbol{\mu})^T\right) + \dots + \text{trace}\left(\Sigma^{-1} (\mathbf{x}_N - \boldsymbol{\mu})(\mathbf{x}_N - \boldsymbol{\mu})^T\right)$$

$$= \text{trace}\left(\Sigma^{-1} (\mathbf{x}_1 - \boldsymbol{\mu})(\mathbf{x}_1 - \boldsymbol{\mu})^T + \dots + \Sigma^{-1} (\mathbf{x}_N - \boldsymbol{\mu})(\mathbf{x}_N - \boldsymbol{\mu})^T\right)$$

$$= \text{trace}\left(\Sigma^{-1} \sum_{k=1}^n (\mathbf{x}_k - \boldsymbol{\mu})(\mathbf{x}_k - \boldsymbol{\mu})^T\right)$$



$$\mathbf{A} = \frac{1}{n} \sum_{k=1}^n (\mathbf{x}_k - \boldsymbol{\mu})(\mathbf{x}_k - \boldsymbol{\mu})^T$$

MLE Example: Multivariate Gaussian

We can now rewrite:

$$\mathbf{A} = \frac{1}{n} \sum_{k=1}^n (\mathbf{x}_k - \boldsymbol{\mu})(\mathbf{x}_k - \boldsymbol{\mu})^T$$

$$\sum_{k=1}^n (\mathbf{x}_k - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_k - \boldsymbol{\mu}) = \text{trace} \left(\Sigma^{-1} \sum_{k=1}^n (\mathbf{x}_k - \boldsymbol{\mu})(\mathbf{x}_k - \boldsymbol{\mu})^T \right) = \text{trace}(n\Sigma^{-1}\mathbf{A}) = n \cdot \text{trace}(\Sigma^{-1}\mathbf{A})$$

$$\begin{aligned} p(\mathbf{x} | \Sigma) &= \prod_{k=1}^n p(\mathbf{x}_k | \Sigma) = \frac{1}{[(2\pi)^d |\Sigma|]^{n/2}} \exp \left[-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right] \\ &= \frac{1}{[(2\pi)^{dn/2} |\Sigma|]^{-n/2}} \exp \left[-\frac{n}{2} \text{trace}(\Sigma^{-1}\mathbf{A}) \right] \end{aligned}$$

- Now define $\mathbf{B} = \Sigma^{-1}\mathbf{A}$ and let $\lambda_1, \lambda_2, \dots, \lambda_d$ be its eigenvalues

$$p(\mathbf{x} | \Sigma) = \frac{1}{[(2\pi)^{dn/2} |\Sigma|]^{-n/2}} \exp \left[-\frac{n}{2} \text{trace}(\mathbf{B}) \right]$$

$$\det(\mathbf{AB}) = \det(\mathbf{A})\det(\mathbf{B})$$

$$\det(\mathbf{A}^{-1}) = \det(\mathbf{A})^{-1}$$

$\text{trace}(\mathbf{A})$ Sum of eigenvalues

$\det(\mathbf{A})$ Product of eigenvalues

$$= \frac{1}{(2\pi)^{dn/2}} \left[\frac{|\mathbf{B}|}{|\mathbf{A}|} \right]^{n/2} \exp \left[-\frac{n}{2} \text{trace}(\mathbf{B}) \right]$$

$$= \frac{1}{(2\pi)^{dn/2}} |\mathbf{A}|^{-n/2} \left(\prod_{i=1}^d \lambda_i \right)^{n/2} \exp \left[-\frac{n}{2} \sum_{i=1}^d \lambda_i \right]$$

MLE Example: Multivariate Gaussian

Now consider the log likelihood

$$P(\mathbf{x} | \Sigma) = (2\pi)^{-dn/2} |\mathbf{A}|^{-n/2} \left(\prod_{i=1}^d \lambda_i \right)^{n/2} \exp\left(-\frac{n}{2} \sum_{i=1}^d \lambda_i\right)$$

$$\log P(\mathbf{x} | \Sigma) = -\frac{dn}{2} \log(2\pi) - \frac{n}{2} \log |\mathbf{A}| + \frac{n}{2} \sum_{i=1}^d \log(\lambda_i) - \frac{n}{2} \sum_{i=1}^d \lambda_i$$

Remember that \mathbf{A} is fixed, and taking the gradient with respect to Σ is now equivalent to taking the derivatives with respect to the eigenvalues of \mathbf{B}

$$\frac{\partial}{\partial \lambda_i} \log P(\mathbf{x} | \Sigma) = \frac{n}{2\lambda_i} - \frac{n}{2} = 0 \Rightarrow \lambda_i = 1 \text{ for } i = 1, \dots, d$$

Thus \mathbf{B} must have all eigenvalues of 1 and is equivalent to \mathbf{I} . This means that the likelihood is maximized if $\Sigma^{-1}\mathbf{A} = \mathbf{I}$ or $\Sigma = \mathbf{A}$

$$\hat{\Sigma}_{\text{ML}} = \frac{1}{n} \sum_{k=1}^n (\mathbf{x}_k - \hat{\mu})(\mathbf{x}_k - \hat{\mu})^T$$

MLE - Sample Covariance

MLE vs The Bayesian Approach

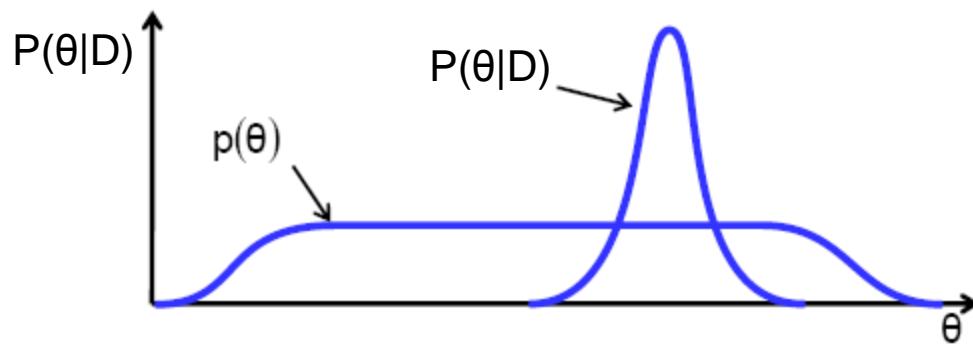
- **Maximum Likelihood Estimation (MLE)**
 - The parameters are assumed to be **FIXED** (i.e. **NON RANDOM**) but unknown
 - The ML seeks the solutions that best explains the data

$$\hat{\theta} = \arg \max_{\theta} \{P(Data | \theta)\}$$

- **Bayesian Estimation (BE)**
 - The parameters are assumed to be **RANDOM VARIABLES** with some known **PRIORI DISTRIBUTION**
 - Bayesian approach aims at estimating the posterior density **$P(\theta|Data)$**
 - **The MAPE (Maximum A Posteriori Estimate)** of θ is the value of θ that maximizes the posterior density (i.e. it is the mode of the posterior)

Bayesian Estimation

- In the Bayesian approach, our uncertainty about the parameters is represented by a PDF
- The parameters are described by a prior density $P(\theta)$ which indicates which parameters are more likely than others
- We make use of Bayes theorem to find the posterior $P(\theta|D)$
- Ideally, we want the training data to “sharpen” the posterior $P(\theta|D)$ or reduce our uncertainty about the parameters



Bayesian Estimation

- We ideally want to estimate a PDF - $P(x)$. Best we can do is estimate it by observing the training data to obtain $P(x|D)$. We also assume that it has a known parametric form. So $P(x|\theta)$ is completely known. But θ is random (unlike in MLE) and has its own PDF.

$$P(x|D) = \int P(x, \theta | D) d\theta \quad \text{Using the theorem of total probability}$$

$$= \int P(x|\theta) P(\theta|D) d\theta$$

This integration is typically very hard

Known

Unknown

θ is random and has its own PDF

Applying Bayes rule:

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} = \frac{P(D|\theta)P(\theta)}{\int P(D|\theta)P(\theta)d\theta}$$

Posterior

$$P(D|\theta) = \prod_{k=1}^N P(x_k|\theta)$$

MAP Estimation (MAPE)

- MLE aims at determining the value $\theta = \theta_{\text{MLE}}$ that maximizes the likelihood

$$\theta_{\text{MLE}} = \arg \max_{\theta} \{P(D | \theta)\}$$

- MAPE assumes that θ is not a fixed, but is an RV with a PDF given by $P(\theta)$ and it aims at maximizing the posterior

$$\theta_{\text{MAP}} = \arg \max_{\theta} \{P(\theta | D)\}$$

where $P(\theta | D) = \frac{P(D | \theta)P(\theta)}{P(D)} = \frac{P(D | \theta)P(\theta)}{\int P(D | \theta)P(\theta)d\theta} = \frac{P(\theta) \prod_{k=1}^N P(x_k | \theta)}{\int P(D | \theta)P(\theta)d\theta}$

Since $P(D)$ is constant we can also view MAP estimate as

$$\theta_{\text{MAP}} = \arg \max_{\theta} \{P(D | \theta)P(\theta)\}$$

- Thus the MAPE of θ is simply the mode of the posterior $P(\theta | D)$ and MAPE differs from MLE as it determines a value of θ which maximizes the posterior instead of the likelihood

MAPE Example: Univariate Gaussian

- Compute $P(\theta|D)$ and the desired PDF $P(x|D)$ where $p(\mu) = N(\mu_0, \sigma_0^2)$
- Assume the known prior knowledge about the mean can be expressed by a *known* prior density assumed to be normal:

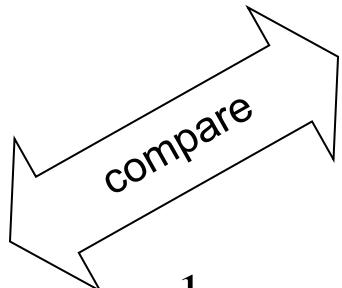
$$p(\mu) = N(\mu_0, \sigma_0^2) \quad \text{Known!}$$

$$P(\mu|D) = \frac{P(D|\mu)P(\mu)}{\int P(D|\mu)P(\mu)d\mu} = \alpha \prod_{k=1}^n P(x_k|\mu)P(\mu)$$

$$\begin{aligned} P(\mu|D) &= \alpha \prod_{k=1}^n \underbrace{\frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x_k - \mu}{\sigma}\right)^2\right]}_{P(x_k|\mu)} \underbrace{\frac{1}{\sqrt{2\pi}\sigma_0} \exp\left[-\frac{1}{2}\left(\frac{\mu - \mu_0}{\sigma_0}\right)^2\right]}_{P(\mu)} \\ &= \alpha' \exp\left[-\frac{1}{2}\left(\sum_{k=1}^n \left(\frac{x_k - \mu}{\sigma}\right)^2 + \left(\frac{\mu - \mu_0}{\sigma_0}\right)^2\right)\right] \\ &= \alpha'' \exp\left[-\frac{1}{2}\left[\left(\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}\right)\mu^2 - 2\left(\frac{1}{\sigma^2} \sum_{k=1}^n x_k + \frac{\mu_0}{\sigma_0^2}\right)\mu\right]\right] \quad \text{GAUSSIAN!} \end{aligned}$$

MAPE Example: Univariate Gaussian

Gaussian Likelihood && Gaussian Prior \rightarrow Gaussian Posterior



$$P(\mu | D) = \alpha \exp \left[-\frac{1}{2} \left[\underbrace{\left(\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2} \right)}_{\text{constant}} \mu^2 - 2 \underbrace{\left(\frac{1}{\sigma^2} \sum_{k=1}^n x_k + \frac{\mu_0}{\sigma_0^2} \right)}_{\text{constant}} \mu \right] \right]$$

$$P(\mu | D) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp \left[-\frac{1}{2} \left(\frac{\mu - \mu_n}{\sigma_n} \right)^2 \right] = \alpha \exp \left[-\frac{1}{2} \left[\mu^2 \underbrace{\left(\frac{1}{\sigma_n^2} \right)}_{\text{constant}} - 2\mu \left(\frac{\mu_n}{\sigma_n^2} \right) + \left(\frac{\mu_n}{\sigma_n} \right)^2 \right] \right]$$

We get $\frac{1}{\sigma_n^2} = \frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}$ and $\frac{\mu_n}{\sigma_n^2} = \frac{1}{\sigma^2} \sum_{k=1}^n x_k + \frac{\mu_0}{\sigma_0^2} = \frac{n}{\sigma^2} \hat{\mu}_n + \frac{\mu_0}{\sigma_0^2}$

where $\hat{\mu}_n = \frac{1}{n} \sum_{k=1}^n x_k$ and is the sample mean

So
$$\sigma_n^2 = \frac{\sigma^2 \sigma_0^2}{n \sigma_0^2 + \sigma^2}$$

$$\mu_n = \left(\frac{n \sigma_0^2}{n \sigma_0^2 + \sigma^2} \right) \hat{\mu}_n + \left(\frac{\sigma^2}{n \sigma_0^2 + \sigma^2} \right) \mu_0$$

MAPE Example: Univariate Gaussian

$$\mu_n = \left(\frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2} \right) \hat{\mu}_n + \left(\frac{\sigma^2}{n\sigma_0^2 + \sigma^2} \right) \mu_0$$

$$\sigma_n^2 = \frac{\sigma^2 \sigma_0^2}{n\sigma_0^2 + \sigma^2}$$

Best guess for μ

Uncertainty

- μ_n is a linear combination of $\hat{\mu}_n$ and μ_0 (always lies between them)
 - For small samples -> More weights on prior ($\hat{\mu}_0$)
 - For large samples -> More weights on observation ($\hat{\mu}_n$)
- If $\sigma_n \neq 0$, then $\mu_n \rightarrow \hat{\mu}_n$ as $n \rightarrow \infty$ (MLE = MAPE)
- If $\sigma_0 = 0$, then $\mu_n = \mu_0$
- If $\sigma_0 \gg \sigma$ then $\mu_n = \hat{\mu}_n$ (MLE = MAPE)

MAPE Example: Univariate Gaussian

- μ_n represents our best guess for μ after observing n samples, and σ_n^2 measures our uncertainty about this guess.

$$\mu_n = \underbrace{\left(\frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2} \right)}_{ML} \hat{\mu}_n + \underbrace{\left(\frac{\sigma^2}{n\sigma_0^2 + \sigma^2} \right)}_{PRIORITY_KNOWLEDGE} \mu_0$$

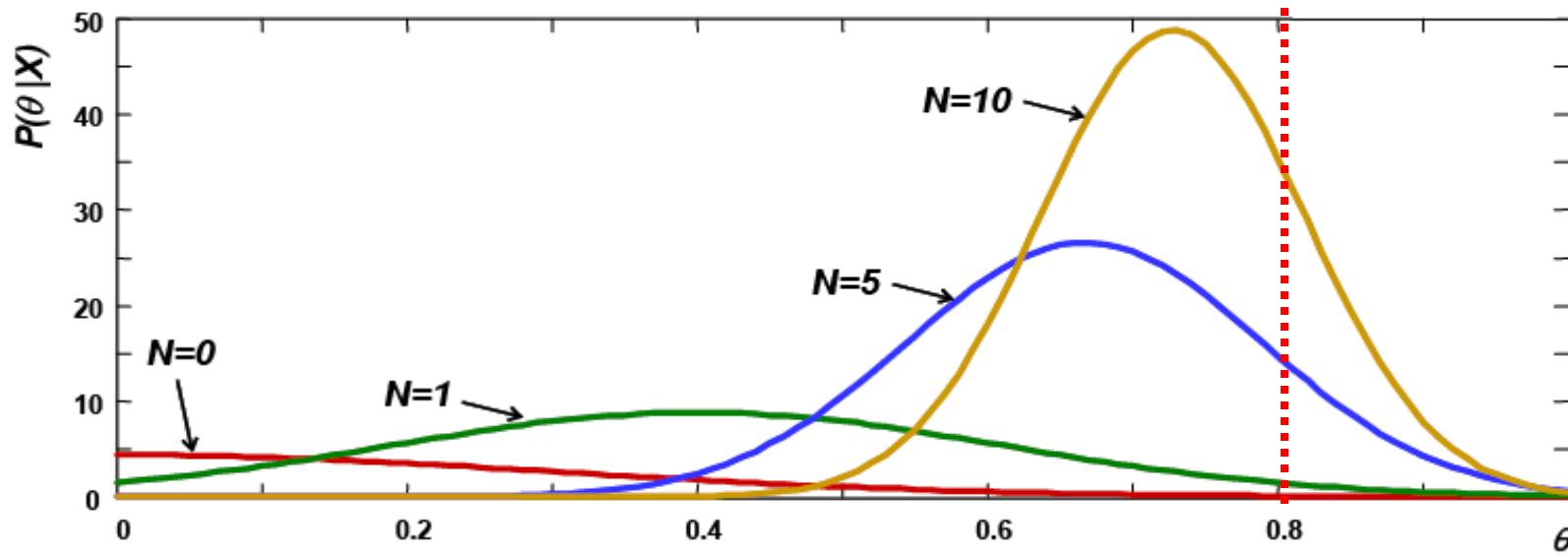
$$\sigma_n^2 = \frac{\sigma^2 \sigma_0^2}{n\sigma_0^2 + \sigma^2}$$

Decreases as $n \rightarrow \infty$

- Each additional observation decreases our uncertainty and $P(\mu|D)$ becomes narrower and sharply peaked around the true value of μ and becomes a Dirac delta function. This is called **Bayesian Learning**.
- A class of PDF $P(\theta)$ is said to be **conjugate** to a class of likelihood functions $P(x|\theta)$ if the resulting posterior distributions $P(\theta|x)$ are in the **same family as $P(\theta)$** .
 - The Gaussian family is conjugate to itself if the likelihood function is Gaussian, choosing a Gaussian prior will ensure that the posterior distribution is also Gaussian

MAPE Example: Gaussians

- Assume that the true mean of the $P(x)$ is $N(0.8, 0.09)$.
 - In reality this is something we cannot know
- We generate a number of examples from this distribution
- We don't know where the mean will be, so we assume a Gaussian prior
 - $P_0(\mu) = N(0, 0.09)$
- As the number of training examples increases, the estimates μ_N approaches its true value of 0.8 and the spread decreases



MAPE: Back To The Gaussian Example

So far we've only figured $P(\mu|D)$. We want $P(x|D)$

$$\begin{aligned}
 P(x|D) &= \int P(x|\mu)P(\mu|D)d\mu \\
 &= \int \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left[-\frac{1}{2}\left(\frac{\mu-\mu_n}{\sigma_n}\right)^2\right] d\mu \\
 &= \frac{1}{2\pi\sigma\sigma_n} \exp\left[-\frac{1}{2}\frac{(x-\mu_n)^2}{\sigma^2 + \sigma_n^2}\right] f(\sigma, \sigma_n) \\
 f(\sigma, \sigma_n) &= \int \exp\left[-\frac{1}{2}\frac{\sigma^2 + \sigma_n^2}{\sigma^2\sigma_n^2}\left(\mu - \frac{\sigma_n^2 x + \sigma^2 \mu_n}{\sigma^2 + \sigma_n^2}\right)^2\right] d\mu
 \end{aligned}$$

Not dependent on μ !!

$$P(x|D) = N(\mu_n, \sigma^2 + \sigma_n^2)$$

Thus $P(x|D)$ is the desired class-conditional density $P(x|\omega_i, D)$ and together with the priors $P(\omega_i)$ it gives us the probabilistic framework needed to design the classifier. This is in contrast with the ML method that only makes **point estimates** rather than estimate a **distribution** for $P(x|D)$

MAPE Example: Multivariate Gaussian

- Compute $P(\mu|D)$ and the desired PDF $P(x|D)$ where $P(x|\mu) = N(\mu, \Sigma)$
- Assume the known prior knowledge about the mean can be expressed by a *known* prior density assumed to be normal:

$$P(\mu) = N(\mu_0, \Sigma_0) \quad \text{Known!}$$

$$P(\mu|D) = \frac{P(D|\mu)P(\mu)}{\int P(D|\mu)P(\mu)d\mu} = \alpha \prod_{k=1}^n P(x_k|\mu)P(\mu)$$

$$\begin{aligned} P(\mu|D) &= \alpha \prod_{k=1}^n P(x_k|\mu)P(\mu) \\ &= \alpha' \exp \left[-\frac{1}{2} \left(\mu^T \left(n\Sigma^{-1} + \Sigma_0^{-1} \right) \mu - 2\mu^T \left(\Sigma^{-1} \sum_{k=1}^n x_k + \Sigma_0^{-1} \mu_0 \right) \right) \right] \end{aligned}$$

Which has a Gaussian Form

$$= \alpha'' \exp \left[-\frac{1}{2} (\mu - \mu_n)^T \Sigma_n^{-1} (\mu - \mu_n) \right]$$

MAPE Example: Multivariate Gaussian

- Thus $P(\mu|D)$ is $N(\mu_n, \Sigma_n)$. Equating coefficients, we obtain:

$$\Sigma_n^{-1} = n\Sigma^{-1} + \Sigma_0^{-1} \quad \text{and} \quad \Sigma_n^{-1}\mu_n = n\Sigma^{-1}\hat{\mu}_n + \Sigma_0^{-1}\mu_0$$

where $\hat{\mu}_n = \frac{1}{n} \sum_{k=1}^n \mathbf{x}_k$ is the sample mean

After some manipulation (which we don't prove) we get:

$$\mu_n = \Sigma_0 \left(\Sigma_0 + \frac{1}{n} \Sigma \right)^{-1} \hat{\mu}_n + \frac{1}{n} \Sigma \left(\Sigma_0 + \frac{1}{n} \Sigma \right)^{-1} \mu_0$$

$$\Sigma_n = \Sigma_0 \left(\Sigma_0 + \frac{1}{n} \Sigma \right)^{-1} \frac{1}{n} \Sigma$$

- Finally, it can be shown that

$$P(\mathbf{x} | D) = N(\mu_n, \Sigma + \Sigma_n)$$

MAPE Example: Gaussians

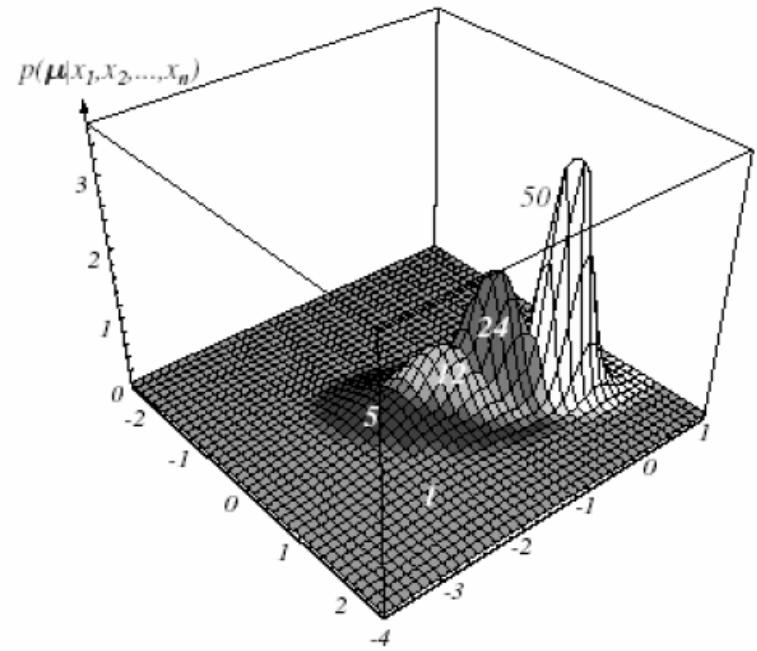
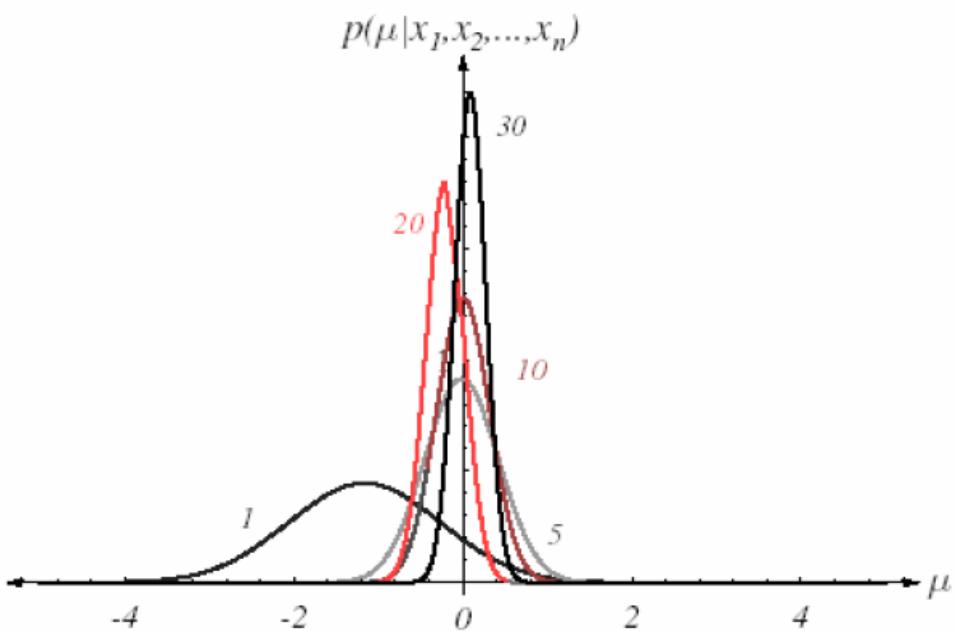


FIGURE 3.2. Bayesian learning of the mean of normal distributions in one and two dimensions. The posterior distribution estimates are labeled by the number of training samples used in the estimation. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Recap: ML vs Bayesian Estimation

- BE assumes that the parameters come from a distribution with known priors
- BE provides a distribution for θ rather than point values. BE provides more information, but is in general much more difficult to compute (integration)
- For most times, if the assumptions are correct, MLE gives good enough results

Recap: ML vs Bayesian Estimation

- Amount of Training data
 - The two methods are equivalent assuming infinite training samples.
 - They differ for smaller training data
- Computational Complexity
 - ML uses **differential calculus** or gradient search
 - Bayesian estimation needs complex multidimensional **integration techniques**
- Solution Complexity
 - ML solution is easy to interpret
 - A Bayes Estimation solution **might not be** of the parametric form assumed
- Prior distribution
 - If the prior $P(\theta)$ is **uniform (flat)**, BE solutions are equivalent to ML solutions

Conjugate Priors

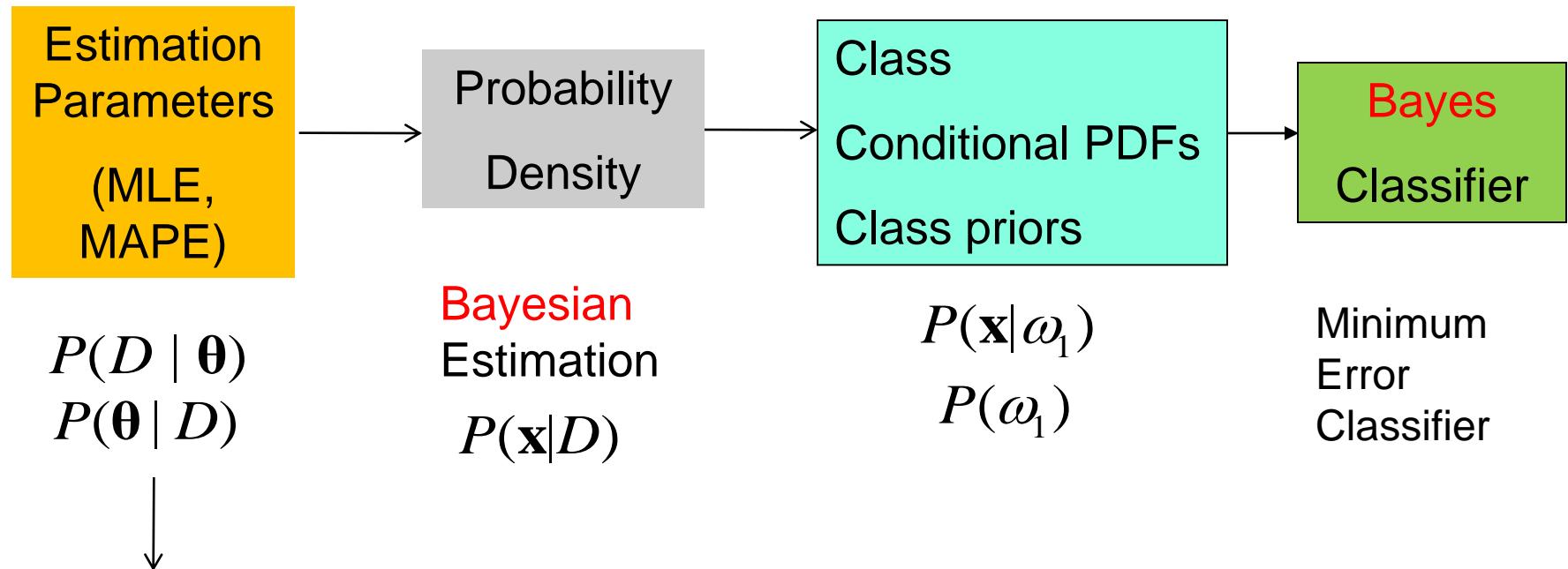
Discrete likelihood distributions

Likelihood	Model parameters	Conjugate prior distribution	Prior hyperparameters	Posterior hyperparameters
Bernoulli	p (probability)	Beta	α, β [4]	$\alpha + \sum_{i=1}^n x_i, \beta + n - \sum_{i=1}^n x_i$
Binomial	p (probability)	Beta	α, β [4]	$\alpha + \sum_{i=1}^n x_i, \beta + \sum_{i=1}^n N_i - x_i$
Poisson	λ (rate)	Gamma	α, β [4]	$\alpha + \sum_{i=1}^n x_i, \beta + n$
Multinomial	p (probability vector)	Dirichlet	$\vec{\alpha}$	$\vec{\alpha} + \sum_{i=1}^n \vec{x}^{(i)}$
Geometric	p_0 (probability)	Beta	α, β [4]	$\alpha + n, \beta + \sum_{i=1}^n x_i$

Continuous likelihood distributions

Likelihood	Model parameters	Conjugate prior distribution	Prior hyperparameters	Posterior hyperparameters
Uniform	$U(0, \theta)$	Pareto	x_m, k	$\max\{x_{(n)}, x_m\}, k + n$
Exponential	λ (rate)	Gamma	α, β [4]	$\alpha + n, \beta + \sum_{i=1}^n x_i$
Normal with known variance σ^2	μ (mean)	Normal	μ_0, σ_0^2	$(\frac{\mu_0}{\sigma_0^2} + \frac{\sum_{i=1}^n x_i}{\sigma^2}) / (\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}), (\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2})^{-1}$
Normal with known precision τ	μ (mean)	Normal	μ_0, τ_0	$(\tau_0 \mu_0 + \tau \sum_{i=1}^n x_i) / (\tau_0 + n\tau), \tau_0 + n\tau$

Big Picture



How To Evaluate
Estimators? (Next Lecture)

Recap

- Maximum Likelihood Estimation (MLE)
 - MLE for Univariate and Multivariate Gaussians
- Bayesian Estimation (BE)
 - BE for Univariate and Multivariate Gaussians
- Maximum A Posteriori Estimation (MAPE)
- Conjugate Priors

Prof. Marios Savvides

Pattern Recognition Theory

Lecture 6 : Principal Component Analysis (PCA) - I

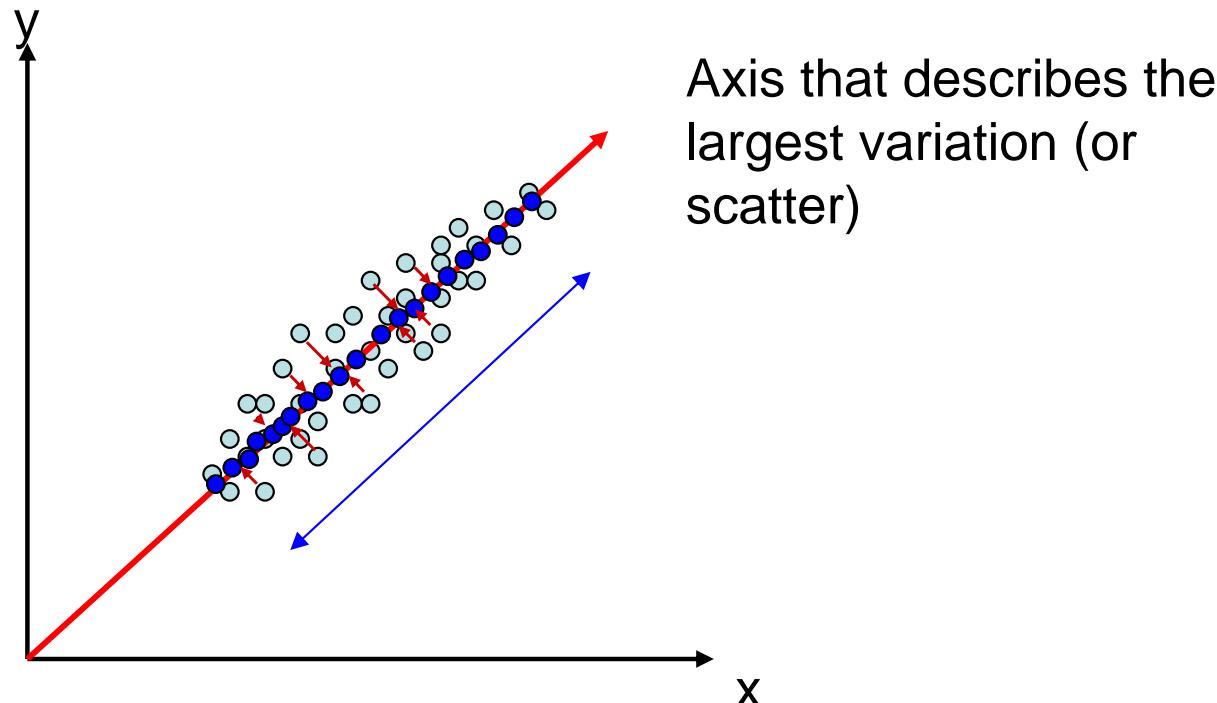
PCA

- Also known as Hotelling Transform or Karhunen Loeve Transform
- Very popular pattern recognition tool...also known as “Eigenfaces” by Turk & Pentland
- Commonly used baseline benchmark when testing your new pattern recognition algorithm..i.e. given same training data and testing configuration, can I do better than PCA?
- So....you must DEFINITELY LEARN PCA!

What is PCA?

What are we trying to do?

- We want to find projections of data (i.e. direction vectors that we can project the data on to) that describe the maximum variation.



PCA formulation

- We want projection vectors ω that when we project the data onto these directions we maximize the scatter or variance.

- i.e. we want to maximize

$$\text{Var} (\omega^T x)$$

Subject to the constraint that ω is a direction vector of unit norm i.e. $\|\omega\|=1$

How is PCA derived?

- Maximize the following objective function:

$$\begin{aligned} J(\omega) &= \text{Var}(\omega^T x) \\ &= E(\omega^T x - \omega^T \mu)^2 \\ &= E(\omega^T x - \omega^T \mu)(x^T \omega - \mu^T \omega) \\ &= \omega^T E(x - \mu)(x - \mu)^T \omega \\ &= \omega^T \Sigma \omega \end{aligned}$$

Where covariance matrix $\Sigma = E(x - \mu)(x - \mu)^T$

Derivation

- We want to maximize $J(\omega)$ subject to the projection vectors ω be unit norm i.e.
- Maximize quadratic term $\omega^T \Sigma \omega$ subject to $\|\omega\|=1$.
- Solution: Form Lagrangian optimization to take care of constraints, take derivative and set to zero to find ω vectors.

Derivation

Solve:

$$L(\omega, \lambda) = \omega^T \Sigma \omega - \lambda(\omega^T \omega - 1)$$

Take derivative w.r.t ω , set it to zero

$$\frac{\partial L(\omega, \lambda)}{\partial \omega} = 2\Sigma\omega - 2\lambda\omega = 0$$

$$\Sigma\omega = \lambda\omega$$

Standard Eigenvalue/Eigenvector problem ($Ax = \lambda x$). i.e.
the vectors ω are the eigenvectors of the covariance matrix Σ .

PCA

- Ok..do eigenanalysis...find eigenvectors ω_i and corresponding eigenvalues λ_i . Which to use though? All of them?
- A $d \times d$ covariance matrix contains max d eigenvector/eigenvalue pairs. Do we need to compute all of them?
- Which ω_i vector and eigenvalue λ_i pairs to use?

PCA

- Lets re-visit our original objective of PCA....
- We want to find projections which describe the biggest variance (or have the maximum scatter of data).
- Remember the variance of projected data is

$$\omega^T \Sigma \omega. \quad (1)$$

- And our solution yielded

$$\Sigma \omega = \lambda \omega \quad (2)$$

- Plug (2) in (1) and we get

$$\begin{aligned} \text{projected variance} &= \omega^T \Sigma \omega = \omega^T \lambda \omega \\ &= \lambda \omega^T \omega \quad (\text{remember } \|\omega\|=1) \\ &= \lambda \end{aligned}$$

PCA

- So this means that the direction vector ω_l has captures λ_l variance.
- So we want to first represent the data using projections with largest variance, so order and keep the ω_l 's with largest λ_{\max} .
- In fact the $d \times d$ covariance matrix will not be full rank. Assume you N training images of size $M \times M = d$, then you have AT MOST $N-1$ non-zero eigenvalues!

Properties of PCA vectors

- Remember we computed eigenvectors ω from covariance matrix Σ .
- IMPORTANT NOTE: Σ matrix is symmetric, i.e. $\Sigma = \Sigma^T$ is how you test for symmetry.

Proof:

$$\begin{aligned}\Sigma &= E(x - \mu)(x - \mu)^T = [E(x - \mu)(x - \mu)^T]^T \\ &= E(x - \mu)(x - \mu)^T = \Sigma^T\end{aligned}$$

So what?

- Symmetric Matrix, has ORTHOGONAL eigenvectors!
- What does this mean?

$$\begin{aligned}\omega_i^T \omega_k &= 0 \text{ for all } i \neq k \\ \omega_i^T \omega_k &= 1 \text{ for all } i = k \quad (\omega^T \omega = 1)\end{aligned}$$

(i.e. the inner-product or dot-product between different eigenvectors is 0.) They are un-correlated, i.e. information about how data varies in the direction of one of the eigenvectors tells you nothing about how data varies in the directions of the eigenvectors.

- This also means we have an ORTHOGONAL basis that we can use to represent our data!

Expanding a signal using a basis

From this slide on: we refer to principal direction vectors ω as eigenvectors v

- Assume you have a discrete vector signal x .
- You have a set of N basis vectors v_i which can use to represent a signal as follows:

$$\mathbf{x} = \sum_{i=1}^N p_i \mathbf{v}_i = p_1 \begin{bmatrix} | \\ \mathbf{v}_1 \\ | \end{bmatrix} + p_2 \begin{bmatrix} | \\ \mathbf{v}_2 \\ | \end{bmatrix} + \dots + p_n \begin{bmatrix} | \\ \mathbf{v}_n \\ | \end{bmatrix} = \mathbf{V}\mathbf{p}$$

i.e. the signal is a linear combination of the basis vectors where the p_i scalars are weight coefficients.

$$\mathbf{V} = \begin{bmatrix} | & | & | & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 & \dots \mathbf{v}_N \\ | & | & | & | \end{bmatrix}$$

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ | \\ p_N \end{bmatrix}$$

Computing the weight coefficients

$$\mathbf{x} = \sum_{i=1}^N p_i \mathbf{v}_i = p_1 \begin{bmatrix} | \\ \mathbf{v}_1 \\ | \end{bmatrix} + p_2 \begin{bmatrix} | \\ \mathbf{v}_2 \\ | \end{bmatrix} + \dots + p_n \begin{bmatrix} | \\ \mathbf{v}_n \\ | \end{bmatrix} = \mathbf{V}\mathbf{p}$$

- Since this is an orthogonal basis, we can easily find the coefficients p . These are just the projections of the signal on to each basis.
- i.e. $p_1 = \mathbf{x}^T \mathbf{v}_1$ or $\mathbf{p} = \mathbf{V}^T \mathbf{x}$

$$\mathbf{V}^T \mathbf{x} = \begin{bmatrix} - & \mathbf{v}_1 & - \\ - & \mathbf{v}_2 & - \\ - & \mathbf{v}_3 & - \end{bmatrix} \begin{bmatrix} | \\ \mathbf{x} \\ | \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

In PCA we model the variations about the mean.

- Don't forget that you model the variance about the mean!
- i.e. Don't forget to subtract the global mean before you analyze your data.

$$\mathbf{p} = \mathbf{V}^T (\mathbf{x} - \mathbf{m})$$

- By Mean we mean the sample mean image/vector of all your training data samples.
- Don't forget to add the mean back before you reconstruct the data!

$$\mathbf{x} = \sum_{i=1}^N p_i \mathbf{v}_i + \mathbf{m} = p_1 \begin{bmatrix} | \\ \mathbf{v}_1 \\ | \end{bmatrix} + p_2 \begin{bmatrix} | \\ \mathbf{v}_2 \\ | \end{bmatrix} + \dots + p_n \begin{bmatrix} | \\ \mathbf{v}_n \\ | \end{bmatrix} + \mathbf{m} = \mathbf{V}\mathbf{p} + \mathbf{m}$$

PCA – Properties of

- These projection vectors form a basis for describing the training data in the Minimum Squared Error (MSE) sense.
- I.e. the eigenvectors with most largest eigenvectors capture most of the signal energy and reconstruction ability.
- N samples of data will have a covariance matrix of rank at most N-1. That means you have at MOST N-1 NON-ZERO eigenvalues. And you ONLY care about these corresponding N-1 eigenvectors.

How do I compute these in Matlab?

- Use MATLAB command “eigs” and “eig”
- “Eigs” is nice because it can find only a few eigenvectors,
- $[v,d]=\text{eigs}(\Sigma, N)$ which finds the N eigenvectors with the largest N eigenvalues.

Practical Issues in computing PCA

- Assume we have image data of size 200x200 pixels.
- This means that our data is 40,000 dimensions!
- What is the size of Covariance Matrix?
$$\Sigma = E(x - \mu)(x - \mu)^T \Rightarrow 40,000 \times 40,000$$
- This is a BIG MATRIX!..infact you CAN NOT create this matrix in MATLAB, no matter how much memory you have!

Also...

- Well we only have N training samples, where $N \ll d$ (d =dimensionality of our problem=40,000).
- Since N is much smaller than d and we will have AT MOST $N-1$ eigenvectors (assuming $N-1$ linearly independent data samples).
- It's not computationally feasible or possible to work with large covariance matrices.

Instead... I show you the Gram Matrix Trick!

- We know that $\Sigma = E(x - \mu)(x - \mu)^T = XX^T$

Must solve $\Sigma v = \lambda v$

$$XX^T v = \lambda v \quad (\text{pre-mult by } X^T) \quad (1)$$

$$X^T XX^T v = \lambda X^T v \quad (v' = X^T v) \quad (2)$$

Solve eigenvalue problem $X^T X v' = \lambda v'$

$X^T X$ is a gram or inner-product matrix, its size is not dependent on dimensionality of data but rather on the number of data samples N . It is of size $N \times N$ and hence easier to compute if $N \ll d$.

Ok..but how do I compute v from v' ?

- Remember Eq(1) and Eq (2):

$$\mathbf{X}\mathbf{X}^T v = \lambda v \quad (1)$$

$$v' = \mathbf{X}^T v \quad (2)$$

Recognizing and subst (2) in (1) we get

$$\mathbf{X}v' = \lambda v$$

Thus $v = \mathbf{X}v'$. We don't care about scaling term because we will anyway make eigenvector orthonormal i.e. $\|v\|=1$.

Lets see an example on real face data.

- We use CMU's AMP Lab facial Expression database.
- 13 people.
- Images are 64x64 cropped and centered facial images.
- Variations are due to varying expressions in the video sequence.
- 75 images in each person's video sequence

AMP Lab Facial Expression DB

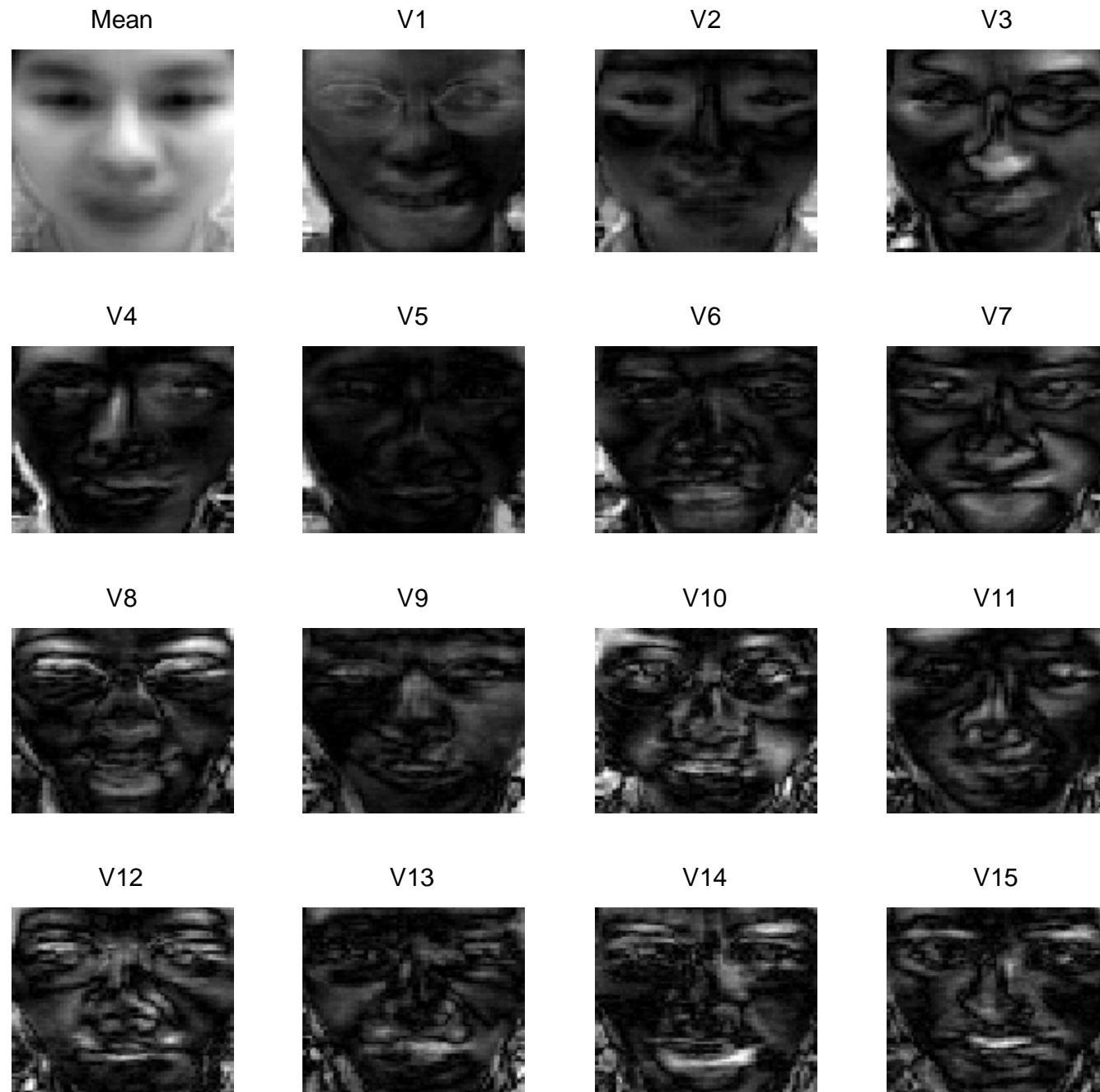


Marios Savvides

Experiment:

- Take the first 5 images per person and use them as training data
- Total of $5 \times 13 = 65$ training images
- Do PCA and compute basis eigenvectors.
- Measure Reconstruction ability
- Perform Dimensionality Reduction (to 3d, i.e. only use a few eigenvectors and look at how data clusters in this 3D linear subspace.)

What do the eigenvectors look like?



How much reconstruction?

- If you want to say reconstruct an image with only 20% MSE (or 80% reconstruction)
- Then use the eigenvectors with largest M eigenvalues from the total of N non-zero eigenvalues satisfying this ratio:

$$reconstruction\% = \frac{\sum_{i=1}^M \lambda_i}{\sum_{i=1}^N \lambda_i} * 100$$

Sum of the eigenvalues of the kept eigenvectors

Sum of all eigenvalues

All 64 eigenvectors..do we need all?



Noisy..not much info..will not contribute to image reconstruction

See how it works...(use only the 3 most dominant eigenvectors)

Mean



+ 230

v1



- 917

v2



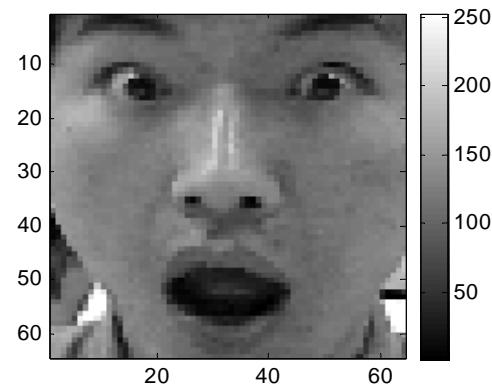
+ 1050

v3



MSE=758.13

=



reconstructed
 $\tilde{\mathbf{X}}$

Original
 \mathbf{X}

How do we compute projection coefficients p?

Mean (m)



+ 230

v1



- 917

v2



+ 1050

v3



x



m



c



Step 1:

Subtract mean image

Step 2:

Project onto
eigenvector

$$\sum_{x=1}^X \sum_{y=1}^Y$$



*

(element multip)

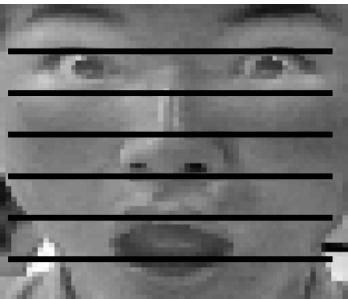
= 230

Projections in vector form

Step 2: Project onto eigenvector

$$\sum_{x=1}^X \sum_{y=1}^Y$$


.* (element multip) = 230


$$\Rightarrow \begin{bmatrix} | \\ | \\ | \\ | \\ | \end{bmatrix} = \mathbf{x}$$

$$\Rightarrow \begin{bmatrix} | \\ | \\ | \\ | \\ | \end{bmatrix} = \mathbf{v}_1$$

$$\mathbf{p}_1 = \mathbf{x}^T \mathbf{v}_1 = 230$$

MSE=1233.16



- If we use only 1 eigenvector, MSE=1233
(max Eigenvalue)

MSE=1027.63



- If we use 2 eigenvectors, MSE=1027

MSE=758.13



- If we use 3 eigenvectors, MSE=758

MSE=634.54



- If we use 4 eigenvectors, MSE=634

MSE=399.08



- If we use 7 eigenvectors, MSE=399

MSE=285.08



- If we use 8 eigenvectors, MSE=285

MSE=216.88



- If we use 13 eigenvectors, MSE=216

MSE=87.93



- If we use 20 eigenvectors, MSE=87

MSE=20.55



- If we use 30 eigenvectors, MSE=20

MSE=6.84



- If we use 45 eigenvectors, MSE=6.8

MSE=2.14



- If we use 50 eigenvectors, MSE=2.1

MSE=0.06



- If we use 60 eigenvectors, MSE=0.06

MSE=0.00



- If we use 64 eigenvectors, MSE=0

(NOTE: We are using ALL NON-ZERO EigenVALUE Eigenvectors!)

Marios Savvides

Feature Extraction via Dimensionality Reduction

- Lets throw away many eigenvectors and only use a couple most dominant ones (this will perform dimensionality reduction)
- We project our data samples onto these vectors and store the projection coefficient for each projected data sample.

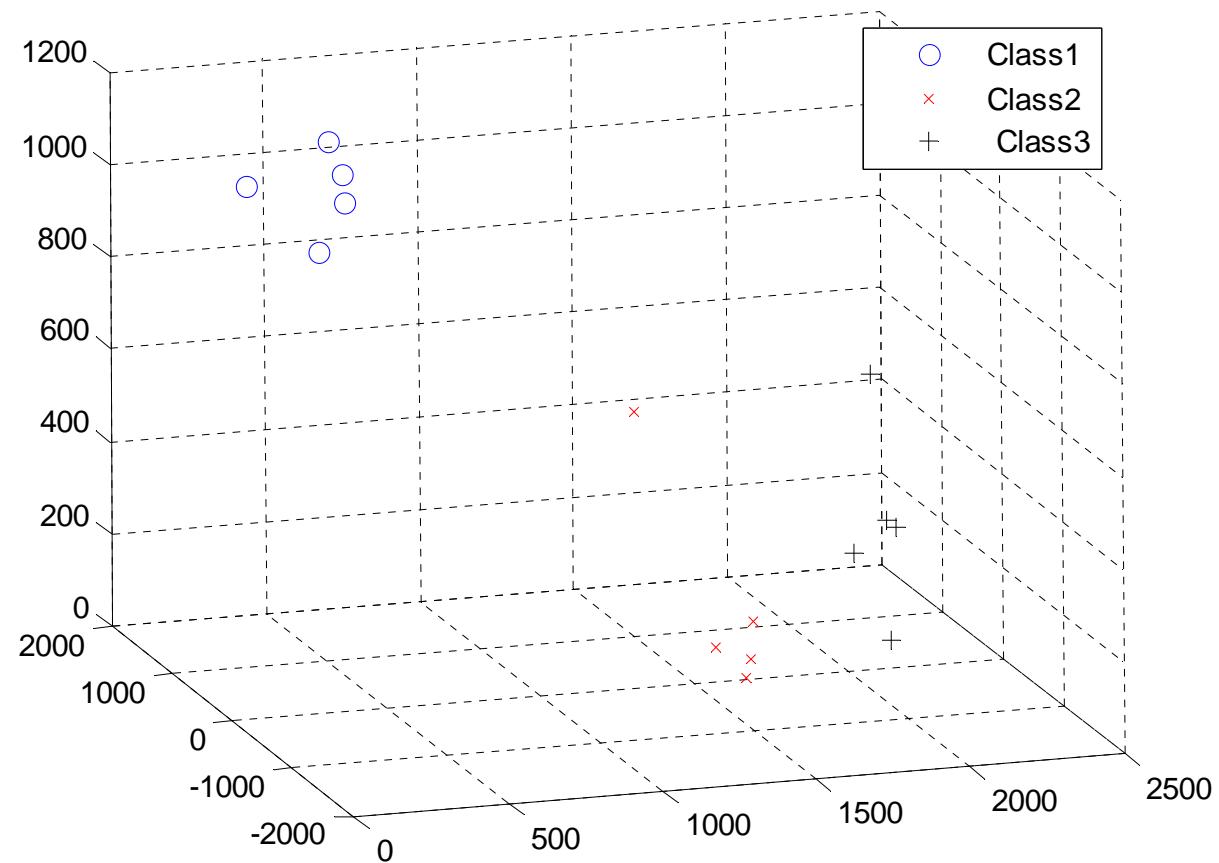
$$\mathbf{p} = \mathbf{V}^T(\mathbf{x} - \mathbf{m})$$

- So if we have N classes with k images per class then we will have $k \times N$ projection vectors \mathbf{p} . We only need to store these vectors and not the original training data to do classification.

E.g.

- In our example we have 64x64 images, however in this toy example we performed PCA (global PCA since it uses all classes) and then only kept the 3 most dominant eigenvectors (i.e. with the 3 largest eigenvalues).
- Project each $64 \times 64 = 4096$ dimensional training image onto this basis to get a 3-dimensional vector. That 3d vector represents each training sample in a 3d subspace.

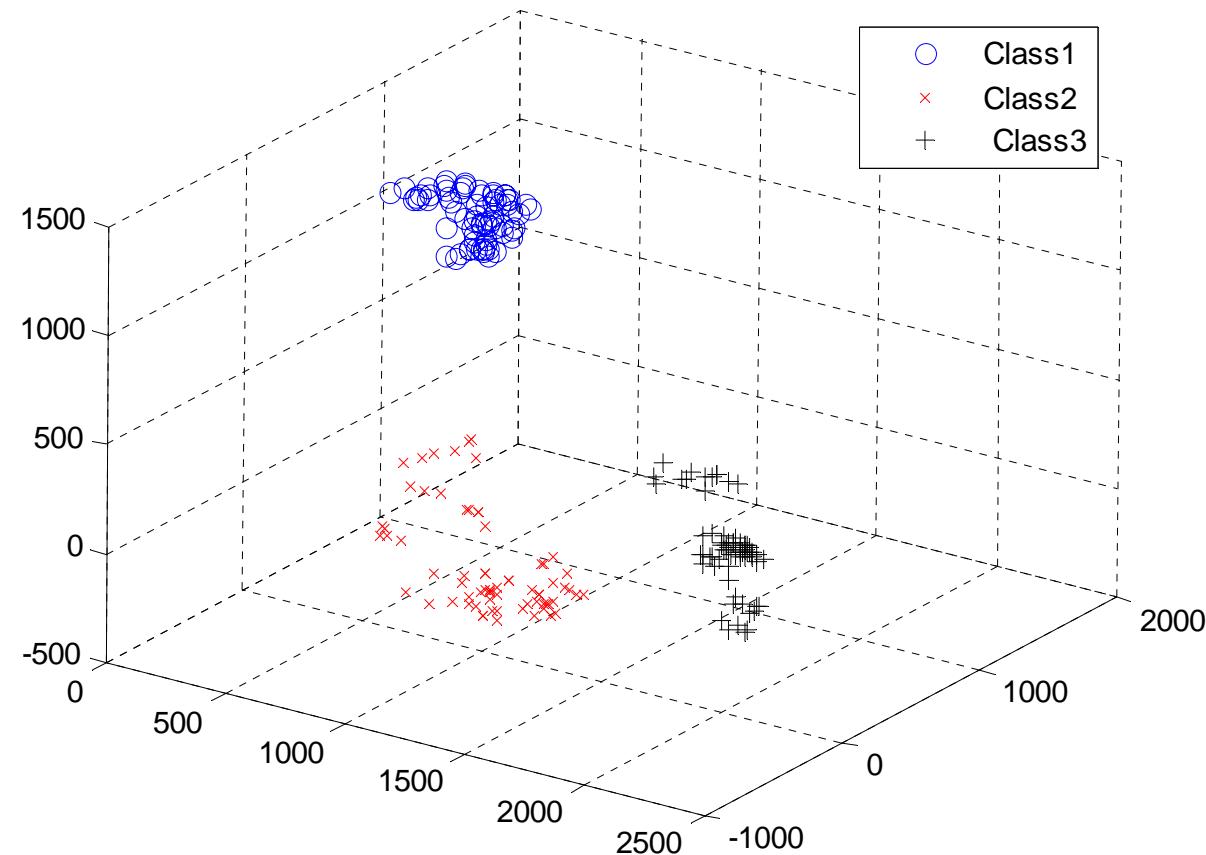
The 3D linear subspace containing the projected training samples from class 1, 2 and 3



How do we classify?

- Given a test face, we project onto this 3D linear subspace (the 3 eigenvectors) to get a 3d-vector.
- Then we can do a simple Nearest Neighbor search (euclidean or L2) distance to all the stored training vectors to find which is closest to and label the data.

Test: We projected all 75 images per person
in the 3D PCA subspace



- We can see the 3 data classes clusters.

PCA Variants

- What I described here was “Global PCA”
i.e. PCA was performed on all classes to find a universal linear subspace.
- However we can also do Individual PCA (IPCA),
i.e. build a subspace for each class using the images from that class.
 - How do we test?
 - We measure the reconstruction error between the test image and the reconstructed image from each subspace.
 - We label the test image with the subspace/basis that yielded the smallest reconstruction error.

Pros & Cons?

- IPCA you need to have more data for each person (won't work with just 1 image/person).
- However, you don't need to re-train the system everytime you add a person in the database (as with Global PCA).

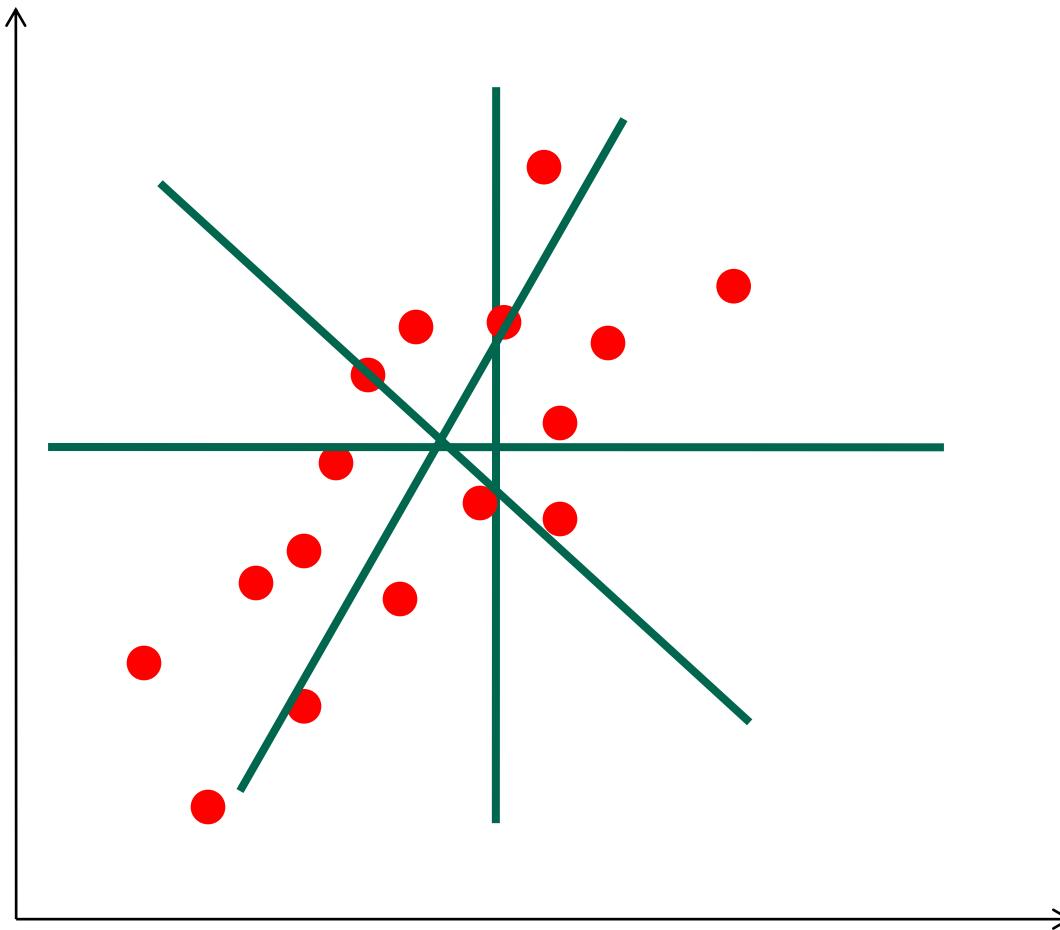
Prof. Marios Savvides

Pattern Recognition Theory

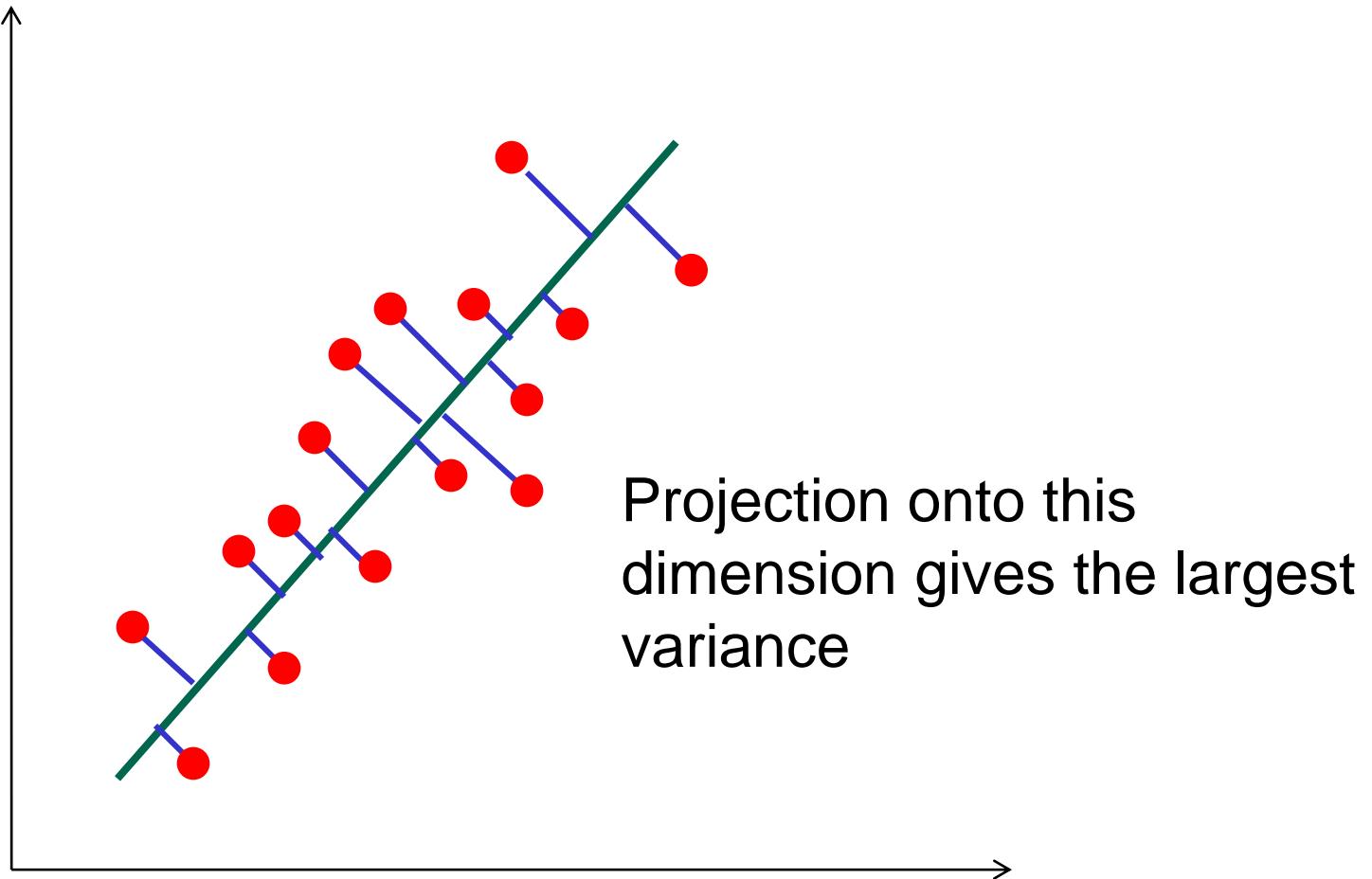
Lecture 7 : Principal Component Analysis II (PCA II)

All graphics from Pattern Classification, Duda, Hart and Stork, Copyright © John Wiley and Sons, 2001

PCA - Recap



PCA - Recap



PCA - Recap

- Find ω that maximizes $\text{Var}(\omega^T x)$

Subject to the constraint that ω is a direction vector of unit norm.

$$L(\omega, \lambda) = \omega^T \Sigma \omega - \lambda (\omega^T \omega - 1)$$

Taking derivative and set to zero. We see that this reduces to an eigen value problem

$$\Sigma \omega = \lambda \omega$$

Points to Note

- $\omega^T \Sigma \omega$ is the variance in the direction ω
- The coraviance matrix is symmetric, so has ORTHOGONAL eigenvectors.

$$\omega_i^T \omega_k = 0 \text{ for all } i \neq k$$

$$\omega_i^T \omega_k = 1 \text{ for all } i = k \quad (\omega^T \omega = 1)$$

- It is positive semidefinite, so has only real and positive eigenvalues.
- They are uncorrelated, and information about how data varies in the direction of one of the eigenvectors tells you nothing about how data varies in the directions of the eigenvectors.
- This means we have an orthogonal basis that we can use to represent the data. Normalize the eigenvectors to unit norm and you will have an orthonormal basis.

Expanding a Signal Using a Basis

- Assume you have a discrete vector signal \mathbf{x}
- You have a set of N basis vectors \mathbf{v}_i which you can use to represent a signal as follows

$$\mathbf{x} = \sum_{i=1}^N p_i \mathbf{v}_i = p_1 \begin{bmatrix} | \\ \mathbf{v}_1 \\ | \end{bmatrix} + p_2 \begin{bmatrix} | \\ \mathbf{v}_2 \\ | \end{bmatrix} + \dots + p_n \begin{bmatrix} | \\ \mathbf{v}_n \\ | \end{bmatrix} = \mathbf{V}\mathbf{p}$$

- The signal is a linear combination of the basis vectors where the p_i are coefficients.

$$\mathbf{V} = \begin{bmatrix} \uparrow & \uparrow & \uparrow \\ \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}$$

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix}$$

Computing the Weight Coefficients

$$\mathbf{x} = \sum_{i=1}^N p_i \mathbf{v}_i = p_1 \begin{bmatrix} | \\ \mathbf{v}_1 \\ | \end{bmatrix} + p_2 \begin{bmatrix} | \\ \mathbf{v}_2 \\ | \end{bmatrix} + \dots + p_n \begin{bmatrix} | \\ \mathbf{v}_n \\ | \end{bmatrix} = \mathbf{V}\mathbf{p}$$

- Since this is an orthogonal basis, we can easily find the coefficients p . These are just the projections of the signal onto each basis.

$$p_1 = \mathbf{x}^T \mathbf{v}_1$$

$$p_2 = \mathbf{x}^T \mathbf{v}_2$$

$$\vdots$$

$$p_n = \mathbf{x}^T \mathbf{v}_n$$

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix} = \begin{bmatrix} \leftarrow & \mathbf{v}_1 & \rightarrow \\ \leftarrow & \mathbf{v}_2 & \rightarrow \\ \vdots & \vdots & \rightarrow \\ \leftarrow & \mathbf{v}_n & \rightarrow \end{bmatrix} \begin{bmatrix} \uparrow \\ \mathbf{x} \\ \downarrow \end{bmatrix} = \mathbf{V}^T \mathbf{x}$$

Model the Variations About the Mean

- Don't forget that you model the variance about the mean!
Don't forget to subtract the global mean before you analyze your test data.

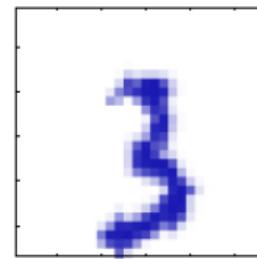
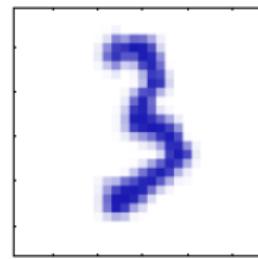
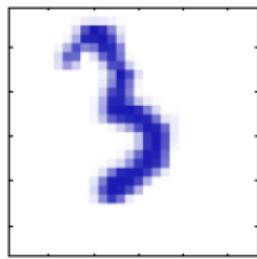
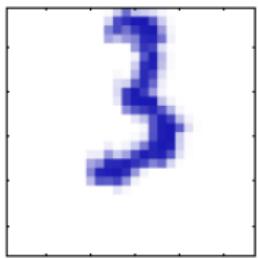
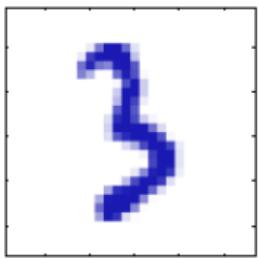
$$\mathbf{p} = \mathbf{V}^T (\mathbf{x} - \mathbf{m})$$

- The mean is the mean of all your training data samples.
- Don't forget to add the mean back before you reconstruct the data !

$$\mathbf{x} = \sum_{i=1}^n p_i \mathbf{v}_i + \mathbf{m} = p_1 \begin{bmatrix} \uparrow \\ \mathbf{v}_1 \\ \downarrow \end{bmatrix} + p_2 \begin{bmatrix} \uparrow \\ \mathbf{v}_2 \\ \downarrow \end{bmatrix} + \cdots + p_n \begin{bmatrix} \uparrow \\ \mathbf{v}_n \\ \downarrow \end{bmatrix} + \mathbf{m} = \mathbf{V}\mathbf{p} + \mathbf{m}$$

A Visual Example

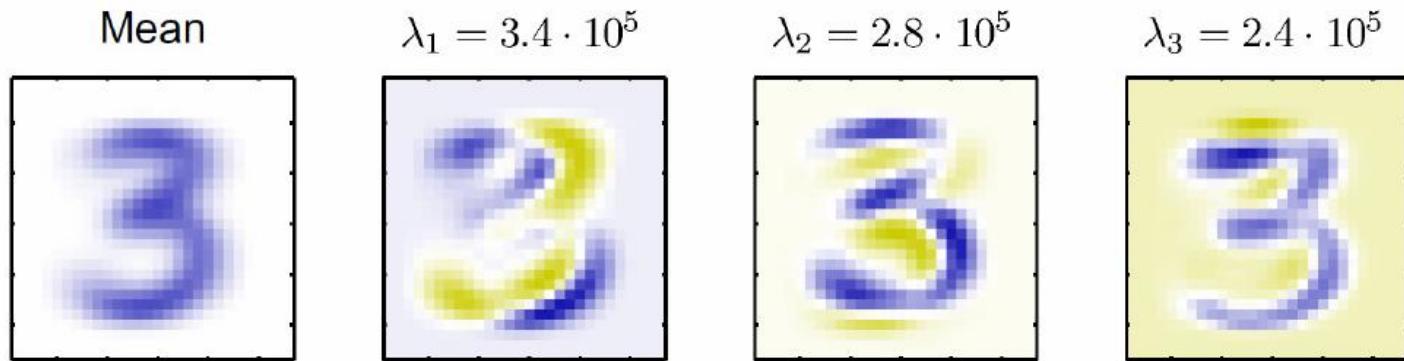
Consider a hand-written digit 3



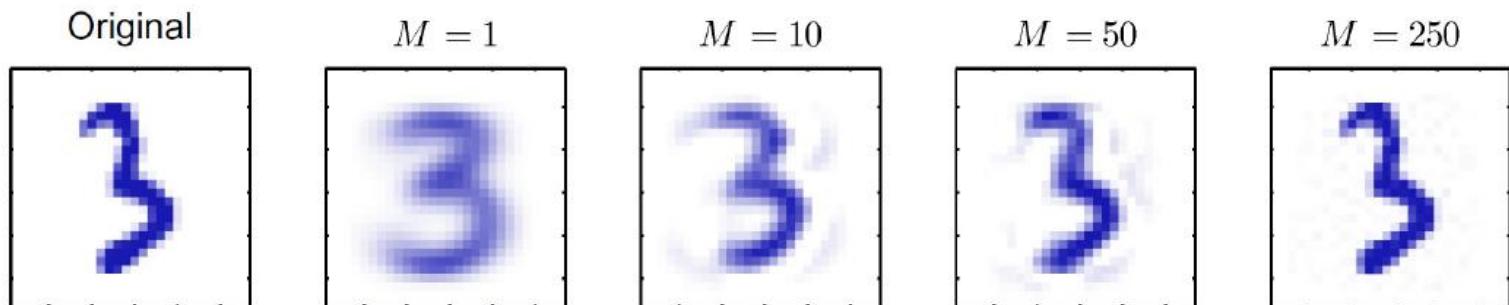
- 28x28 pixel images → number of dimensions, $d = 784$.
- These images were constructed by translating and rotating one image.

A ‘visual’ example

The mean image and the eigen vectors corresponding to the largest three eigen values



Reconstructing the digit using the first M eigen vectors

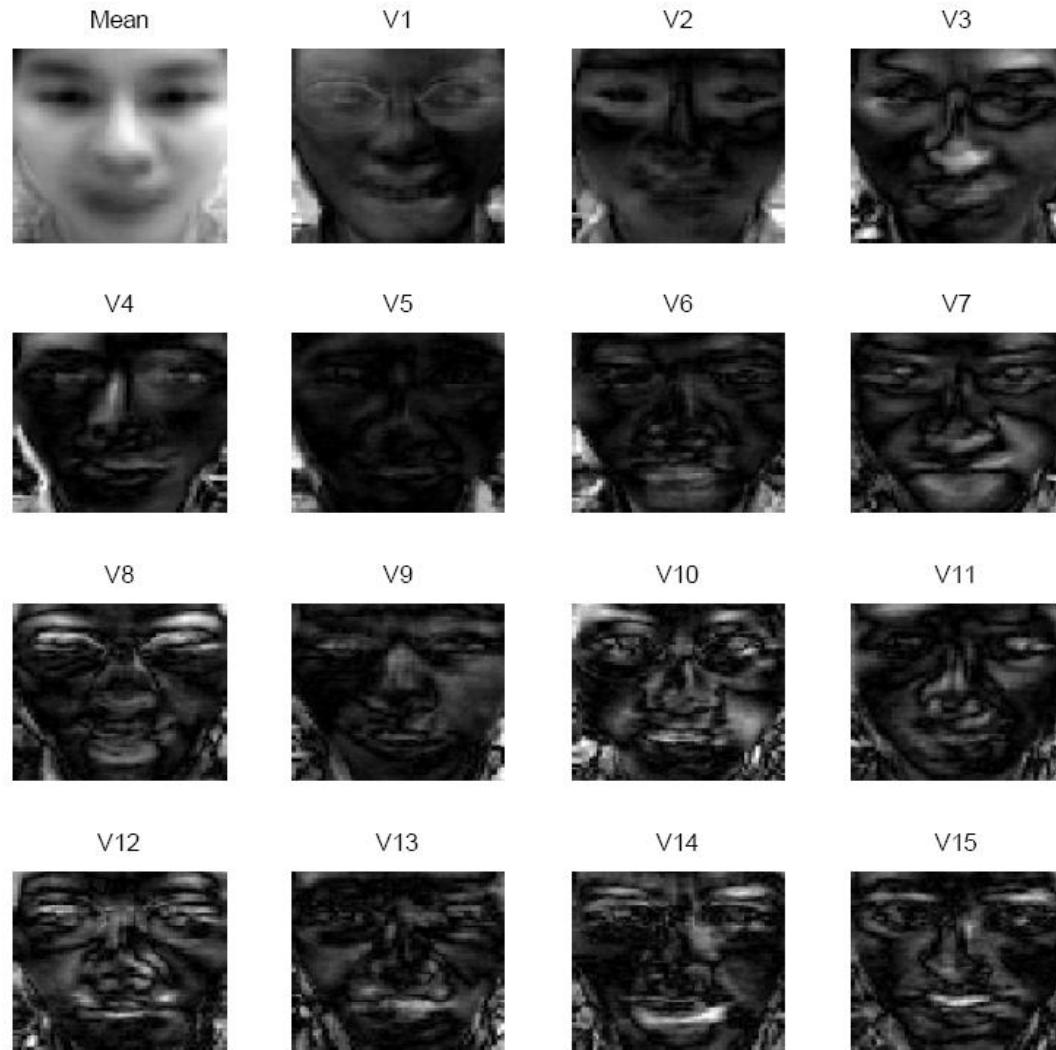


Another Example: Human Faces

- AMP Lab face expression database
 - 13 people
 - Images are 64x64 cropped and aligned.
 - Variations are due to varying expressions in the video sequence.
 - 75 images in each person's sequence
-
- Experiment
 - Take first 5 images per person as training
 - Total $5 \times 13 = 65$ training images
 - Do PCA and extract basis eigenvectors
 - Measure reconstruction ability
 - Perform dimensionality reduction to 3D (take only the first 3 eigenvectors and look at how the data clusters)



PCA Example: Eigenfaces



How Much Reconstruction Error

- If you want to say reconstruct an image with only 20% MSE (or 80% reconstruction)
- Then use the eigenvectors with largest M eigenvalues from the total of N non-zero eigenvalues satisfying this ratio:

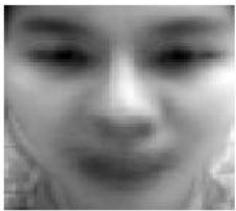
$$\text{reconstruction\%} = \frac{\sum_{i=1}^M \lambda_i}{\sum_{i=1}^N \lambda_i} * 100$$

Sum of the eigenvalues of the kept eigenvectors

Sum of all eigenvalues

Using Only 3 Most Dominant Eigenvectors

Mean

 $+ 230$

v1

 $- 917$

v2

 $+ 1050$

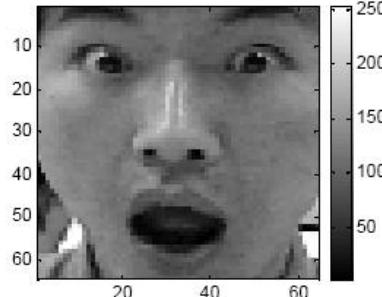
v3



MSE=758.13

 $=$ 

reconstructed



Original

Computing Coefficients

Mean (m)



+ 230

v_1



- 917

v_2



+ 1050

v_3



Step 1:

Subtract mean image



=



Step 2:

Project onto
eigenvector

$$\sum_{x=1}^X \sum_{y=1}^Y$$



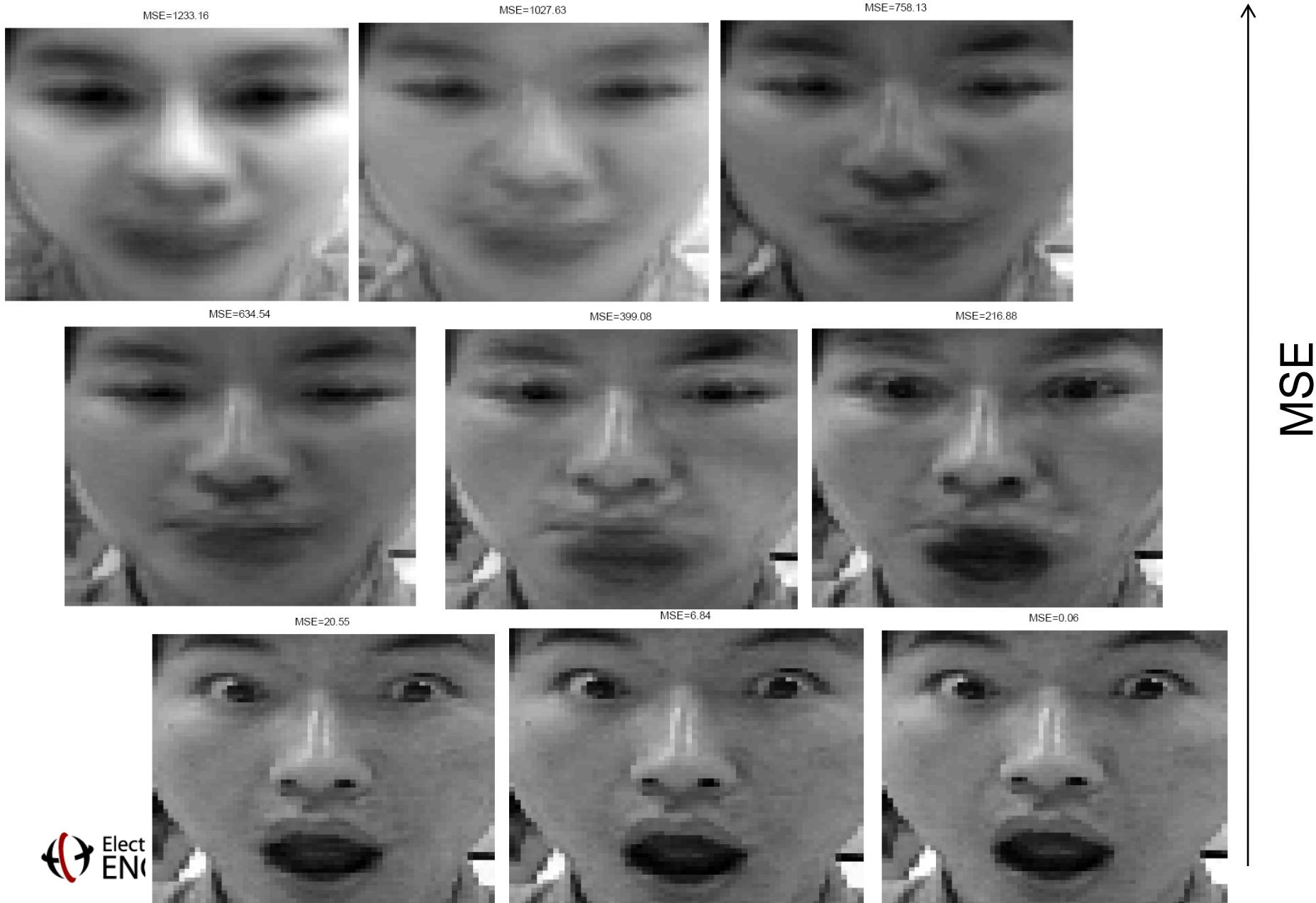
*



(element multip)

= 230

Reconstruction



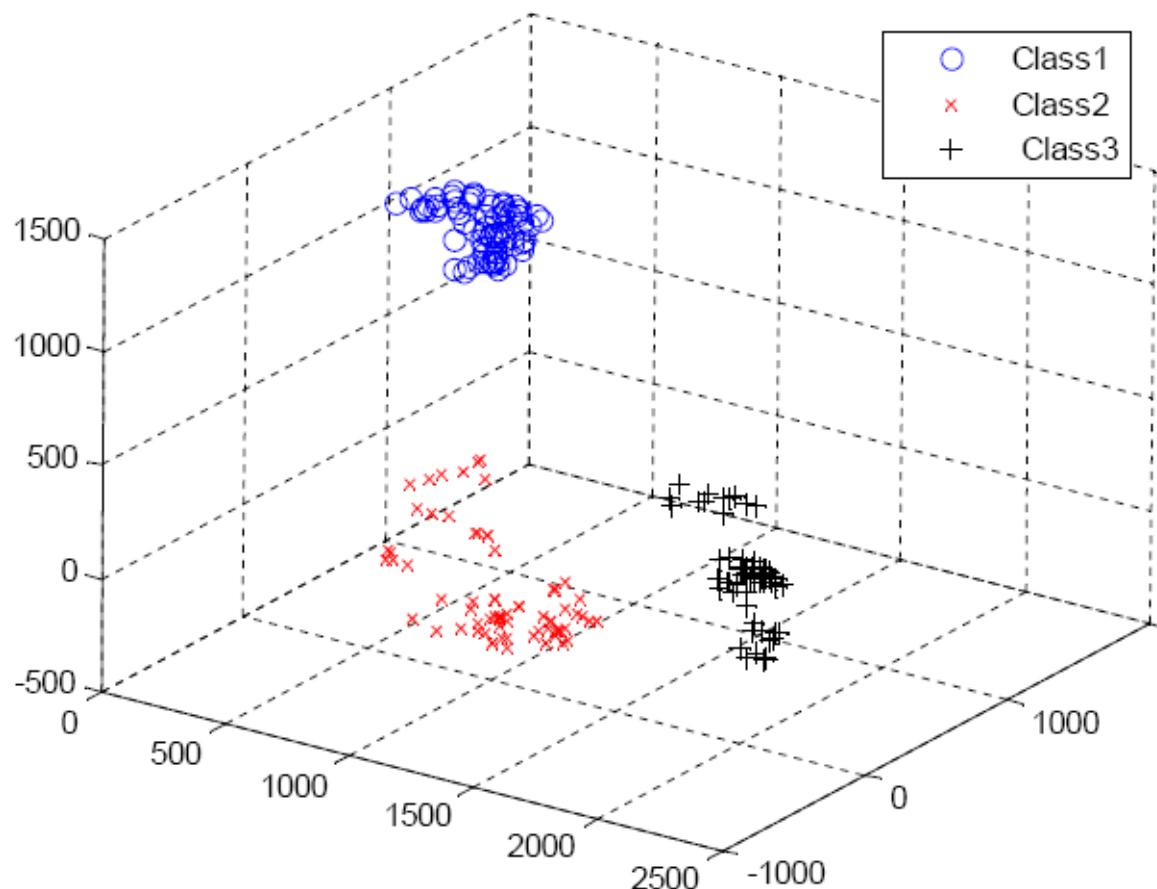
Feature Extraction Via Dimensionality Reduction

- Dimensionality reduction → throw away many eigenvectors and use only a subset (corresponding to the more dominant ones)
- Project the data samples onto these vectors and store the projection coefficients for each data sample → we thus obtain a **feature vector**.
- Why feature reduction? – because now we need to store only the projection coefficients and not the entire image!

Dimensionality Reduction to 3D Subspace

How to classify?

- Project onto the first 3 eigen vectors. Now we can visualize the coefficients in a 3D space.
- Use Nearest-Neighbor search (based on a distance metric) to all stored vectors and find the closest in order to classify.



Practical Issues in Computing PCA

- Assume we have image data of size 200x200 pixels.
- This means our data vector is $d=40,000$ dimensions.
- The size the covariance matrix will be $40,000 \times 40,000$
- This is a BIG MATRIX. You will run out of memory when you try to create a matrix this big.
- Also when usually we have N training samples, where $N \ll d$.
- We will have at most $N-1$ eigenvectors and nonzero-eigenvalues.
- Computing the big $40,000 \times 40,000$ matrix is therefore a waste.

The Gram Matrix Trick

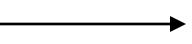
- We know that $\Sigma = E[(\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^T] = \mathbf{X}\mathbf{X}^T$

- WE must solve $\Sigma\mathbf{v} = \lambda\mathbf{v}$

$$\mathbf{X}\mathbf{X}^T\mathbf{v} = \lambda\mathbf{v}$$

Premultiply by \mathbf{X}^T 

$$\mathbf{X}^T\mathbf{X}\mathbf{X}^T\mathbf{v} = \lambda\mathbf{X}^T\mathbf{v}$$

Replace $\mathbf{X}^T\mathbf{v}$ by \mathbf{v}'   Now solve this new eigen value/eigen vector problem

$$\mathbf{X}^T\mathbf{X}\mathbf{v}' = \lambda\mathbf{v}'$$

- $\mathbf{X}^T\mathbf{X}$ is a gram or inner-product matrix. It is of size $N \times N$, which is not dependent on the dimensionality of the data (d), but rather on the number of data samples N . It will be easier to compute if $N \ll d$.

The Gram Matrix Trick

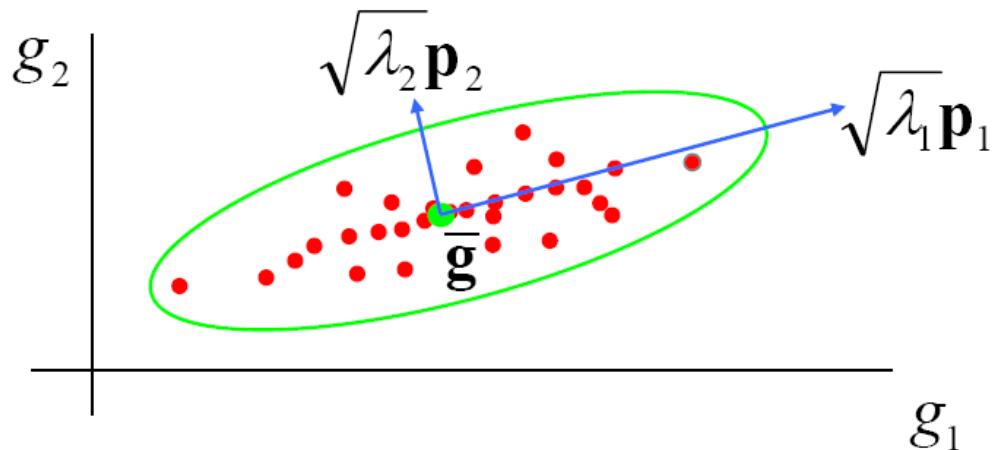
- Now that we've obtained all the eigenvectors \mathbf{v}' of the Gram matrix, how do we obtain the eigenvectors of the covariance matrix $\mathbf{X}\mathbf{X}^T$?
- Remember from the previous equations that:

$$\mathbf{X}\mathbf{X}^T \mathbf{v} = \lambda \mathbf{v} \quad \text{and} \quad \mathbf{v}' = \mathbf{X}^T \mathbf{v}$$

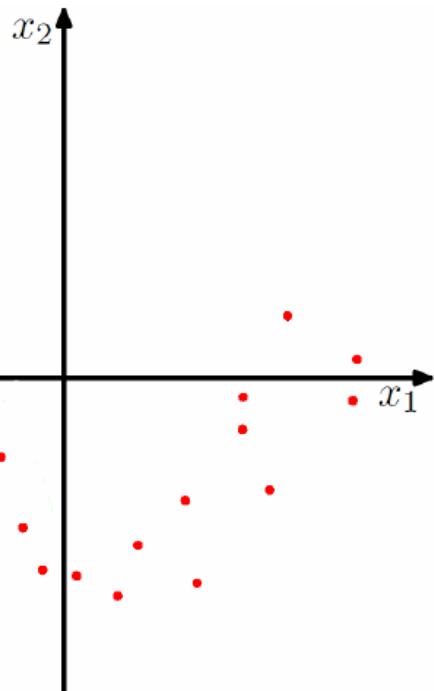
- Substitute the second equation in the first equation we obtain
- $$\mathbf{X}\mathbf{v}' = \lambda \mathbf{v}$$
- Thus $\mathbf{v} \cong \mathbf{X}\mathbf{v}'$. We do not care about the scaling term because we will unit-normalize the eigenvectors anyway to obtain an orthonormal basis.

PCA: Afterthoughts

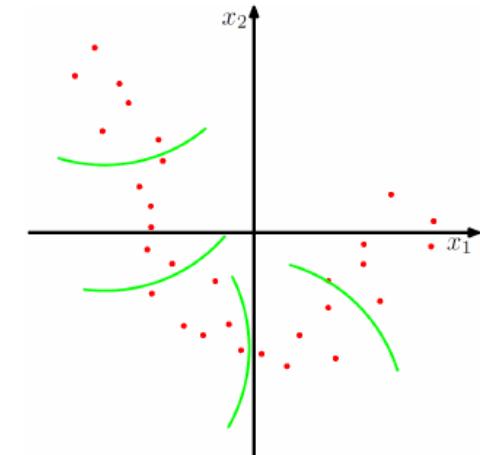
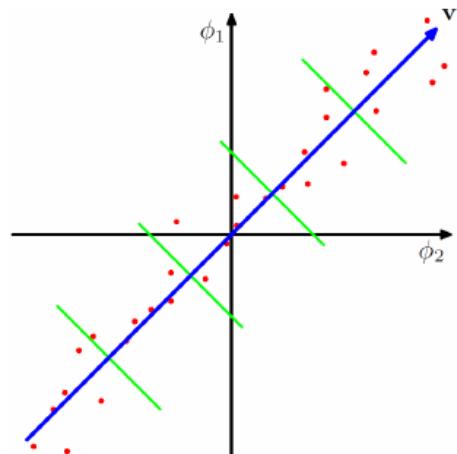
- The mean and the covariance specify a Gaussian model
- So PCA seems to be related to fitting a Gaussian model to the data
 - Find eigen vectors & eigen values of covariance matrix
 - Transform the original coordinate system by translation and rotation into a new space
 - Mean becomes origin
 - “Largest” eigen vector becomes 1st axis, second “largest” eigvec becomes 2nd axis, and so on.
- PCA de-correlates the data.
- Dimensionality reduction is obtained by using only a subset of the new axes that account for most of the variance.



Later: Kernel PCA

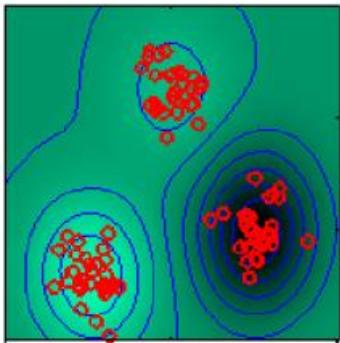


- If data is not normally distributed, it could be much more efficient to project the data on a curve
- Finding this curve is much harder than the linear case → Kernels come to the rescue.
- Map data to higher dimensional space
- Do standard PCA there
- Corresponds to non-linear PCA in lower dimensional space.

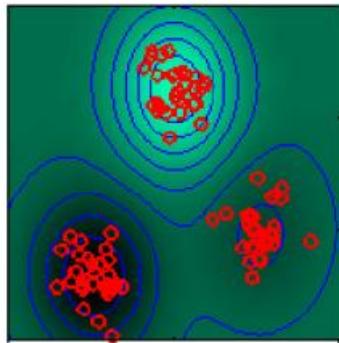


Later: Kernel PCA

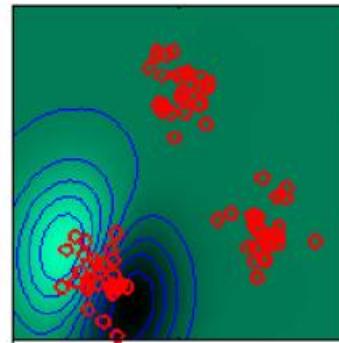
Eigenvalue=21.72



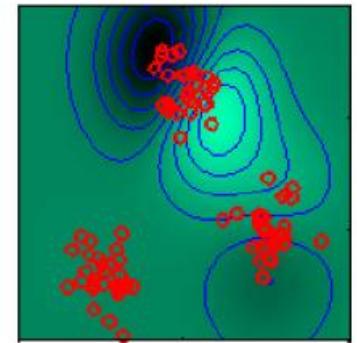
Eigenvalue=21.65



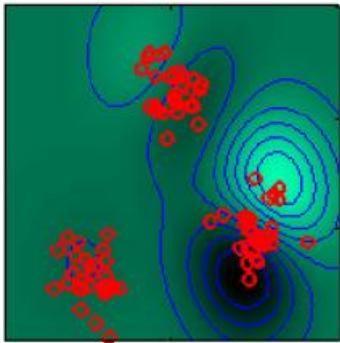
Eigenvalue=4.11



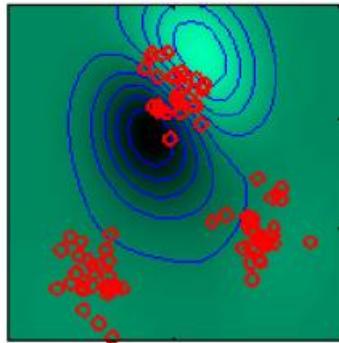
Eigenvalue=3.93



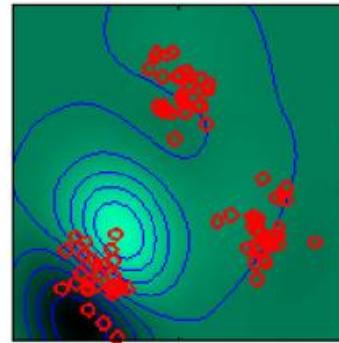
Eigenvalue=3.66



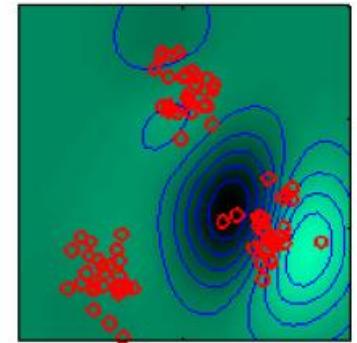
Eigenvalue=3.09



Eigenvalue=2.60



Eigenvalue=2.53



KernelPCA with Gaussian kernel. Data is clustered. First two components encode cluster. Higher components encode structure within clusters

Recap

- PCA
- Using an Eigen basis
- Reconstruction Error
- PCA on Faces
- Gram Trick

Prof. Marios Savvides

Pattern Recognition Theory

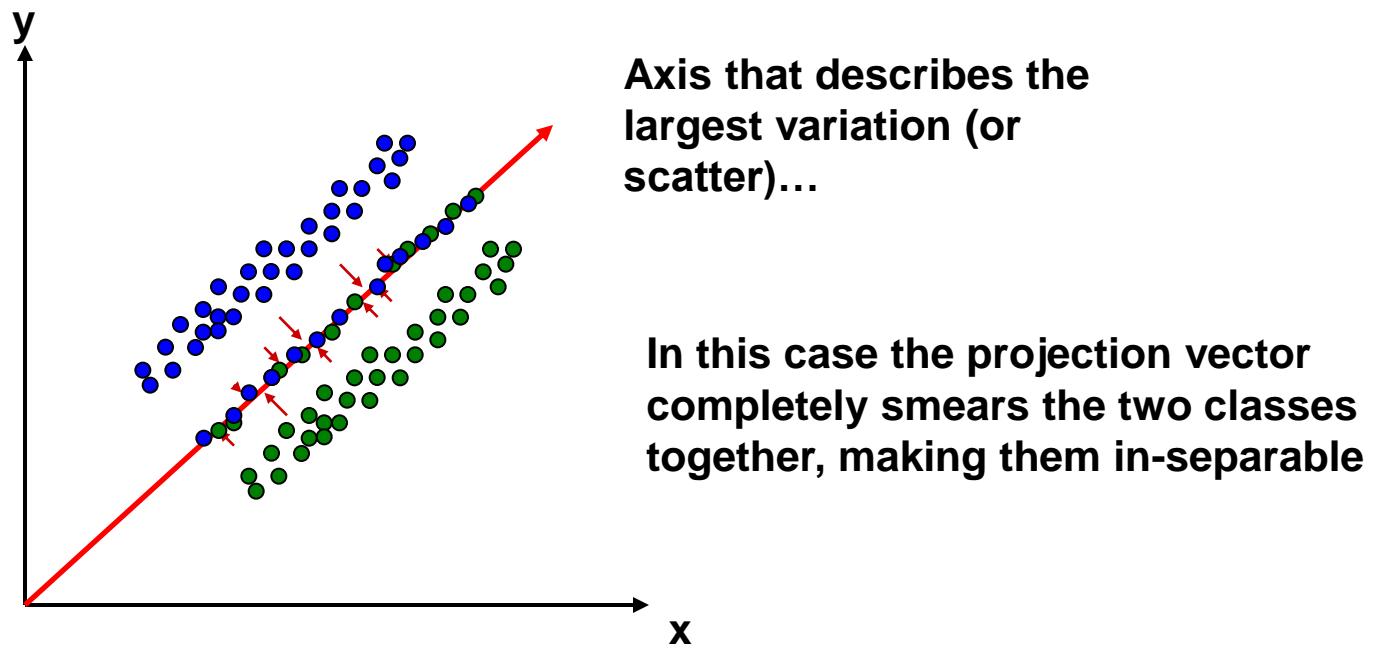
Lecture 8 : Linear Discriminant Analysis

All graphics from Pattern Classification, Duda, Hart and Stork, Copyright © John Wiley and Sons, 2001

What is PCA?

What are we trying to do?

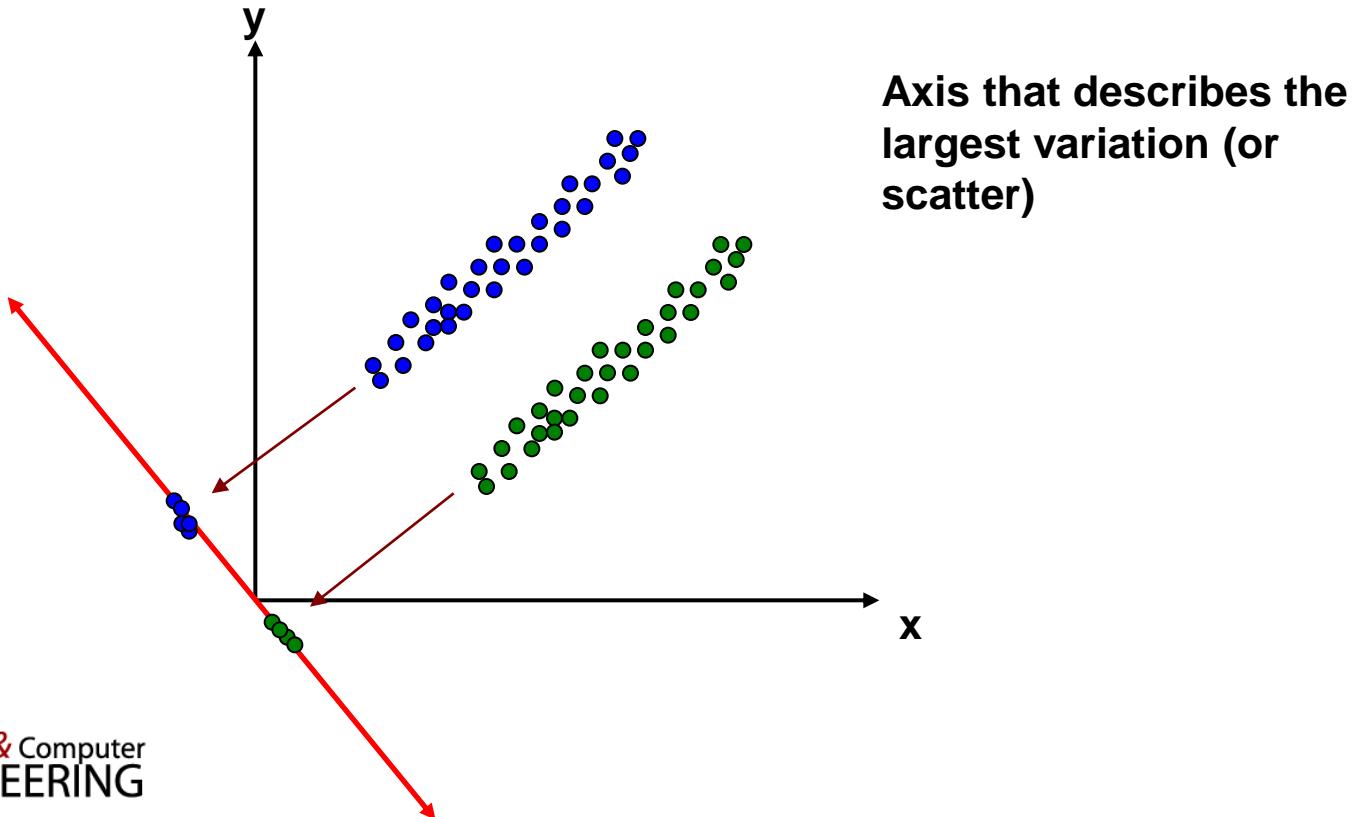
- We want to find projections of data (i.e. direction vectors that we can project the data on to) that describe the maximum variation.



What is LDA?

What are we trying to do?

- We want to find projections that separate the classes.
(Assume unimodal Gaussian modes – maximize distance between two means and minimize variance
=>will lead to minimize overall probability of error)



Case 1: Simple 2 Class Problem

- We want to maximize the distance between the projected means:

e.g. maximize $|(\tilde{\mu}_1 - \tilde{\mu}_2)|^2$

Where $\tilde{\mu}_1$ is the projected mean μ_1 of class onto LDA direction vector \mathbf{w} , i.e.

$$\tilde{\mu}_1 = \mathbf{w}^T \mu_1$$

and for class 2: $\tilde{\mu}_2 = \mathbf{w}^T \mu_2$ thus

$$\begin{aligned} |(\tilde{\mu}_1 - \tilde{\mu}_2)|^2 &= |(\mathbf{w}^T \mu_1 - \mathbf{w}^T \mu_2)|^2 \\ &= \mathbf{w}^T (\mu_1 - \mu_2) (\mu_1 - \mu_2)^T \mathbf{w} \\ &= \mathbf{w}^T \mathbf{S}_B \mathbf{w} \end{aligned}$$

Between Class Scatter Matrix S_B

$$\begin{aligned}(\tilde{\mu}_1 - \tilde{\mu}_2)^2 &= (\mathbf{w}^T \boldsymbol{\mu}_1 - \mathbf{w}^T \boldsymbol{\mu}_2)^2 \\&= \mathbf{w}^T (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \mathbf{w} \\&= \mathbf{w}^T S_B \mathbf{w}\end{aligned}$$

We want to maximize $\mathbf{w}^T S_B \mathbf{w}$ where S_B is the between class scatter matrix defined as:

$$S_B = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$$

NOTE: S_B is rank 1. This will be useful later on to find closed form solution for 2-class LDA

We also want to minimize....

- The variance or scatter of the projected samples from each class (i.e. we want to make each class more compact or closer to its mean). The scatter from class 1 defined as s_1 is given as

$$\tilde{s}_1^2 = \sum_{i=1}^{N_1} (\tilde{x}_i - \tilde{\mu}_1)^2$$

- Thus we want to minimize the scatter of class 1 and class 2 in projected space, i.e.

minimize the total scatter $\tilde{s}_1^2 + \tilde{s}_2^2$

Fisher Linear Discriminant Criterion Function

- Objective [1]-We want to **maximize** the between class scatter defined as: $|(\tilde{\mu}_1 - \tilde{\mu}_2)|^2$
- [2]- We want to **minimize** the within-class scatter.

$$\tilde{s}_1^2 + \tilde{s}_2^2$$

- Thus we define our objective function $J(\mathbf{w})$ as the following ratio that we want to **maximize** in order to achieve [1] and [2]:

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

Scatter s_1 of Class 1

- Thus we want to find the vector \mathbf{w} that maximizes $J(\mathbf{w})$.
- Lets expand on scatter

$$\tilde{s}_1^2 = \sum_{i=1}^{N_1} (\tilde{x}_i - \tilde{\mu}_1)^2$$

$$= \sum_{i=1}^{N_1} (\mathbf{w}^T \mathbf{x}_i - \mathbf{w}^T \boldsymbol{\mu}_1)^2$$

$$= \sum_{i=1}^{N_1} \mathbf{w}^T (\mathbf{x}_i - \boldsymbol{\mu}_1)(\mathbf{x}_i - \boldsymbol{\mu}_1)^T \mathbf{w}$$

$$= \mathbf{w}^T \mathbf{S}_1 \mathbf{w}$$

Same for Scatter s_2 of Class 2

$$\tilde{s}_2^2 = \sum_{i=1}^{N_2} (\tilde{x}_i - \tilde{\mu}_2)^2$$

$$= \sum_{i=1}^{N_2} (\mathbf{w}^T \mathbf{x}_i - \mathbf{w}^T \boldsymbol{\mu}_2)^2$$

$$= \sum_{i=1}^{N_2} \mathbf{w}^T (\mathbf{x}_i - \boldsymbol{\mu}_2) (\mathbf{x}_i - \boldsymbol{\mu}_2)^T \mathbf{w}$$

$$= \mathbf{w}^T \mathbf{S}_2 \mathbf{w}$$

Total Within-Class Scatter Matrix

- We want to minimize total *within*-class scatter. i.e.

$$\tilde{s}_1^2 + \tilde{s}_2^2$$

- Which is equivalent to $\mathbf{w}^T \mathbf{S}_w \mathbf{w}$

$$\mathbf{S}_w = \sum_{i=1}^C \sum_{j=1}^{N_i} (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T$$

$C=2$, N_i =No. of images in i th class

Solving LDA

- Maximize $J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$
- We need to find the optimal \mathbf{w} which will maximize the above ratio (or quotient).
- So what do we do now?
Take derivative and solve for \mathbf{w}

Solving LDA

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \frac{2\mathbf{w}^T \mathbf{S}_W \mathbf{w} \mathbf{S}_B \mathbf{w} - 2\mathbf{w}^T \mathbf{S}_B \mathbf{w} \mathbf{S}_W \mathbf{w}}{(\mathbf{w}^T \mathbf{S}_W \mathbf{w})^2} = 0$$

$$\Rightarrow \frac{\mathbf{w}^T \mathbf{S}_W \mathbf{w} \mathbf{S}_B \mathbf{w} - \mathbf{w}^T \mathbf{S}_B \mathbf{w} \mathbf{S}_W \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} = 0$$

$$\Rightarrow \mathbf{S}_B \mathbf{w} - J(\mathbf{w}) \mathbf{S}_W \mathbf{w} = 0$$

Solving LDA

$$\mathbf{S}_B \mathbf{w} - J(\mathbf{w}) \mathbf{S}_W \mathbf{w} = 0$$

$$\mathbf{S}_B \mathbf{w} - \lambda \mathbf{S}_W \mathbf{w} = 0$$

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_W \mathbf{w}$$

Generalized Eigenvalue problem

$$\mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{w} = \lambda \mathbf{w}$$

If \mathbf{S}_W is non-singular and invertible

We want to
maximize $J(\mathbf{w})$
thus we want
the eigenvector
 \mathbf{w} with the
largest
eigenvalue!

Prof. Marios Savvides

Pattern Recognition Theory

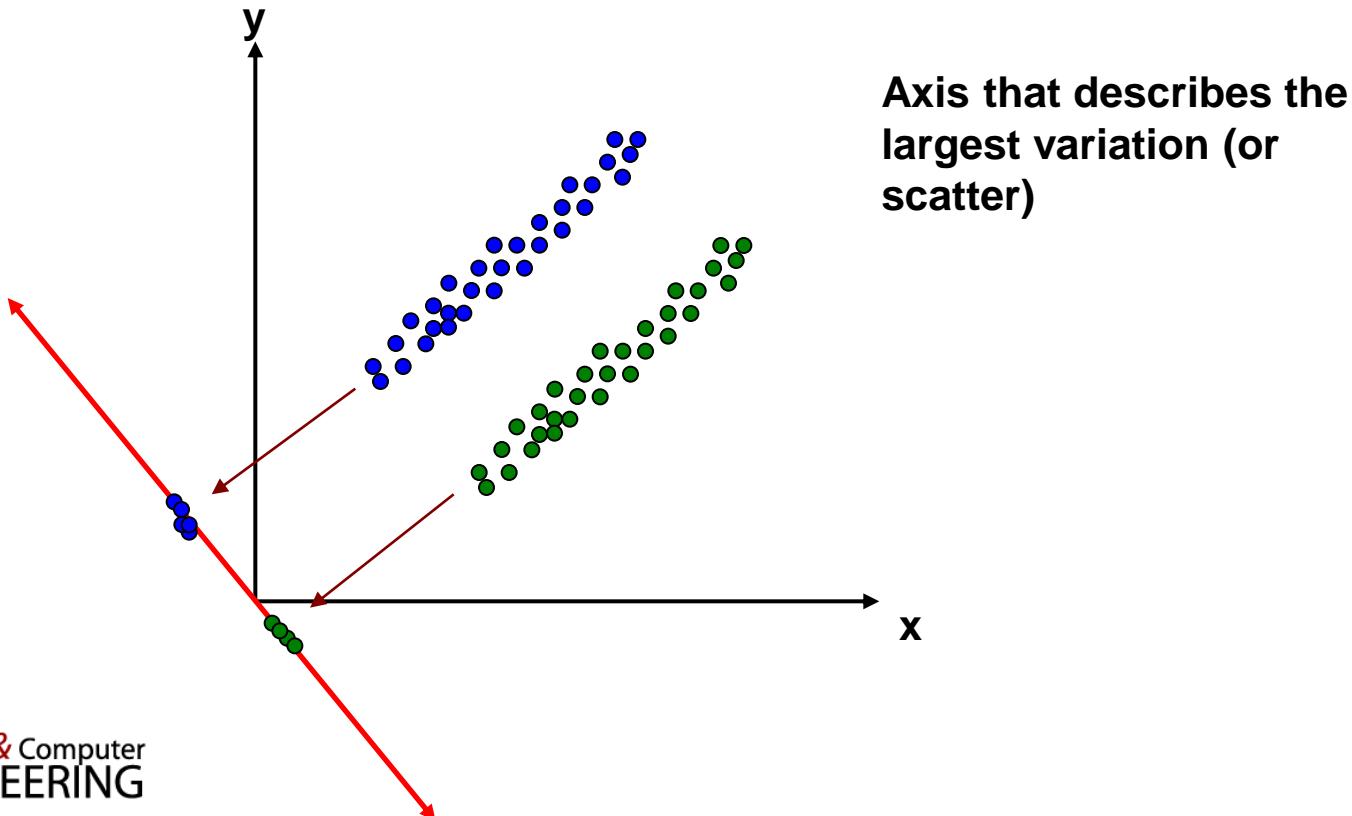
Lecture 9 : Linear Discriminant Analysis

All graphics from Pattern Classification, Duda, Hart and Stork, Copyright © John Wiley and Sons, 2001

What is LDA?

What are we trying to do?

- We want to find projections that separate the classes.
(Assume unimodal Gaussian modes – maximize distance between two means and minimize variance
=>will lead to minimize overall probability of error)



Case 1: Simple 2 Class Problem

- We want to maximize the distance between the projected means:

e.g. maximize $|(\tilde{\mu}_1 - \tilde{\mu}_2)|^2$

Where $\tilde{\mu}_1$ is the projected mean μ_1 of class onto LDA direction vector \mathbf{w} , i.e.

$$\tilde{\mu}_1 = \mathbf{w}^T \mu_1$$

and for class 2: $\tilde{\mu}_2 = \mathbf{w}^T \mu_2$ thus

$$\begin{aligned} |(\tilde{\mu}_1 - \tilde{\mu}_2)|^2 &= |(\mathbf{w}^T \mu_1 - \mathbf{w}^T \mu_2)|^2 \\ &= \mathbf{w}^T (\mu_1 - \mu_2) (\mu_1 - \mu_2)^T \mathbf{w} \\ &= \mathbf{w}^T \mathbf{S}_B \mathbf{w} \end{aligned}$$

Between Class Scatter Matrix S_B

$$\begin{aligned}(\tilde{\mu}_1 - \tilde{\mu}_2)^2 &= (\mathbf{w}^T \boldsymbol{\mu}_1 - \mathbf{w}^T \boldsymbol{\mu}_2)^2 \\&= \mathbf{w}^T (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \mathbf{w} \\&= \mathbf{w}^T S_B \mathbf{w}\end{aligned}$$

We want to maximize $\mathbf{w}^T S_B \mathbf{w}$ where S_B is the between class scatter matrix defined as:

$$S_B = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$$

NOTE: S_B is rank 1. This will be useful later on to find closed form solution for 2-class LDA

We also want to minimize....

- The variance or scatter of the projected samples from each class (i.e. we want to make each class more compact or closer to its mean). The scatter from class 1 defined as s_1 is given as

$$\tilde{s}_1^2 = \sum_{i=1}^{N_1} (\tilde{x}_i - \tilde{\mu}_1)^2$$

- Thus we want to minimize the scatter of class 1 and class 2 in projected space, i.e.

minimize the total scatter $\tilde{s}_1^2 + \tilde{s}_2^2$

Fisher Linear Discriminant Criterion Function

- Objective [1]-We want to **maximize** the between class scatter defined as: $|(\tilde{\mu}_1 - \tilde{\mu}_2)|^2$
- [2]- We want to **minimize** the within-class scatter.

$$\tilde{s}_1^2 + \tilde{s}_2^2$$

- Thus we define our objective function $J(\mathbf{w})$ as the following ratio that we want to **maximize** in order to achieve [1] and [2]:

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

Scatter s_1 of Class 1

- Thus we want to find the vector \mathbf{w} that maximizes $J(\mathbf{w})$.
- Lets expand on scatter

$$\tilde{s}_1^2 = \sum_{i=1}^{N_1} (\tilde{x}_i - \tilde{\mu}_1)^2$$

$$= \sum_{i=1}^{N_1} (\mathbf{w}^T \mathbf{x}_i - \mathbf{w}^T \boldsymbol{\mu}_1)^2$$

$$= \sum_{i=1}^{N_1} \mathbf{w}^T (\mathbf{x}_i - \boldsymbol{\mu}_1)(\mathbf{x}_i - \boldsymbol{\mu}_1)^T \mathbf{w}$$

$$= \mathbf{w}^T \mathbf{S}_1 \mathbf{w}$$

Same for Scatter s_2 of Class 2

$$\tilde{s}_2^2 = \sum_{i=1}^{N_2} (\tilde{x}_i - \tilde{\mu}_2)^2$$

$$= \sum_{i=1}^{N_2} (\mathbf{w}^T \mathbf{x}_i - \mathbf{w}^T \boldsymbol{\mu}_2)^2$$

$$= \sum_{i=1}^{N_2} \mathbf{w}^T (\mathbf{x}_i - \boldsymbol{\mu}_2) (\mathbf{x}_i - \boldsymbol{\mu}_2)^T \mathbf{w}$$

$$= \mathbf{w}^T \mathbf{S}_2 \mathbf{w}$$

Total Within-Class Scatter Matrix

- We want to minimize total *within*-class scatter. i.e.

$$\tilde{s}_1^2 + \tilde{s}_2^2$$

- Which is equivalent to $\mathbf{w}^T \mathbf{S}_w \mathbf{w}$

$$\mathbf{S}_w = \sum_{i=1}^C \sum_{j=1}^{N_i} (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T$$

$C=2$, N_i =No. of images in i th class

Solving LDA

- Maximize $J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$
- We need to find the optimal \mathbf{w} which will maximize the above ratio (or quotient).
- So what do we do now?
Take derivative and solve for \mathbf{w}

Solving LDA

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \frac{2\mathbf{w}^T \mathbf{S}_W \mathbf{w} \mathbf{S}_B \mathbf{w} - 2\mathbf{w}^T \mathbf{S}_B \mathbf{w} \mathbf{S}_W \mathbf{w}}{(\mathbf{w}^T \mathbf{S}_W \mathbf{w})^2} = 0$$

$$\Rightarrow \frac{\mathbf{w}^T \mathbf{S}_W \mathbf{w} \mathbf{S}_B \mathbf{w} - \mathbf{w}^T \mathbf{S}_B \mathbf{w} \mathbf{S}_W \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} = 0$$

$$\Rightarrow \mathbf{S}_B \mathbf{w} - J(\mathbf{w}) \mathbf{S}_W \mathbf{w} = 0$$

Solving LDA

$$\mathbf{S}_B \mathbf{w} - J(\mathbf{w}) \mathbf{S}_W \mathbf{w} = 0$$

$$\mathbf{S}_B \mathbf{w} - \lambda \mathbf{S}_W \mathbf{w} = 0$$

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_W \mathbf{w}$$

Generalized Eigenvalue problem

$$\mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{w} = \lambda \mathbf{w}$$

If \mathbf{S}_W is non-singular and invertible

We want to
maximize $J(\mathbf{w})$
thus we want
the eigenvector
 \mathbf{w} with the
largest
eigenvalue!

Special Case: LDA Solution for 2 Class Problems

- Lets replace what \mathbf{S}_B is for two classes and see how we can simplify to get a closed form solution
i.e. a solution of the vector \mathbf{w} for the 2-class case.
- We know that in two class case, there is only 1 \mathbf{w} vector. Lets use this knowledge cleverly...

S_B is Rank 1

$$S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T = \mathbf{m}\mathbf{m}^T$$

$$S_B = \mathbf{m}\mathbf{m}^T = \begin{bmatrix} & & & \\ | & | & | & \\ m(1)\mathbf{m} & m(2)\mathbf{m} & m(N)\mathbf{m} \\ & | & | & | \end{bmatrix}$$

S_B has only 1 linearly independent column vector => Rank 1 matrix

2-Class LDA

$$\mathbf{S}_B = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$$

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_w \mathbf{w}$$

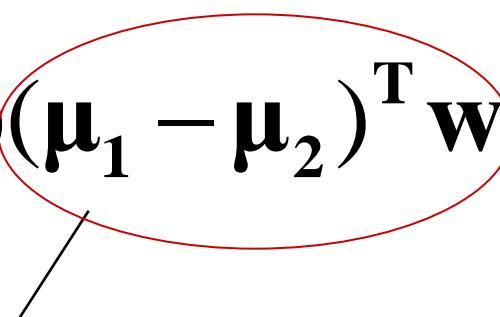
Basically in this generalized eigenvalue/eigenvector problem, the number of valid eigenvectors with non-zero eigenvalue is determined by the *MIN* rank of matrix \mathbf{S}_B and \mathbf{S}_w .

So *in this case*, there is only 1 valid eigenvector with a non-zero eigenvalue! i.e. there is only one valid \mathbf{w} vector solution.

2-Class LDA

Simplify the 2 class case:

$$(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \mathbf{w} = \lambda S_w \mathbf{w}$$


$$(\mu_1 - \mu_2)^T \mathbf{w} = scalar = \beta$$

Which gives

$$(\mu_1 - \mu_2) \beta = \lambda S_w \mathbf{w}$$

2-Class LDA - Closed Form Solution

$$(\mu_1 - \mu_2) \beta = \lambda S_w w$$

$$(\mu_1 - \mu_2) = \frac{\lambda}{\beta} S_w w$$

$$S_w^{-1} (\mu_1 - \mu_2) = \frac{\lambda}{\beta} w$$

Since we can normalize w , we don't have to worry about the constants

$$w = S_w^{-1} (\mu_1 - \mu_2)$$

2-Class LDA - Closed Form Solution

Nice closed form solution for 2-class LDA.

$$\mathbf{W} = \mathbf{S}_w^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

We know that this is the right solution as we showed before that there is only ONE solution, so the closed-form solution we found is the optimal one.

Multi-Class LDA

- What if we have more than 2 classes...what then?
- Answer is we need more than one w projection vector to provide separability.
- Lets look at our math framework to see what changes.

Multi-Class LDA

- Maximize

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$

- Lets start with the Between-Class Scatter matrix for 2 class.

$$\mathbf{S}_B = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$$

- However \mathbf{S}_B now is the between class scatter matrix for many classes. Now we want to make all the class means furthest from each other. One way is to push them as far away from their global mean.

$$\mathbf{S}_B = \sum_{i=1}^C (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T$$

Multi-Class LDA

- Solution

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_W \mathbf{w}$$

Here, the number of valid eigenvectors are bound by the MINIMUM rank of matrix ($\mathbf{S}_B, \mathbf{S}_W$). In this case \mathbf{S}_B is typically lowest rank which is sum of C outer-product matrices. (Since they subtract the global mean, the rank is $C-1$).

$$\mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{w} = \lambda \mathbf{w}$$

If \mathbf{S}_W is non-singular and invertible.

So for C classes we have at most **$C-1$ w vectors** which we can project on to.

Dealing With High Dimensional Data

- What happens when we deal with high-dimensional data.
- If we have more dimensions d than data samples N , then we run into more problems.
- **Sw is singular**. It will still have at most $N-C$ non-zero eigenvalues.
- (N is the total number of samples from all classes, C is the number of classes)

$$\mathbf{S}_w = \sum_{i=1}^C \sum_{j=1}^{N_i} (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T$$

Fisherfaces

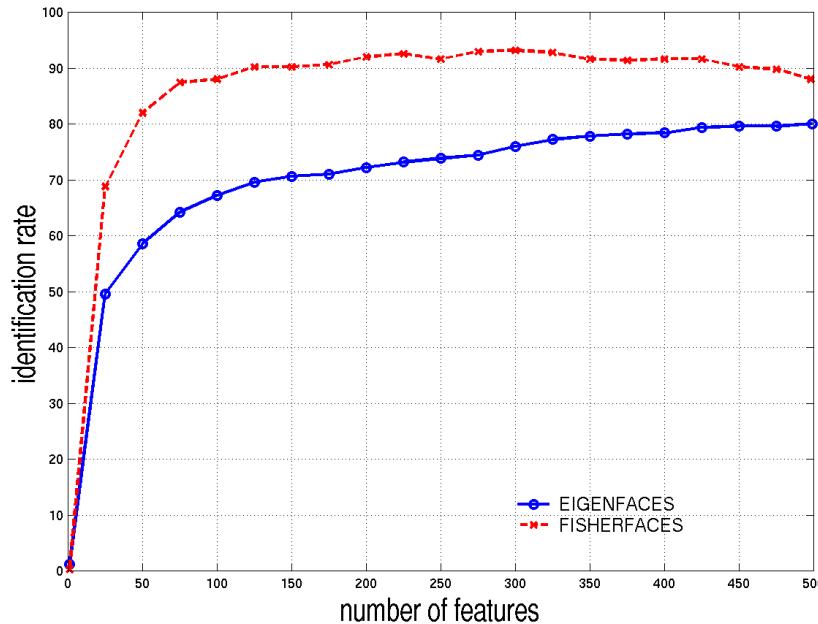
- Solution? Fisherfaces.....
- First do PCA and keep $N-C$ eigenvectors.
Project your data on to these $N-C$ eigenvectors. (\mathbf{S}_w will now be full rank)
- Then do LDA and compute the $C-1$ projections in this lower-dimensional $N-C$ subspace.
- PCA+LDA=Fisherfaces

Fisherfaces vs Eigenfaces

- Fisherfaces
 - Better discriminant for classification
 - The subject of a test image must be in the database
- Eigenfaces
 - No distinction between inter-class and intra-class faces
 - Optimal for representation but not for discrimination

Fisherfaces vs Eigenfaces

- FERET database



Best ID rates: eigenfaces - 80.0%, fisherfaces - 93.2%

Recap

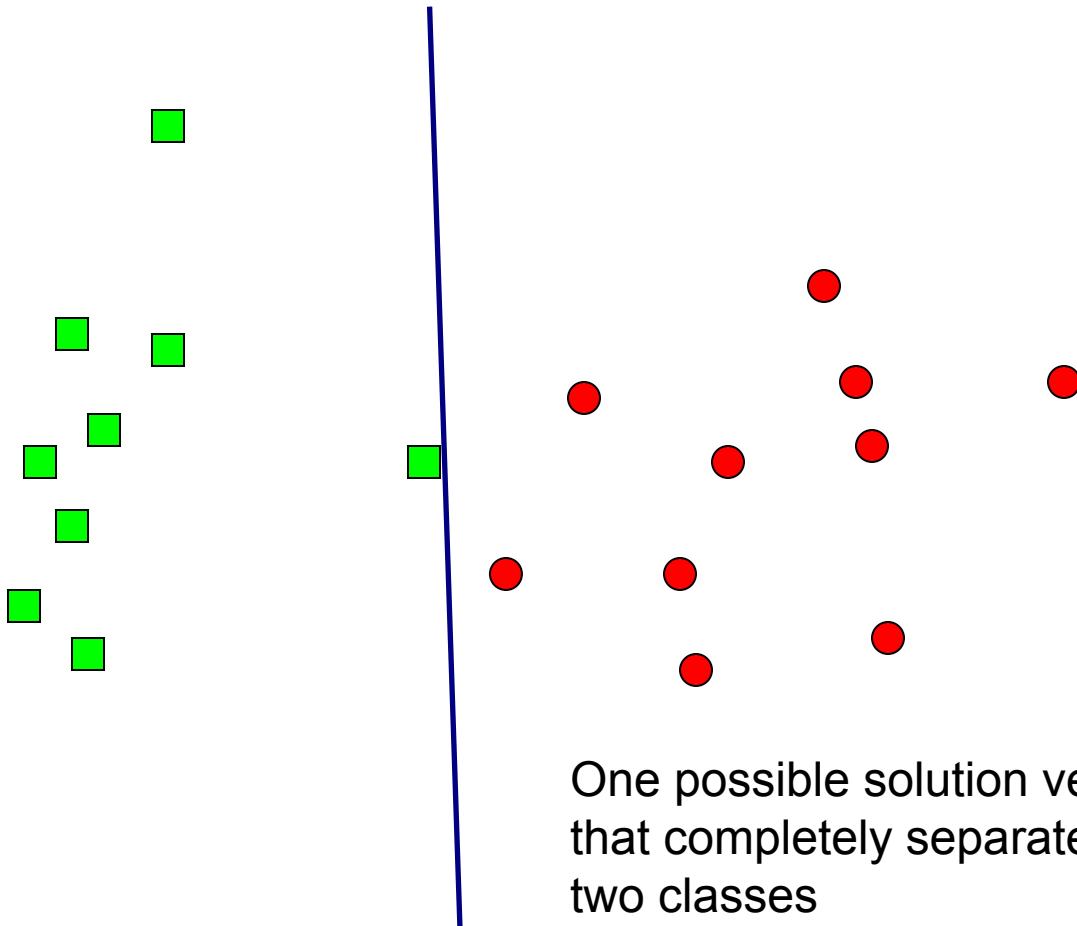
- LDA vs PCA
- LDA Objective Function
- Closed Form Expression for 2 Class Case
- Multi-Class LDA
- Fisherfaces

Prof. Marios Savvides

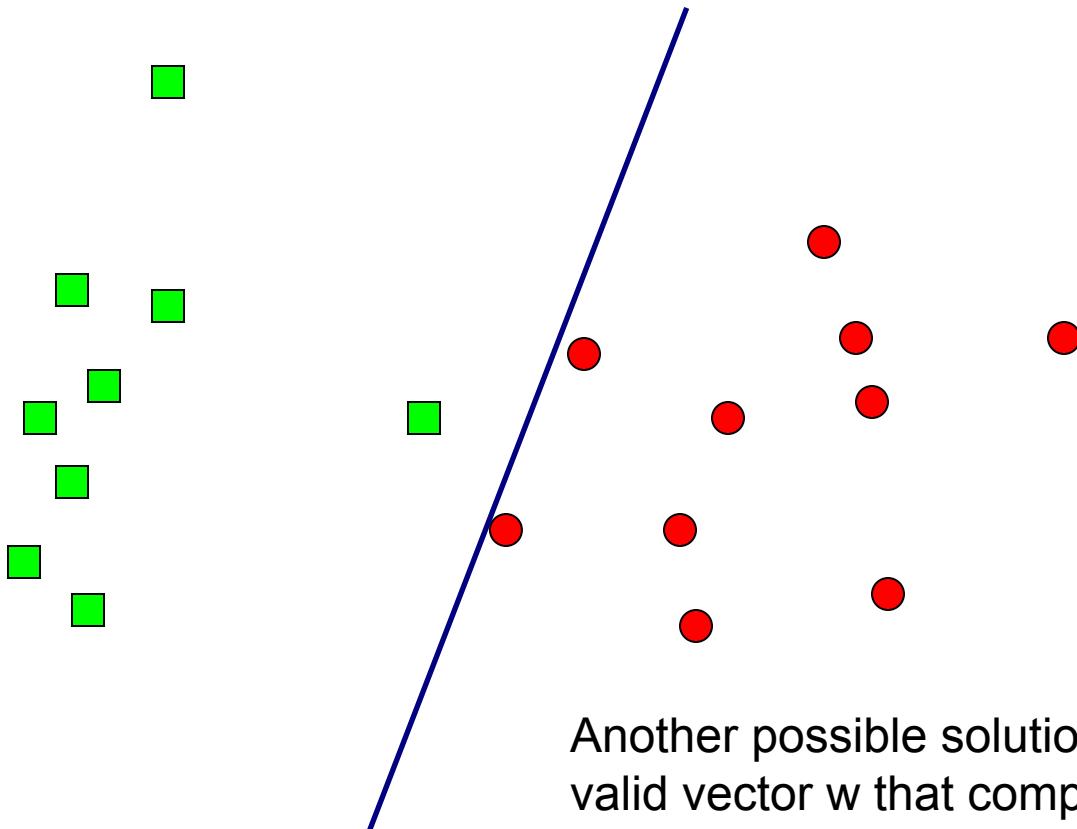
Pattern Recognition Theory

Lecture 11 : Support Vector Machines (SVMs)

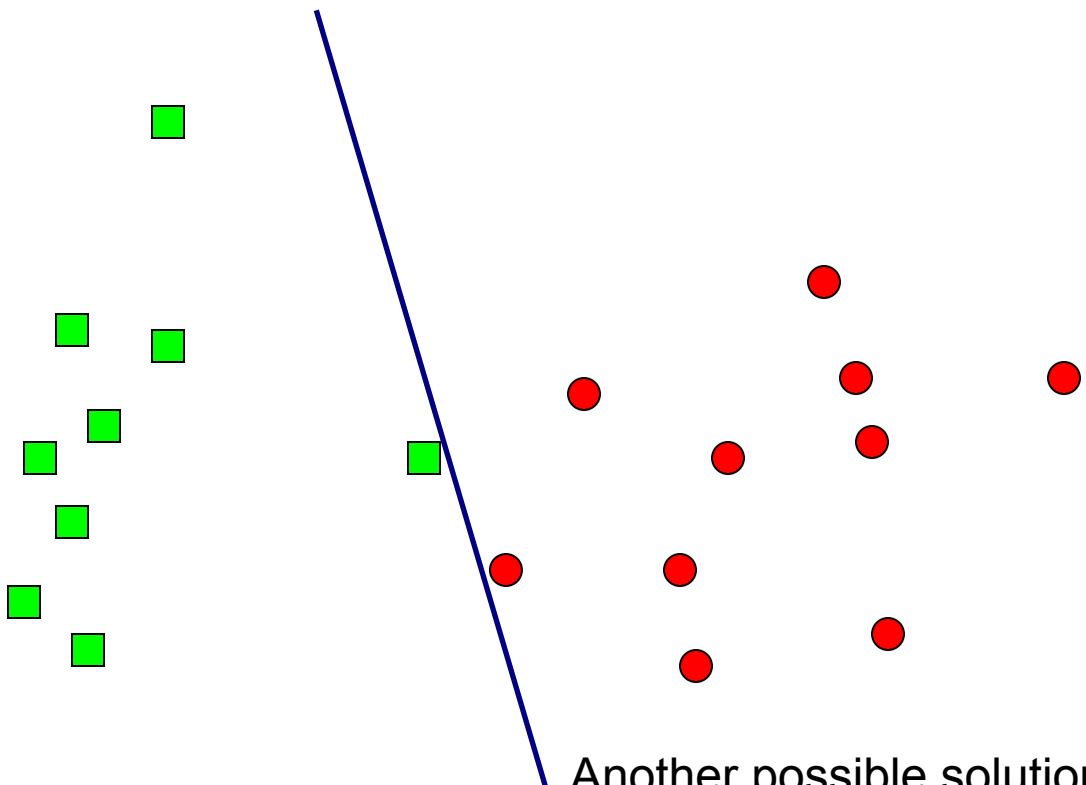
Neural Classifier



Neural Classifier

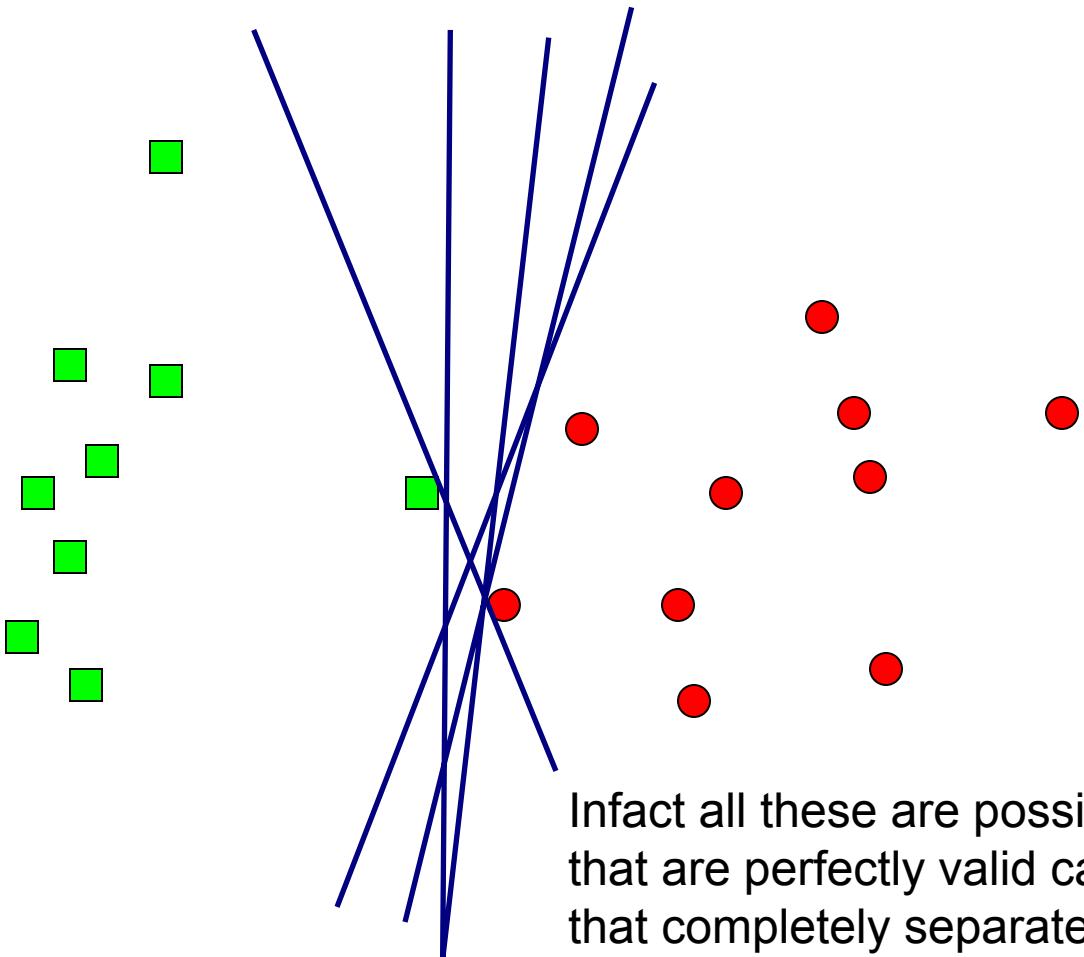


Neural Classifier



Another possible solution and perfectly valid vector w that completely separates the two classes

Neural Classifier



Neural Classifiers

- If data is separable, you will always converge to candidate solution vector w which completely separates the classes (perceptron, Ho-Kashyap, MLP, NNs)
- Objective function is to find vector that separates the two classes, and as we have seen there are many vectors that satisfy this criteria.

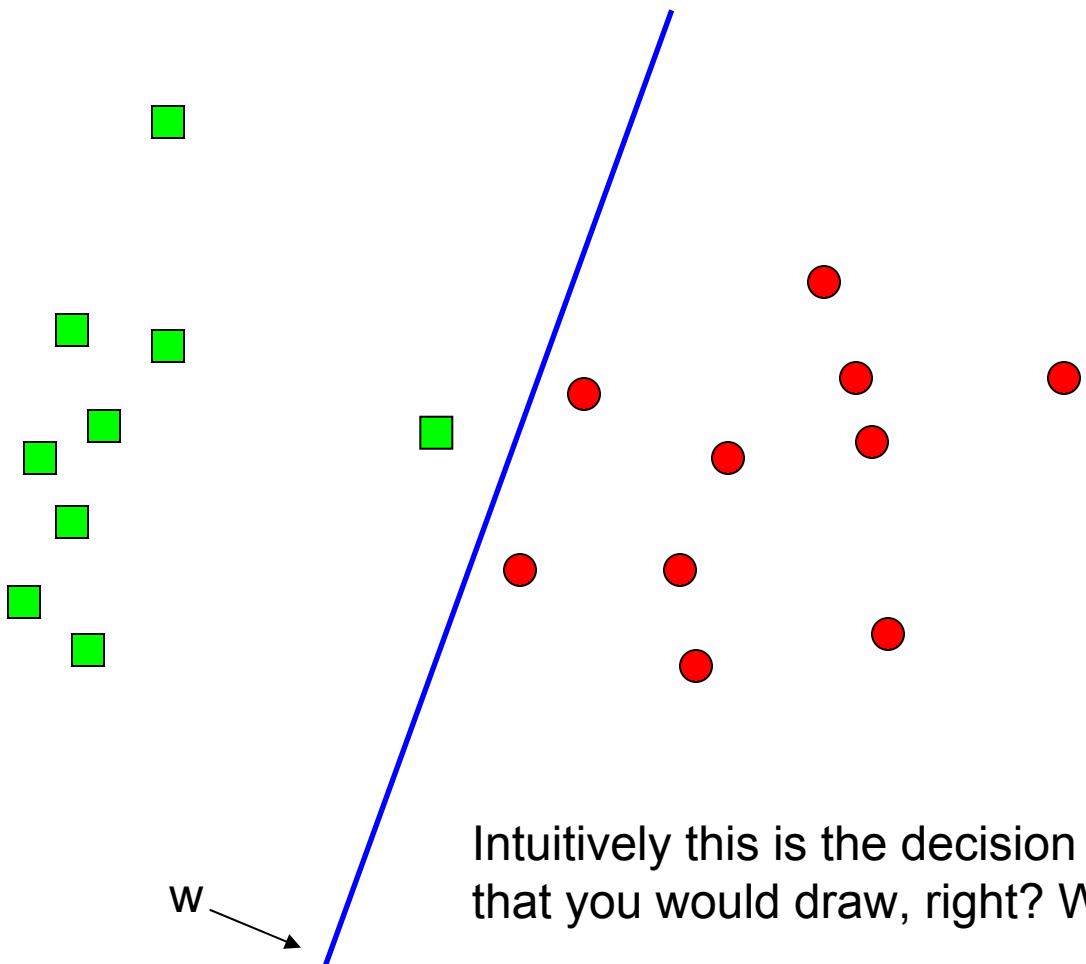
Neural Classifiers

- Neural Nets in general converge to different solution due to the initial weight vectors (usually random).
- That is why in practice (in the old times where we used NN everyday), you would re-initialize NNets and re-train to see if the generalization solution vector that converged offered better generalization (on the cross-validation set).
- So what do we learn, we need an objective function that offers better “generalization”.

Neural Net Generalization

- What do we mean by generalization?
 - We want the hyperplane or decision boundary defined by the solution vector w to not only separate the training vectors but also work well with un-seen test data.
- So...what would you do?

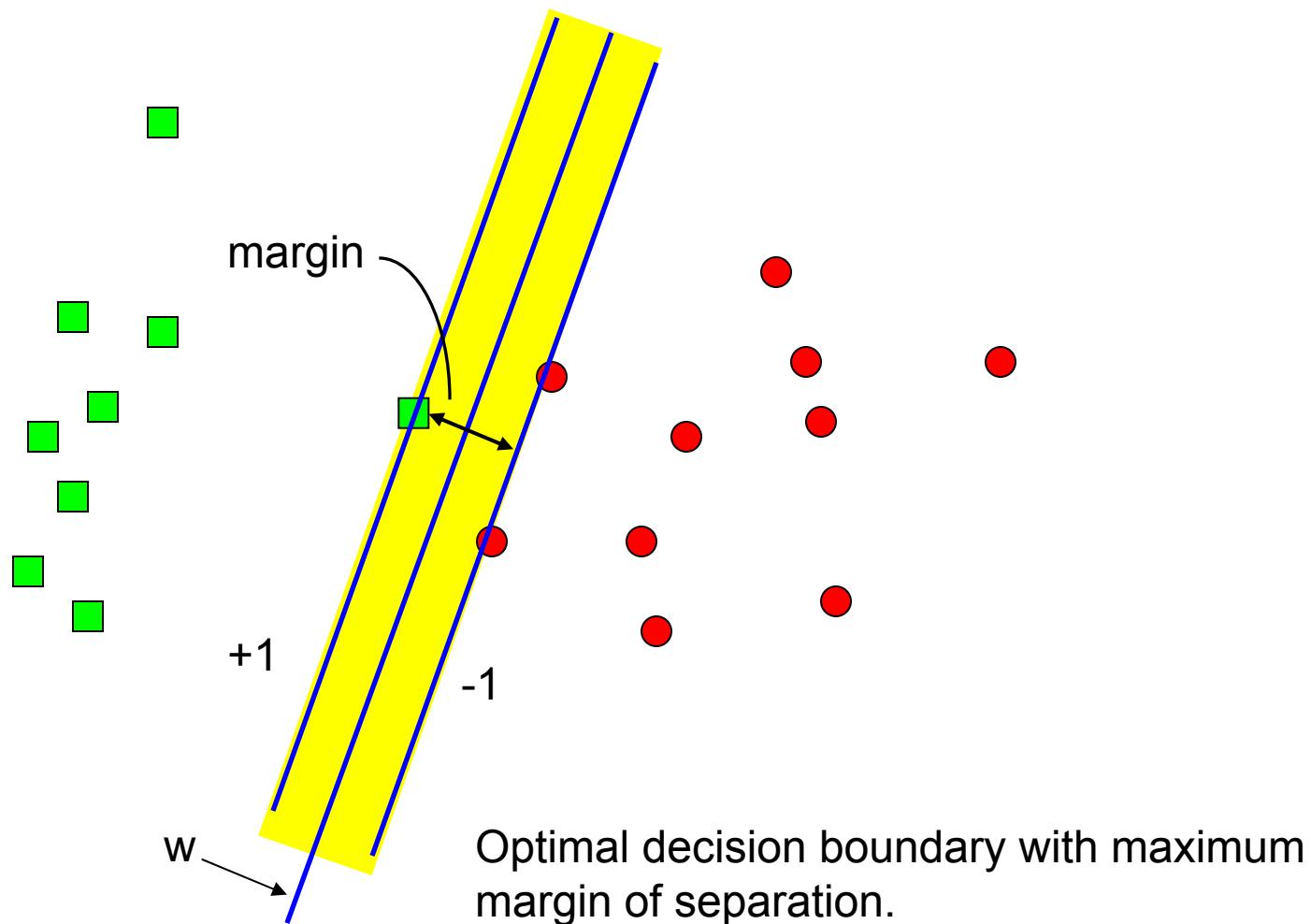
Support Vectors



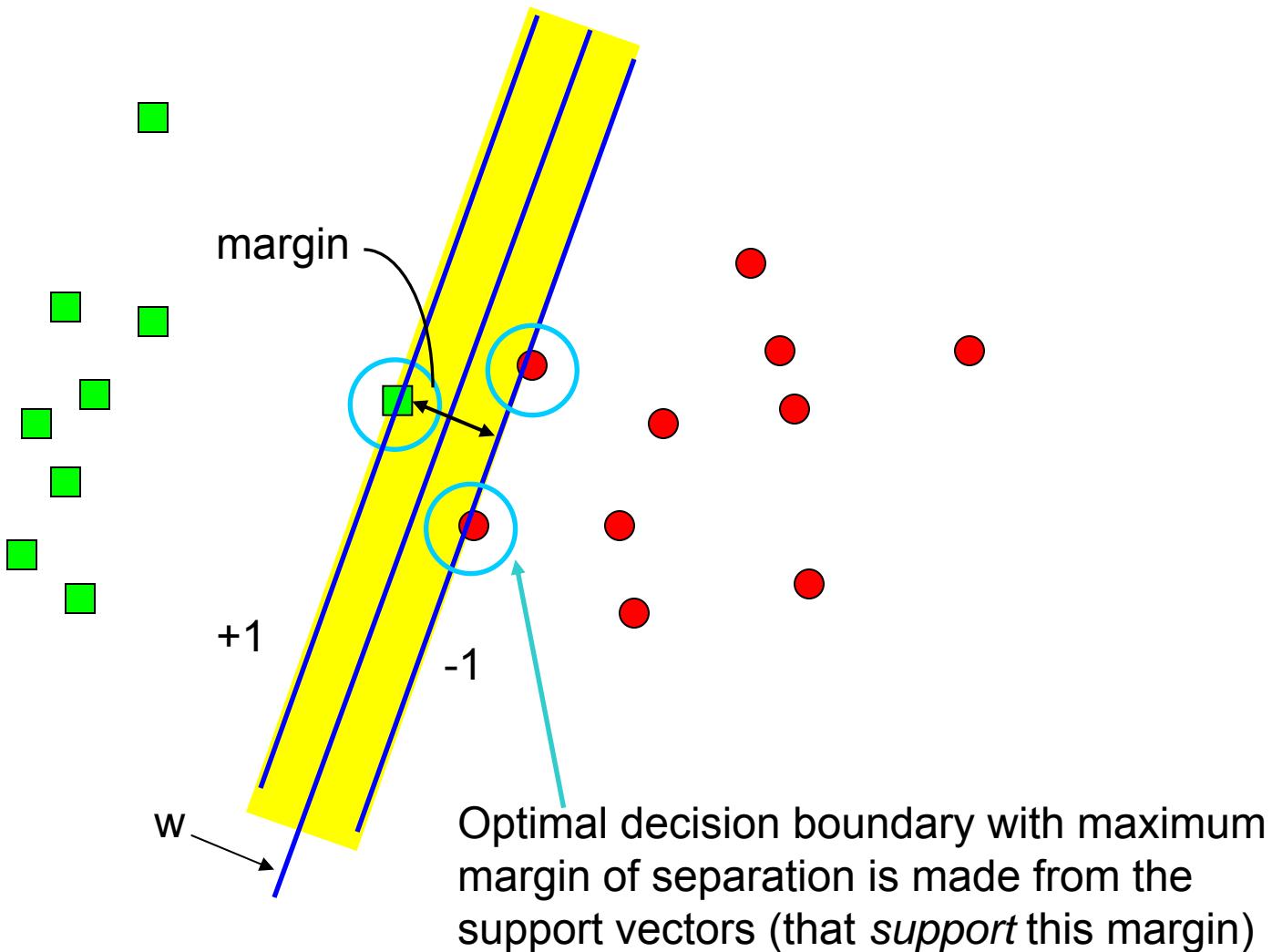
Support Vector Machines

- Goal is to improve generalization.
- How?
- By finding a decision boundary solution vector that “Maximizes” the margin boundary between the two classes.
- So..we don't care about data samples that are easily classified (i.e. the samples away from the margin). We care about where the errors will lie..i.e. near the margin..which of the data samples are affected.

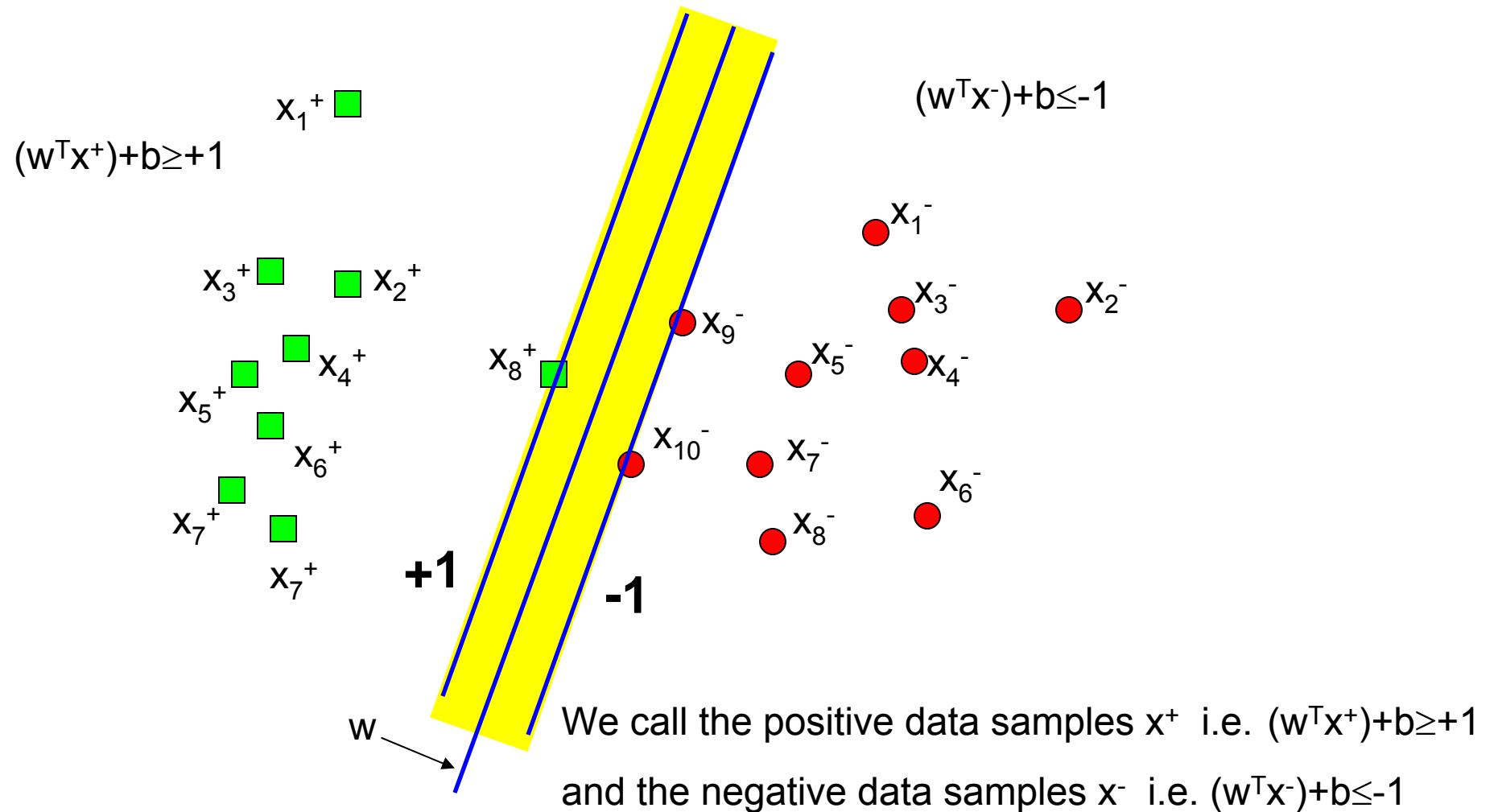
Support Vectors



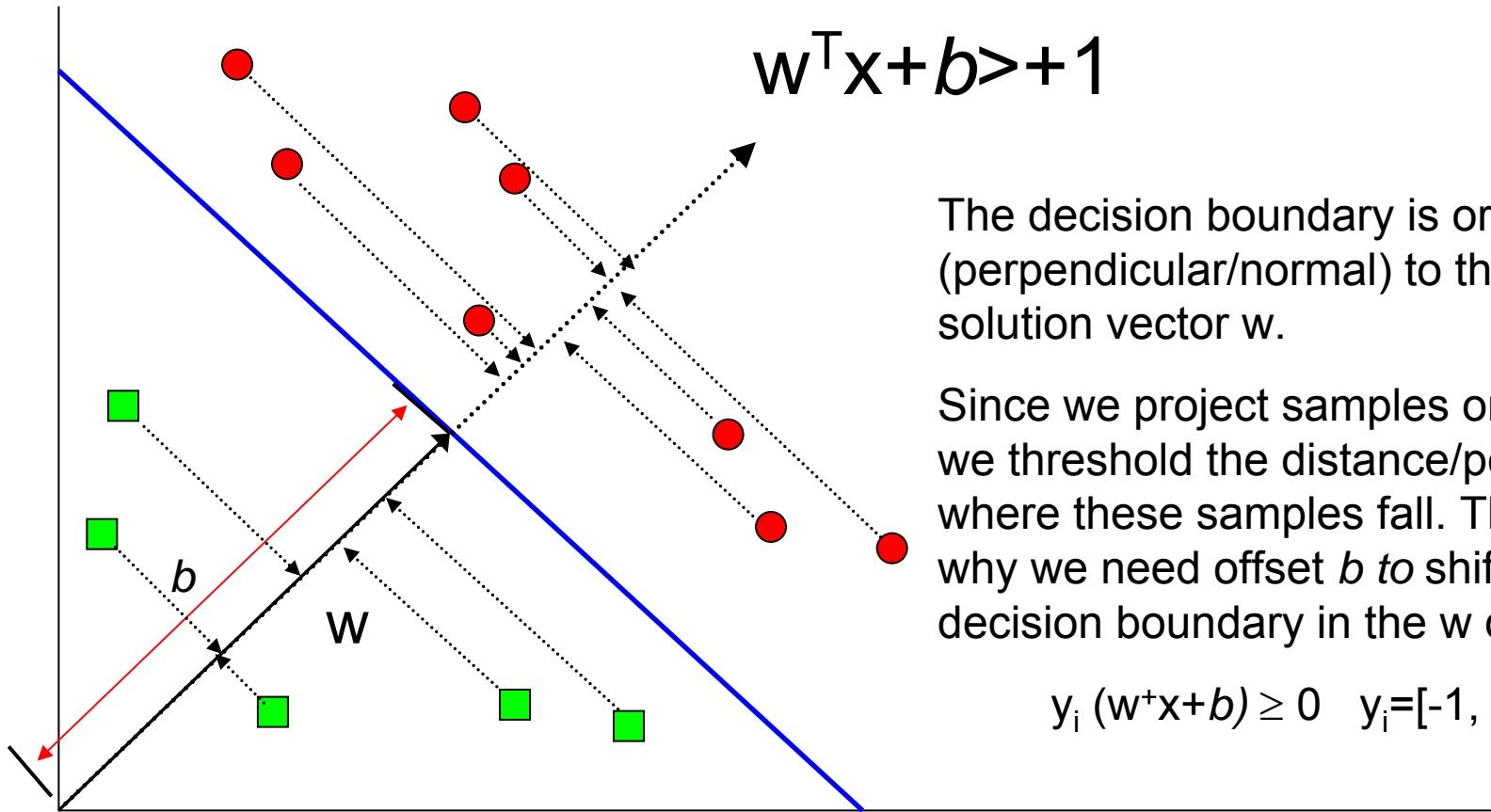
Support Vectors



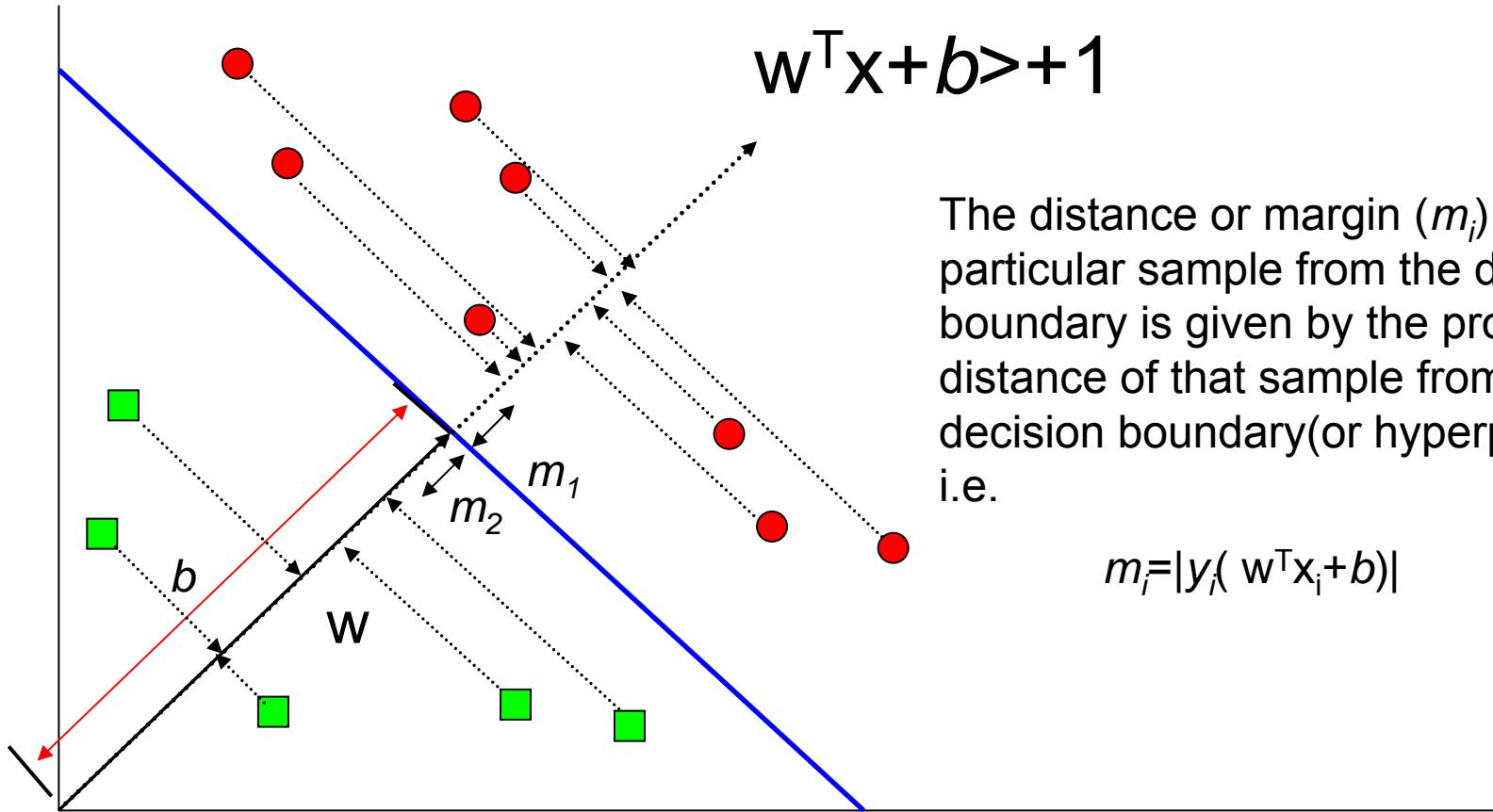
Support Vectors



Linear Discriminant - revisited



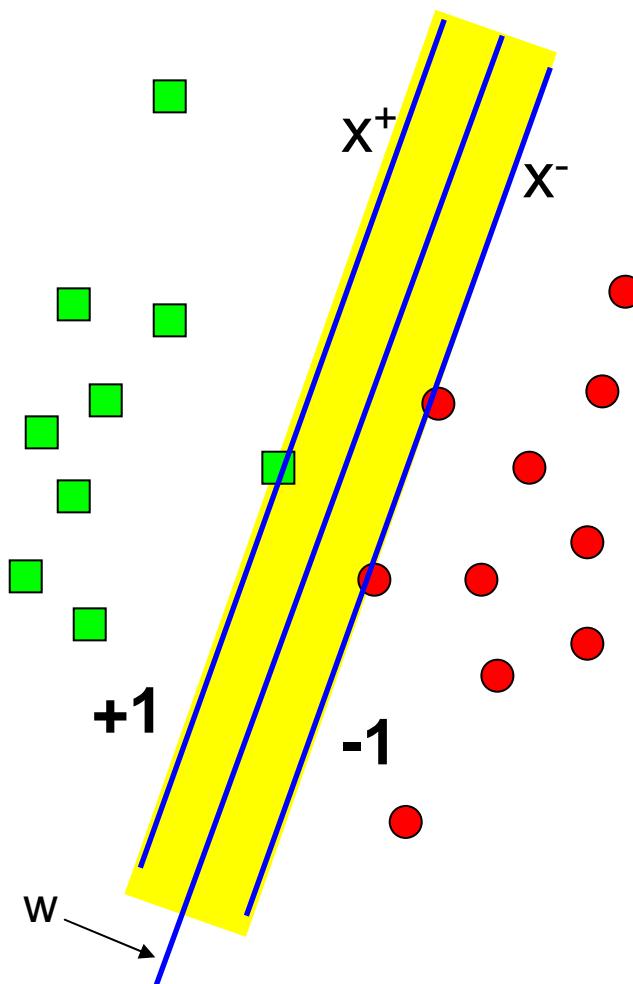
Linear Discriminant - margins



The distance or margin (m_i) of a particular sample from the decision boundary is given by the projected distance of that sample from the decision boundary(or hyperplane) i.e.

$$m_i = |y_i(w^T x_i + b)|$$

Support Vectors



Let x^+ denote a positive point with functional margin of 1 and x^- denote a negative point respectively.

This implies:

$$w^T x^+ + b = +1$$

$$w^T x^- + b = -1$$

The functional margin of the resulting classifier m is

$$m = \left(\left\langle \frac{w}{\|w\|}, x^+ \right\rangle - \left\langle \frac{w}{\|w\|}, x^- \right\rangle \right)$$

$$= \frac{1}{\|w\|} \left(\langle w, x^+ \rangle - \langle w, x^- \rangle \right)$$

$$= \frac{2}{\|w\|}$$

Recall

- If we want to **maximize** the margin $\frac{2}{\|\mathbf{w}\|}$ is

equivalent to **minimizing** $\langle \mathbf{w}, \mathbf{w} \rangle = \mathbf{w}^T \mathbf{w}$

Thus for a support vector machine we want a hyperplane that maximizes this margin!!

Support Vector Optimization Criteria

- So...we want to maximize the margin by minimizing

$$\mathbf{w}^T \mathbf{w}$$

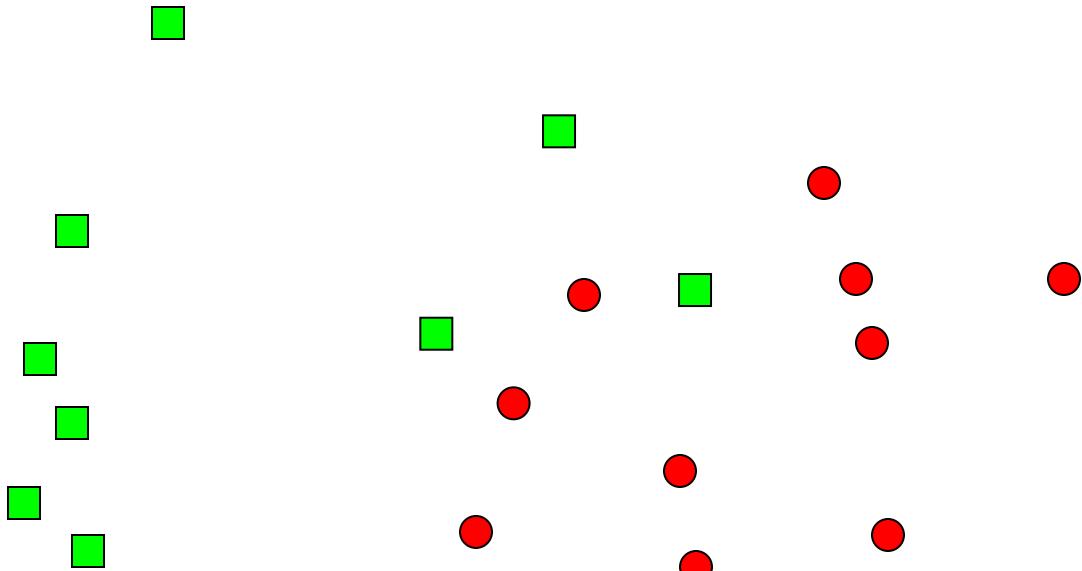
Subject the inequality constraints

$$y_i (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$$

i.e. positive samples must fall on the positive side (vice-versa for negative data samples) $y_i = [-1, +1]$

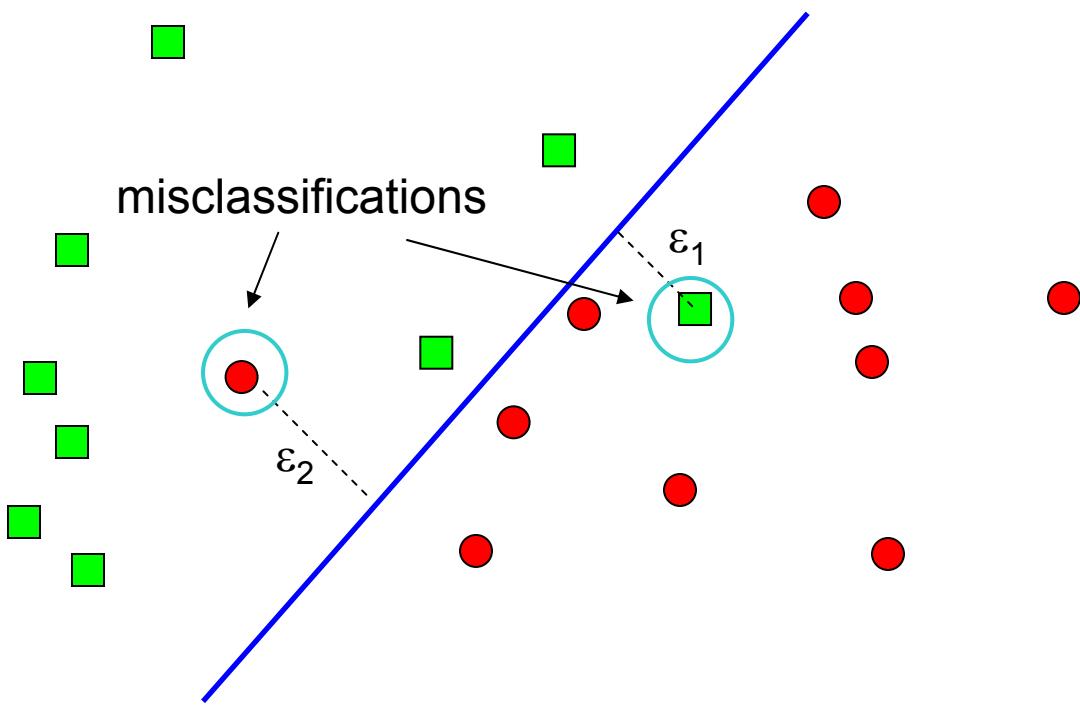
- Can be solved via Quadratic Programming (QP)..programs exist in matlab..never need to write them yourself!

Houston...we have a problem!



- What happens when data is non-separable?
We can not draw a linear hyperplane to separate the data..so..do we pack our bags and go home?....ofcourse not....atleast not yet..

Introduce error metric ε



- We still can have a hyperplane that can not fully separate the true classes but tries to maximize margin while minimizing the total error $\sum \varepsilon_i$

Slack variables

- We call the error distances ε the ‘slack’ variables.
- We want to cut some slack to the support vector machine so they can get the job done.

So what do we optimize now?

- In non-separable cases we still want to maximize the margin but we also want to minimize the error distances.
- So.... Our optimization formulation says we:
- **MINIMIZE**

$$\frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^N \varepsilon_i$$

Is that all? What about our inequality constraints?

- We add some slack to our data so we minimize:

$$\frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^N \varepsilon_i$$

- Subject to the following inequality constraints:

$$\mathbf{w}^T \mathbf{x}_i + b \geq 1 - \varepsilon_i \quad \text{for } +ve \text{ class}$$

$$\mathbf{w}^T \mathbf{x}_i + b \leq -1 + \varepsilon_i \quad \text{for } -ve \text{ class}$$

A Dual problem in a dual universe?

- The optimization problem set-up was is what's called its 'primal' form.
- We can re-formulate the optimization problem of maximizing margin in what's called a 'Dual' form..why?
- We can do cool things, like use Kernels that allow us to work in a higher dimensional space (without actually computing this space). – what's called the kernel trick....but first.....

Primal Langrangian Form

- Primal form:

- Where α_i are the lagrange multipliers $\alpha_i \geq 0$
 - We want to optimize this function

Primal Langrangian Form contd..

- Primal form:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1]$$

- Differentiate with respect \mathbf{w} to find

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i = \mathbf{0}$$

$$\mathbf{w} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i$$

Primal Langrangian Form contd..

- Primal form:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \alpha_i \left[y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 \right]$$

- Differentiate with respect b to find

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial b} = \sum_{i=1}^N \alpha_i y_i = 0$$

$$\sum_{i=1}^N \alpha_i y_i = 0$$

Dual Langrangian Form contd..

- Primal form:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1]$$

- Substitute $\mathbf{w} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i$ back into above Eq.

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1]$$

$$= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \sum_{i=1}^N \alpha_i$$

$$= \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

What do we optimize?

- Dual form:

$$\sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle$$

- Subject to the constraints $\sum_{i=1}^N \alpha_i y_i = \mathbf{0}$ and $\alpha_i \geq 0$
- Again this is solvable with QP.

Ok...what the important thing to get from this Dual formulation?

- Remember when we optimized the dual lagrangian we got

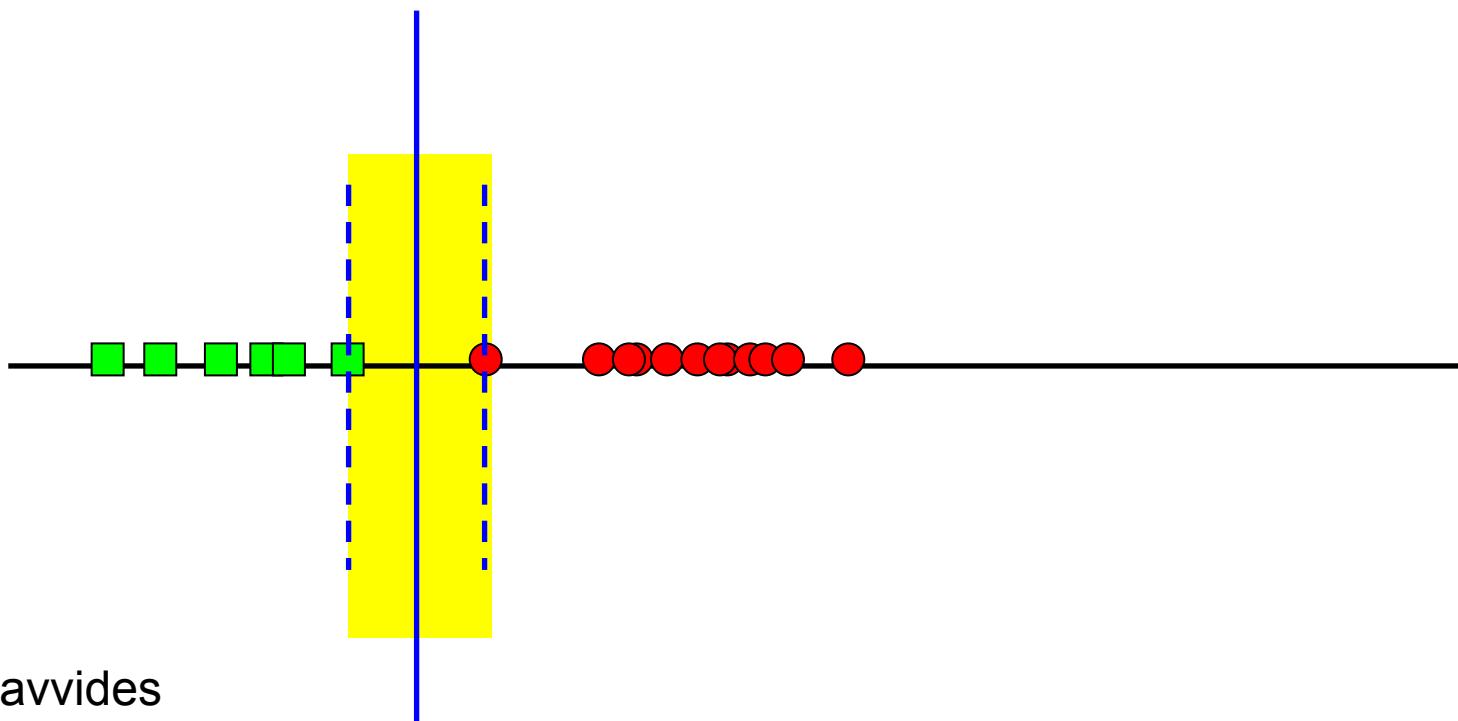
$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i = \mathbf{0}$$

$$\mathbf{w} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i \quad \alpha_i \geq 0 \quad y_i \in \{-1, +1\}$$

- The important thing to note is that the optimal hyperplane \mathbf{w} with maximum margin is a linear combination of the training sets. More importantly it is linear combination of the support vectors (the lagrange multipliers α_i specify which samples to use and how much or it can be 0.

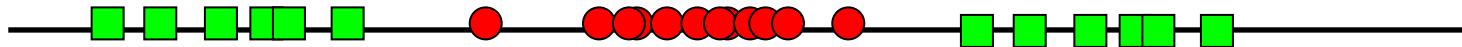
Example SVM

- No problemo....



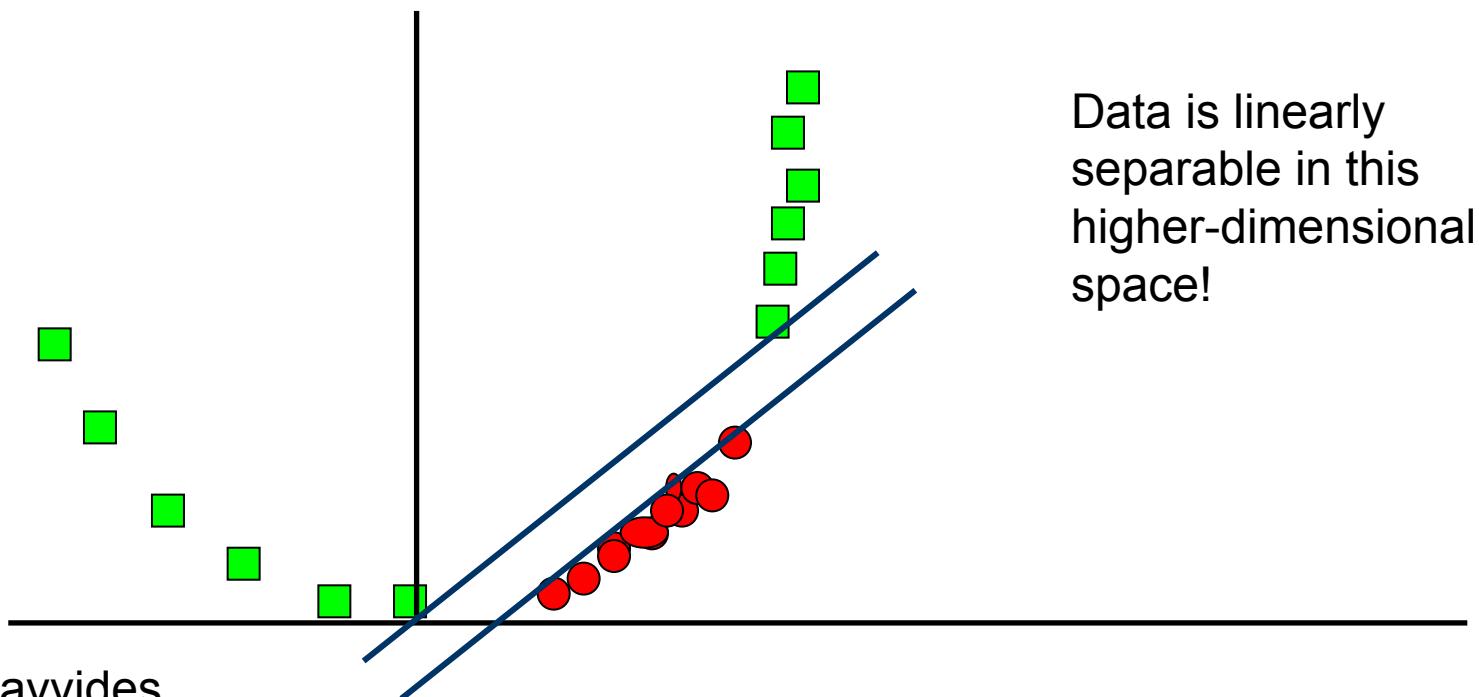
Example SVM

- Major problem...what now?



Example SVM

- Remember permitting a non-linear transformation to higher dimensional space. $F_k = (x_k, x_k^2)$



Wooow...now...Hold on a second..

- Don't we usually say....we want to reduce the dimensionality of data in pattern recognition problems?
- Correct..PCA does dimensionality reduction...but as we just saw there are cases where the data is NOT separable and we need to map our data into a *higher-dimensional space* where it might be linearly separable!

Mapping functions

$$\phi: X \rightarrow F$$

This is a non-linear map from the input space to some feature space. Looking back to our SVM linear machine we now have:

$$\sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j \langle \Phi(x_i), \Phi(x_j) \rangle$$

$$\mathbf{w} = \sum_{i=1}^N \alpha_i y_i \Phi(\mathbf{x}_i)$$

Project on to
direction vector

$$\mathbf{w}^T \Phi(\mathbf{x}) = \sum_{i=1}^N \alpha_i y_i \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}) \rangle$$

Mapping functions

$$\phi: X \rightarrow F$$

This is a non-linear map from the input space to some feature space. Looking back to our SVM linear machine we now have:

$$\sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j \langle \Phi(x_i), \Phi(x_j) \rangle$$

$$\mathbf{w} = \sum_{i=1}^N \alpha_i y_i \Phi(\mathbf{x}_i)$$

Project on to direction vector

$$\mathbf{w}^T \Phi(\mathbf{x}) = \sum_{i=1}^N \alpha_i y_i \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}) \rangle$$

Marios Savvides

This is what we need to compute efficiently without actually having to form the mapping.

Mapping functions

$$\phi: X \rightarrow F$$

This is a non-linear map from the input space to some feature space. Looking back to our SVM linear machine we now can define a kernel function $K(x,y)$

$$K(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle$$

What are examples of some commonly used kernel functions?

The efficiency trick is that we need to compute $K(x,y)$ without actually compute the mapping of our data into higher dimensions (i.e. compute $K(x,y)$ without having to form $\Phi(x)$ and $\Phi(y)$) as computation and memory may be too great! – Kernel's must satisfy Mercer's condition to ensure that kernel actually forms a dot product in some higher dimensional space.

Example Kernel Functions

- Polynomial

$$K(a,b) = (\langle a,b \rangle + 1)^p$$

$$\begin{aligned} \text{Expand } \langle a,b \rangle^2 &= \langle (a(1)b(1) + a(2)b(2))^2 \rangle \\ &= [a(1)b(1)]^2 + 2a(1)b(1)a(2)b(2) + [a(2)b(2)]^2 \end{aligned}$$

$\xrightarrow{R^2 \rightarrow R^3}$

- Radial basis Style Kernel Function

$$K(a,b) = \exp(-\frac{(a-b)^2}{2\sigma^2})$$

- Neural Net Style Kernel Function

$$K(a,b) = \tanh(k \langle a,b \rangle - \delta)$$

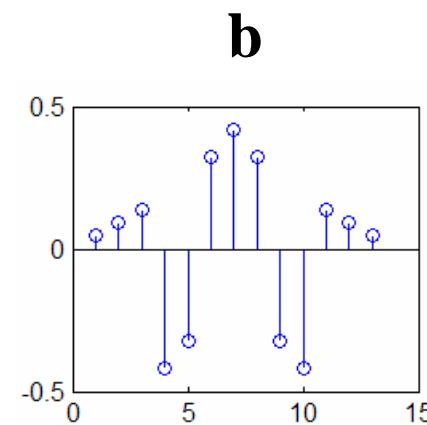
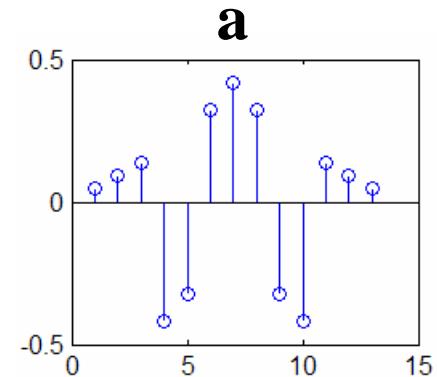
Prof. Marios Savvides

Pattern Recognition Theory

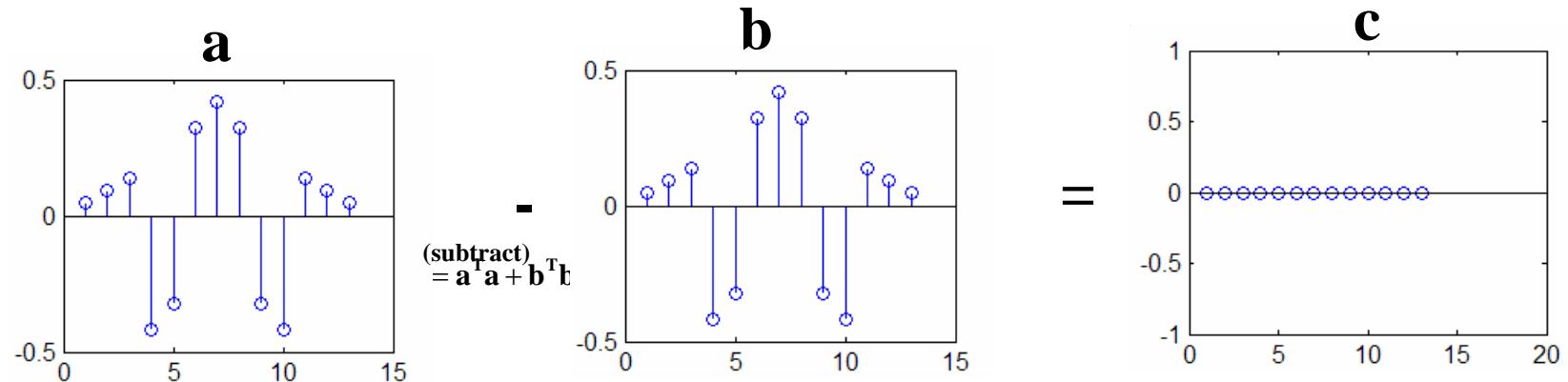
Lecture 12 : Correlation Filters

Pattern Matching

- How to match two patterns?
- How do you locate where the pattern is in a long sequence of patterns?
- Are these two patterns the same?
- How to compute a match metric?



Pattern Matching



Lets define mean squared error, i.e.

$$e = \|\mathbf{a} - \mathbf{b}\|^2 = (\mathbf{a} - \mathbf{b})^T (\mathbf{a} - \mathbf{b}) = \mathbf{a}^T \mathbf{a} + \mathbf{b}^T \mathbf{b} - 2\mathbf{a}^T \mathbf{b}$$

$$\mathbf{a}^T \mathbf{a} = \sum_{i=1}^N a(i)^2 = \text{energy_of_a}$$

$$\mathbf{a}^T \mathbf{b} = \sum_{i=1}^N \mathbf{a}(i)\mathbf{b}(i) = \text{correlation_term}$$

$$\mathbf{b}^T \mathbf{b} = \sum_{i=1}^N \mathbf{b}(i)^2 = \text{energy_of_b}$$

Pattern Matching

$$e = \|\mathbf{a} - \mathbf{b}\|^2 = (\mathbf{a} - \mathbf{b})^T(\mathbf{a} - \mathbf{b}) = \mathbf{a}^T\mathbf{a} + \mathbf{b}^T\mathbf{b} - 2\mathbf{a}^T\mathbf{b}$$

$$\mathbf{a}^T\mathbf{a} = \sum_{i=1}^N a(i)^2 = \text{energy_of_a}$$

$$\mathbf{b}^T\mathbf{b} = \sum_{i=1}^N b(i)^2 = \text{energy_of_b}$$

$$\mathbf{a}^T\mathbf{b} = \sum_{i=1}^N \mathbf{a}(i)\mathbf{b}(i) = \text{correlation_term}$$

- Assume we normalize energy of a and b to 1 i.e. $\mathbf{a}^T\mathbf{a}=\mathbf{b}^T\mathbf{b}=1$
- Then to minimize error, we seek to maximize the correlation term.
- So performing correlation, the maximum correlation point is the location where the two pattern match best.

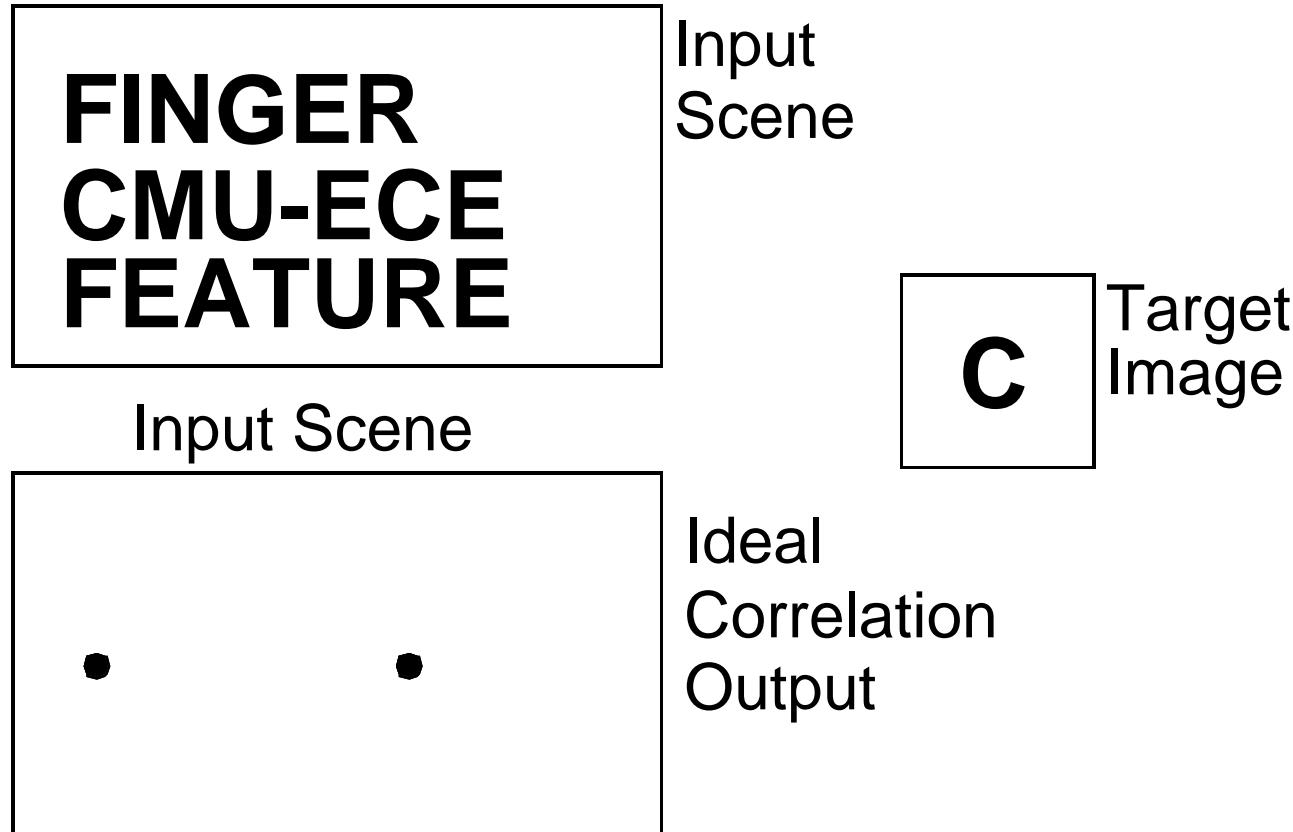
Correlation Pattern Recognition

- $r(x)$ test pattern
- $s(x)$ reference pattern

$$-1 \leq \frac{\mathbf{a}^T \mathbf{b}}{\sqrt{(\mathbf{a}^T \mathbf{a})(\mathbf{b}^T \mathbf{b})}} \leq 1$$

- Normalized correlation between $a(x)$ and $b(x)$ gives 1 if they match perfectly (i.e. only if $a(x) = b(x)$) and close to 0 if they don't match.
- **Problem:** Reference patterns rarely have same appearance
- **Solution:** Find the pattern that is consistent (i.e., yields large correlation) among the observed variations.

Object Recognition using correlation

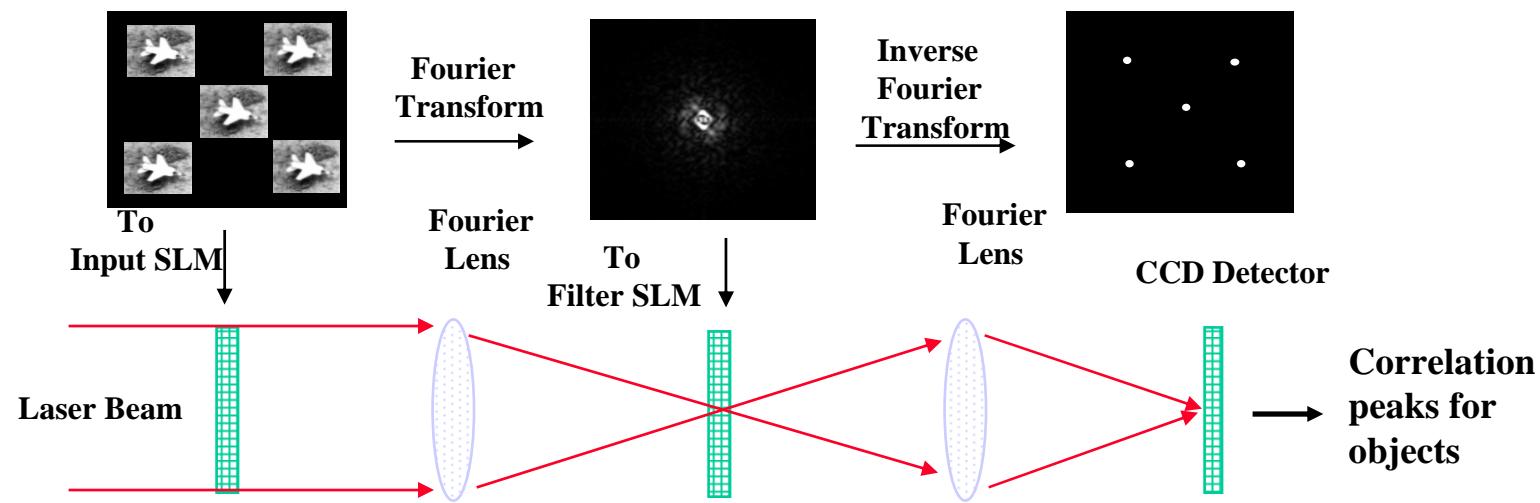


Goal: Locate all occurrences of a target in the input scene

Why correlation filters?

- Built-in Shift-Invariance: shift in the input image leads to a corresponding shift in the output peak. Classification result remains unchanged.
- Matched filters are just replicas of the object that we are trying to find. Problem is we need as many matched filters as the different appearances that object can look under different conditions. (i.e. a matched filter for every pose, illumination and expression variation).
- Using Matched filters is computationally and memory very expensive.
- We can synthesize distortion tolerant filters that can recognize more than one view of an object.
- We can build different types of distortion-tolerance in each filter (e.g. scale, rotation, illumination etc).
- We will show advanced correlation filters exhibit graceful degradation with input image distortions.

Optical Correlation @ light speed



SLM: Spatial Light Modulator

CCD: Charge-Coupled Detector

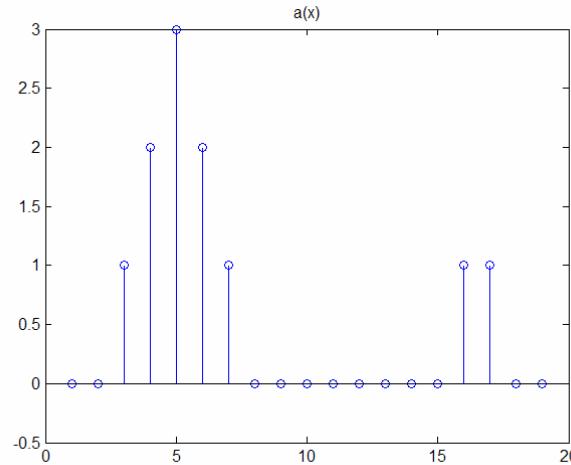
How to do Correlations Digitally and Efficiently?

- Use Fast Fourier Transforms...
- How? Fourier Transform property tells us:
 - ▼ Convolution Theorem:
 - ▼ Convolution of two functions $a(x)$ and $b(x)$ in the spatial domain is equivalently computed in the Fourier domain by multiplying the $\text{FT}\{a(x)\}$ with the $\text{FT}\{b(x)\}$.
 - ▼ i.e. in matlab this would be
 - $\text{Convolution}=\text{IFFT}(\text{FFT}(a) .* \text{FFT}(b))$ (assuming 1 D)
 - ▼ Correlation Theorem:
 - ▼ Similar to convolution except the correlation of functions $a(x)$ and $b(x)$ in the spatial domain is equivalently computed in the Fourier domain by multiplying $\text{FT}\{a(x)\}$ with conjugate of $\text{FT}\{b(x)\}$.
 - $\text{Correlation}=\text{IFFT}(\text{FFT}(a) .* \text{conj}(\text{FFT}(b)))$

Some Digital Signal Processing basics

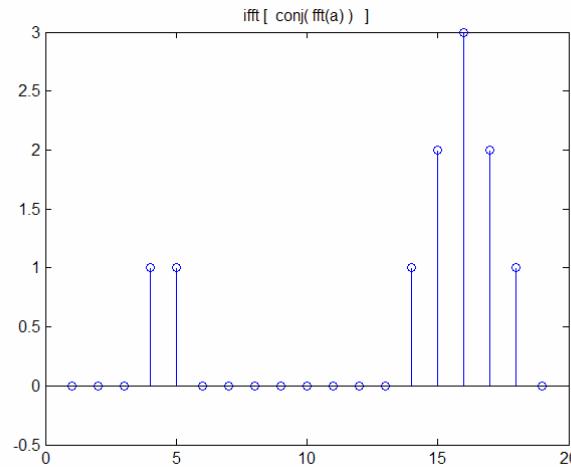
- Take a signal $a(x)$
- $a = [0\ 0\ 1\ 2\ 3\ 2\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0]$;

Looks like this ->



- [1] Compute its Discrete Fourier Transform or FFT,
 $a(x) \rightarrow A(u)$.
- [2] Take the conjugate: $\text{conj}(A(u))$
 - [3] Take inverse Fourier Transform of $\text{conj}(A(u))$

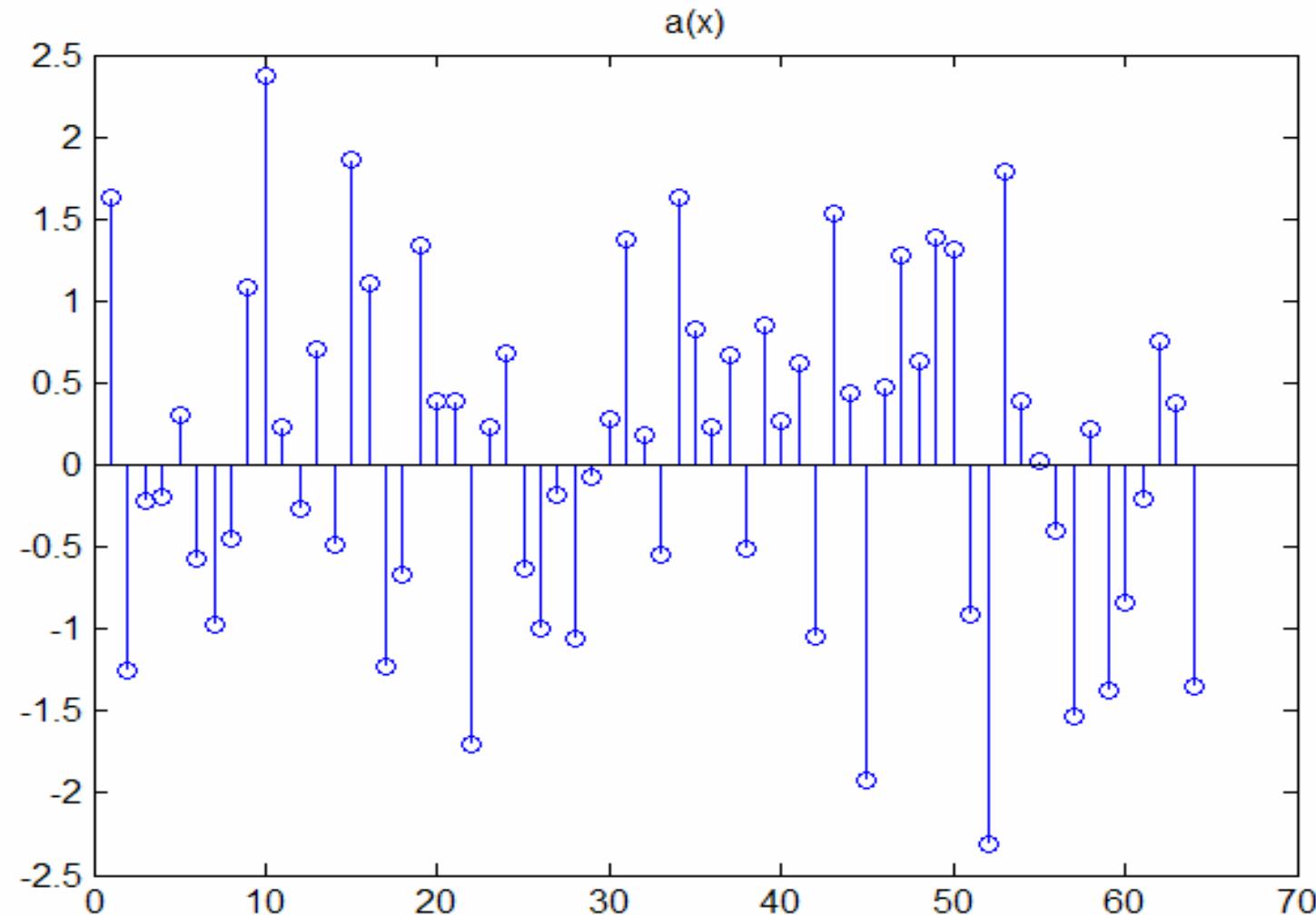
**Conjugation in Frequency domain
leads to time-reversal in the time
domain**



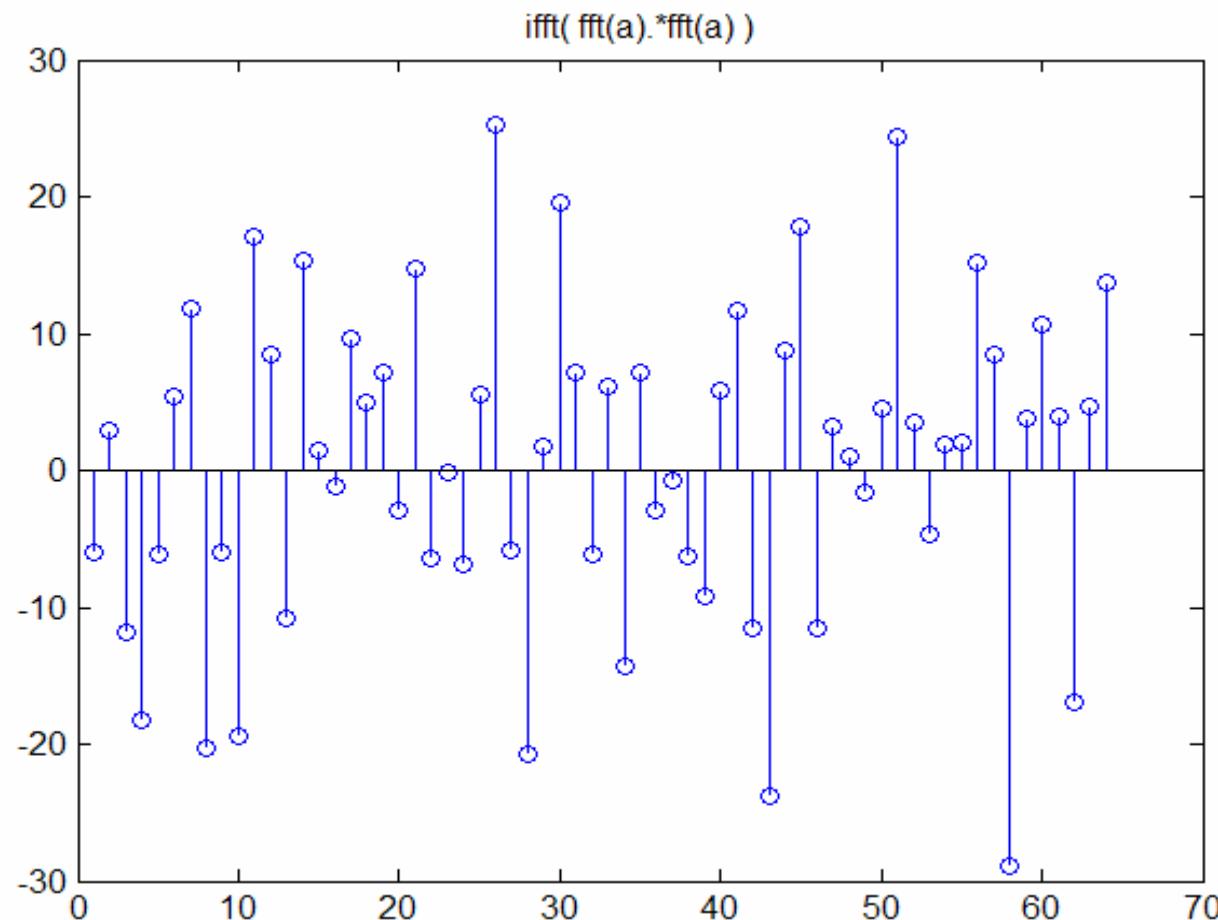
So Correlation in Fourier domain is.....

- Is just like convolution except we convolve the test signal $t(x)$ with a time-reversed signal $h(x)$.
- Taking the conjugate in the Fourier domain time-reverses the signal in the time domain.
- Since convolution automatically time-reverses $h(x)$ also..(there is a double time reversal, which cancels out, meaning that you end up computing inner-products with the reference signal and the test signal in the Fourier domain.

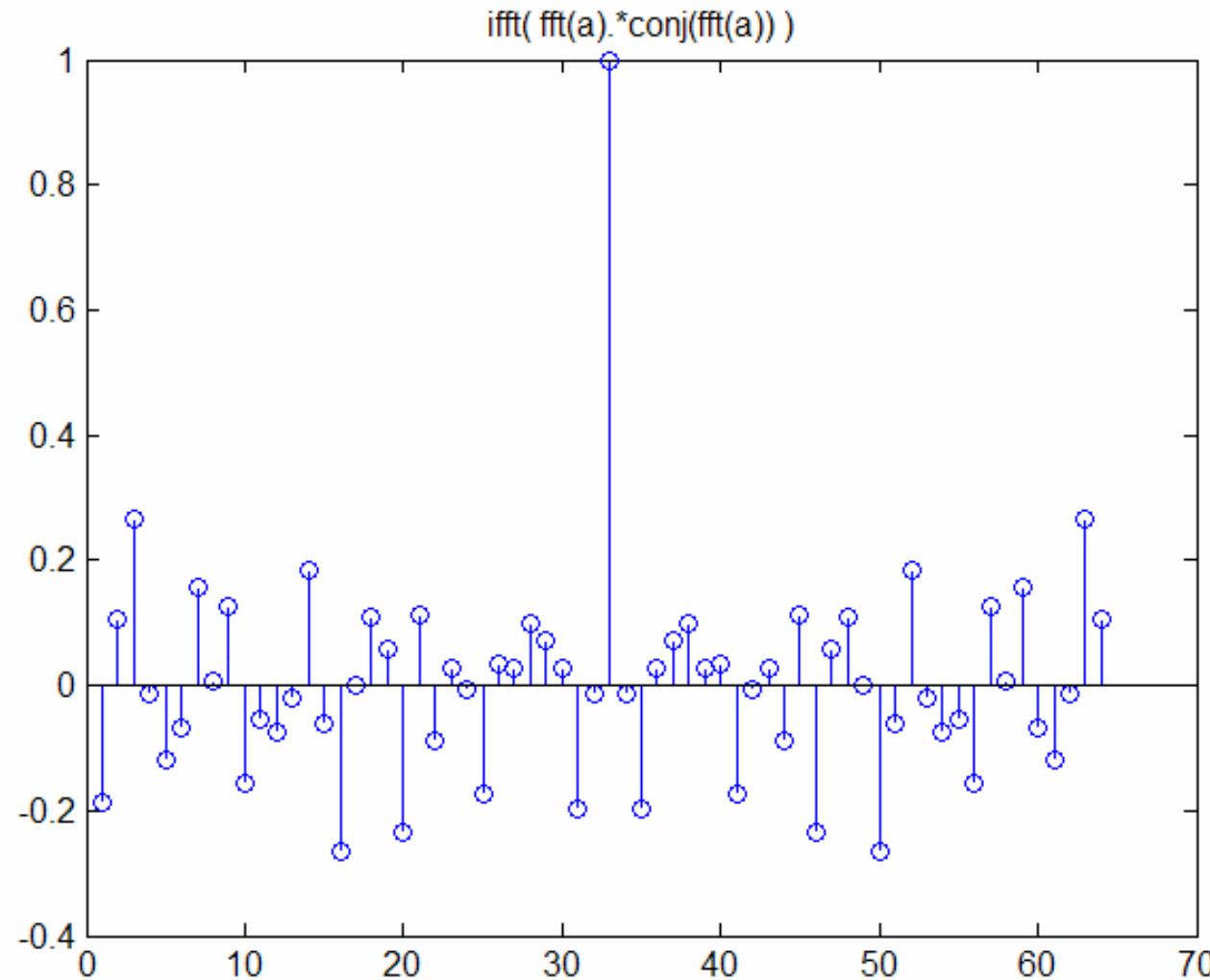
Lets start with a random sample signal $a(x)$



Here is the result of convolving $a(x)$ with $a(x)$



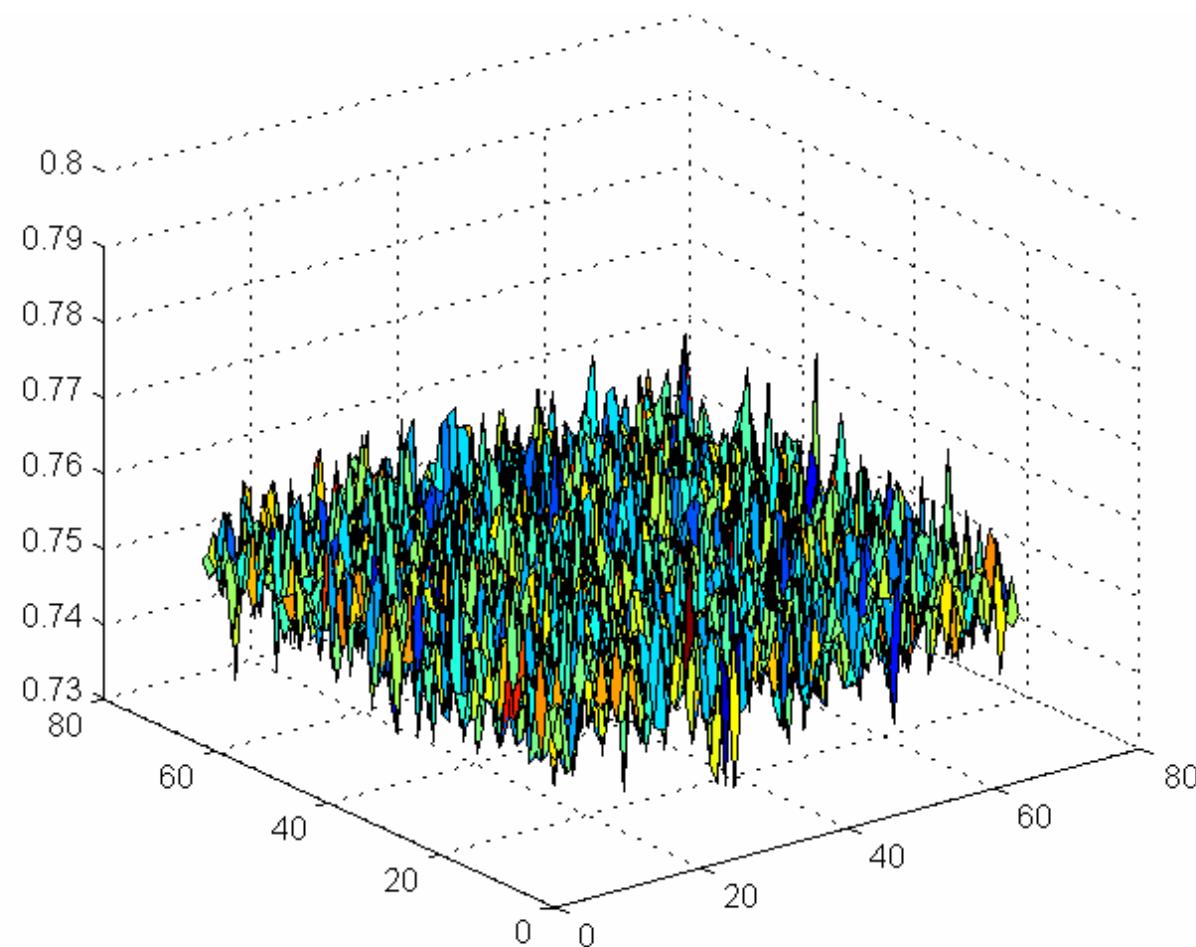
Here is the result of correlating $a(x)$ with $a(x)$



2D random image $s(x,y)$

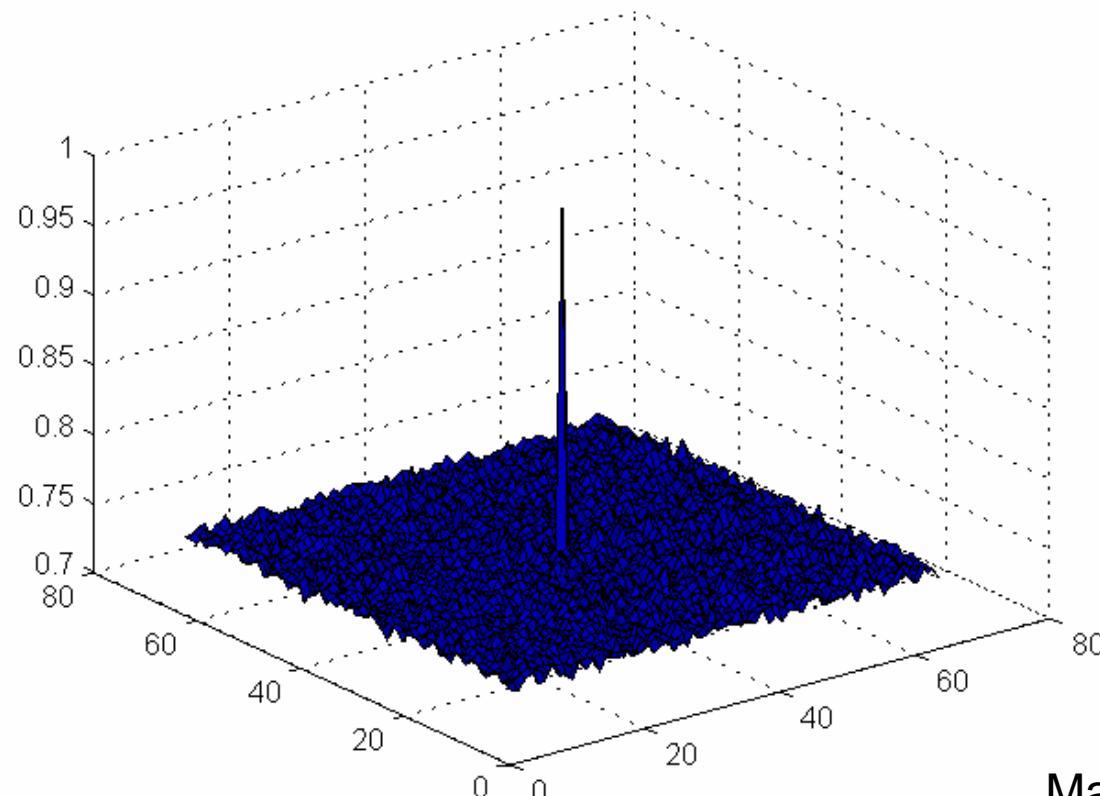


Convolution of $s(x,y)$ with $s(x,y)$



Correlation of $s(x,y)$ with $s(x,y)$ (auto-correlation)

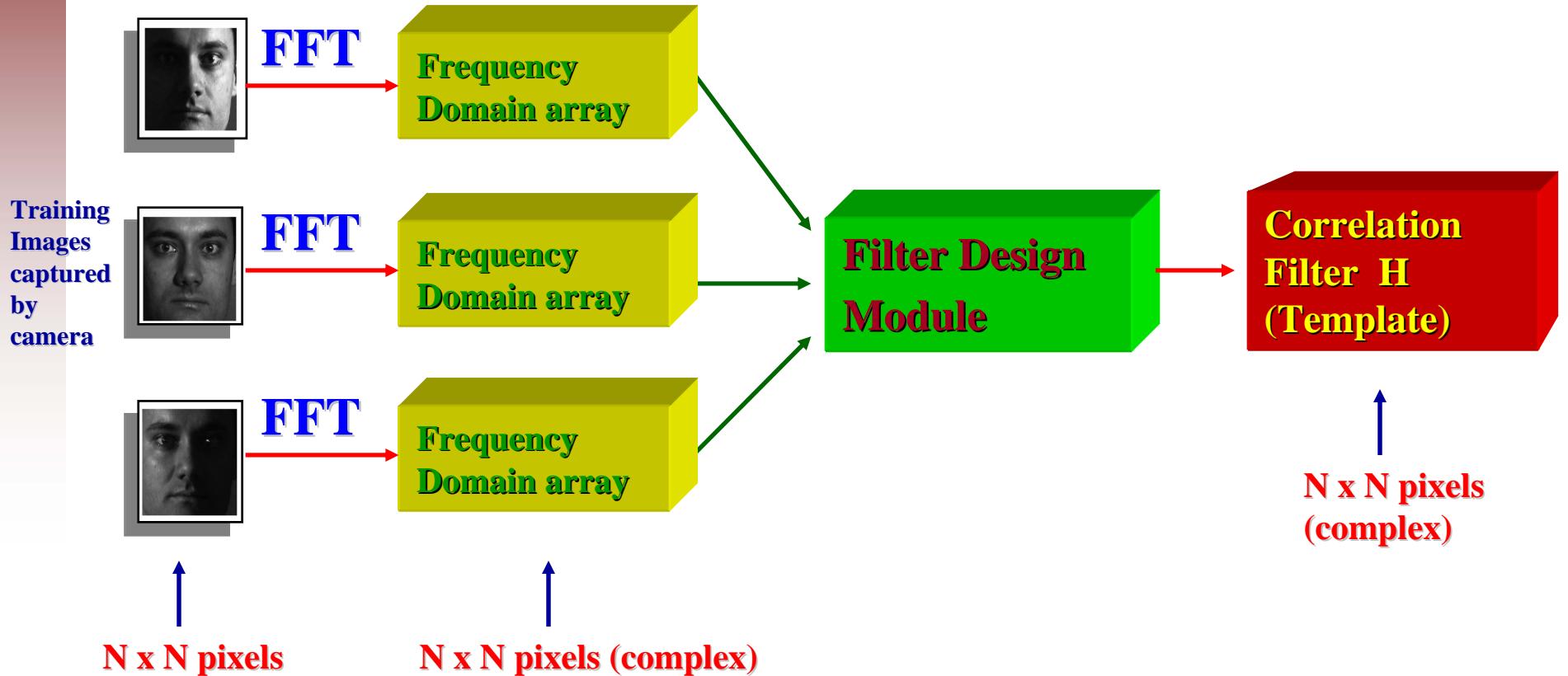
- Notice nice peak..with height of 1 at the location where the images perfectly align. The peak height indicates confidence of match. Because $s(x,y)$ is random signal, no other shifts of the signal match and there is only a single peak appearing exactly where the two signals are aligned.



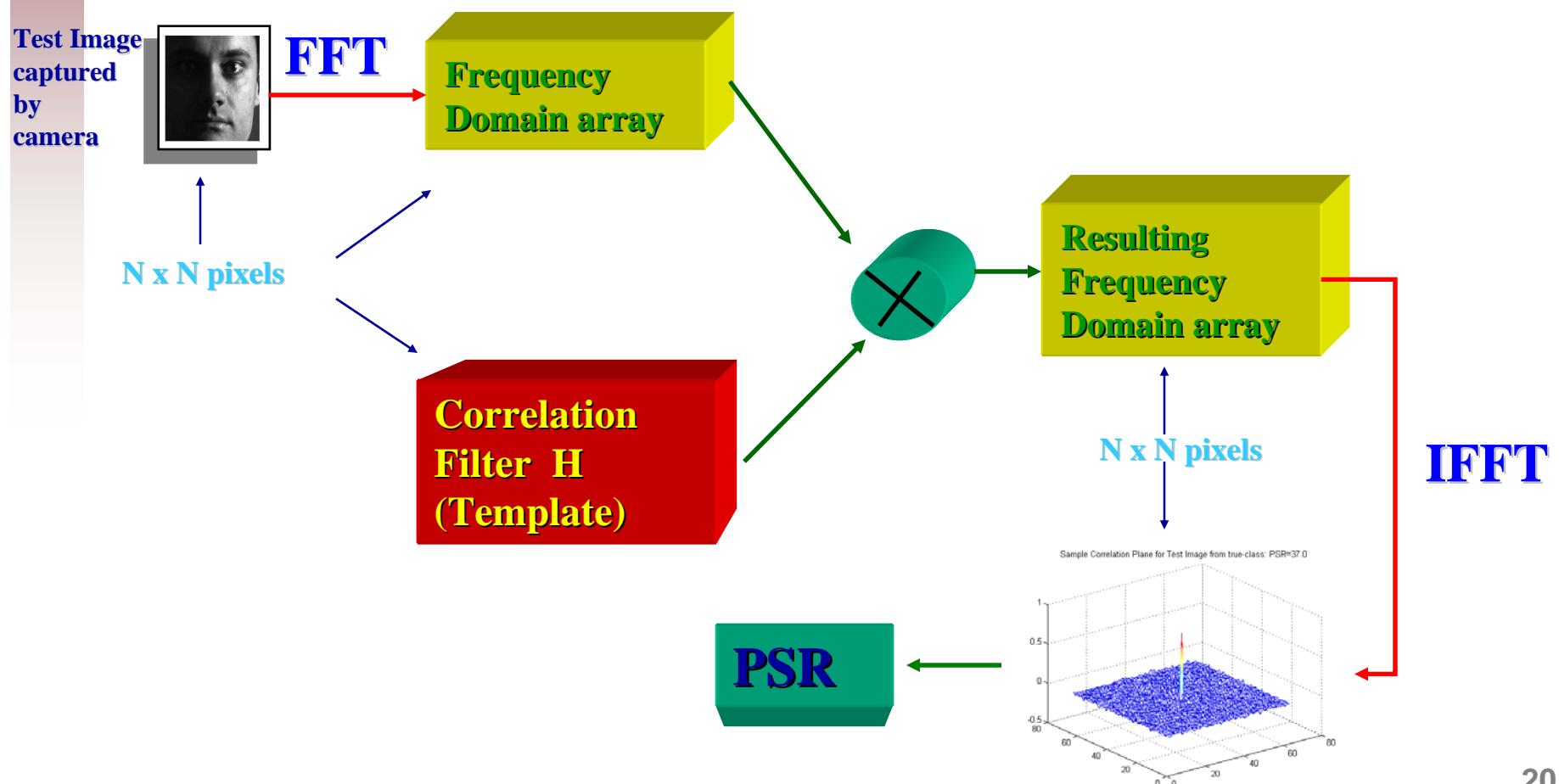
Matched Filter

- Matched Filter is simply the reference image that you want to match in the scene.
- Matched Filter is optimal for detecting the known reference signal in additive white Gaussian noise (AWGN) – it has maximum Signal-to-Noise Ratio.
- OK...but what are the short-comings of Matched Filters?
 - ▼ Matched Filters only match with the reference signal, and even slight distortions will not produce a match.
 - ▼ Thus you need as many matched filters as number of training images (N).
 - ▼ Not feasible from a computational viewpoint as you will need to perform N correlations and store all N filters.

Typical Enrollment for Correlation Filters



Recognition stage



Equal Correlation Peak Synthetic Discriminant Function (ECP-SDF) Filter

- Idea is to overcome the limitations of MFs, by building a single correlation filter from multiple training images that will be able to recognize all of them.
- How?
- So we want to keep the ideas from MF, in that we want the peak at the origin (i.e. when the two signals align) to be 1 for all the training reference images.
- And we want the filter to be in the convex hull of the training images, i.e. it is made up of a linear combinations of MF (we need to determine the weights).

ECP-SDF Filter - details

- Assume h is a vector containing the impulse response of the filter (vectorized from 2D).
- Let x_i be vectorized i th training image into a column vector.
- We want the peak at the origin to be 1, i.e. we want the inner-product (since correlations are just inner-products of the filter h and signal x at different spatial shifts).

$$h^T x_i = 1$$

So for all $i=1..N$ images we write this as:

$$X^T h = c$$

- Where matrix $X = [x_1 \ x_2 \ x_3 \ x_4]$ contains the images x_i in vectorized format along its columns.
- c is a column vector containing the peak constraints- in this case a vector of all 1's

ECP-SDF Filter continued....

- We also said that the ECP-SDF filter is a linear combination of Matched Filters (or equivalently) a linear weighted combination of the training images.

- i.e.

$$\mathbf{h} = \mathbf{X} \mathbf{a}$$

Where

$$\begin{bmatrix} | \\ \mathbf{h} \\ | \end{bmatrix} = \begin{bmatrix} | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_N \\ | & & | \end{bmatrix} \begin{bmatrix} | \\ \mathbf{a} \\ | \end{bmatrix} = a_1 \mathbf{x}_1 + a_2 \mathbf{x}_2 + a_3 \mathbf{x}_3 + \dots + a_N \mathbf{x}_N$$

Subst $\mathbf{h} = \mathbf{X} \mathbf{a}$ in $\mathbf{X}^T \mathbf{h} = \mathbf{c}$ to get

$$\mathbf{X}^T \mathbf{X} \mathbf{a} = \mathbf{c}$$

ECP-SDF Filter continued....

- Use $X^T X a = c$ to solve for the linear combination weights a

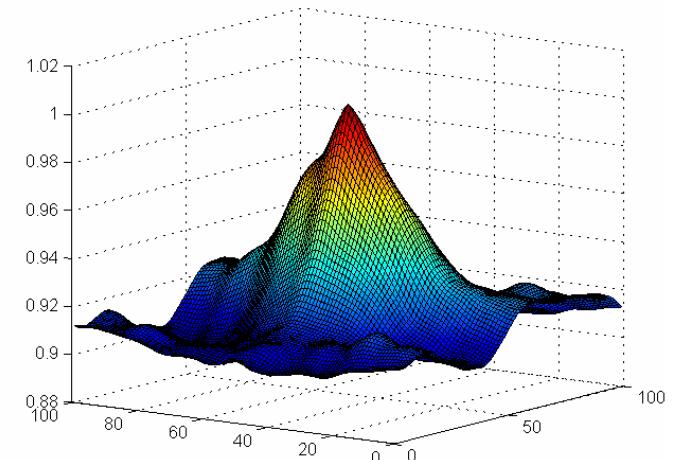
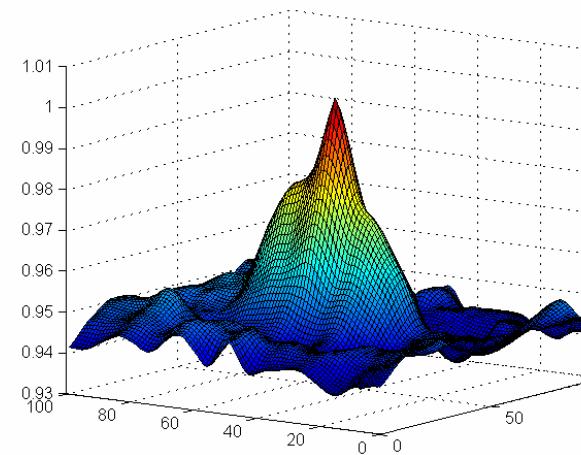
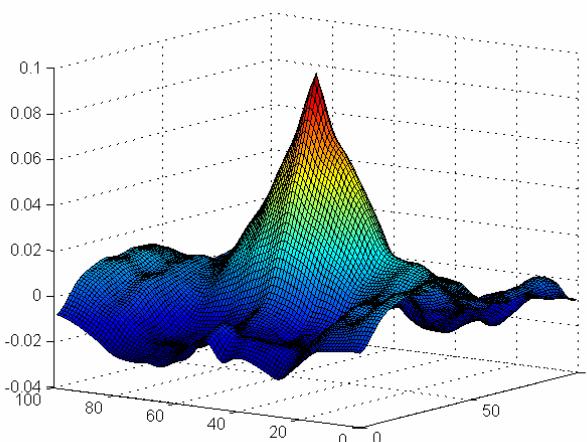
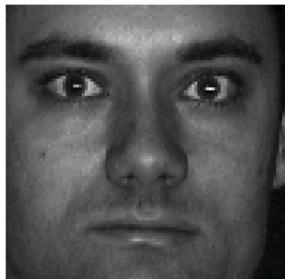
$$a = (X^T X)^{-1} c$$

- $X^T X$ is an inner-product matrix or Gram matrix of size $N \times N$ where N is the number of training images, thus computationally efficient.
- OK.. Now we know the linear combination weights so lets plug a back in our filter equation $h = X a$ to get

$$h = X (X^T X)^{-1} c$$

Example Correlation Outputs

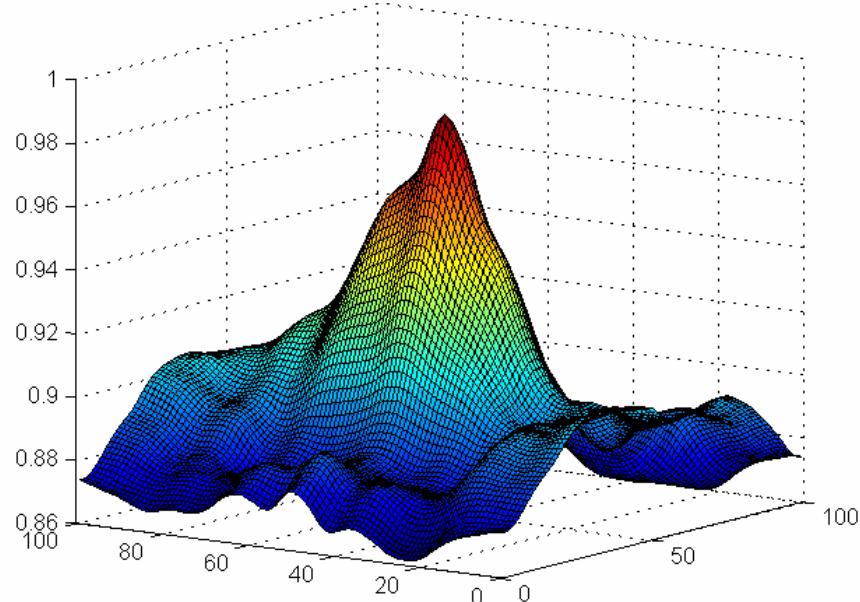
- Use these images for training (peak=1 for all correlation outputs)



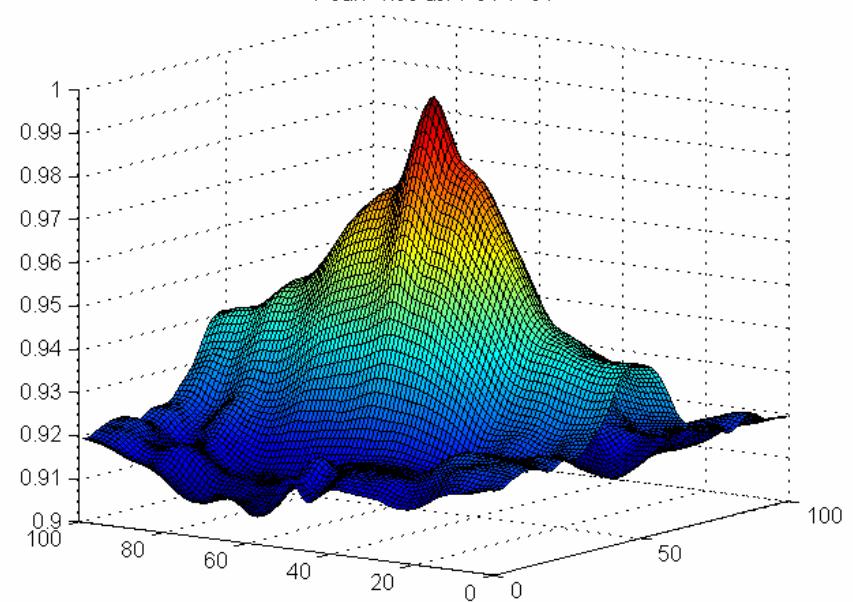
Example Correlation Output Impostors using ECP-SDF



Peak=0.99 at X=51 Y=50



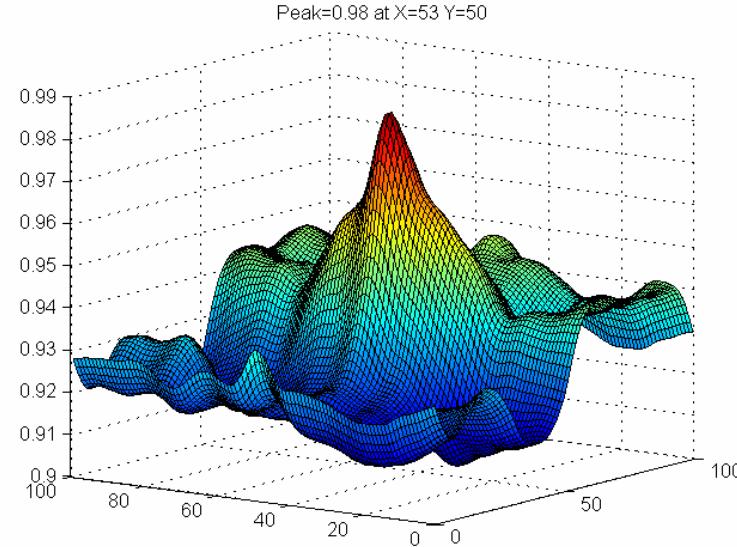
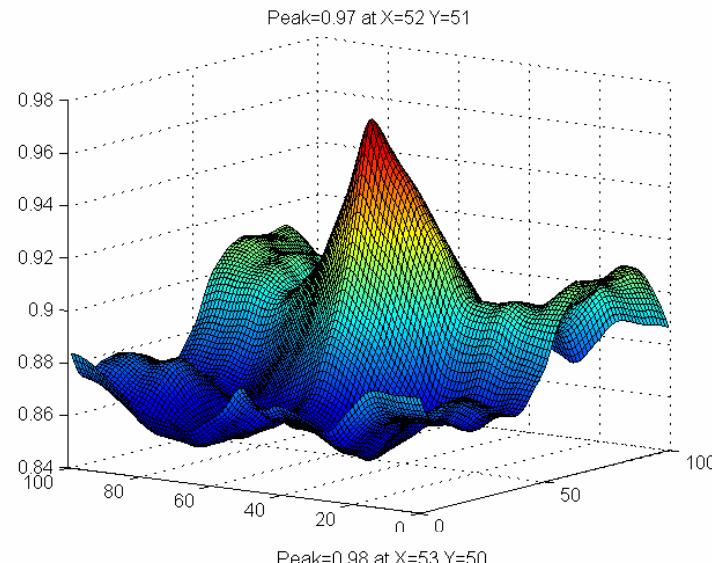
Peak=1.00 at X=51 Y=51



Very high peaks (0.99 and 1.0 for impostor class! Not good!

²⁶
Marios Savvides

Example correlation output for authentic people (but slightly different illumination)



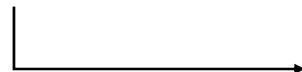
Conclusion – ECP-SDF

- ECP-SDF does not generalize well...atleast not for illumination variations.
- Discrimination is poor, as we saw we got high peaks for impostor classes.
- Does not guarantee peak is at origin, we may get a sharp peak at some other location.
- Solution:- Minimum Average Correlation Filters (MACE) filters, to produce sharp peaks.

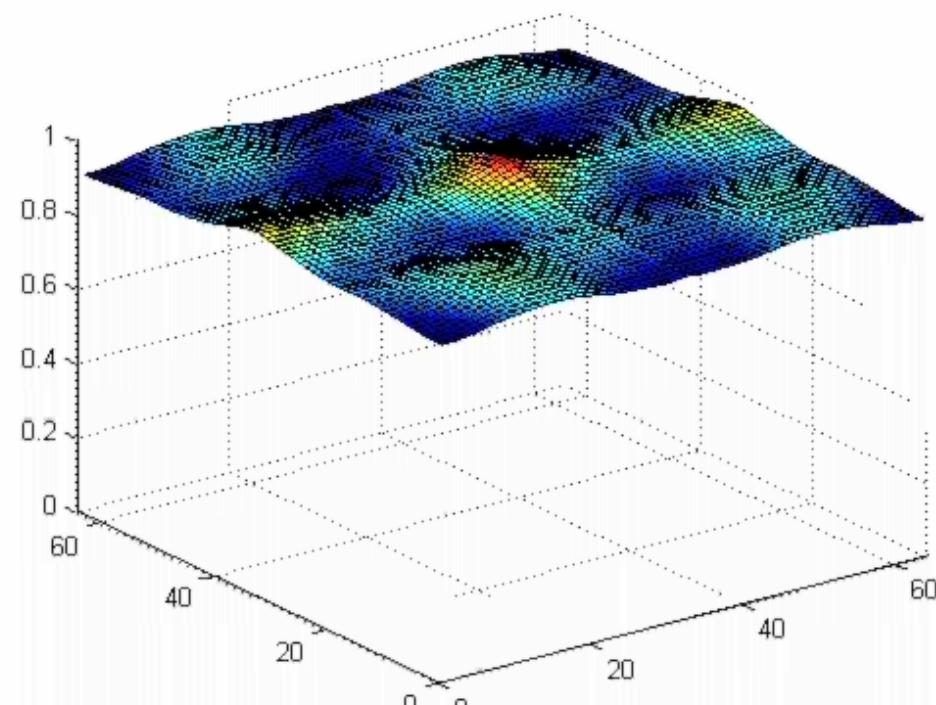
Minimum Average Correlation Energy (MACE) Filters

- MACE filter minimizes the average energy of the correlation outputs while maintaining the correlation value at the origin at a pre-specified level.
- Sidelobes are reduced greatly and discrimination performance is improved

MACE Filter produces a sharp peak at the origin



Correlation Plane



MACE Filter Formulation

- Minimizing spatial correlation energy can be done directly in the frequency domain by expressing the i th correlation plane (c_i) energy E_i as follows:

$$E_i = \frac{1}{d} \sum_{p=1}^d |c_i(p)|^2 = \frac{1}{d} \sum_{k=1}^d |H(k)|^2 |X_i(k)|^2 = \mathbf{h}^+ \mathbf{X}_i \mathbf{X}_i^* \mathbf{h} = \mathbf{h}^+ \mathbf{D}_i \mathbf{h}$$

↑
Parseval's Theorem!

- The Average correlation plane energy for the N training images is given by E_{ave}

$$E_{ave} = \frac{1}{N} \sum_{i=1}^N E_i = \mathbf{h}^+ \left[\frac{1}{N} \sum_{i=1}^N \mathbf{X}_i \mathbf{X}_i^* \right] \mathbf{h} = \mathbf{h}^+ \mathbf{D} \mathbf{h}$$

MACE Filter Formulation (Cont'd.)

- The value at the origin of the correlation of the i -th training image is:

$$c_i(0) = \sum_{p=1}^d H(p)^* X_i(p) e^{j2\pi 0 p} = \sum_{p=1}^d H(p)^* X_i(p) = \mathbf{h}^+ \mathbf{x}_i$$

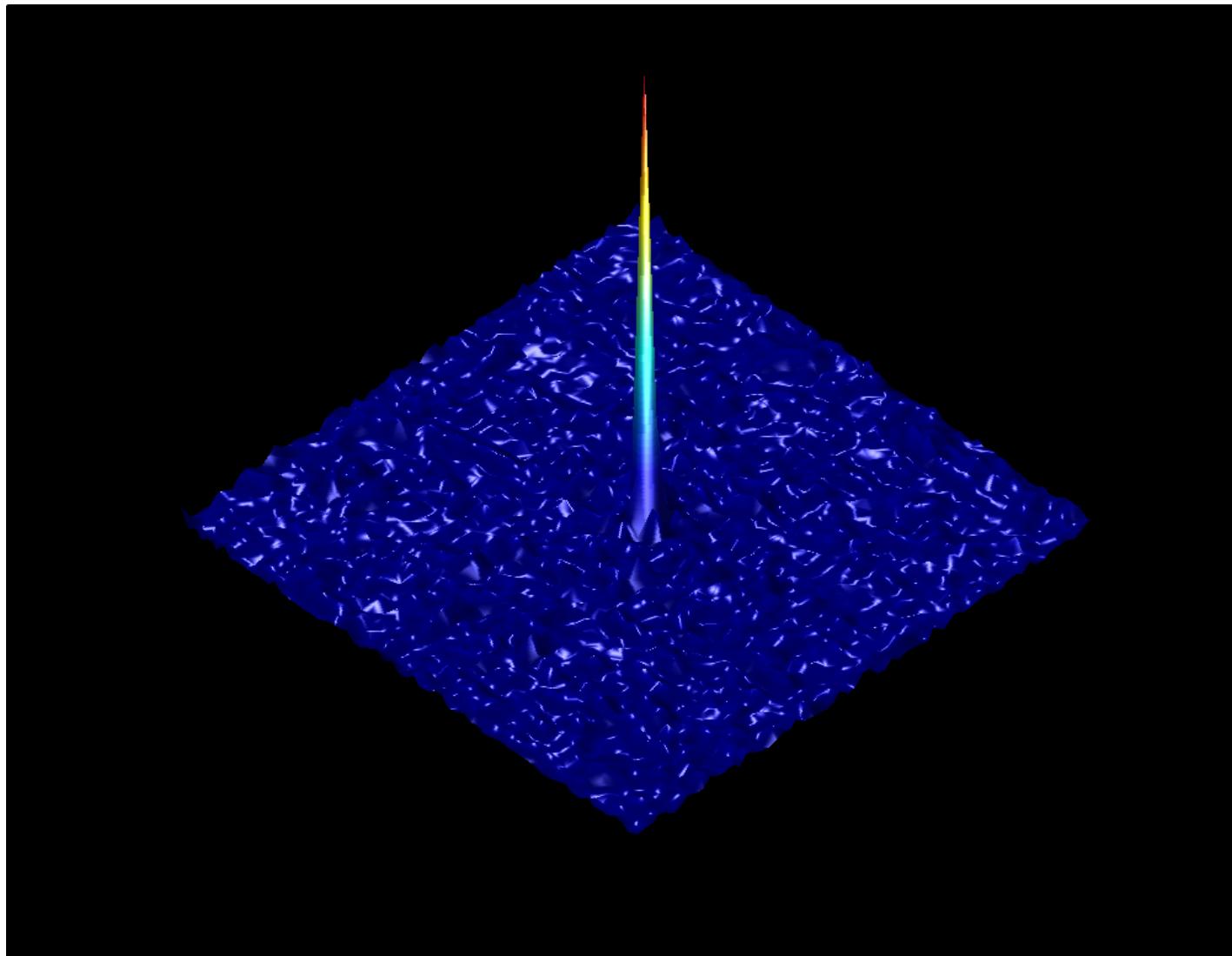
- Specify the correlation peak values for all the training image using column vector \mathbf{u}

$$\mathbf{X}^+ \mathbf{h} = \mathbf{u}$$

- Minimizing the average correlation energies $\mathbf{h}^+ \mathbf{D} \mathbf{h}$ subject to the constraints $\mathbf{X}^+ \mathbf{h} = \mathbf{u}$ leads to the MACE filter solution.

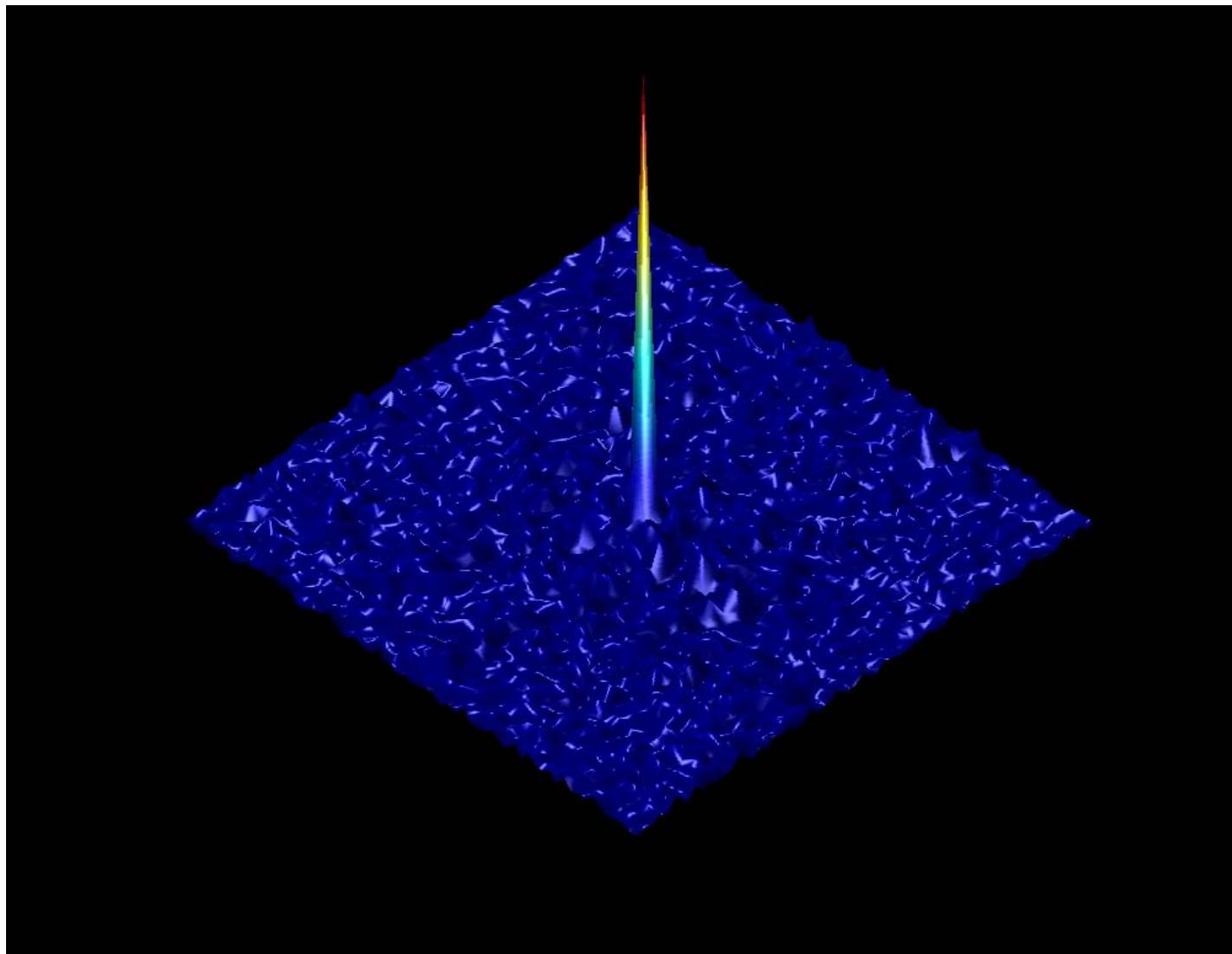
$$\mathbf{h}_{\text{MACE}} = \mathbf{D}^{-1} \mathbf{X} (\mathbf{X}^+ \mathbf{D}^{-1} \mathbf{X})^{-1} \mathbf{u}$$

Example Correlation Outputs from an Authentic



32

Example Correlation Outputs from an Impostor



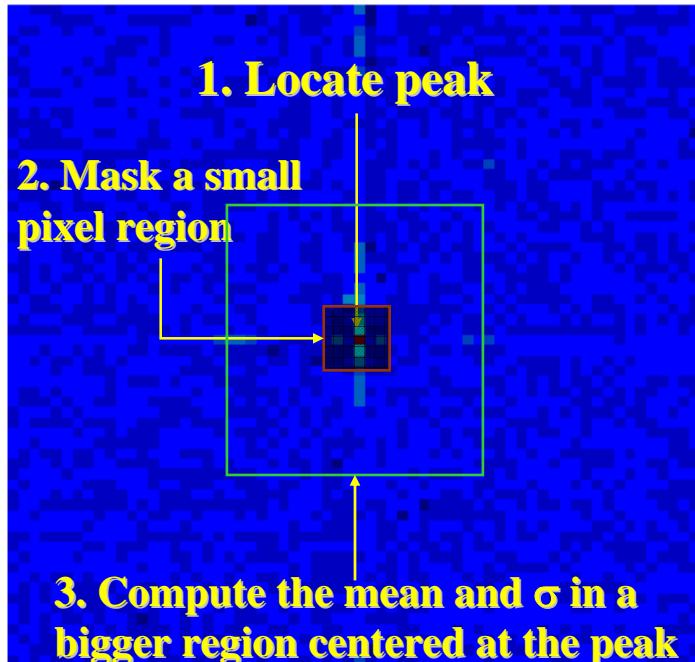
33

Different Figure of Merit (FOM)

- For Matched Filter we showed that the peak height is what was used for recognition.
- For MACE filters, the optimization is not only to keep the peak height at 1 but also to create sharp peaks.
- Thus it makes sense to use a metric that can measure whether we have achieved this optimization
- We need a metric that can measure the Peak sharpness! As images that resemble the training classes will produce a sharp peak whereas impostor classes will not produce sharp peaks as they were not optimized to do so!

Peak to Sidelobe Ratio (PSR)

- PSR invariant to constant illumination changes

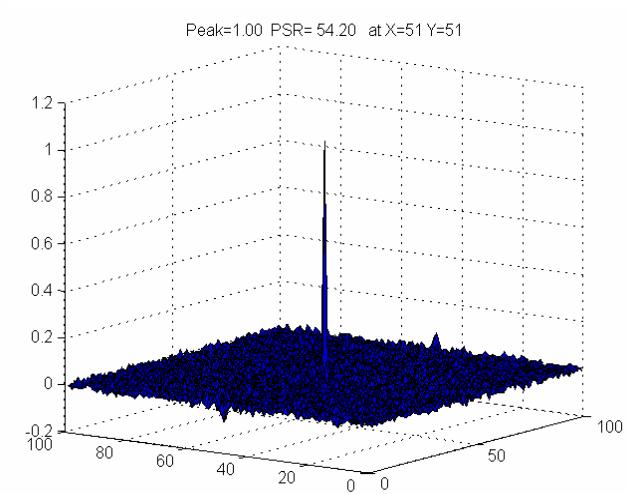
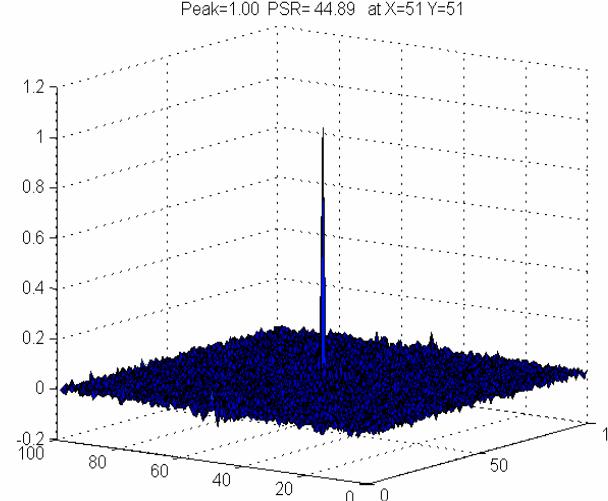
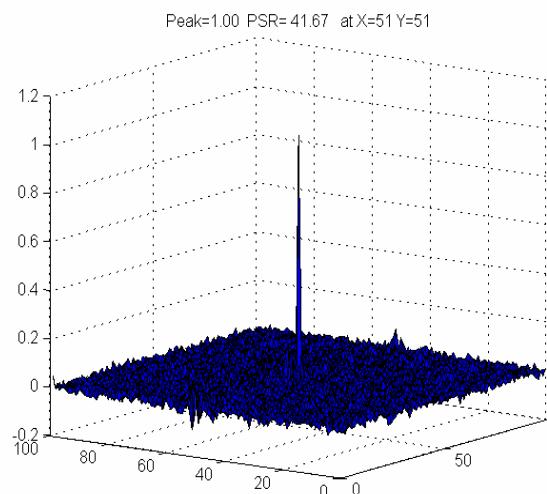
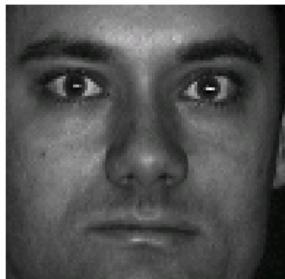


$$PSR = \frac{Peak - mean}{\sigma}$$

- Match declared when PSR is large, i.e., peak must not only be large, but sidelobes must be small.

Example Correlation Outputs

- Use these images for training (peak=1 for all correlation outputs)



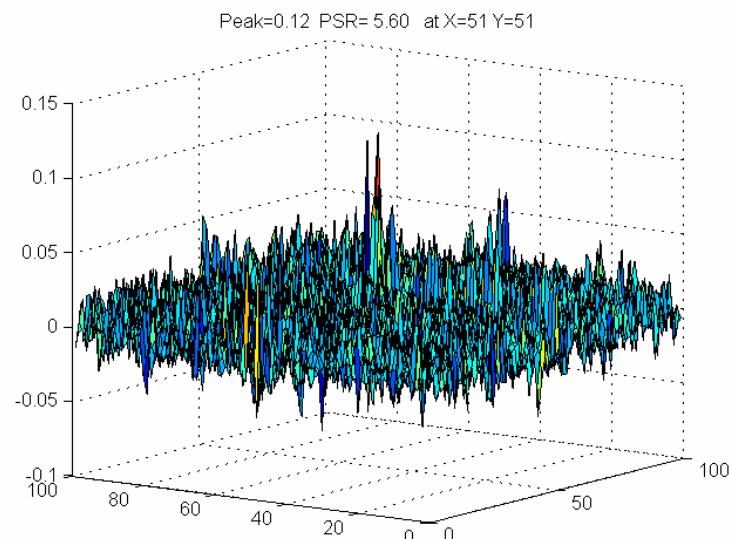
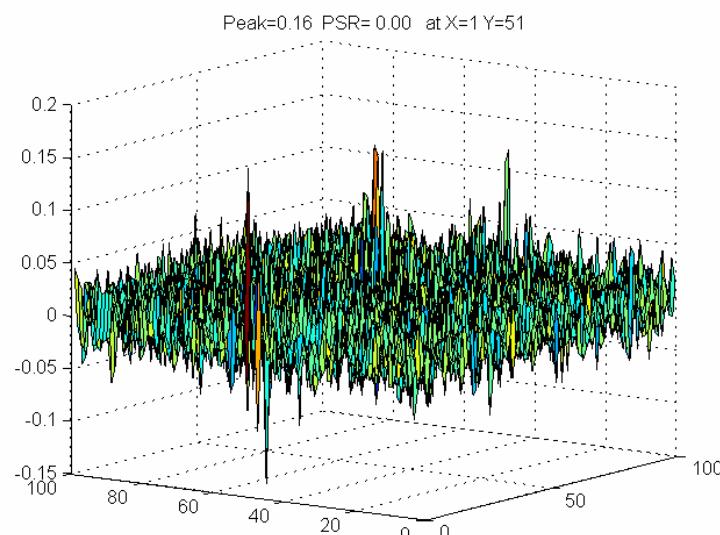
PSR: **41.6**

44.8

54.2

36

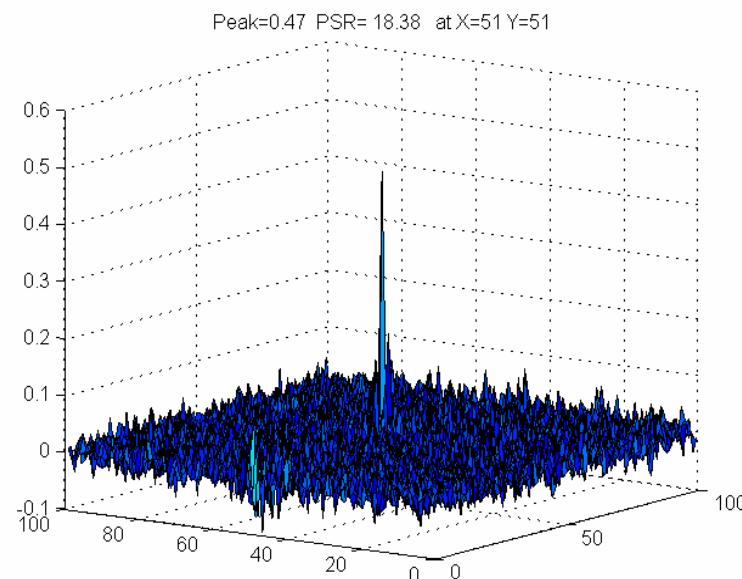
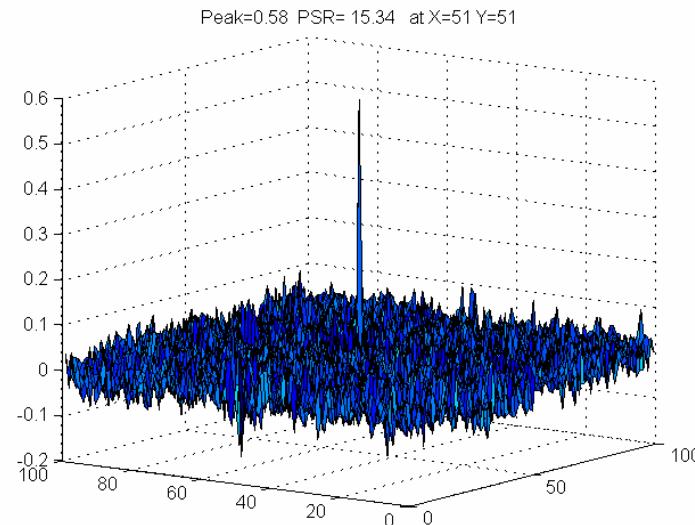
Example Correlation Output Impostors using MACE



No discernible peaks (0.16 and 0.12 and PSR 5.6 for impostor class! Very good!

37

Example correlation output for authentic people (but slightly different illumination)

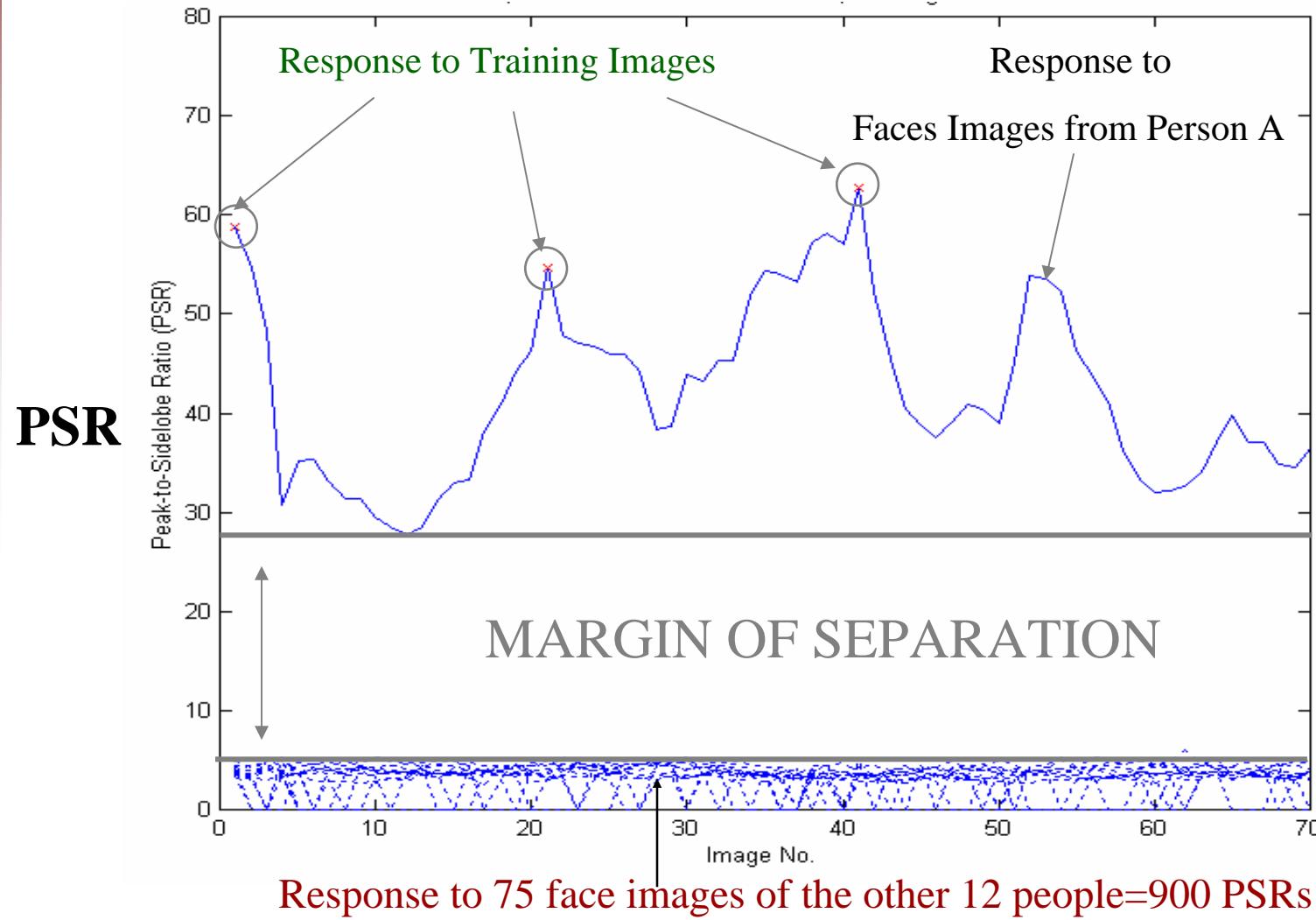


Facial Expression Database

- Facial Expression Database (AMP Lab, CMU)
 - 13 People
 - 75 images per person
 - Varying Expressions
 - 64x64 pixels
 - Constant illumination
- 1 filter per person made from 3 training images



PSRs for the Filter Trained on 3 Images



49 Faces from PIE Database illustrating the variations in illumination

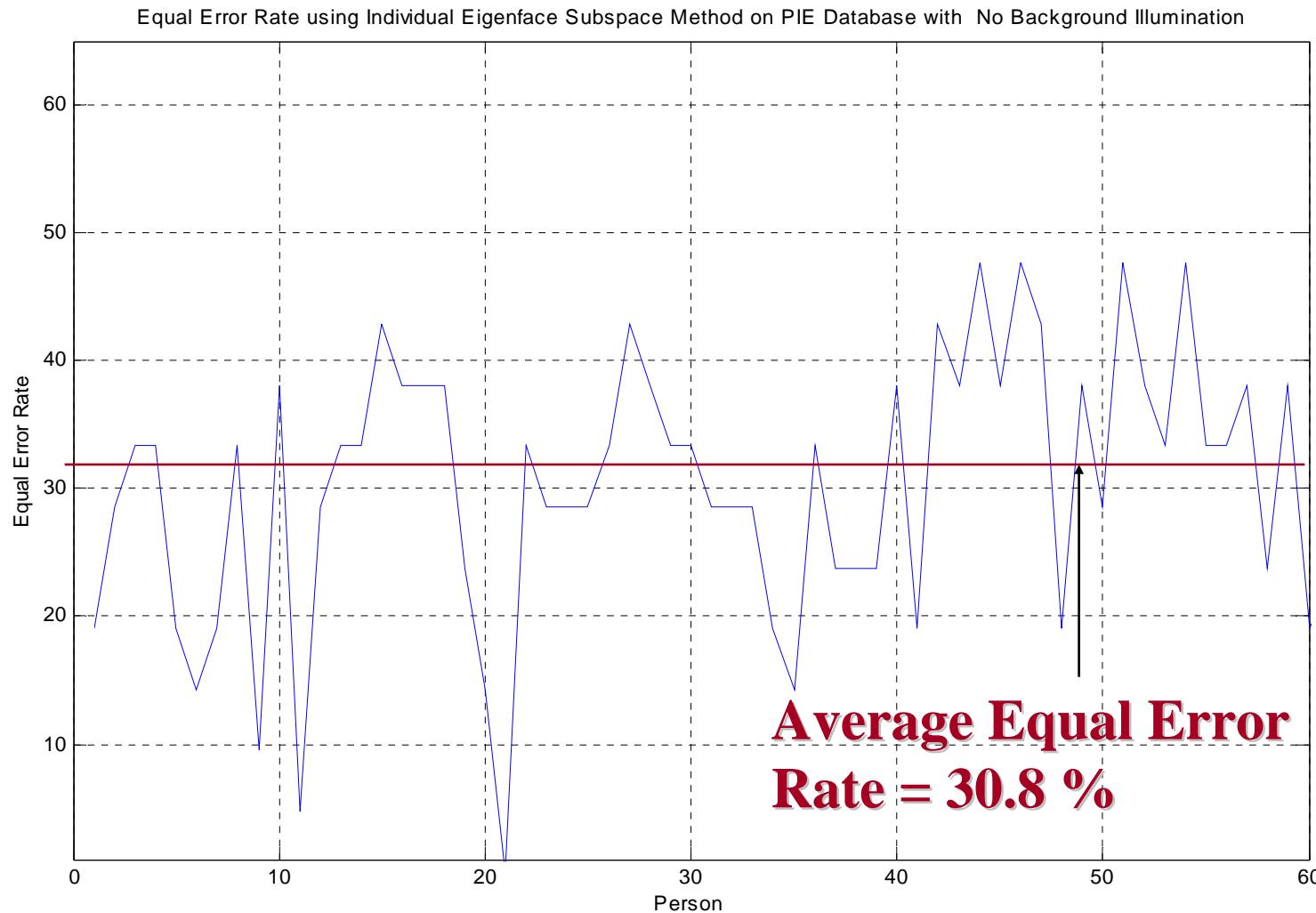


Training Image selection

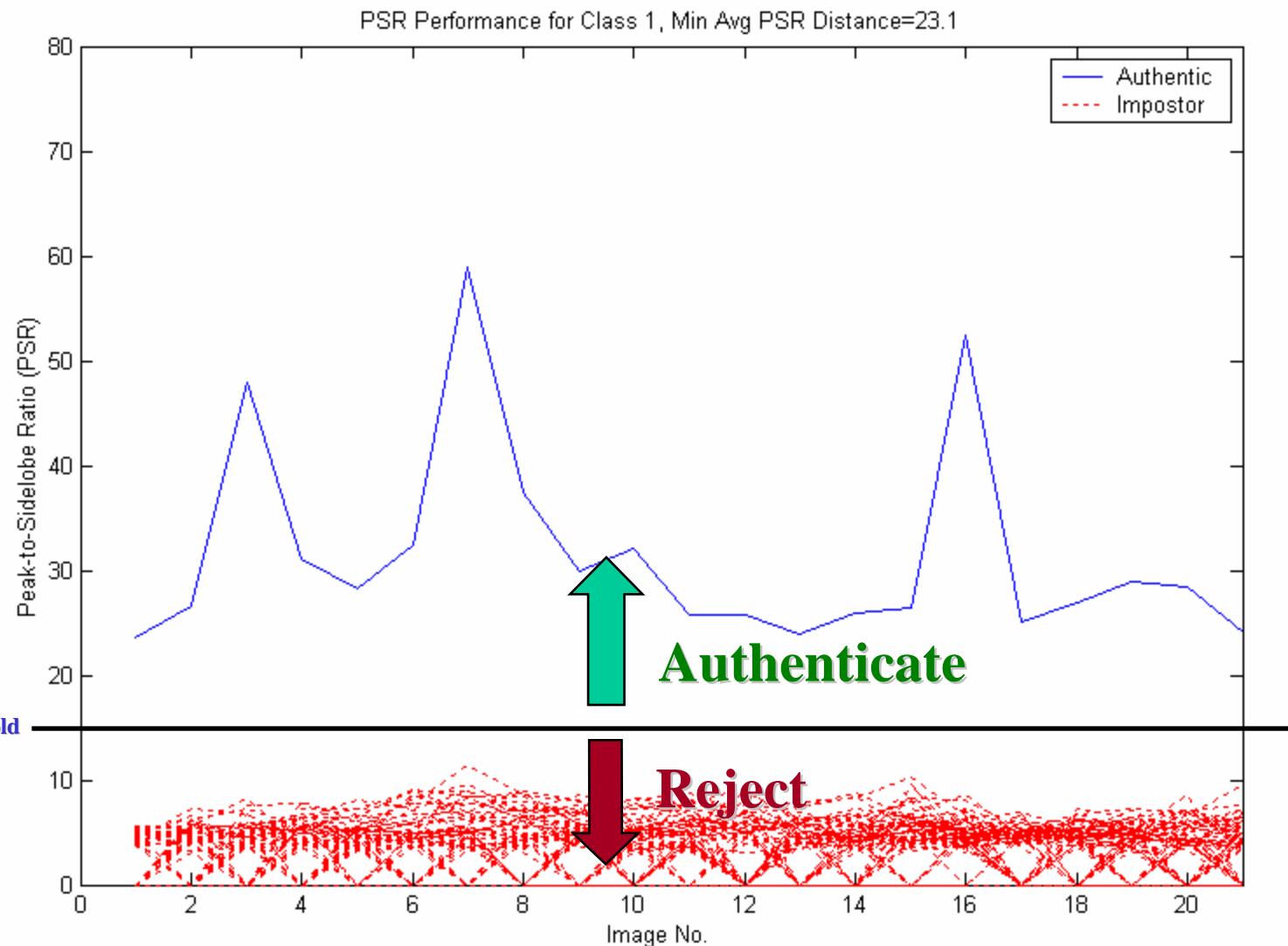
- We used *three* face images to synthesize a correlation filter
- The three selected training images consisted of 3 extreme cases (dark left half face, normal face illumination, dark right half face).

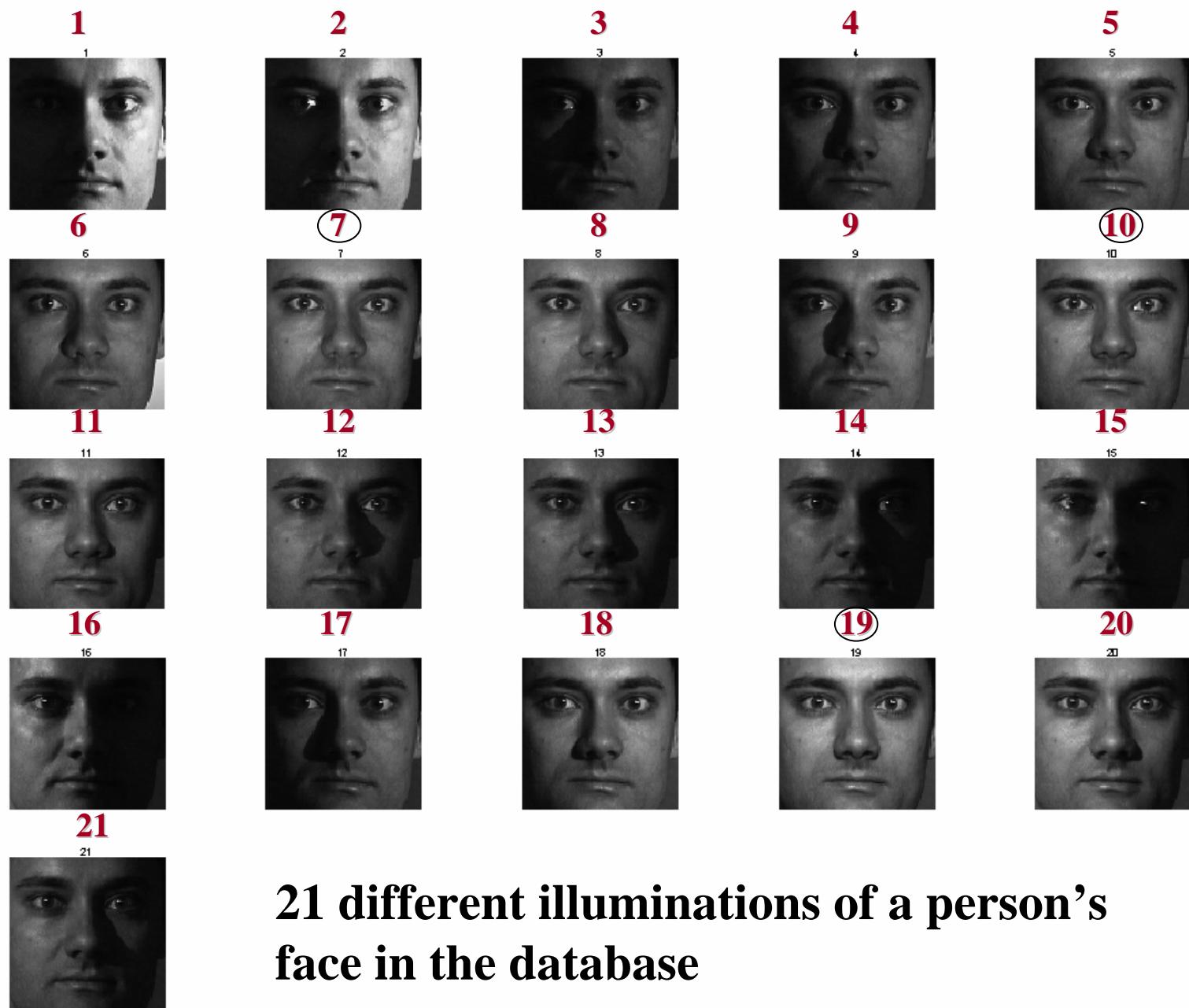


EER using IPCA with no Background illumination



EER using Filter with no Background illumination





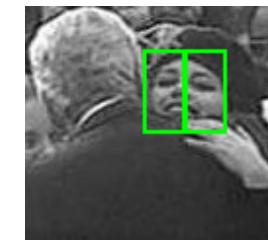
21 different illuminations of a person's face in the database

Recognition Accuracy using Frontal Lighting Training Images

PIE dataset (face images captured with room lights off)										
Frontal Lighting	IPCA		3D Linear Subspace		Fisherfaces		MACE Filters		UMACE Filters	
Training Images	# Errors	%Rec Rate	# Errors	%Rec Rate	# Errors	%Rec Rate	# Errors	% Rec Rate	# Errors	% Rec Rate
5,6,7,8,9,10, 11,18,19,20	33	97.6%	31	97.3%	36	97.3%	0	100%	0	100%
5,6,7,8,9, 10	110	91.4%	40	97.1%	145	89.3%	1	99.9%	0	100%
5,7,9,10	337	72.4%	93	93.2%	390	71.4%	1	99.9%	3	99.7%
7,10,19	872	36.1%	670	50.9%	365	73.3%	10	99.1%	10	99.1%
8,9,10	300	78.0%	30	97.8%	244	82.1%	1	99.9%	1	99.9%
18,19,20	122	91.0%	22	98.4%	79	94.2%	2	99.9%	1	99.9%

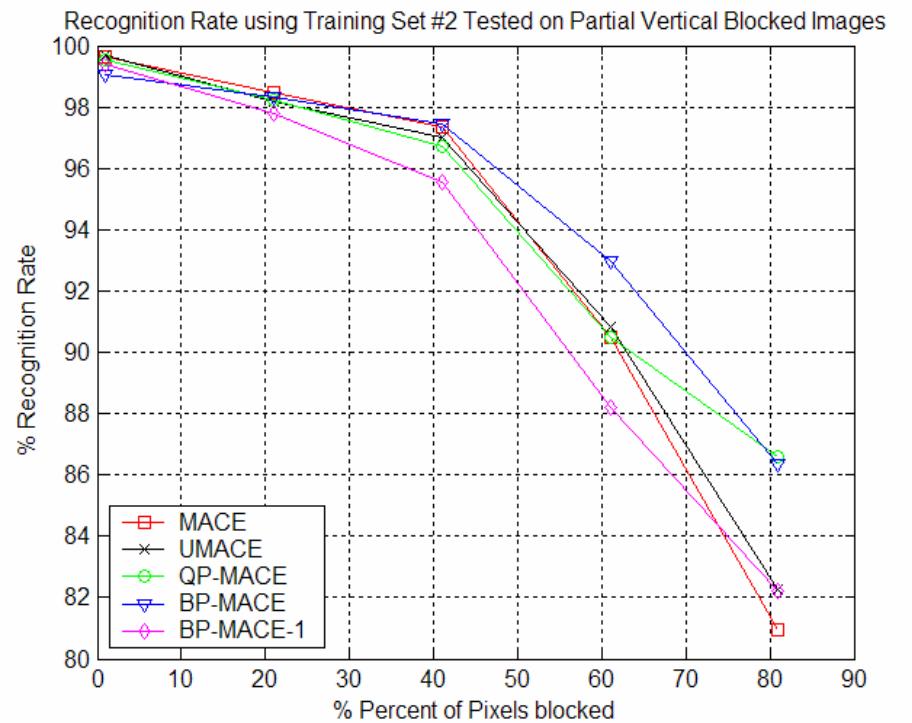
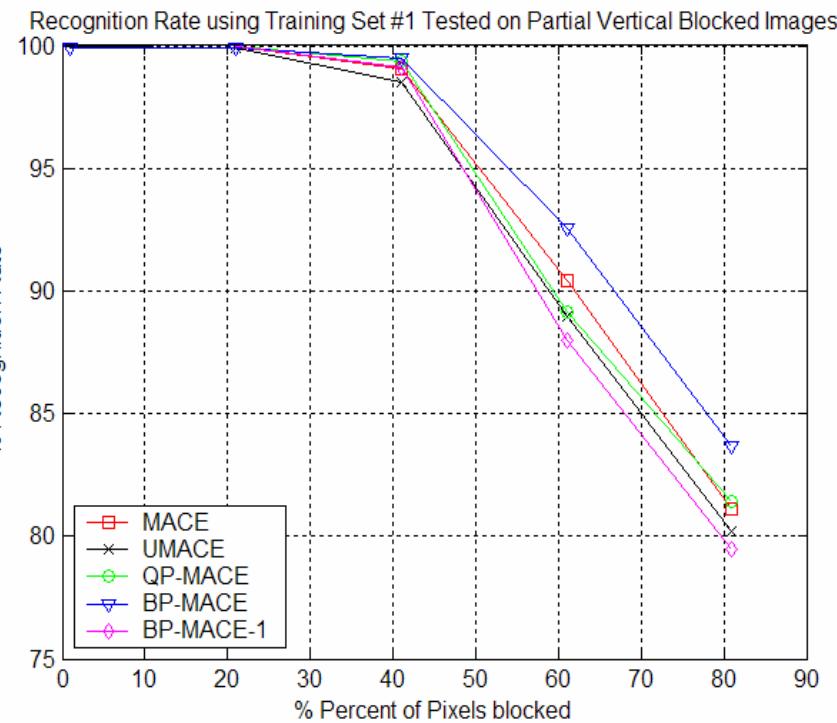
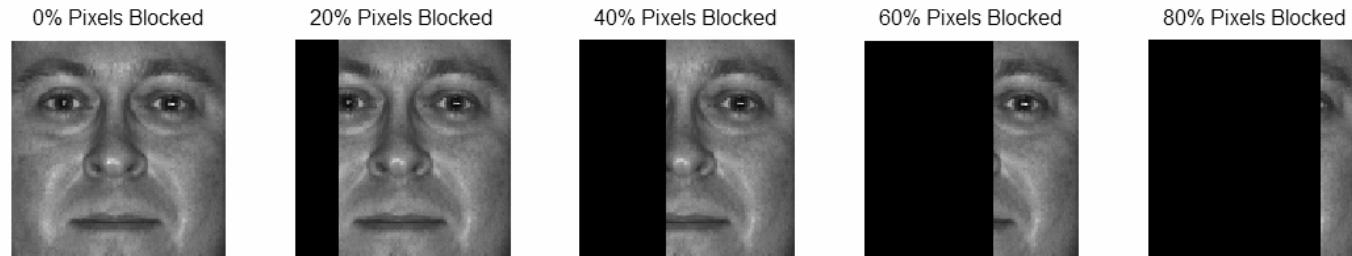
Face Identification from Partial Faces

- We have shown that these correlation filters seem to be tolerant to illumination variations, even when half the face is completely black and still achieve excellent recognition rates.
- What about partial face recognition?
- In practice a face detector will detect and retrieve part of the face (another type of registration error). In many cases, occluded by another face or object. Other face recognition methods fail in this circumstance.



*M. Savvides, B.V.K. Vijaya Kumar and P.K. Khosla, "Robust, Shift-Invariant Biometric Identification from Partial Face Images", Defense & Security Symposium, special session on Biometric Technologies for Human Identification (OR51) 2004.

Vertical cropping of test face image pixels (correlation filters are trained on FULL size images)



Using Training set #1 (3
extreme lighting images)

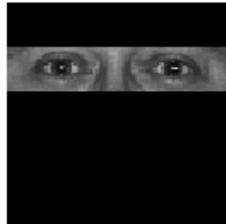
Using Training set #2 (3
frontal lighting images)

Recognition using selected face regions

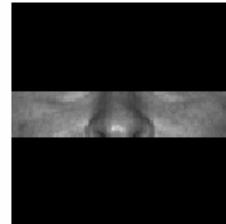
Face Section #1



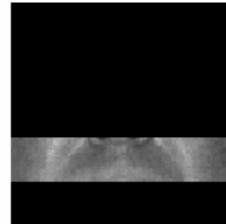
Face Section #2



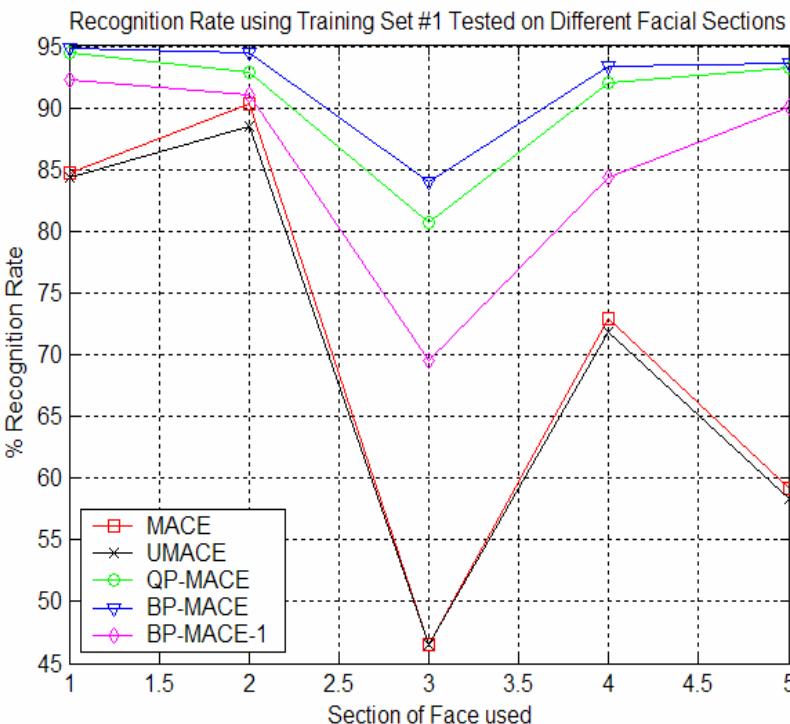
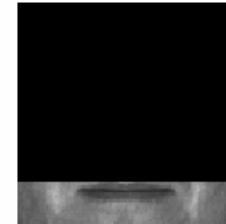
Face Section #3



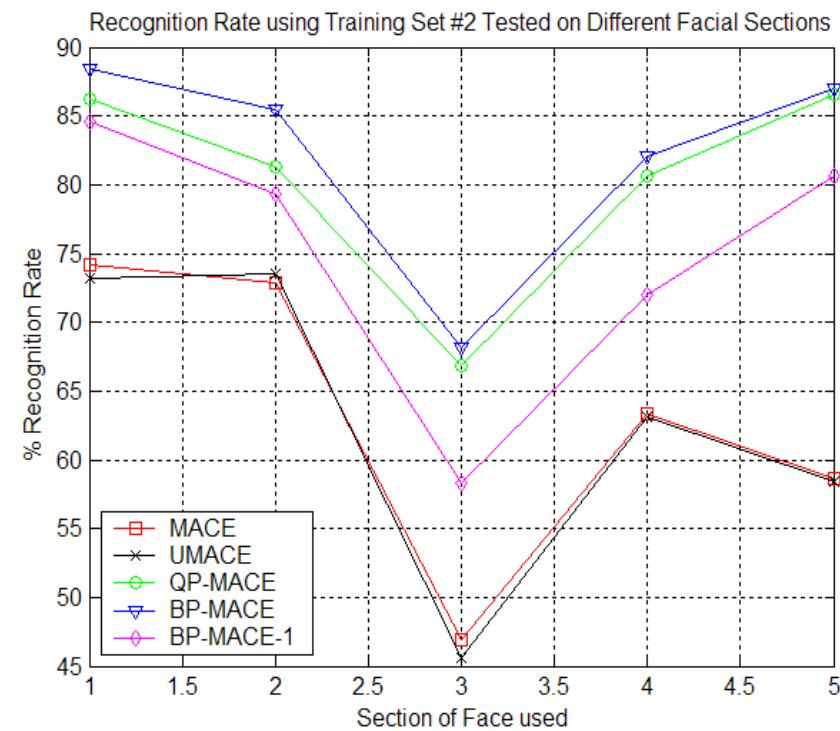
Face Section #4



Face Section #5

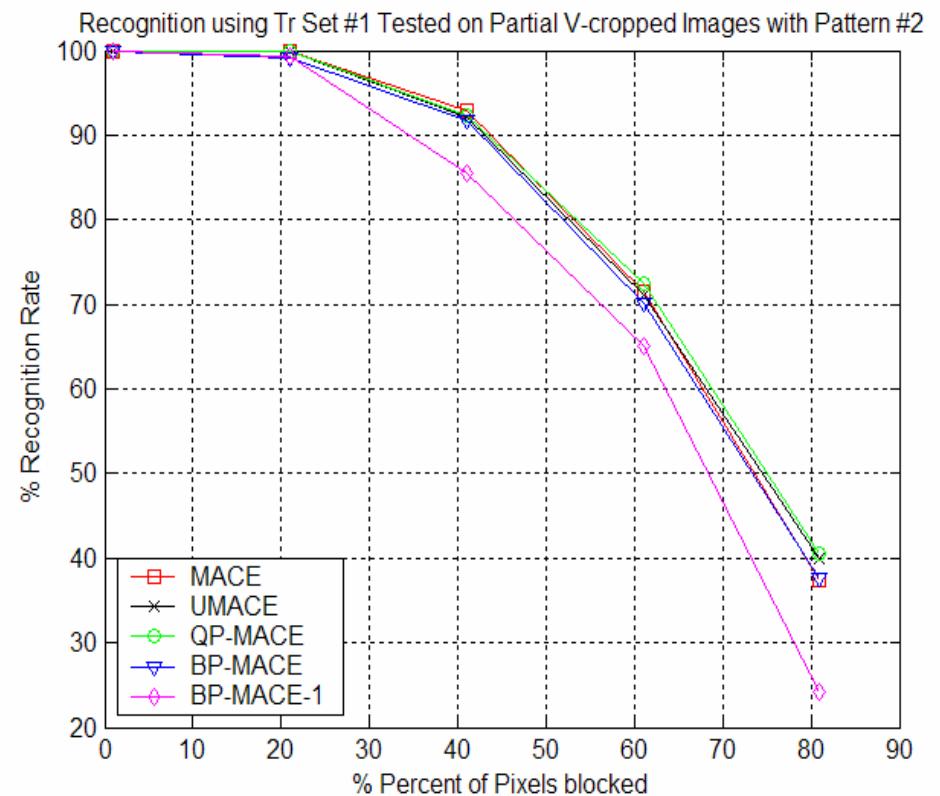
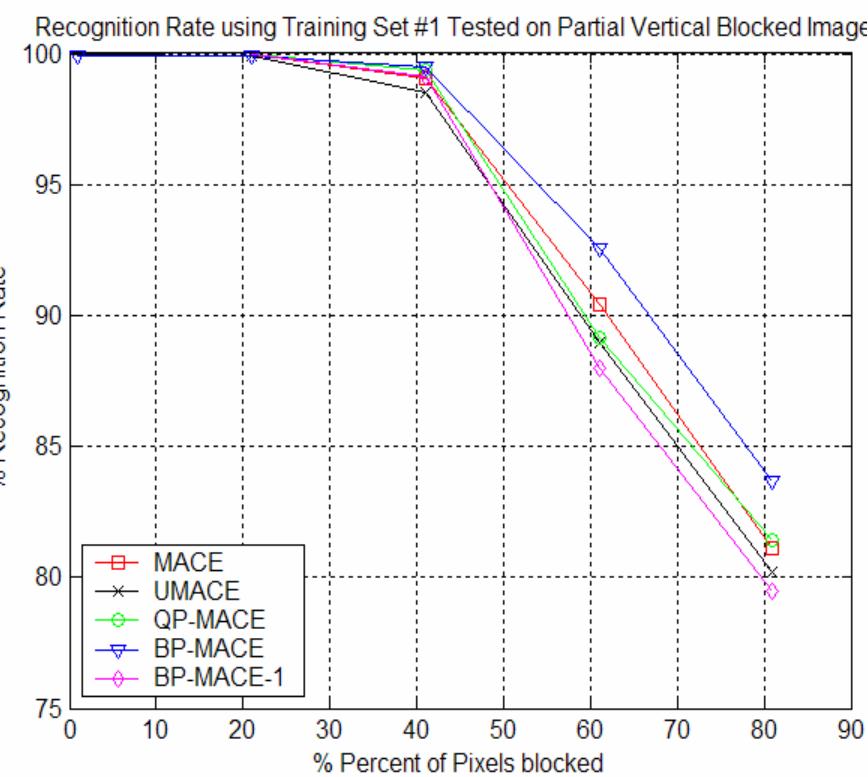
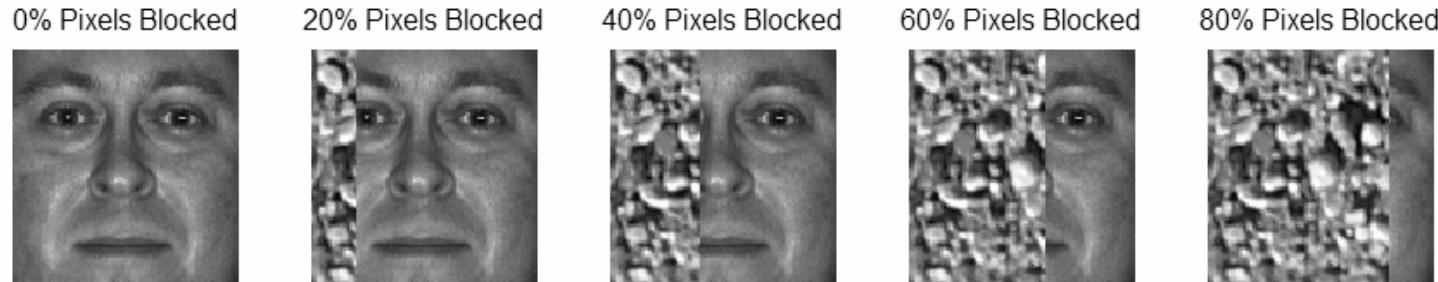


Using Training set #1 (3
extreme lighting images)



Using Training set #2 (3
frontal lighting images)

Vertical crop + texture #2

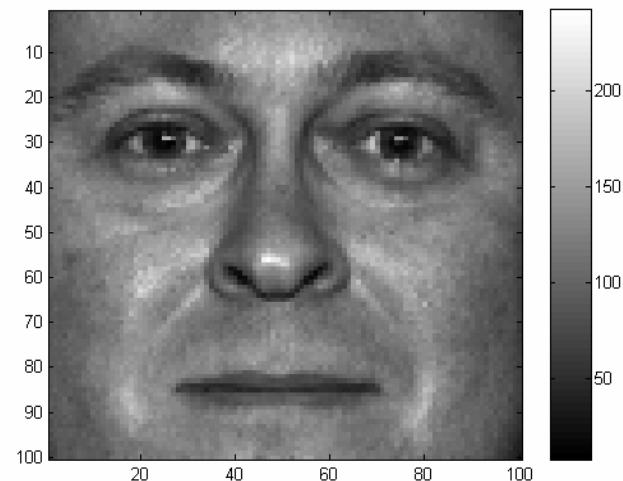


Zero intensity background

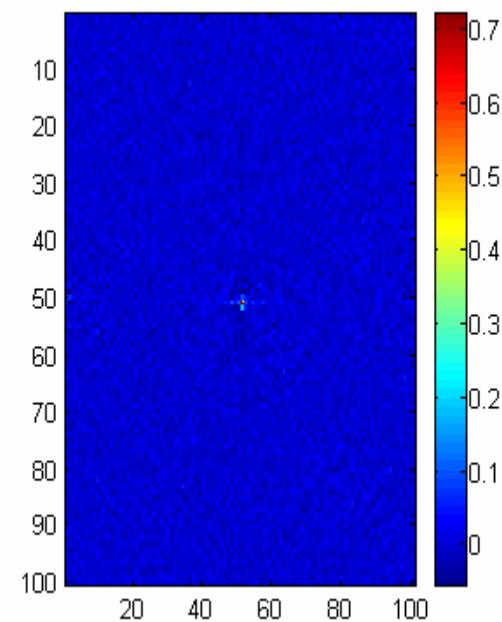
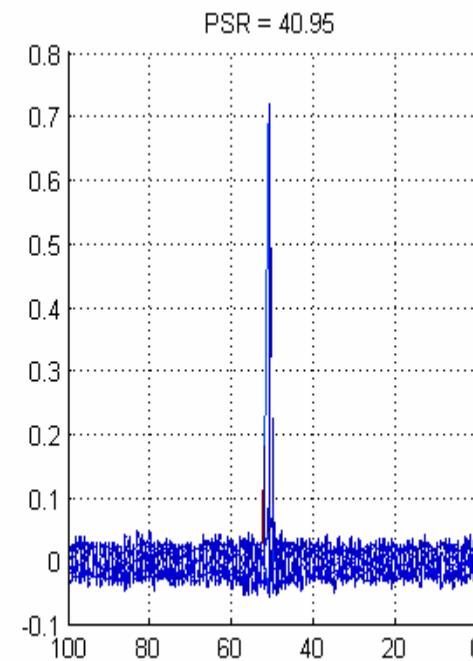
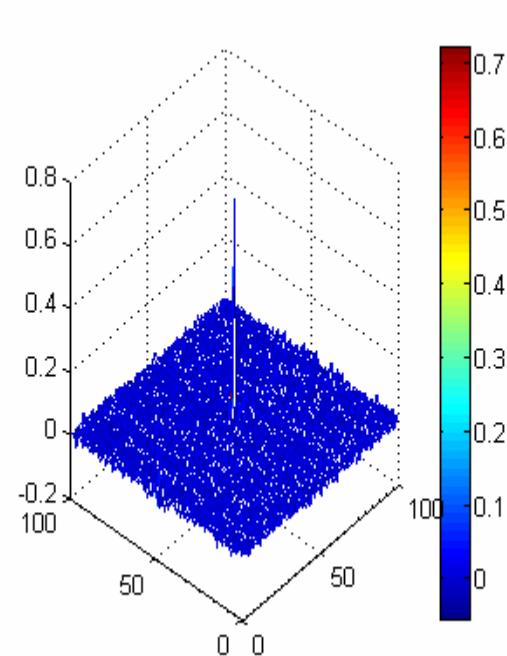
Textured background

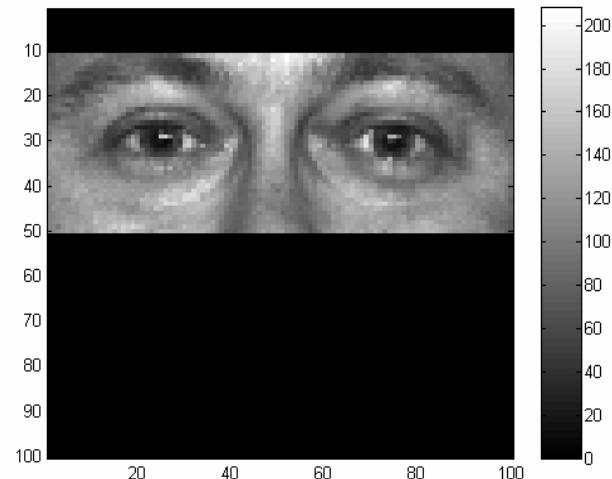
50

*M. Savvides, B.V.K. Vijaya Kumar and P.K. Khosla, "Robust, Shift-Invariant Biometric Identification from Partial Face Images", Defense & Security Symposium, special session on Biometric Technologies for Human Identification (OR51) 2004.

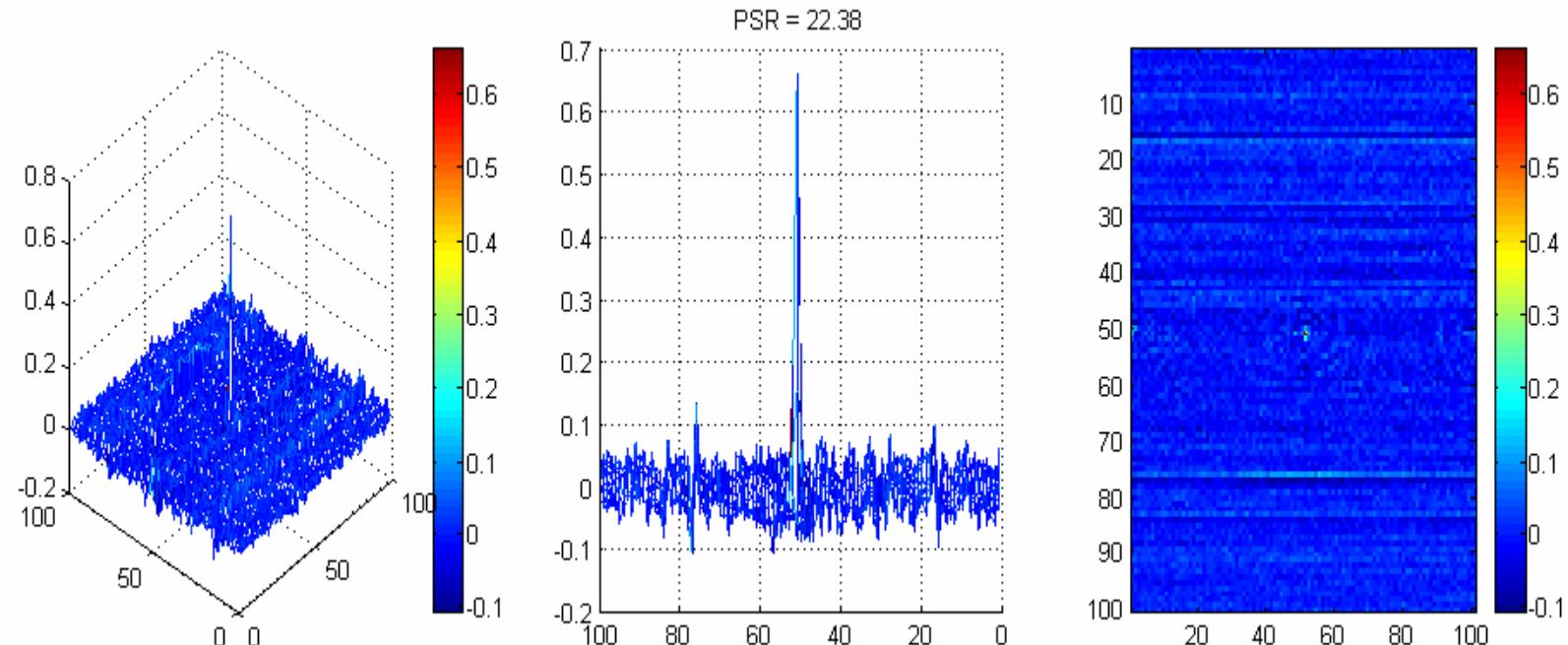


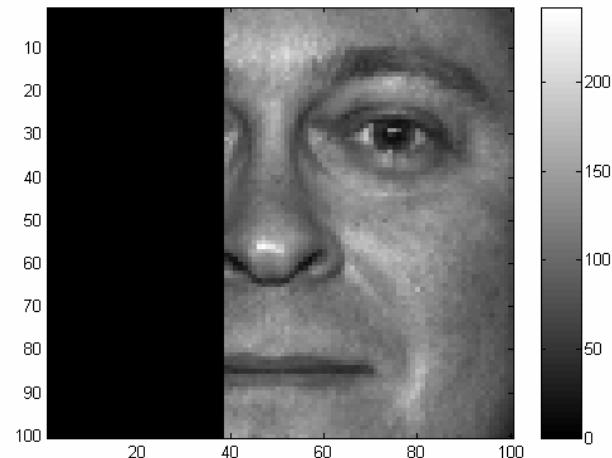
**Train filter on illuminations 3,7,16.
Test on 10.**



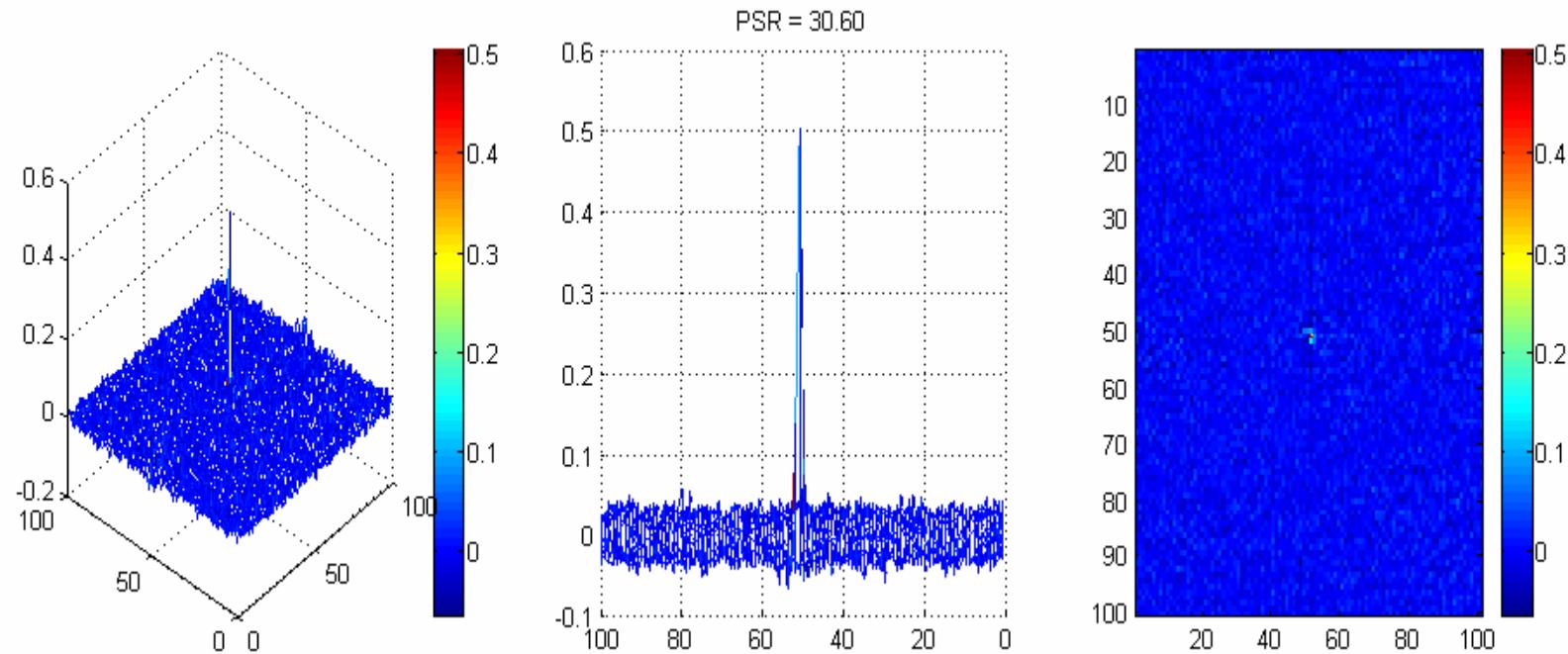


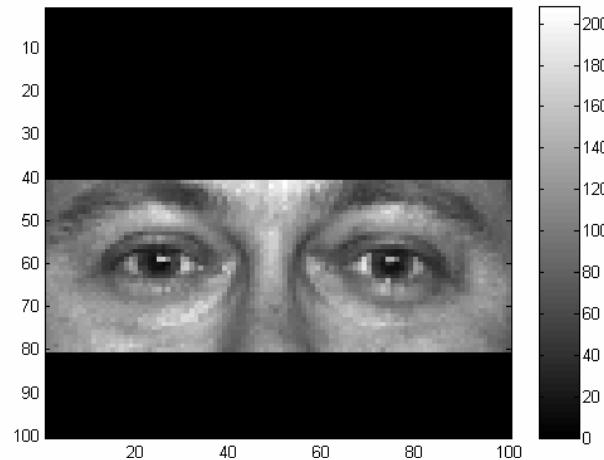
**Using same Filter trained before,
Perform cross-correlation on
cropped-face shown on left**





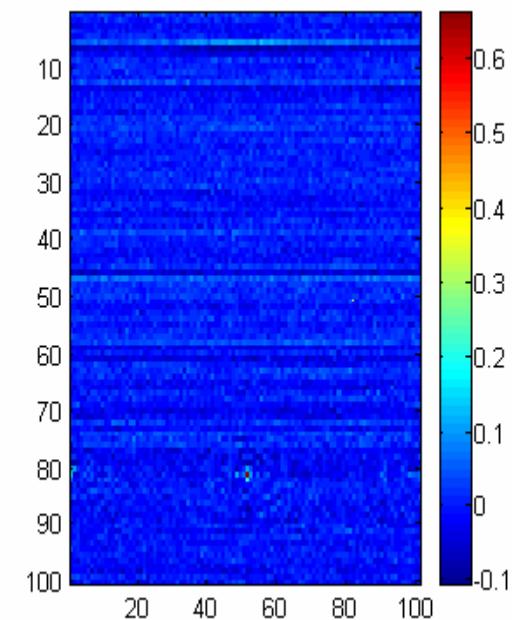
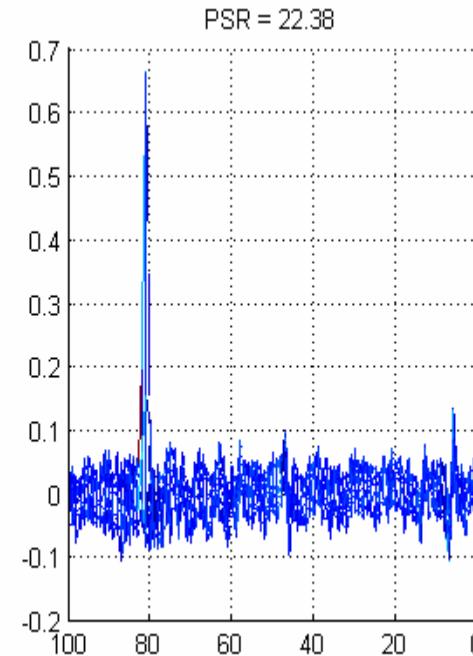
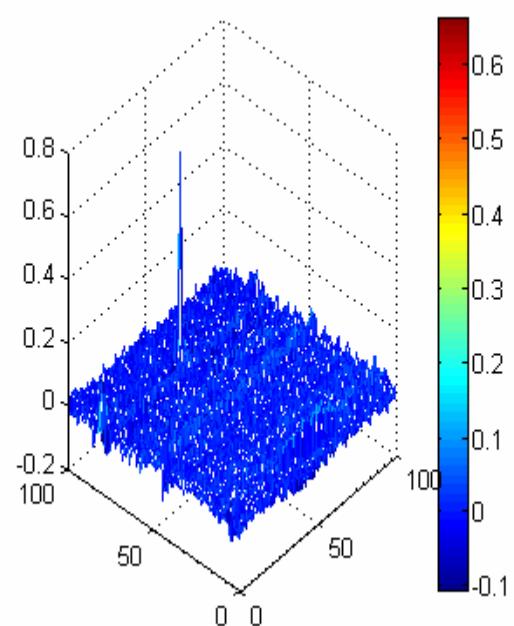
**Using same Filter trained before,
Perform cross-correlation on
cropped-face shown on left.**



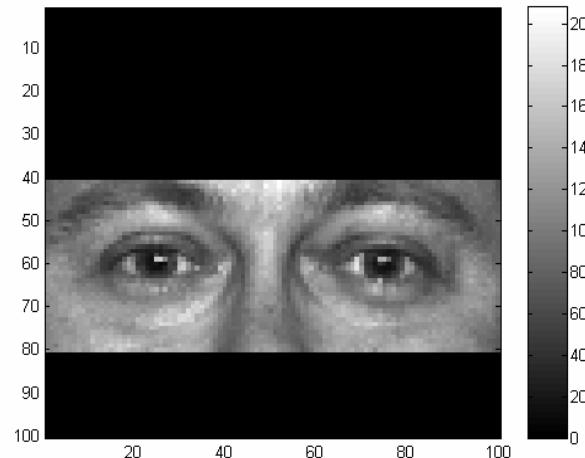


- CORRELATION FILTERS ARE SHIFT-INVARIANT

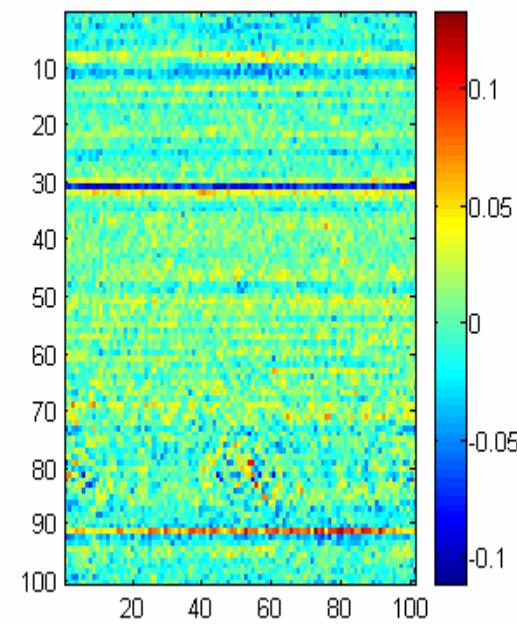
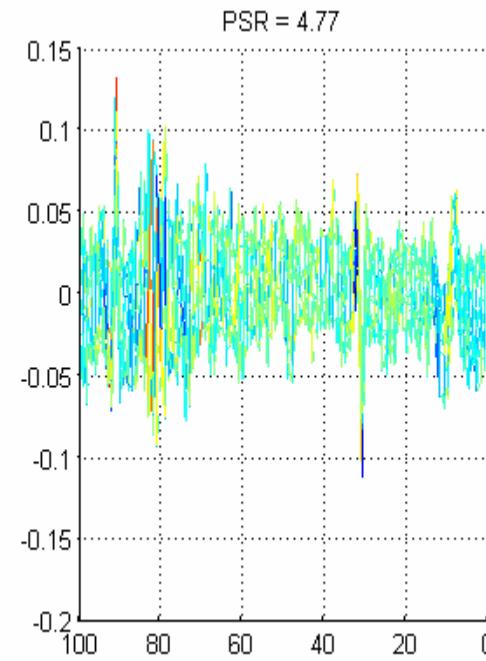
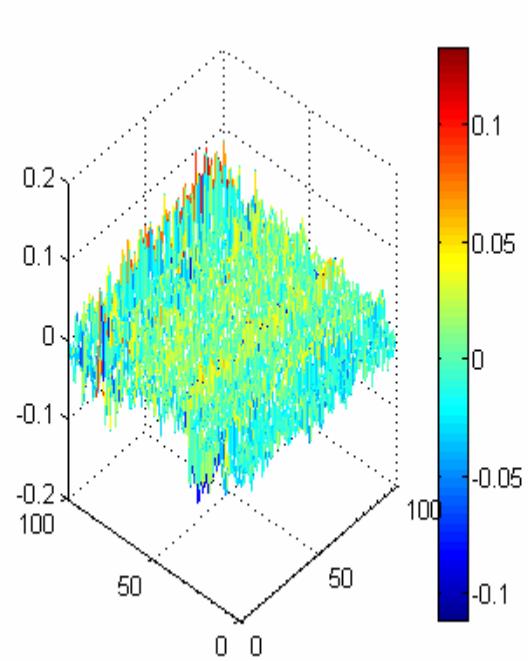
- Correlation output is shifted down by the same amount of the shifted face image, PSR remains SAME!



*M.Savvides and B.V.K. Vijaya Kumar, "Efficient Design of Advanced Correlation Filters for Robust Distortion-Tolerant Face Identification", IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS) 2003.



- Using SOMEONE ELSE'S Filter,....
- Perform cross-correlation on cropped-face shown on left.
- As expected very low PSR.

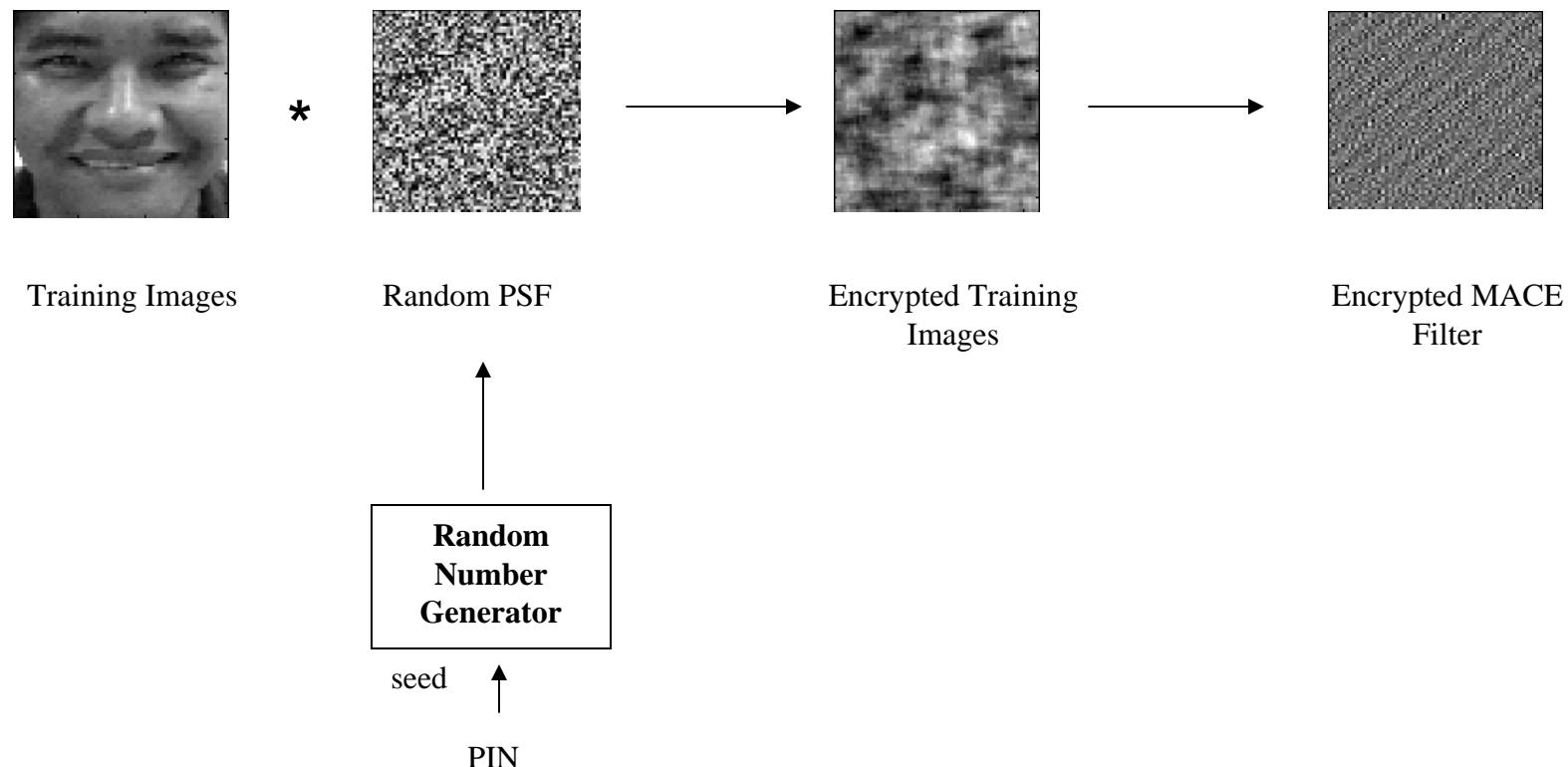


Cancellable Biometric Filters:-practical ways of deploying correlation filter biometrics

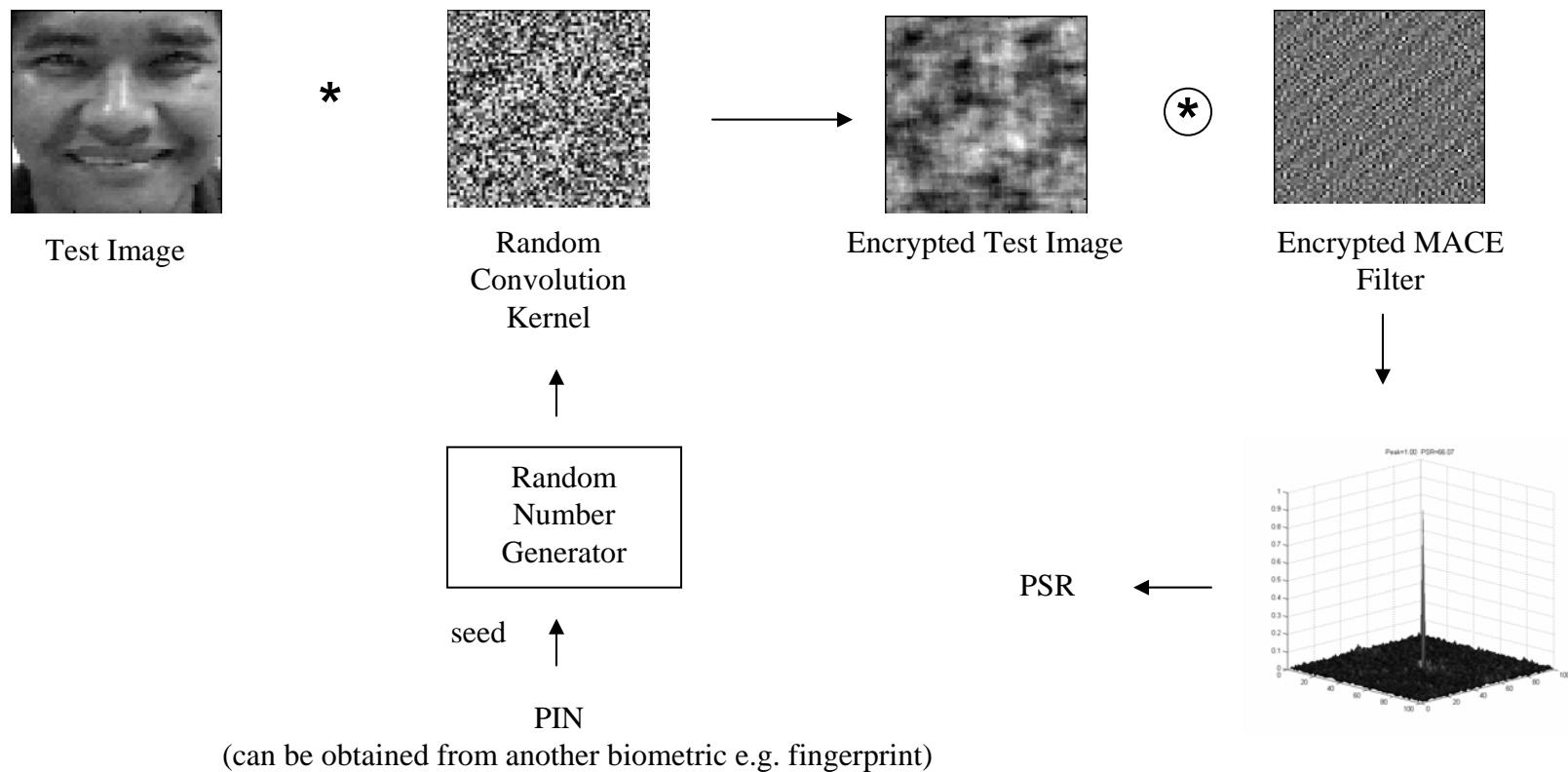
- A biometric filter (stored on a card) can be lost or stolen
 - ▶ Can we re-issue a different one (just as we re-issue a different credit card)?
 - ▶ There are only a limited set of biometric images per person (e.g., only one face)
 - ▶ We can use standard encryption methods to encrypt the biometrics and then decrypt them for use during the recognition stage, however there is a ‘window’ of opportunity where a hacker can obtain the decrypted biometric during the recognition stage.
 - ▶ We have figure out a way to encrypt them and ‘work’ or authenticate in the encrypted domain and NOT directly in the original biometric domain.

*M. Savvides, B.V.K. Vijaya Kumar and P.K. Khosla, "Authentication-Invariant Cancelable Biometric Filters for Illumination-Tolerant Face Verification", Defense & Security Symposium, special session on Biometric Technologies for Human Identification, 2004.

Enrollment Stage



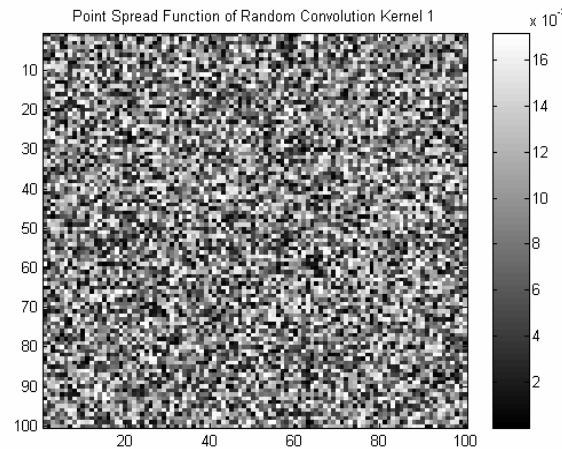
Authentication Stage



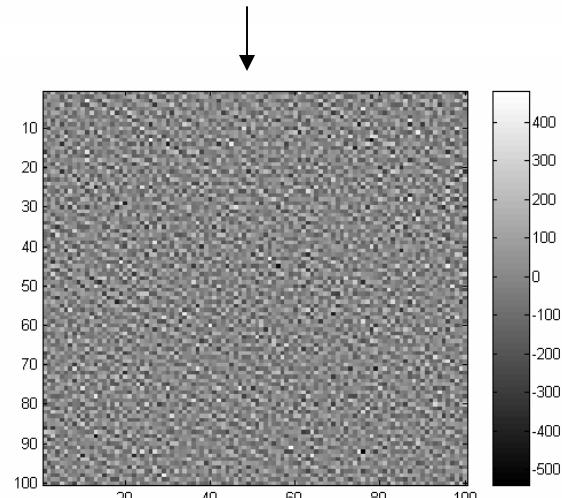
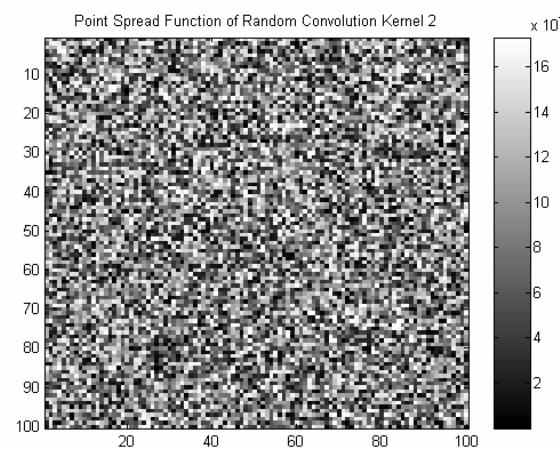
What about performance?

- We can show theoretically that performing this convolution pre-processing step does not affect resulting Peak-to-Sidelobe ratios.
- Thus, working in this encrypted domain does not change the verification performance

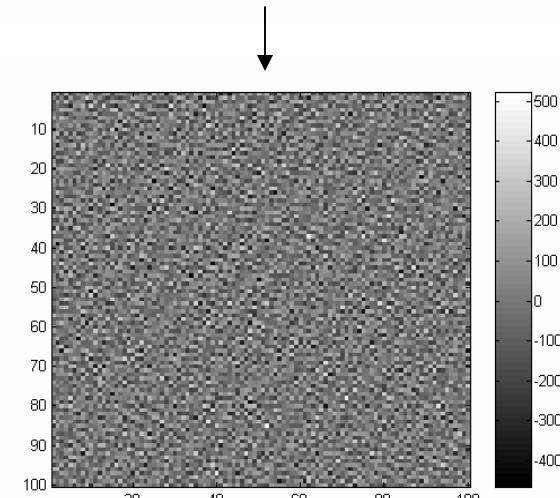
Random Convolution Kernel 1



Random Convolution Kernel 2



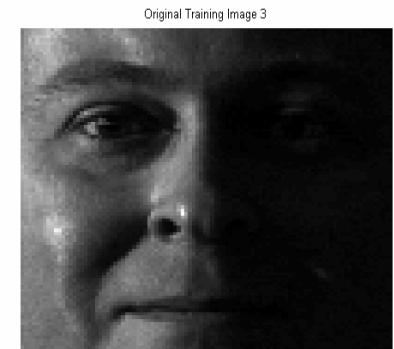
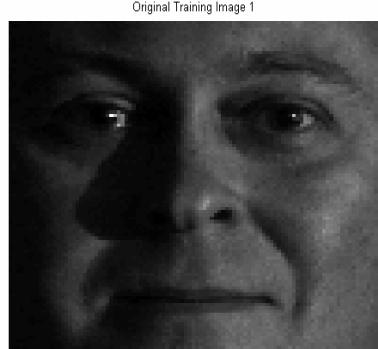
Encrypted MACE
Filter 1



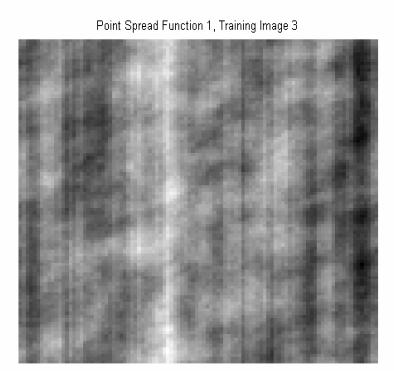
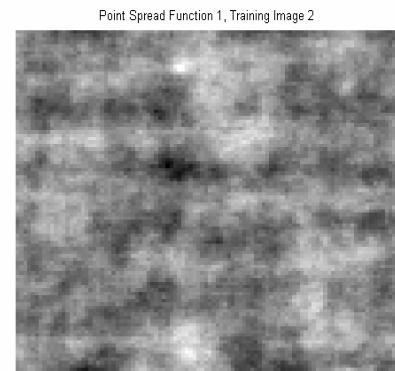
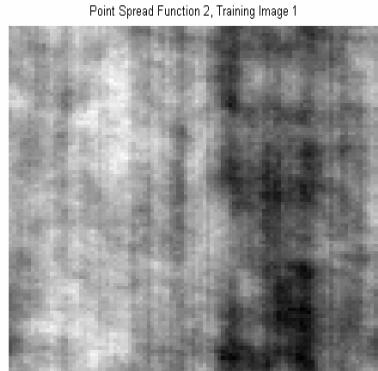
Encrypted MACE
Filter 2

60

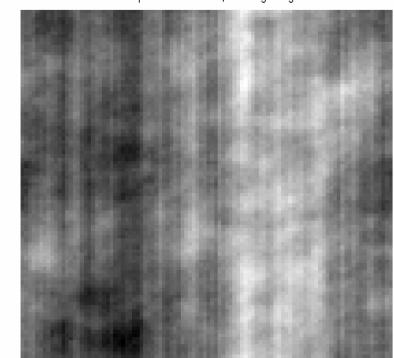
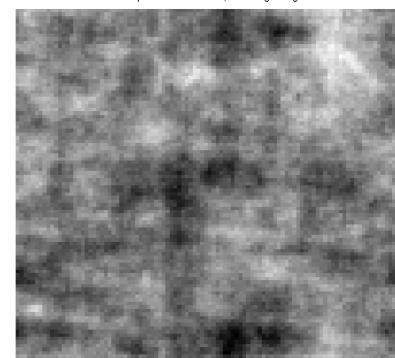
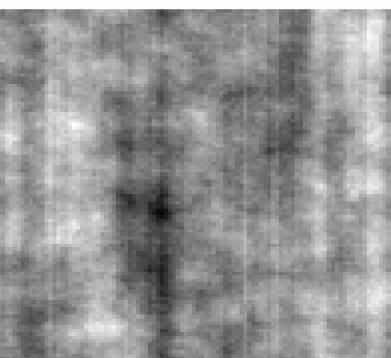
Original Training Images

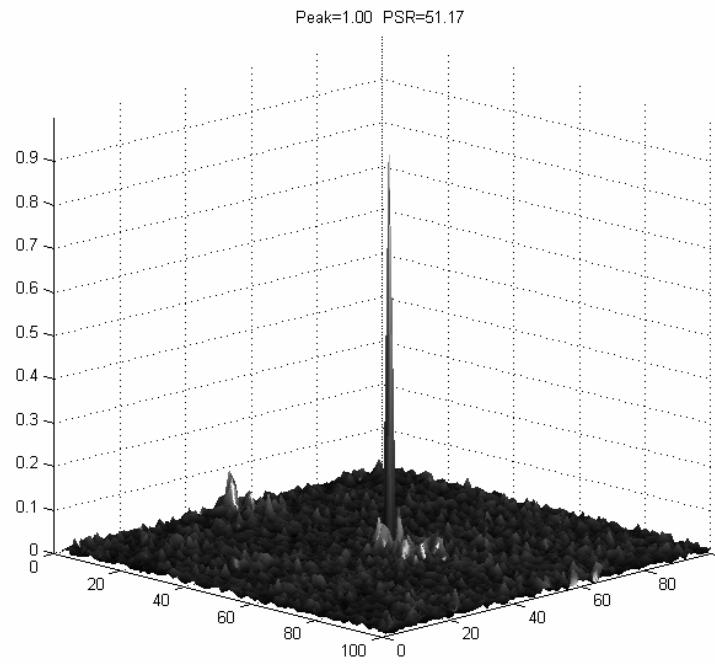


Convolved with Random Convolution Kernel 1

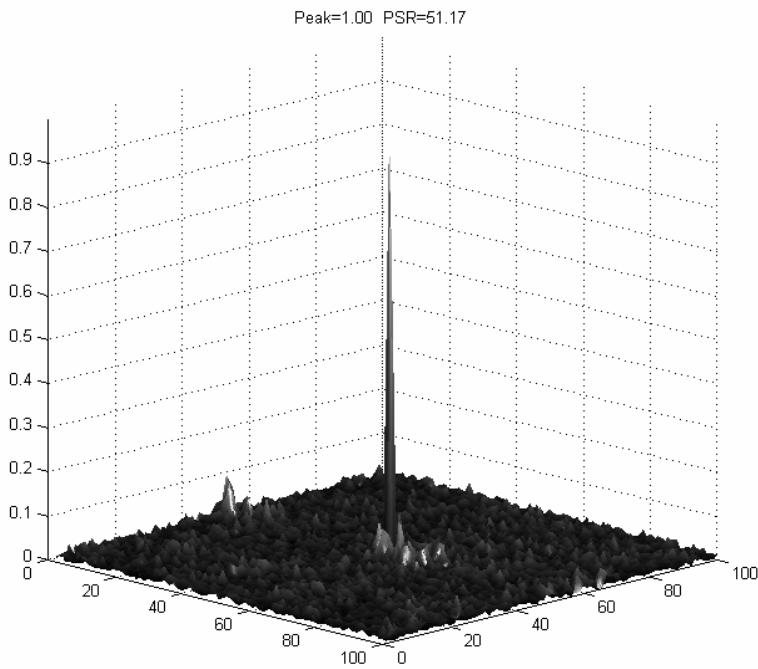


Convolved with Random Convolution Kernel 2





**Correlation Output from
Encrypted MACE Filter 1**



**Correlation Output from
Encrypted MACE Filter 2**

Prof. Marios Savvides

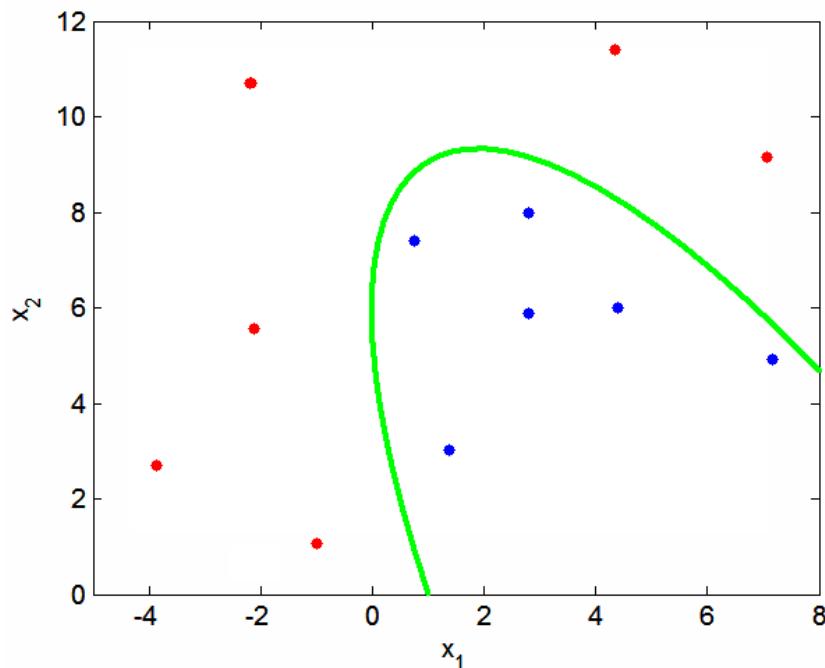
Pattern Recognition Theory

Lecture 12 : Perceptron Learning And Neural Networks

All graphics from Pattern Classification, Duda, Hart and Stork, Copyright © John Wiley and Sons, 2001

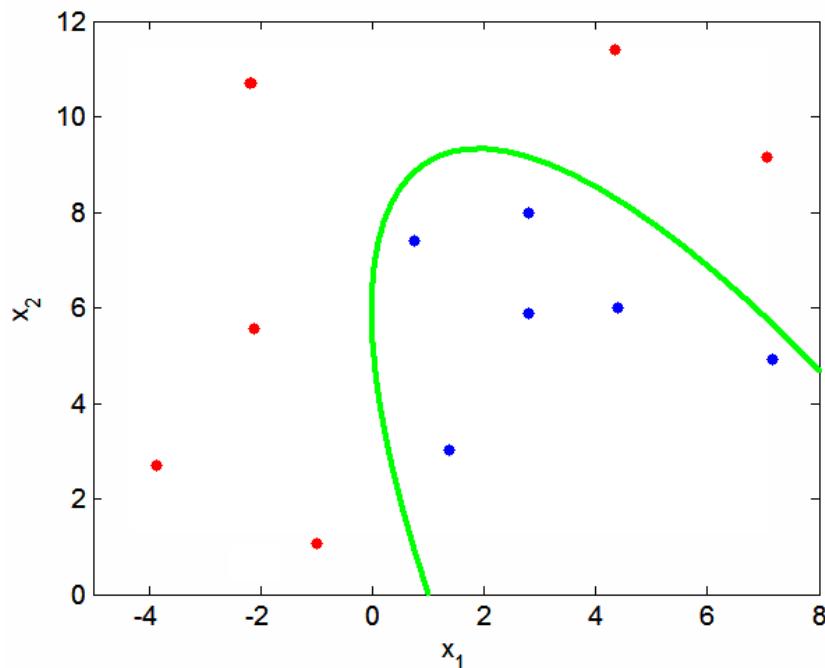
Discriminant Functions

- A discriminant function is a function of the feature pattern x that **leads to a classification rule**.
- We can use **any monotonic transformation** of the discriminant function and it will lead to the same decision rule
- In contrast to previous approaches, now the **form of the DF is specified**, and is not imposed by the underlying distribution
→ Discriminative approach (as opposed to Generative approach)



Discriminant Function Estimation

- Specify a **parametric form** of the decision boundary (for ex, linear, quadratic, etc..)
- Find the “**best**” decision boundary of the specified form using a set of training examples.
- This best decision boundary is achieved by **minimizing some criterion function**
 - For example, minimize the *training error*



Generative vs. Discriminative Methods

- **Generative Methods**

- Model class-conditional pdfs and prior probabilities.
- New data points can be **generated**.
- Examples : Bayesian Minimum Error , Gaussians, Mixture of Gaussians, HMMs...

- **Discriminative Methods**

- Directly estimate posterior probabilities.
 - No need to model underlying probability distributions.
- Use existing data points
- Examples : *Support Vector Machines. Nearest Neighbor. Neural Networks.*

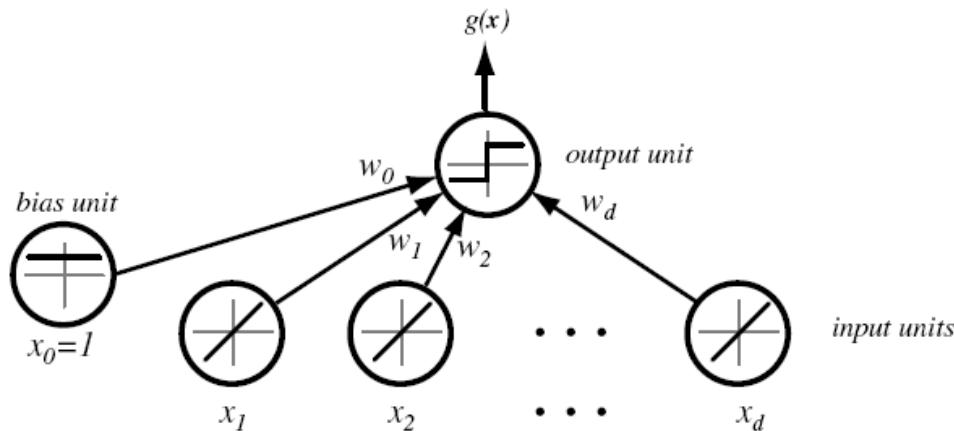
Linear Discriminant Functions

- A linear discriminant function is a linear combination of its components:

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = \sum_{i=1}^d w_i x_i + w_0$$

- \mathbf{w} is the weight vector
- w_0 is the bias (or threshold weight)

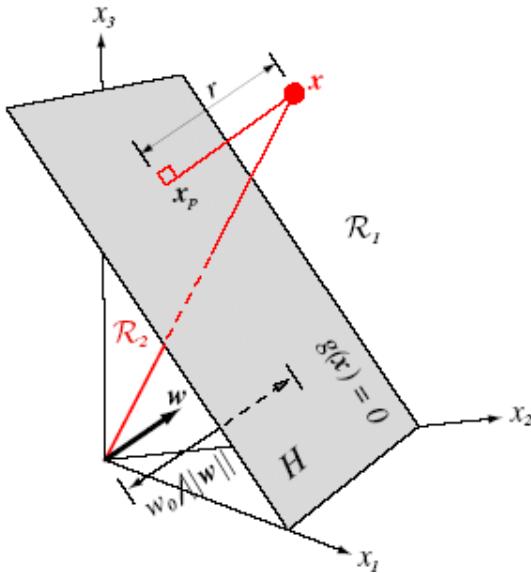
- Decide ω_1 if $g(\mathbf{x}) > 0$ and ω_2 if $g(\mathbf{x}) < 0$
 - If $g(\mathbf{x})=0$ then \mathbf{x} is on the decision boundary and can be assigned to either class
- If $g(\mathbf{x})$ is linear, the decision boundary is a **hyperplane**.



Decision Boundary

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = \sum_{i=1}^d w_i x_i + w_0$$

- The orientation of the hyperplane is determined by \mathbf{w} and its location by w_0
 - \mathbf{w} is the normal to the hyperplane
 - If $w_0=0$, the hyperplane passes through the origin

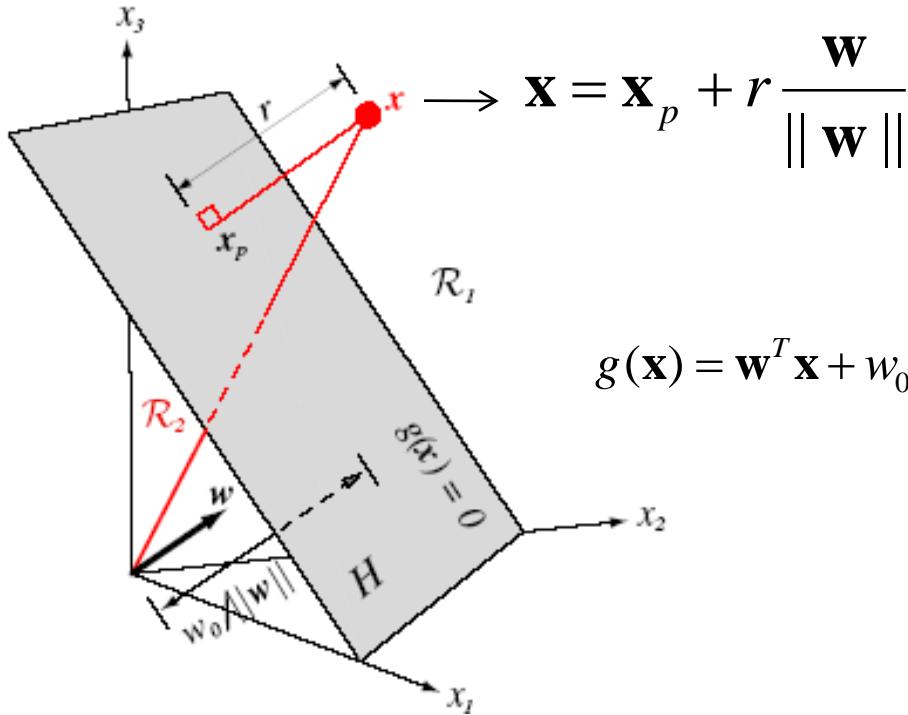


- If x_1 and x_2 are both on the decision surface, then

$$\mathbf{w}^T \mathbf{x}_1 + w_0 = \mathbf{w}^T \mathbf{x}_2 + w_0$$

$$\mathbf{w}^T (\mathbf{x}_1 - \mathbf{x}_2) = 0$$
- \mathbf{w} is normal to any vector lying in the hyperplane.

Decision Boundary



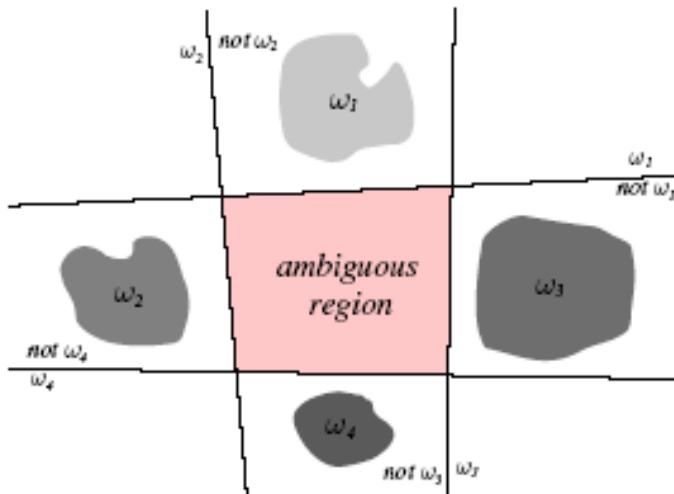
$$\begin{aligned}
 g(\mathbf{x}) &= \mathbf{w}^T \mathbf{x} + w_0 = \mathbf{w}^T (\mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|}) + w_0 = \mathbf{w}^T \mathbf{x}_p + r \frac{\mathbf{w}^T \mathbf{w}}{\|\mathbf{w}\|} + w_0 \\
 &= \underbrace{\left(\mathbf{w}^T \mathbf{x}_p + w_0 \right)}_{g(\mathbf{x}_p)=0} + r \|\mathbf{w}\| = r \|\mathbf{w}\|
 \end{aligned}$$

-This gives the distance of \mathbf{x} from H : $r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}$

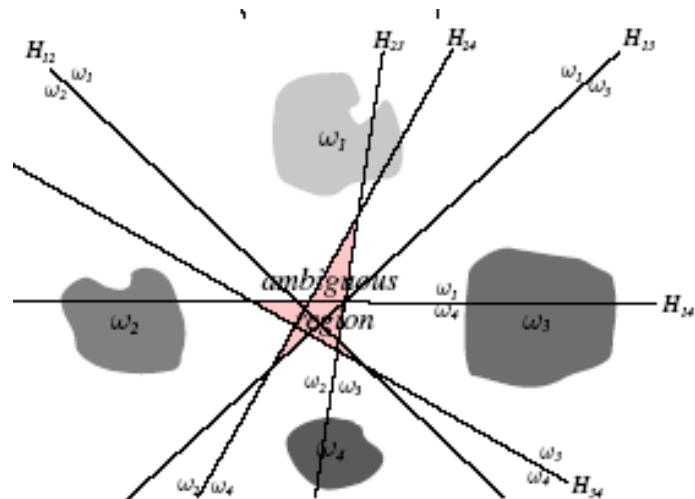
- w_0 determines the distance of the hyperplane from the origin: $\frac{w_0}{\|\mathbf{w}\|}$

LDF: Multi-Category Case

- There are several ways to devise multiclass classifiers using LDFs:
 - One against the rest (ω_i /not ω_i dichotomies)
 - A total of $c-1$ two-class problems
 - One against another (ω_i / ω_j dichotomies)
 - A total of $c(c-1)/2$ pairs of classes
- Both of these methods produce ambiguous regions



c-1 two class problems

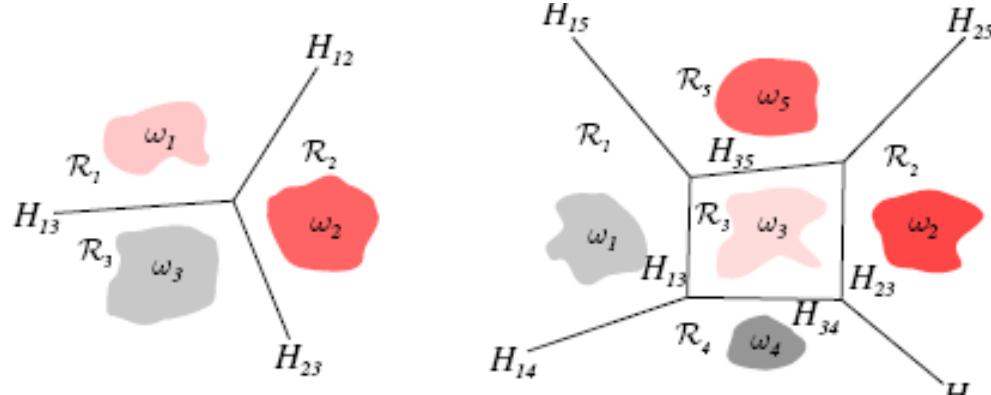


c(c-1)/2 two class problems

LDF: Multi-Category Case

- To avoid the problem of ambiguous regions:
 - Define c linear discriminant functions
 - Assign \mathbf{x} to ω_i if $g_i(\mathbf{x}) > g_j(\mathbf{x})$ for $i \neq j$
- The resulting classifier is called a **linear machine**:

**CONVEX &
SIMPLY
CONNECTED
DECISION
REGIONS**



- The boundary between two regions is a portion of the hyperplane given by:

$$g_i(\mathbf{x}) = g_j(\mathbf{x}) \quad or$$

$$(\mathbf{w}_i - \mathbf{w}_j)^t \mathbf{x} + (w_{i0} - w_{j0}) = 0$$

Higher Order DFs

- High Order DFs can produce more complicated decision boundaries.
 - Quadratic discriminant: obtained by adding terms corresponding to product of pairs of components of \mathbf{x}

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^d w_i x_i + \sum_{i=1}^d \sum_{j=1}^d x_i x_j w_{ij}$$

- Polynomial discriminant: obtained by adding terms such as $x_i x_j x_k w_{ijk}$
- The Generalized Linear Discriminant Function:

$$g(\mathbf{x}) = \sum_{i=1}^{\hat{d}} a_i y_i(x) = \mathbf{a}^T \mathbf{y}$$

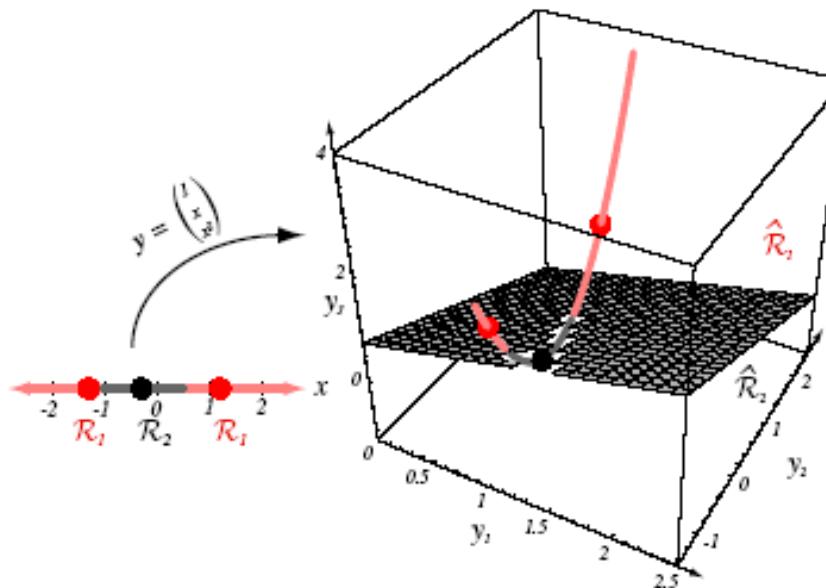
- \mathbf{A} is a d^A dimensional weight vector
- The functions $y_i(x)$ map points from the d -dimensional space to the d^A dimensional space. (usually $d^A \gg d$)
- The resulting DF is **not linear in \mathbf{x} but linear in \mathbf{y}**
- The generalized discriminant separates points in the transformed space

Generalized DF Example:

$$g(x) = -1 + x + 2x^2$$

$$\mathbf{a} = [-1 \quad 1 \quad 2]^T, \mathbf{y} = [1 \quad x \quad x^2]^T$$

- Maps a line in **x**-space to a parabola in **y**-space
- The plane $\mathbf{a}^T \mathbf{y} = 0$ divides the **y**-space in two decision regions
- The corresponding decision regions in the original **x**-space are not simply connected.



Alternative Notation: Augmented Feature/Weight Vectors

- Exploit the convenience of writing $g(\mathbf{x})$ in the homogeneous form

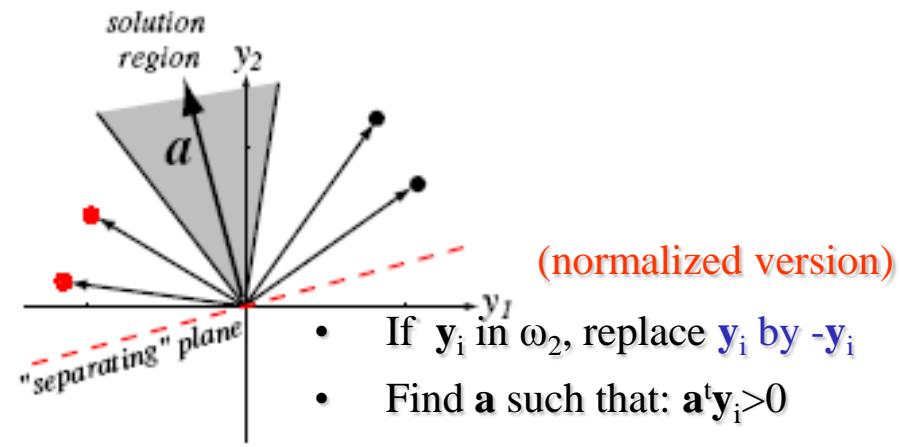
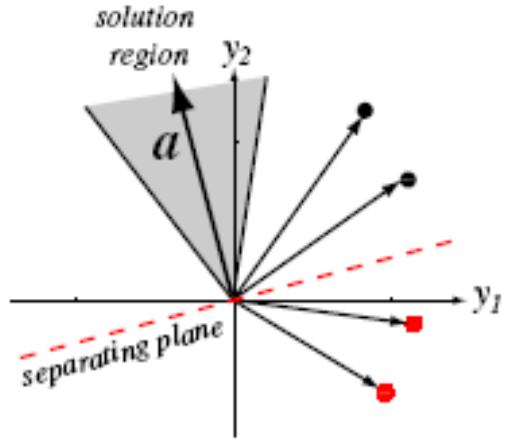
$$g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0 = \sum_{i=1}^d w_i x_i + x_0 w_0 = \sum_{i=0}^d w_i x_i = \mathbf{a}^t \mathbf{y}$$

$$\mathbf{a} = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_d \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ x_1 \\ \dots \\ x_d \end{bmatrix} \quad x_0 = 1$$

- Decision hyperplane passes through origin in \mathbf{y} -space.

Two-Class, Linearly Separable

- Given a linear DF $g(\mathbf{x}) = \mathbf{a}^T \mathbf{y}$, the goal is to learn the weights using a set of n labeled samples.
- Classification rule:
 - If $\mathbf{a}^T \mathbf{y} > 0$, assign y_i to ω_1 else if $\mathbf{a}^T \mathbf{y} < 0$, assign y_i to ω_2
- Every training sample places a constraint on the weight vector \mathbf{a}
- Given n samples, the solution must lie in the intersection of n half-spaces.
- Solution vector is usually not unique! Impose constraints to enforce uniqueness.



Perceptron Learning 1

- Now consider the problem of learning a binary classification problem with a linear discriminant function.

- Remember that our objective is to find a vector \mathbf{a} such that

$$g(\mathbf{x}) = \mathbf{a}^T \mathbf{y} = \begin{cases} > 0 & \mathbf{x} \in \omega_1 \\ < 0 & \mathbf{x} \in \omega_2 \end{cases}$$

- To simplify the derivation, we will normalize the training set by replacing all samples from class ω_2 by their negative:

$$\mathbf{y} \leftarrow [-\mathbf{y}] \quad \forall \mathbf{y} \in \omega_2$$

- Now we can ignore the class labels and look for a weight vector such that

$$g(\mathbf{x}) = \mathbf{a}^T \mathbf{y} > 0$$

- To find the solution we must define the criterion function $J(\mathbf{a})$

- One choice is known as the **Perceptron criterion function** $J_P(\mathbf{a}) = \sum_{\mathbf{y} \in Y_M} -\mathbf{a}^T \mathbf{y}$
- Y_M is the set of misclassified samples by \mathbf{a}
- Note that $J_p(\mathbf{a})$ is **non-negative since $\mathbf{a}^T \mathbf{y} < 0$ for the misclassified**

Perceptron Learning 2

- To find the minimum of this criterion function we use gradient descent.

$$J_P(\mathbf{a}) = \sum_{\mathbf{y} \in Y_M} -\mathbf{a}^T \mathbf{y}$$

$$\nabla_{\mathbf{a}} J_P(\mathbf{a}) = \sum_{\mathbf{y} \in Y_M} -\mathbf{y}$$

Y_M is the set of misclassified samples by \mathbf{a}

- The gradient descent update rule becomes:

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta \sum_{\mathbf{y} \in Y_M(k)} \mathbf{y}$$

Perceptron rule

- This is known as the Perceptron batch update rule
- If the classes are linearly separable, the Perceptron rule is guaranteed to converge to a valid solution.
- However, if the 2 classes are not linearly separable, the Perceptron rule will not converge.
 - Since no weight vector \mathbf{a} can correctly classify every sample, the corrections in the Perceptron rule will never diminish.

Iterative Optimization

- To find the solution to the set of inequalities $\mathbf{a}^t \mathbf{y}_i > 0$
- Define a criterion function $J(\mathbf{a})$ that is minimized if \mathbf{a} is a solution vector
- Gradient descent is the general method for function minimization
 - Recall that the minimum of a function $J(x)$ is defined by the zeros of the gradient.
$$\hat{\mathbf{x}} = \arg \min [J(\mathbf{x})] \rightarrow \nabla_{\mathbf{x}} J(\mathbf{x}) = 0$$
 - Only in very specific cases that this minimization function has a closed form solution.
 - In some other cases a closed form solution may exist but can be numerically unstable or too expensive/impractical.
- Gradient descent finds the minimum in an iterative way by moving in the direction of steepest descent.

Gradient Descent

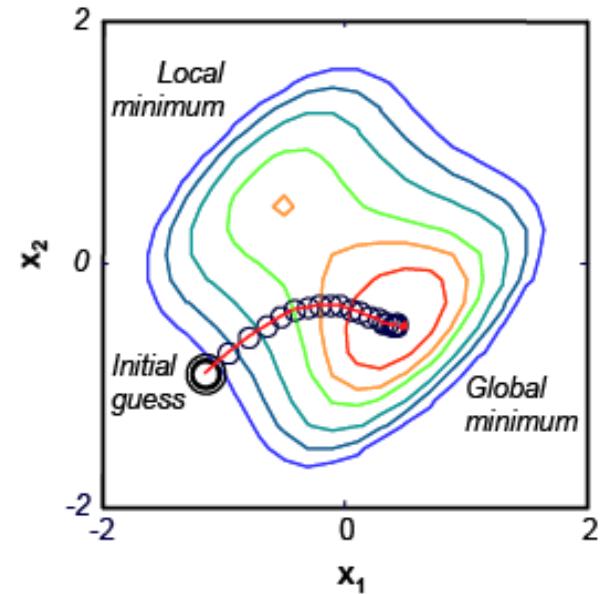
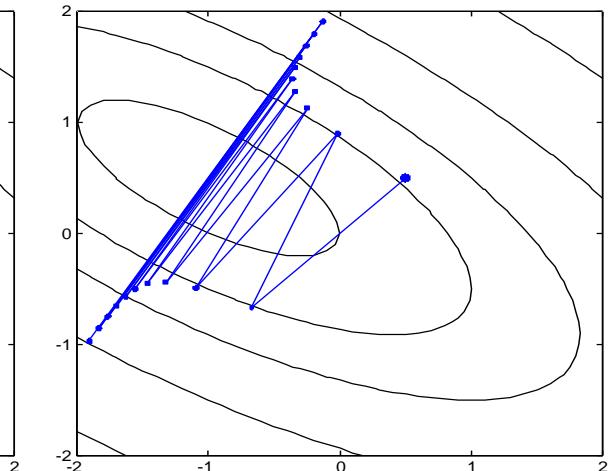
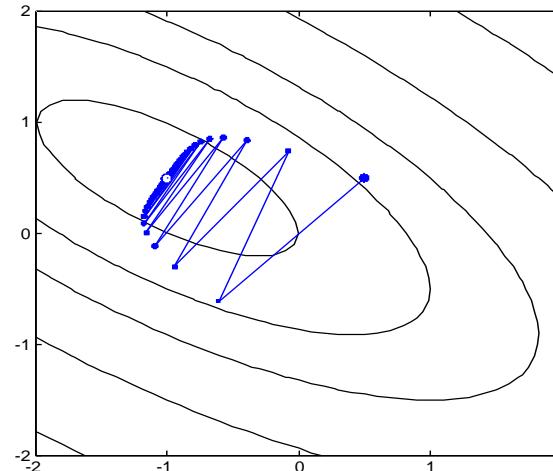
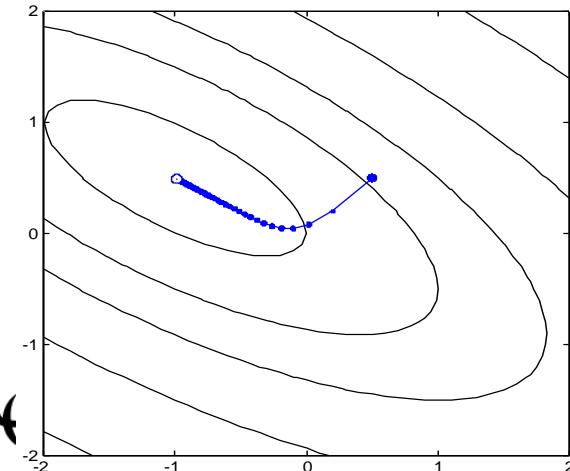
- Gradient descent finds the minimum in an iterative way by moving in the direction of steepest descent.

- Start with an arbitrary initial solution (guess) $\mathbf{x}(0)$
- Compute the gradient $\nabla_{\mathbf{x}} J(\mathbf{x}(k))$
- Move in the direction of steepest descent

$$\mathbf{x}(k+1) = \mathbf{x}(k) - \eta \nabla_{\mathbf{x}} J(\mathbf{x}(k))$$

- Repeat until convergence.

- Effect of the learning rate on convergence:



Linearly Non-Separable

- **Linearly separable**
 - The Perceptron procedure
- **Linearly non-separable**
 - The Minimum Squared Error procedure
 - The Least Mean Squared Error procedure
 - The Ho-Kashyap procedure

Minimum Squared Error Solution

- The traditional Minimum Squared Error (MSE) criterion provides an alternative to the Perceptron rule.
 - The Perceptron rule seeks a weight vector \mathbf{a} that satisfies $\mathbf{a}^T \mathbf{y}_i > 0$
 - The perceptron rule only considers misclassified samples, since these are the only ones that violate the inequality
 - Instead, the MSE criterion looks for a solution to the equality $\mathbf{a}^T \mathbf{y}_i = b_i$ where b are some pre-specified target values (for example class labels).
 - The MSE solution uses ALL the samples in the training set.
- The system of equations solved by MSE is

$$\begin{bmatrix} y_0^1 & y_1^1 & \dots & y_d^1 \\ y_0^2 & y_1^2 & & y_d^2 \\ \vdots & \ddots & \vdots \\ y_0^n & y_1^n & \dots & y_d^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix} = \begin{bmatrix} b^1 \\ b^2 \\ \vdots \\ b^n \end{bmatrix} \Leftrightarrow \mathbf{Y}\mathbf{a} = \mathbf{b}$$

- where \mathbf{a} is the weight vector, each row in \mathbf{Y} is a training sample, and each row in \mathbf{b} is the corresponding class label.

MSE Solution: Pseudo Inverse

- An exact solution to $\mathbf{Y}\mathbf{a} = \mathbf{b}$ can be found sometimes.
 - If the number of equations (n) is equal to the number of unknowns ($d+1$), the exact solution is defined by
- In practice, \mathbf{Y} will be singular and the inverse does not exist.
 - \mathbf{Y} will have more rows (samples) than columns (dimensions) which yields an over-determined system.
- The solution in this case is to find a weight vector that minimizes some function of the error between $\mathbf{a}\mathbf{Y}$ and the desired output \mathbf{b} .

$$J_{MSE}(\mathbf{a}) = \sum_{i=1}^n (\mathbf{a}^T \mathbf{y}^i - \mathbf{b}^i)^2 = \|\mathbf{Y}\mathbf{a} - \mathbf{b}\|^2$$

- The gradient of J is given by $\nabla_{\mathbf{a}} J_{MSE}(\mathbf{a}) = 2\mathbf{Y}^T(\mathbf{Y}\mathbf{a} - \mathbf{b}) = 0$
- If $\mathbf{Y}^T\mathbf{Y}$ is nonsingular, the MSE solution becomes:

$$\mathbf{a} = (\mathbf{Y}^T \mathbf{Y})^{-1} \mathbf{Y}^T \mathbf{b} = \mathbf{Y}^\dagger \mathbf{b}$$

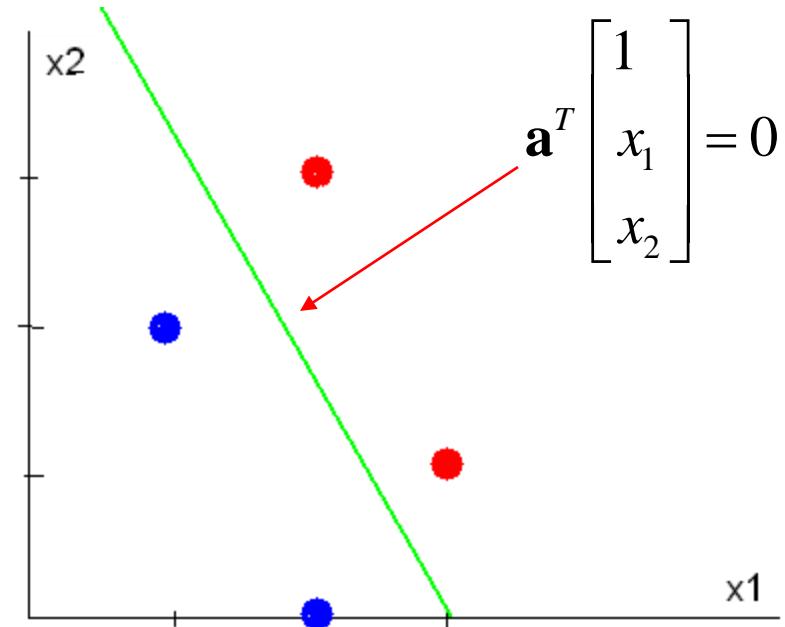
Pseudoinverse solution

Known as the normal equations

MSE Numerical Example

- Consider the following samples:
 - Class 1: [1, 2] and [2, 0]
 - Class 2: [3, 1], and [2, 3]
- Sample matrix ($d = 1+2$, $n = 4$)

$$\mathbf{Y} = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 0 \\ -1 & -3 & -1 \\ -1 & -2 & -3 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{y} \leftarrow [-\mathbf{y}] \quad \forall \mathbf{y} \in \omega_2$$



$$\mathbf{Y}^\dagger = (\mathbf{Y}^T \mathbf{Y})^{-1} \mathbf{Y}^T = \begin{bmatrix} 5/4 & 13/12 & 3/4 & 7/12 \\ -1/2 & -1/6 & -1/2 & -1/6 \\ 0 & -1/3 & 0 & -1/3 \end{bmatrix} \quad \mathbf{a} = \mathbf{Y}^\dagger \mathbf{b} = \begin{bmatrix} 11/3 \\ -4/3 \\ -2/3 \end{bmatrix}$$

Perceptron vs MSE Example

- Consider the following dataset:

 - $X_1 = \{(1,6), (7,2), (8,9), (9,9)\}$
 - $X_2 = \{(2,1), (2,2), (2,4), (7,1)\}$

- The Perceptron learning approach:

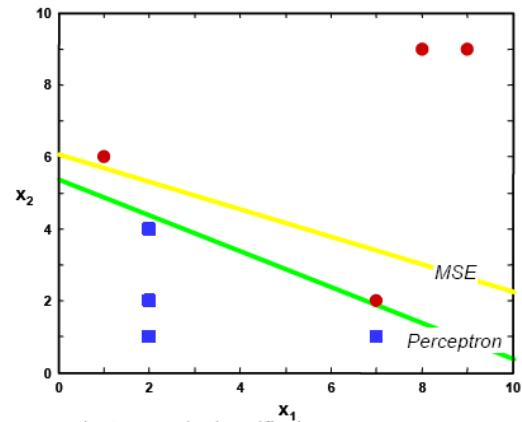
 - Assume $\eta=0.1$ and $a(0)=[0.1, 0.1, 0.1]$
 - Using an online update rule:

 - Normalize the data
 - Iterate through all the examples and update $a(k)$ on the ones that were misclassified

 - $Y(1) \rightarrow [1 1 6] * [0.1 0.1 0.1]^t > 0 \rightarrow$ no update
 - $Y(2) \rightarrow [1 7 2] * [0.1 0.1 0.1]^t > 0 \rightarrow$ no update
 - ...
 - $Y(5) \rightarrow [-1 -2 -1] * [0.1 0.1 0.1]^t < 0 \rightarrow$ update $a(1) = [0.1 0.1 0.1] + \eta[-1 2 -1] = [0 -0.1 0]$
 - $Y(6) \rightarrow [-1 -2 -2] * [0 -0.1 0]^t > 0 \rightarrow$ no update
 - ...
 - $Y(1) \rightarrow [1 1 6] * [0 -0.1 0]^t < 0 \rightarrow$ update $a(2) = [0 0 -0.1 0] + \eta[1 1 6] = [0.1 0 0.6]$

 - After 175 iterations, the resulting $a=[-3.5 0.3 0.7]$

$$Y = \begin{bmatrix} 1 & 1 & 6 \\ 1 & 7 & 2 \\ 1 & 8 & 9 \\ 1 & 9 & 9 \\ -1 & -2 & -1 \\ -1 & -2 & -2 \\ -1 & -2 & -4 \\ -1 & -7 & -1 \end{bmatrix}$$



- MSE approach

 - The MSE solution found is $a=(Y^t Y)^{-1} Y^t b = [-1.18 0.07 0.19]$

Least Mean Squares Solution

- The criterion function of the MSE $J_{MSE}(\mathbf{a}) = \sum_{i=1}^n (\mathbf{a}^T \mathbf{y}^i - b^i)^2 = \|\mathbf{Y}\mathbf{a} - \mathbf{b}\|^2$ can also be found using a gradient descent procedure
 - This avoids inverting large matrices
 - It is especially useful when $\mathbf{Y}^T \mathbf{Y}$ is singular
- The update rule now becomes

$$\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k) \mathbf{Y}^T (\mathbf{b} - \mathbf{Y}\mathbf{a}(k))$$

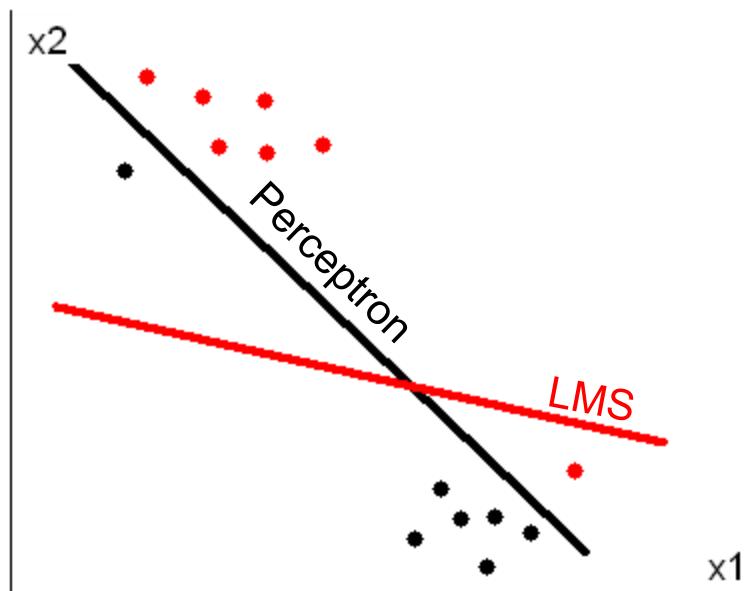
LMS rule – Batch update
 - It can be shown that if $n(k)=n(1)/k$ where $n(1)$ is any positive constant, this rule generates a sequences of weight vectors that converge to a solution to $\mathbf{Y}^T(\mathbf{Y}\mathbf{a}-\mathbf{b})=0$
- Another advantage of this method is that the storage requirement can be further reduced by considering each sample sequentially

$$\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k) (b^i - y^i \mathbf{a}(k)) y^i$$

LMS rule – Sequential update

Perceptron Rule vs MSE Rule

- Perceptron rule:
 - The perceptron rule always finds a solution if the classes are linearly separable but does not converge in case of non-separability.
- MSE/LMS rule
 - The MSE/LMS solution guarantees convergence, but it may not find a separating hyperplane if the classes are linearly separable.
 - It just tries to minimizes the sum of the square of the distances of the samples to the hyperplane, as opposed to finding the actual hyperplane.



The Ho-Kashyap Procedure

- The main issue with the MSE criterion is the lack of guarantees that a separating hyperplane will be found in the linearly separable case
 - The MSE rule just minimizes $J_{mse} = \|\mathbf{Y}\mathbf{a} - \mathbf{b}\|^2$. Whether it will find a separating hyperplane or not depends on how the outputs \mathbf{b} are selected.
- If the two classes are linearly separable, there must be vectors \mathbf{a}^* and \mathbf{b}^* such that $\mathbf{Y}\mathbf{a} = \mathbf{b} > 0$
 - If \mathbf{b}^* were known, one could simply use the MSE solution to compute the separating hyperplane.
 - However, since \mathbf{b} is also unknown, one must then solve for BOTH \mathbf{a} & \mathbf{b} .
 - We need to find the good labeling (\mathbf{b}) .
- This gives an alternative training algorithm for LDFs known as the Ho-Kashyap procedure
 - Find the target values \mathbf{b} through gradient descent
 - Compute the weight vector \mathbf{a} from the MSE solution

The Ho-Kashyap Procedure

- The gradients are defined by:

$$J_{MSE}(\mathbf{a}, \mathbf{b}) = \|\mathbf{Y}\mathbf{a} - \mathbf{b}\|^2$$

$$\nabla_{\mathbf{a}} J_{MSE}(\mathbf{a}, \mathbf{b}) = -2\mathbf{Y}^t(\mathbf{Y}\mathbf{a} - \mathbf{b}) \quad \nabla_{\mathbf{b}} J_{MSE}(\mathbf{a}, \mathbf{b}) = -2(\mathbf{Y}\mathbf{a} - \mathbf{b})$$

- For any value of \mathbf{b} , we can always use the pseudoinverse solution

$$\mathbf{a} = \mathbf{Y}^\dagger \mathbf{b}$$

- Constraints on \mathbf{b} :

- $\mathbf{b} > 0$

- We don't want $\mathbf{b}(k) \rightarrow 0$. So start with a $\mathbf{b} > 0$ and refuse to reduce any of its components.

$$\mathbf{b}(k+1) = \mathbf{b}(k) - \eta(k) \nabla_{\mathbf{b}} J_{MSE} = \mathbf{b}(k) - 2\eta(k)(\mathbf{Y}\mathbf{a} - \mathbf{b})$$

- The final iterative procedure is given by:

$$\mathbf{b}(k+1) = \mathbf{b}(k) - 2\eta(k)\mathbf{e}^+(k)$$

$$\mathbf{a}(k+1) = \mathbf{Y}^\dagger \mathbf{b}(k+1)$$

**Ho-Kashyap
procedure**

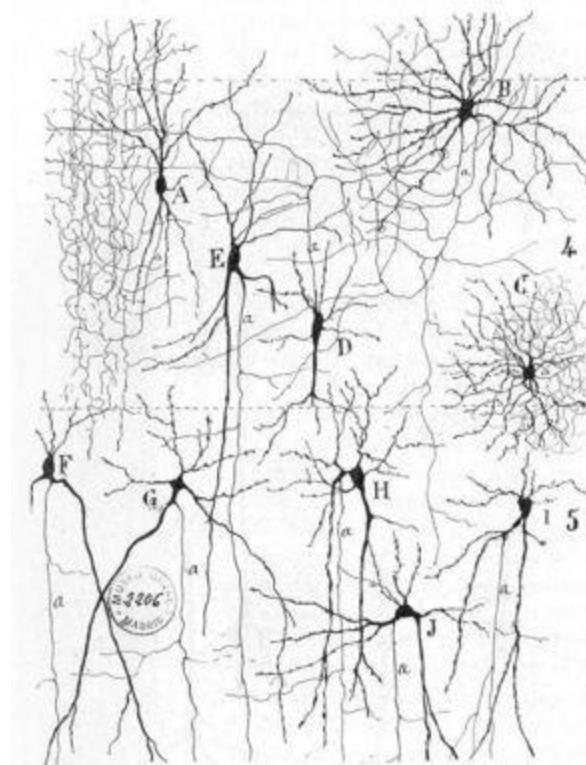
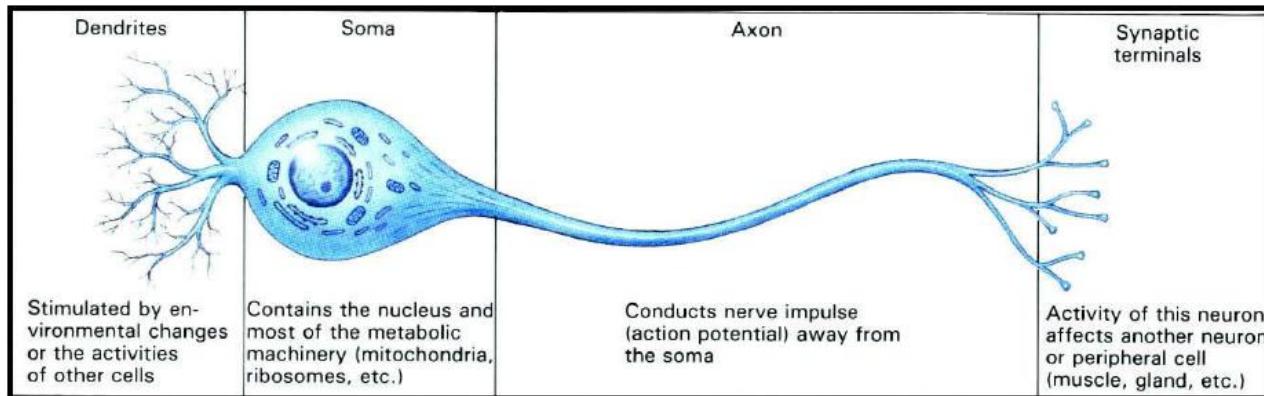
Error vector

$$\longrightarrow \mathbf{e}(k) = \mathbf{Y}\mathbf{a}(k) - \mathbf{b}(k)$$

$$\longrightarrow \mathbf{e}^+(k) = \frac{1}{2}(\mathbf{e}(k) + |\mathbf{e}(k)|)$$

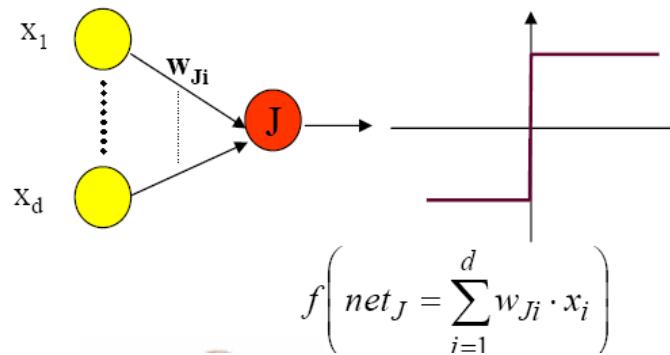
Artificial Neural Networks

- An **artificial neural network** (ANN), often just called a "neural network" (NN), is a mathematical model or computational model, based on biological neural networks.
- It can be looked upon as consisting of an interconnected group of artificial neurons
- During the learning phase, this structure undergoes changes. The final structure is used for the final classification problem



History Overview 1

- McCulloch & Pitts, 1943
- Hebb, 1949
- Rosenblatt, 1958
 - Introduced the Perceptron
 - A single neuron with adjustable synaptic weights and a threshold activation function
 - Also developed an error-correction rule to adapt the weights (Perceptron learning rule)
 - Proved that if 2 classes are linearly separable, the algorithm would converge (Perceptron convergence theorem).

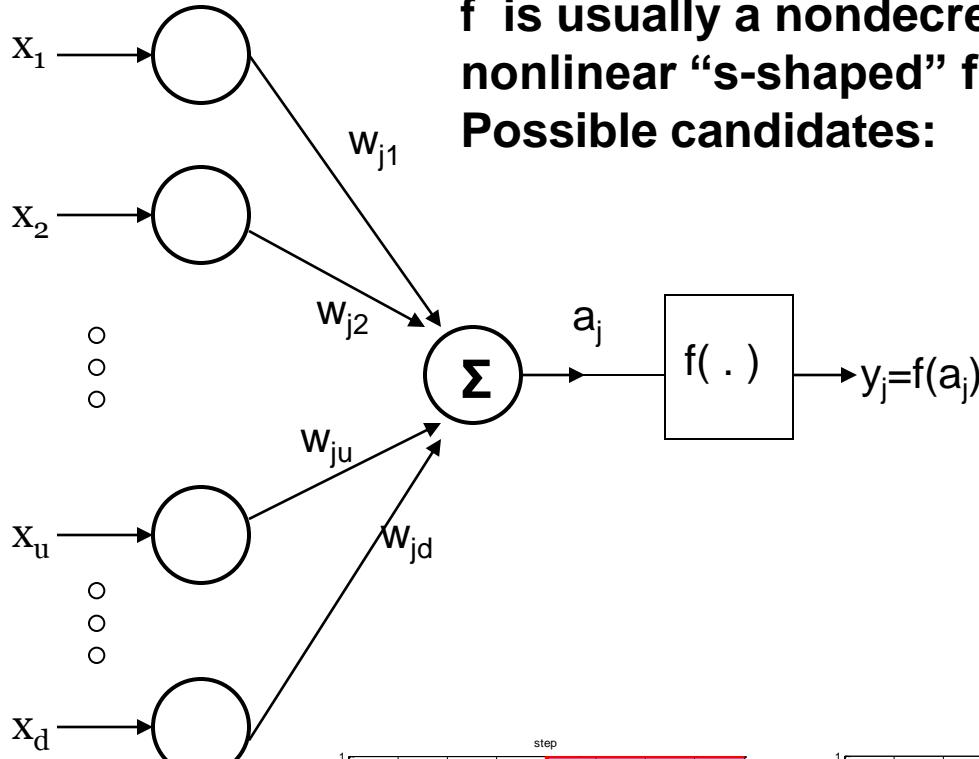


- Widrow & Hoff, 1960
 - Introduced the LMS algorithm
 - Hoff is also credited with the invention of the microprocessor

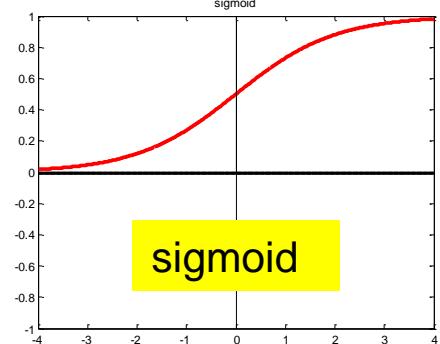
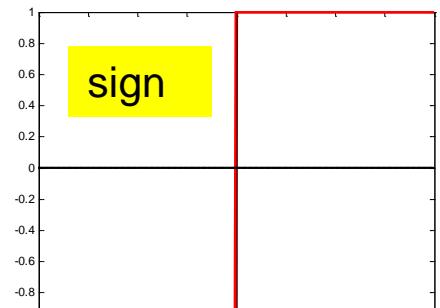
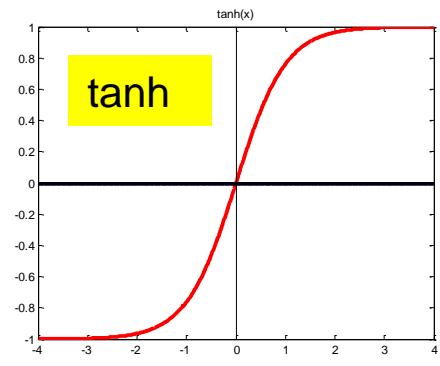
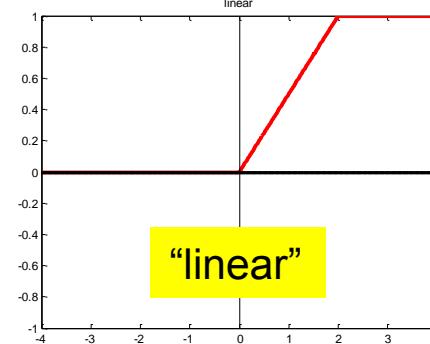
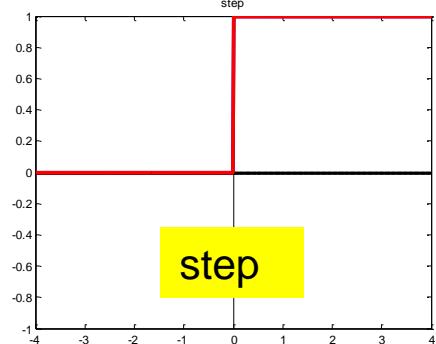
History Overview 2

- Minsky & Papert, 1969
 - Pioneers of AI
 - Proved the limitations of the perceptron:
 - can't solve the XOR problem
- Rumelhart, Hinton & Williams, 1986
 - Developed a simple algorithm for training multilayer networks, learning nonlinearly separable decision boundaries through backward propagation of errors, a generalization of the LMS algorithm.
 - Overcame the limitations of the Perceptron.
 - Original inventor of the backpropagation algorithm was Paul Werbos (Ph.D. thesis Harvard 1974)

Main Perceptron Unit

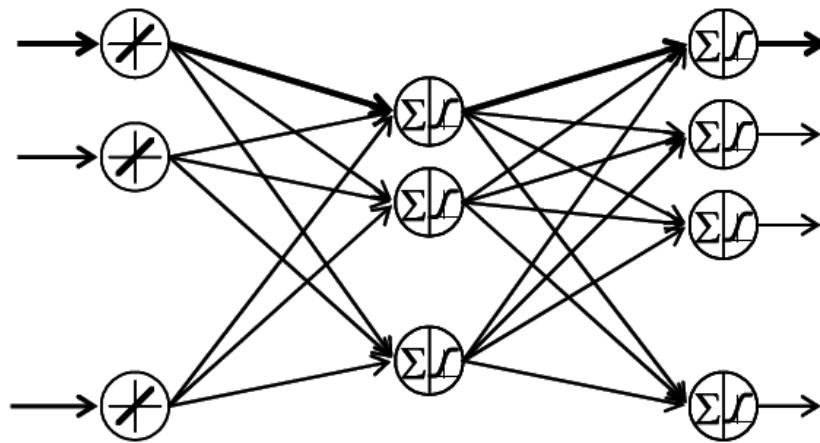


f is usually a nondecreasing nonlinear “s-shaped” function.
Possible candidates:



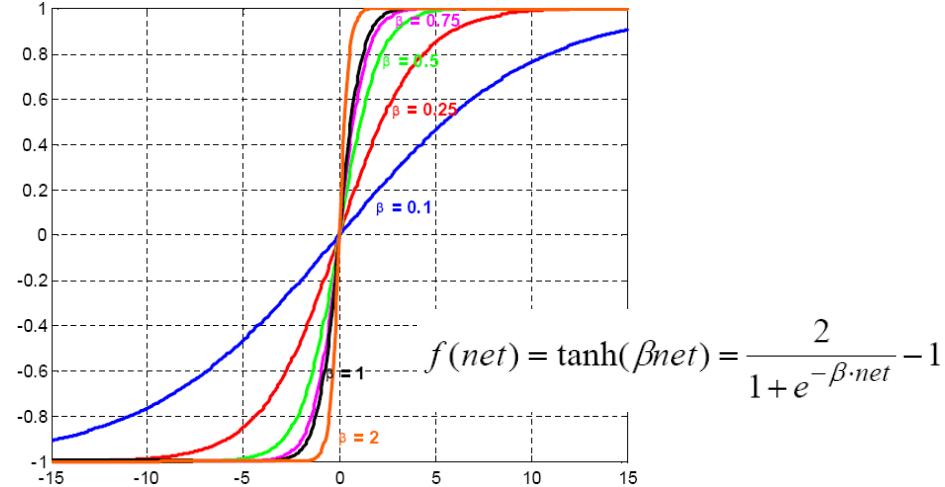
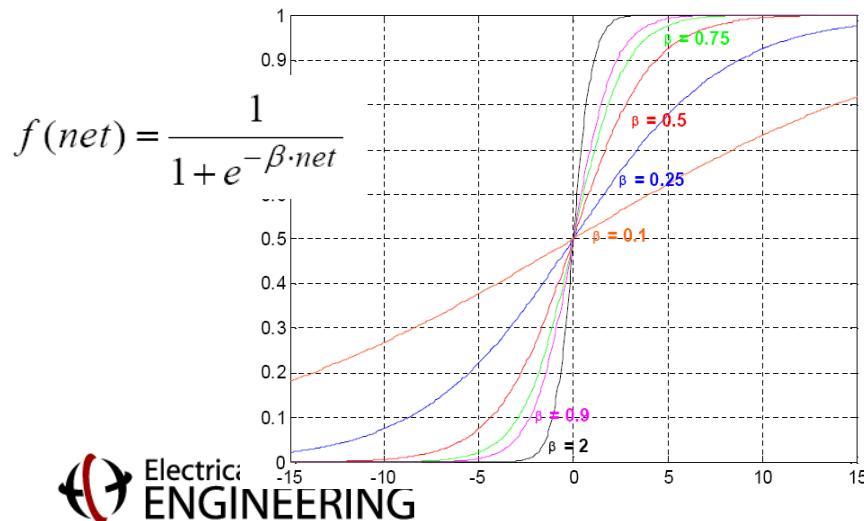
Multilayer Perceptrons

- MLPs are feed-forward networks of simple processing units with at least ONE hidden layer.
- What truly separates an MLP from a regular simple Perceptron is the non-linear threshold function, also known as the activation function.
 - If a linear thresholding function is used, the MLP can be replaced with a series of simple Perceptrons which can only solve linearly separable problems.

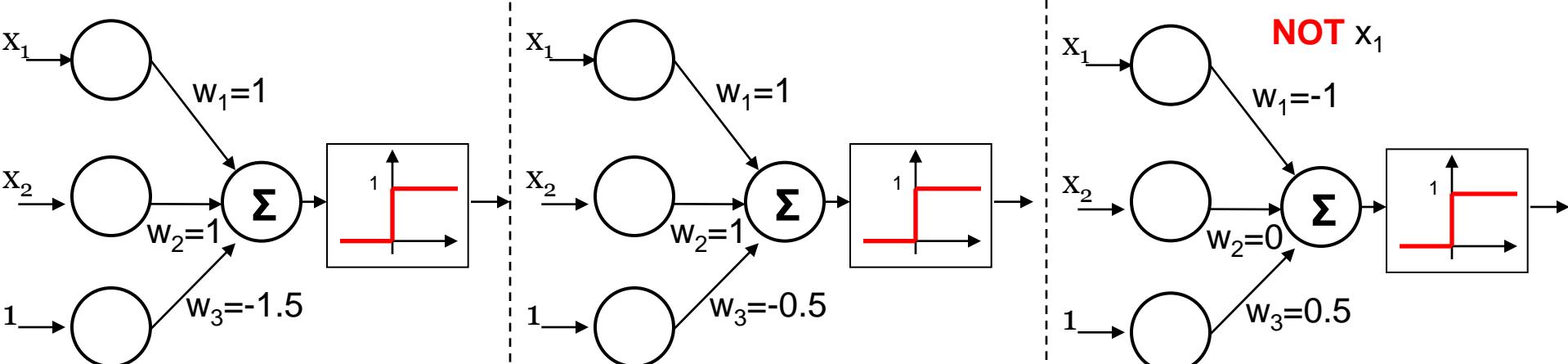
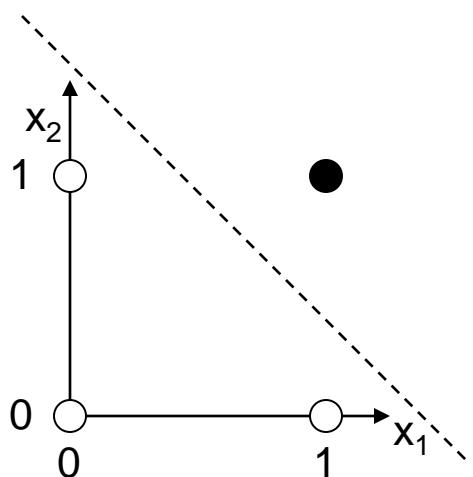
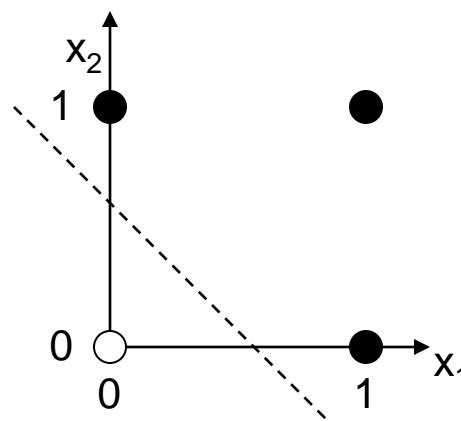
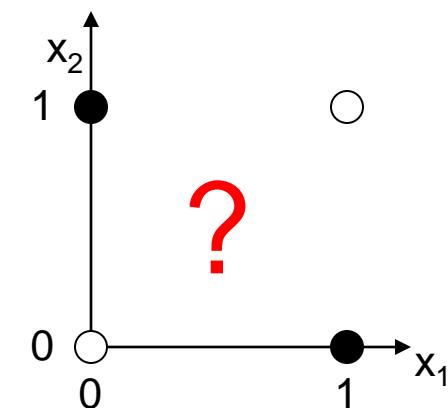


Activation Functions

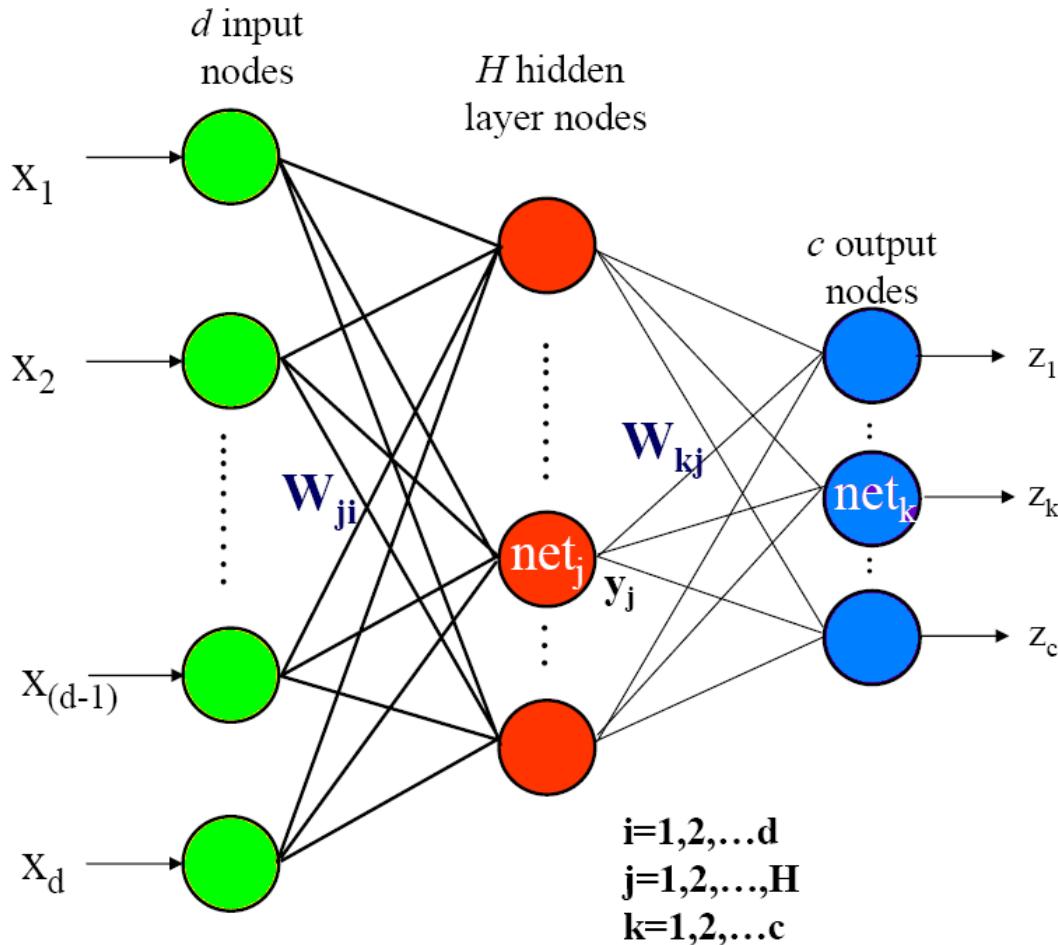
- Some desirable attributes of an activation function
 - Nonlinearity: gives the MLP the power or generating nonlinear decision boundaries
 - Saturation (for classification problems) so that the outputs can be limited between a min and a max limit (e.g. $\rightarrow -1 \& 1$).
 - Continuity and smoothness – so we can take its derivative
 - Monotonicity – so that the activation function itself does not introduce additional local minima
- The logarithmic sigmoid and the tangential sigmoid satisfy all attributes



What Can Perceptrons Represent?

 $x_1 \text{ AND } x_2$  $x_1 \text{ OR } x_2$ How about $x_1 \text{ XOR } x_2$?

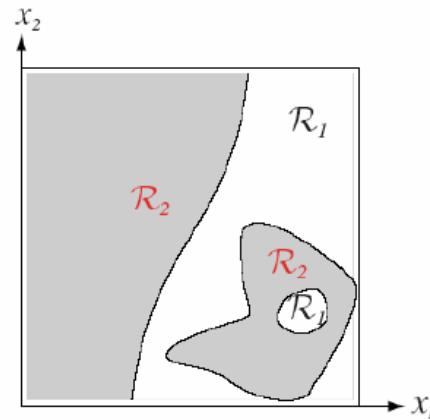
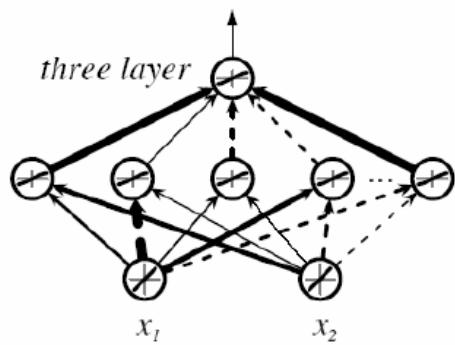
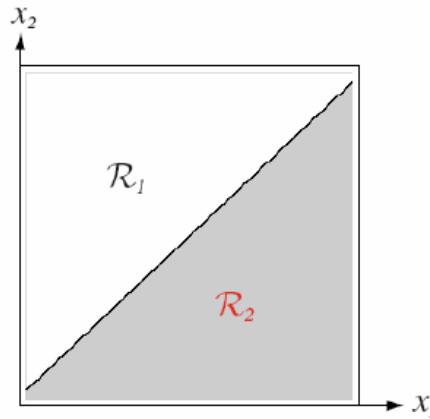
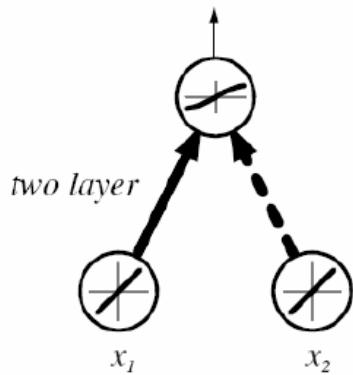
Multilayer Perceptrons – Feedforward Operation



$$y_j = f(net_j) = f\left(\underbrace{\sum_{i=1}^d w_{ji}x_i}_{net_j}\right)$$

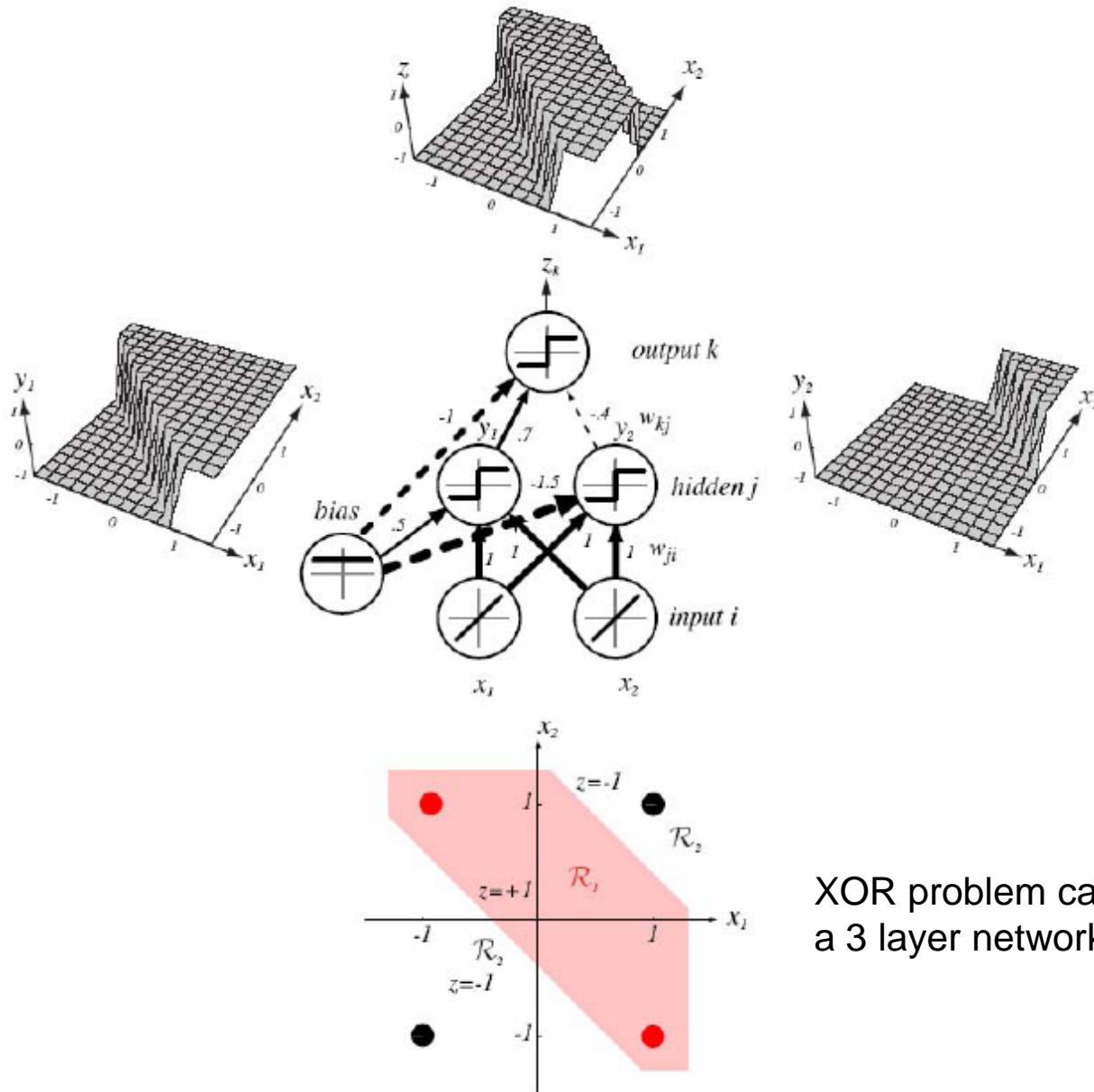
$$z_k = f(net_k) = f\left(\underbrace{\sum_{j=1}^H w_{kj}y_j}_{net_k}\right)$$

Decisions Boundaries



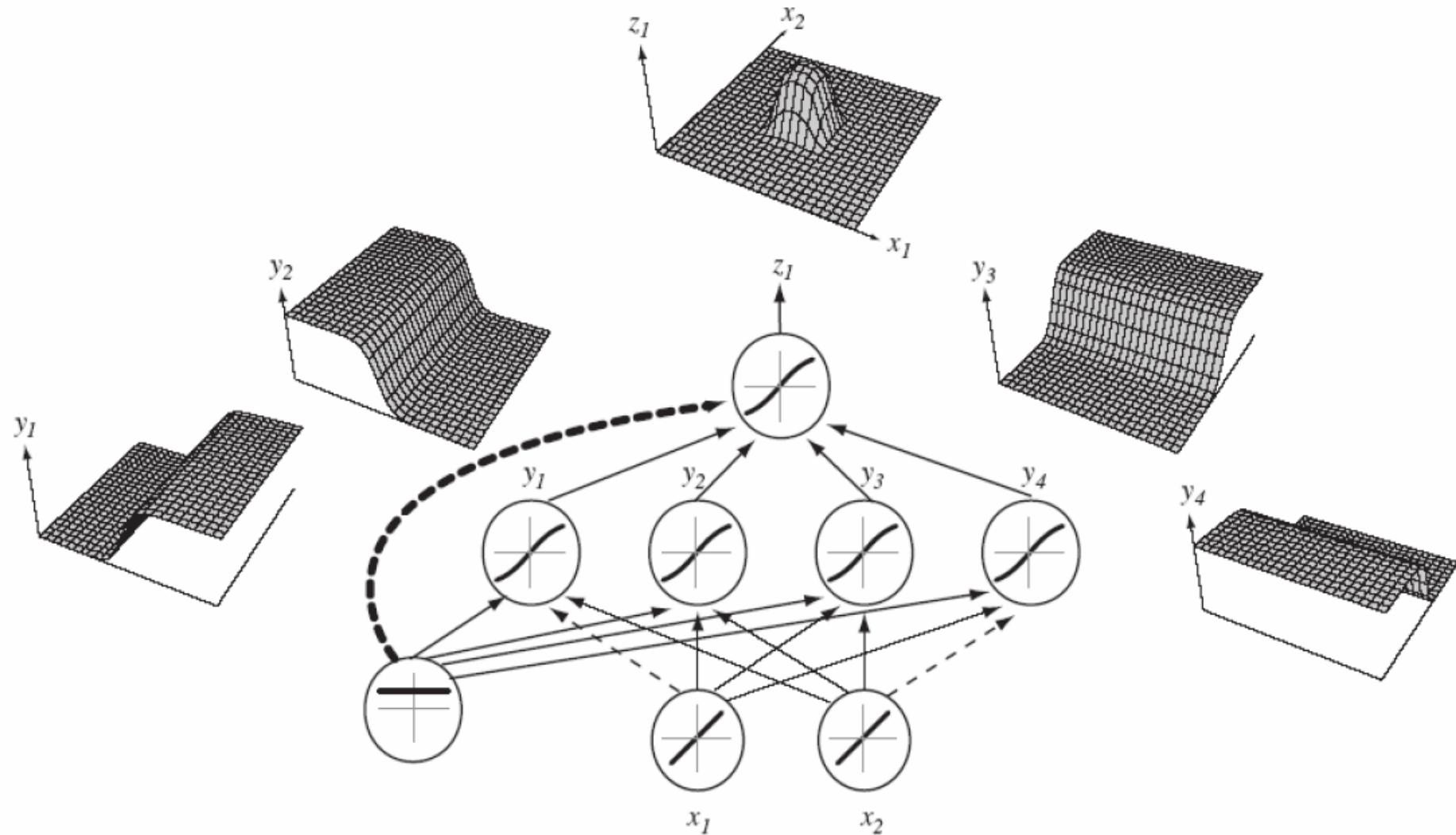
A two layer network classifier can only implement a linear decision boundary. But given an adequate number of hidden units, higher-layer networks can implement arbitrary decision boundaries.

XOR: Simple Three-Layer ANN



2-4-1 ANN

Given a sufficiently large number of hidden units, any continuous function from input to output can be approximated arbitrarily well by such a network. Hence a MLP can solve any non-linearly separable classification problem.



Notation

- x_i is the i^{th} input to the network
- $w_{ij}^{<1>}$ is the weight connection the i^{th} input to the j^{th} hidden neuron
- $\text{net}_j^{<1>}$ is the dot product at the j^{th} hidden neuron
- y_j is the output of the j^{th} hidden neuron
- $w_{jk}^{<2>}$ is the weight connecting the j^{th} hidden neuron to the k^{th} output
- $\text{net}_k^{<2>}$ is the dot product at the k^{th} output neuron
- t_k is the target or desired output at the k^{th} output neuron

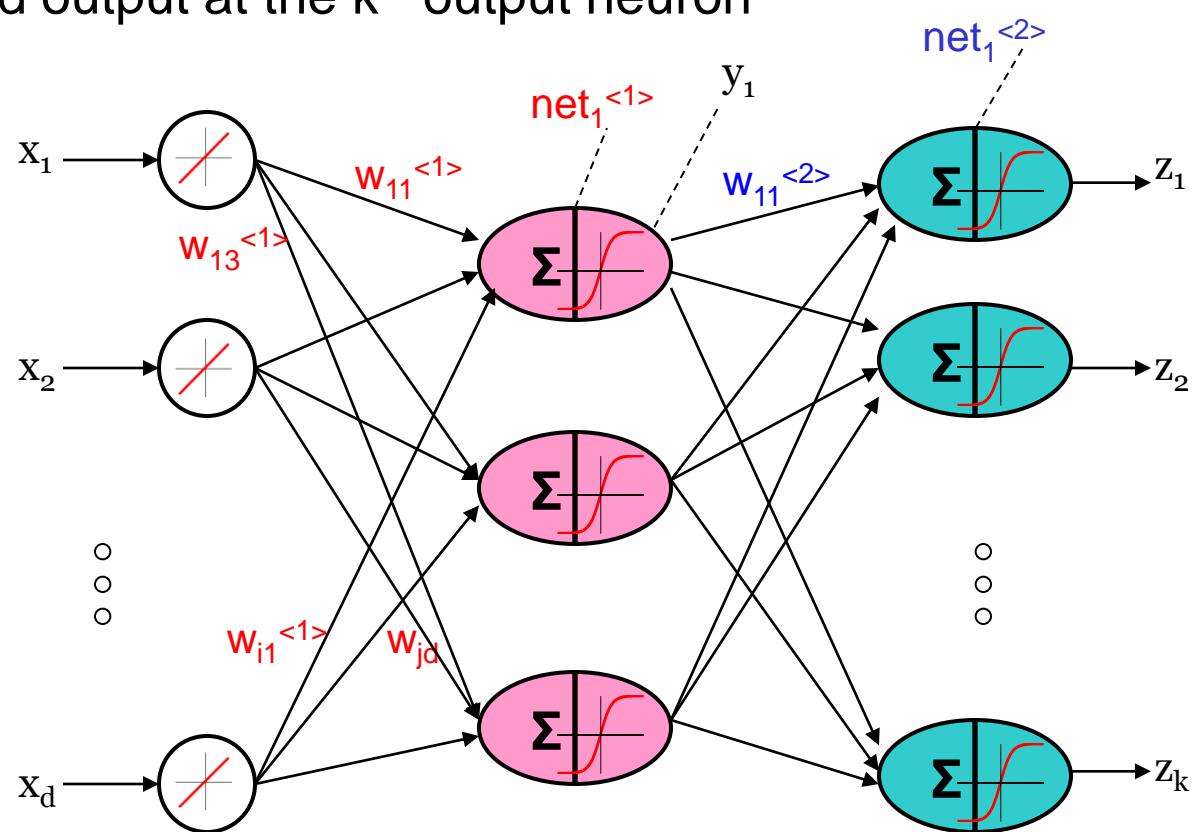
Example:

$$\text{net}_j^{<1>} = \sum_{i=1}^{N_{\text{input}}} x_i w_{ij}^{<1>}$$

$$y_j^{<1>} = f(\text{net}_j^{<1>}) = \frac{1}{1 + \exp(-\text{net}_j^{<1>})}$$

$$\text{net}_k^{<2>} = \sum_{j=1}^{N_{\text{hidden}}} y_j w_{jk}^{<2>}$$

$$y_k^{<2>} = f(\text{net}_k^{<2>}) = \frac{1}{1 + \exp(-\text{net}_k^{<2>})}$$



Backpropagation Algorithm 1

- The learning problem is finding the weight vector \mathbf{w} that captures the input/output mapping implicit in a set of training examples
 - If we use the sum-squared error at the output as our criterion

$$J(\mathbf{w}) = \sum_{n=1}^{Nsamples} \sum_{k=1}^{Noutput} \frac{1}{2} (t_k^{(n)} - z_k^{(n)})^2$$

- where $t_k^{(n)}$ is the desired target of the k^{th} output neuro for the n^{th} sample
- where $z_k^{(n)}$ is the output of the k^{th} output neuron for the n^{th} sample
- $\mathbf{w}=\{w_{ij}^{<1>} , w_{jk}^{<2>} , \dots\}$ is the set of all weights.

- Backprop learns the weights through gradient descent

$$\mathbf{w} = \mathbf{w} + \Delta \mathbf{w} = \mathbf{w} - \eta \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$$

- For simplicity, we first assume that the number of samples is 1

$$J(\mathbf{w}) = \sum_{k=1}^{Noutput} \frac{1}{2} (t_k^{(n)} - z_k)^2$$

Backpropagation Algorithm 2

- Computing dJ/dw for **hidden-to-output** (HO) weights
 - Using the **chain rule**, the derivative of $J(W)$ with respect to a HO weight is

$$\frac{\partial J(\mathbf{w})}{\partial w_{jk}^{<2>}} = \frac{\partial J(\mathbf{w})}{\partial z_k} \frac{\partial z_k}{\partial \text{net}_k^{<2>}} \frac{\partial \text{net}_k^{<2>}}{\partial w_{jk}^{<2>}}$$

- Compute each one separately

$$1. \quad \frac{\partial J(\mathbf{w})}{\partial z_k} = \frac{\partial}{\partial z_k} \left[\sum_{k=1}^{N_{\text{output}}} \frac{1}{2} (t_k^{(n)} - z_k^{(n)})^2 \right] = z_k - t_k^{(n)}$$

$$2. \quad \frac{\partial z_k}{\partial \text{net}_k^{<2>}} = \frac{\partial}{\partial \text{net}_k^{<2>}} \left[\frac{1}{1 + \exp(-\text{net}_k^{<2>})} \right] \\ = \frac{\exp(-\text{net}_k^{<2>})}{(1 + \exp(-\text{net}_k^{<2>}))^2} = \left[\frac{\exp(-\text{net}_k^{<2>}) + 1 - 1}{1 + \exp(-\text{net}_k^{<2>})} \right] \left[\frac{1}{1 + \exp(-\text{net}_k^{<2>})} \right] = (1 - z_k) z_k$$

$$3. \quad \frac{\partial \text{net}_k^{<2>}}{\partial w_{jk}^{<2>}} = \frac{\partial}{\partial w_{jk}^{<2>}} \left[\sum_{n=1}^{N_{\text{hidden}}} w_{nk}^{<2>} y_n \right] = y_j$$

- Final Result for HO is

$$\boxed{\frac{\partial J(\mathbf{w})}{\partial w_{jk}^{<2>}} = (z_k - t_k^{(n)}) (1 - z_k) z_k y_j}$$

Backpropagation Algorithm 3

- Computing dJ/dw for **input-to-hidden (IH)** weights
 - Using the **chain rule**, the derivative of $J(W)$ with respect to a IH weight is

$$\frac{\partial J(\mathbf{w})}{\partial w_{ij}^{<1>}} = \frac{\partial J(\mathbf{w})}{\partial y_j} \frac{\partial y_j}{\partial \text{net}_j^{<1>}} \frac{\partial \text{net}_j^{<1>}}{\partial w_{ij}^{<1>}}$$
 - Following similar approach as before, we can easily compute the last 2 terms:
 - 1. $\frac{\partial y_j^{<1>}}{\partial \text{net}_j^{<1>}} = (1 - y_j) y_j$
 - 2. $\frac{\partial \text{net}_j^{<1>}}{\partial w_{ij}^{<1>}} = x_i$
 - The first term poses a problem and is not straightforward since we ignore what the outputs of the hidden neurons will be
 - This is known as the “credit assignment problem” [Minsky, 1961], and confused scientists for 2 decades
 - The trick to solve it was to realize that the hidden neurons do not make errors but only contribute to the errors of the output nodes.
 - The derivative of the error with respect to a hidden node’s output is the sum of that hidden node’s contribution to the errors of all the output neurons

$$\frac{\partial J(\mathbf{w})}{\partial y_j} = \sum_{n=1}^{N_{\text{output}}} \frac{\partial J(\mathbf{w})}{\partial z_n} \frac{\partial z_n}{\partial \text{net}_n^{<2>}} \frac{\partial \text{net}_n^{<2>}}{\partial y_j}$$

Backpropagation Algorithm 4

- The derivative of the error with respect to a hidden node's output is the sum of that hidden node's contribution to the errors of all the output neurons

$$\frac{\partial J(\mathbf{w})}{\partial y_j} = \sum_{n=1}^{N_{output}} \frac{\partial J(\mathbf{w})}{\partial z_n} \frac{\partial z_n}{\partial \text{net}_n^{<2>}} \frac{\partial \text{net}_n^{<2>}}{\partial y_j}$$

- Using similar techniques to the previous step, we can get the first 2 terms:

$$\frac{\partial J(\mathbf{w})}{\partial z_n} = z_n - t_n^{(n)} \quad \frac{\partial z_n}{\partial \text{net}_n^{<2>}} = (1 - z_n) z_n$$

- The last term is given by:

$$\frac{\partial \text{net}_n^{<2>}}{\partial y_j} = w_{jn}^{<2>}$$

- Merging all terms together, we obtain:

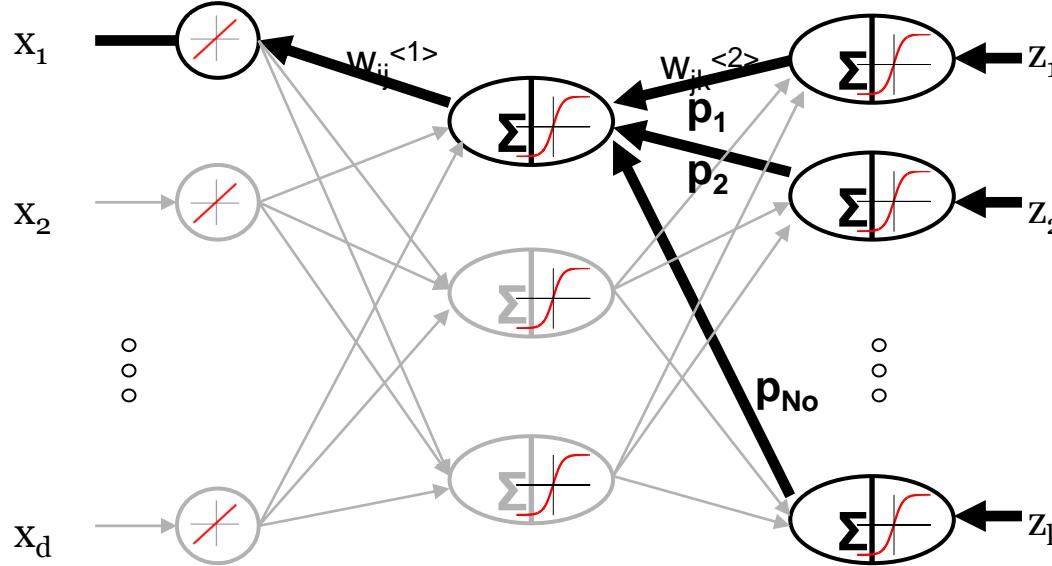
$$\frac{\partial J(\mathbf{w})}{\partial y_j} = \sum_{n=1}^{N_{output}} \frac{\partial J(\mathbf{w})}{\partial z_n} \frac{\partial z_n}{\partial \text{net}_n^{<2>}} \frac{\partial \text{net}_n^{<2>}}{\partial y_j} = \sum_{n=1}^{N_{output}} (z_n - t_n^{(n)}) (1 - z_n) z_n w_{jn}^{<2>}$$

Backpropagation Algorithm 5

- The derivative of the error with respect to a hidden node is given by:

$$\frac{\partial J(\mathbf{w})}{\partial y_j} = \sum_{n=1}^{N_{output}} \underbrace{(z_n - t_n^{(n)}) (1 - z_n) z_n}_{p_n} w_{jn}^{<2>}$$

- What we are actually doing is propagating the error term p_n backwards through the hidden-to-output weights



- The combined expression for $dJ(w)/dw$ for input-to-hidden weights is

$$\frac{\partial J(\mathbf{w})}{\partial w_{ij}^{<1>}} = \left[\sum_{n=1}^{N_{output}} (z_n - t_n^{(n)}) (1 - z_n) z_n w_{jn}^{<2>} \right] (1 - y_j) y_j x_j$$

Backprop: Recap 1

- The weights are determined through a gradient descent error minimization of a criterion function $J(\mathbf{w})$

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t) \Rightarrow \Delta\mathbf{w} = -\eta \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$$

- We need to express $J(\mathbf{w})$ in terms of \mathbf{w} for both output and hidden layer nodes. Output nodes are easy since we know the functional representation of J with respect to \mathbf{w} through the chain rule:

$$\frac{\partial J(\mathbf{w})}{\partial w_{jk}} = \frac{\partial J(\mathbf{w})}{\partial net_k} \frac{\partial net_k}{\partial w_{jk}} = \frac{\partial J(\mathbf{w})}{\partial z_k} \frac{\partial z_k}{\partial net_k} \frac{\partial net_k}{\partial w_{jk}} = - \underbrace{\left(t_k - z_k \right) f'(net_k)}_{\delta_k} y_j$$

↑
Output node sensitivity

For output layer weights

$$\Delta w_{jk} = \eta \delta_k y_j = \eta (t_k - z_k) f'(net_k) y_j$$

Backprop: Recap 2

- For the hidden layer, things are more complicated because we do not know the desired values of the hidden layer node outputs. However, by the appropriate use of the chain rule we obtain:

$$\frac{\partial J(\mathbf{w})}{\partial w_{ij}} = \frac{\partial J(\mathbf{w})}{\partial y_j} \frac{\partial y_j}{\partial w_{kj}} = \frac{\partial J(\mathbf{w})}{\partial y_j} \frac{\partial y_j}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial w_{ij}} = \frac{\partial J(\mathbf{w})}{\partial y_j} f'(\text{net}_j) x_i$$

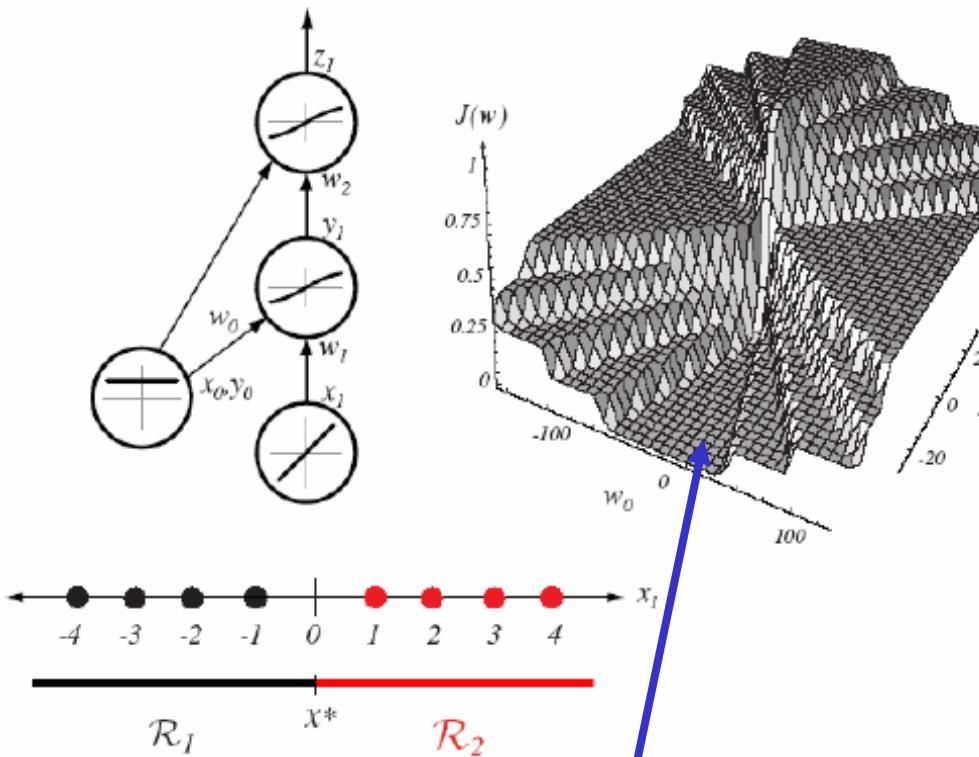
$$\frac{\partial J(\mathbf{w})}{\partial y_j} = \frac{\partial}{\partial y_j} \left[\frac{1}{2} \sum_{n=1}^{N\text{output}} (t_k - z_k)^2 \right] = - \sum_{n=1}^{N\text{output}} (t_k - z_k)^2 = - \sum_{n=1}^{N\text{output}} (t_k - z_k) \frac{\partial z_k}{\partial y_j}$$

$$= - \underbrace{\sum_{n=1}^{N\text{output}} (t_k - z_k)}_{\delta_k} f'(\text{net}_k) w_{jk}$$

$$\Delta w_{ij} = -\eta \frac{\partial J}{\partial w_{ij}} = \eta \sum_{k=1}^{N\text{output}} \delta_k w_{jk} f'(\text{net}_j) x_i = \eta \delta_j x_i$$

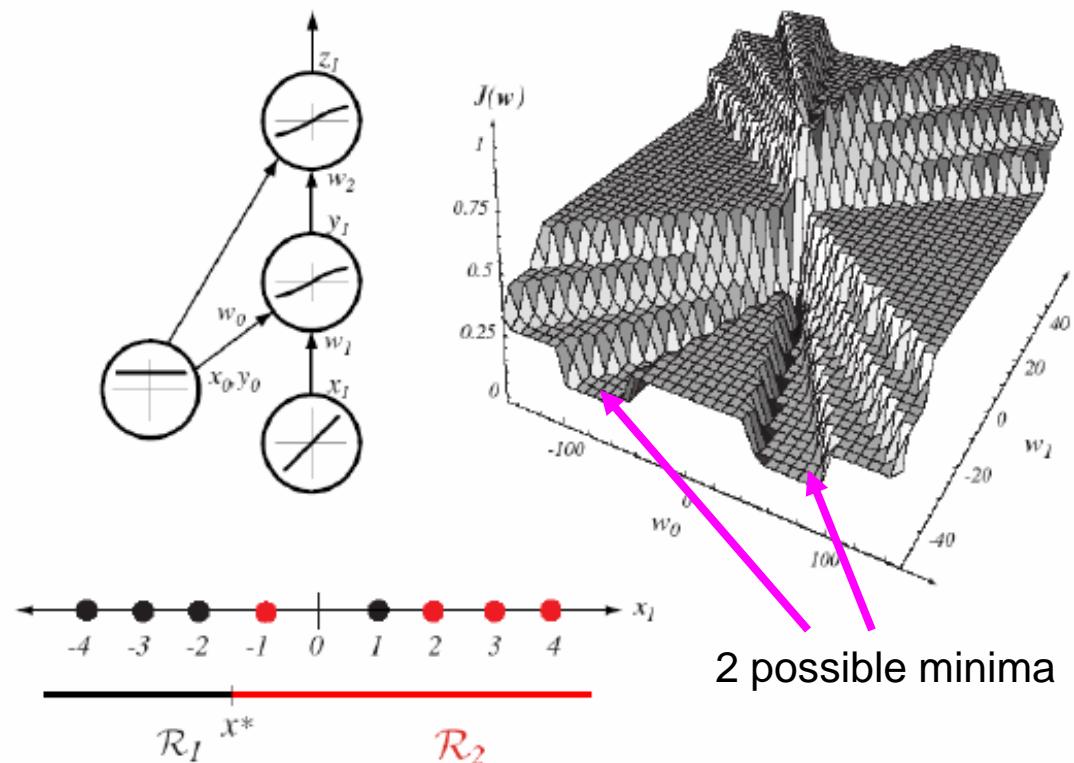
The parameter ζ represents the sensitivity of the criterion function (error) with respect to a hidden/output layer node. Note that the sensitivity of a hidden layer node is a weighted sum of the output sensitivities, scaled by $f(\text{net}_j)$, where the weights are the hidden-to-output layer weights.

Error Surfaces



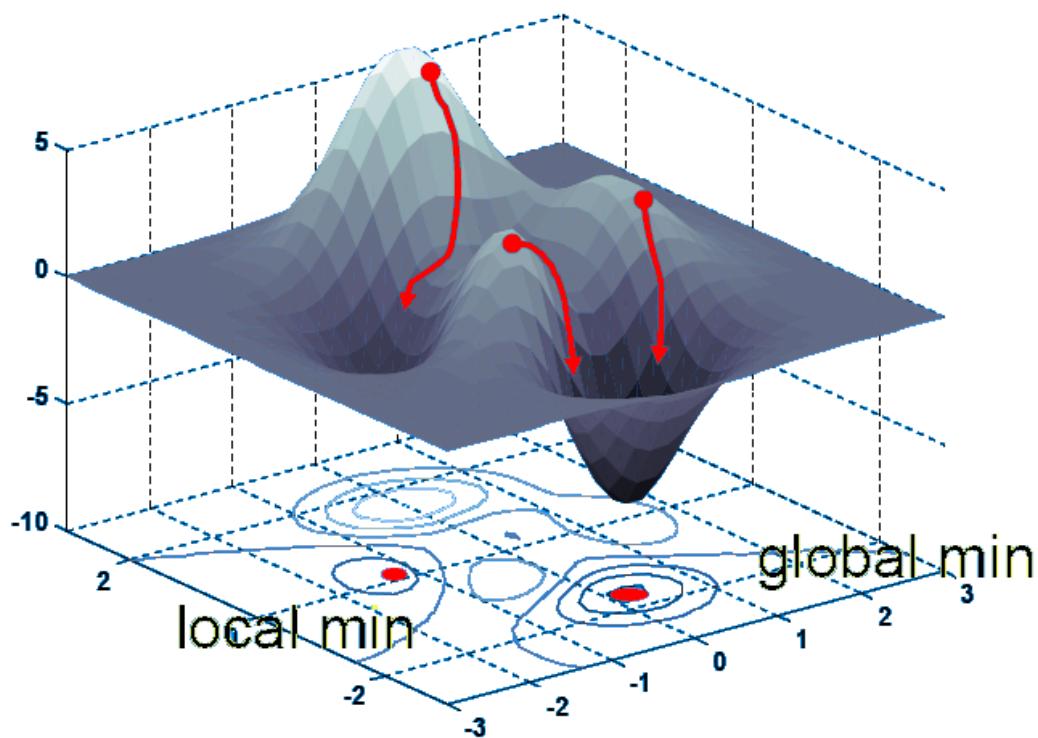
Linearly Separable case: A low error solution exists which leads to a decision boundary separating the sample points

Non-Linearly Separable case:
The error surface is slightly higher and two forms of minimum error solution exist.



General Multimodal Cost Surface

- One of the main problem of backprop (and gradient descent in general) is the local minima which creates a suboptimal solution

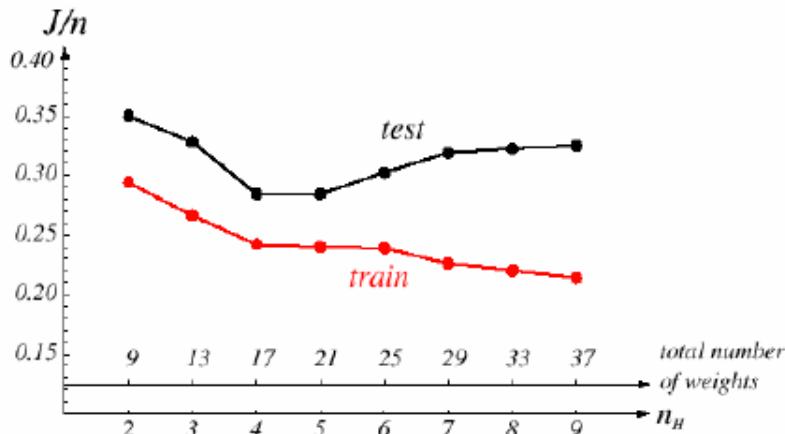


Implementation Issues 1

- Choosing an activation function
- Input normalization
 - If two features vary orders of magnitude in their values, network cannot learn effectively. For stability reasons, individual features need to be in the same order of magnitude.
- Target Values
 - Usually 0/1 for sigmoid function and -1/1 for tanh.
- Number of Hidden Units
 - It defines the *expressive power* of the network.
 - Too many units cause overfitting.
 - Too few may not be able to solve a complicated problem.
 - A rule of thumb is to choose number of hidden units such that the total number of weights remains less than $N/10$ where N is the total number of training samples.

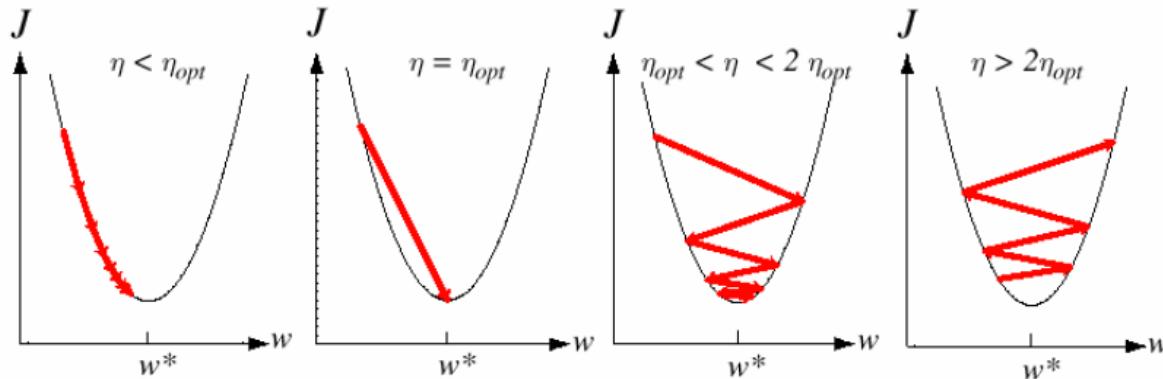
Network trained with 90 2D features from 2 categories $\rightarrow n = 180$

The minimum of the test error occurs in the range $N \rightarrow [17, 21]$ corresponding to number of hidden units $\rightarrow [4, 5]$



Implementation Issues 2

- Initializing Weights
 - Determines the final solution in the case of a complex error surface.
 - A rule of thumb is to randomly choose the weights from a uniform distribution:
- Learning Rates
 - In theory only affects convergence time, but in practice can lead to divergence:



- Momentum
 - Adding a momentum term can control the speed of learning
- Stopping Criterion
 - The algorithm should be stopped before it reaches its minimum error, because too small error causes the noise in the data to be learned at the expense of the general pattern.
- Regularization
 - Is the smoothing of the error curve so that the optimum solution can be found.

Recap

- Discriminant Functions
- Perceptron Learning
- Gradient Descent
- Minimum Square Error Solution
- Least Mean Square Solution
- Ho-Kashyap Procedure
- Artificial Neural Networks
- Activation Functions and Multilayer Perceptrons
- Backpropagation Algorithm
- Implementation Issues

Prof. Marios Savvides

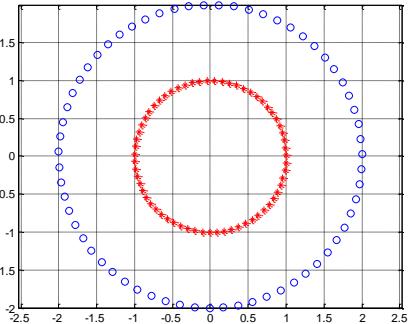
Pattern Recognition Theory

Lecture 14 : Kernel Feature Analysis

All graphics from Pattern Classification, Duda, Hart and Stork, Copyright © John Wiley and Sons, 2001

Linear vs Nonlinear Methods

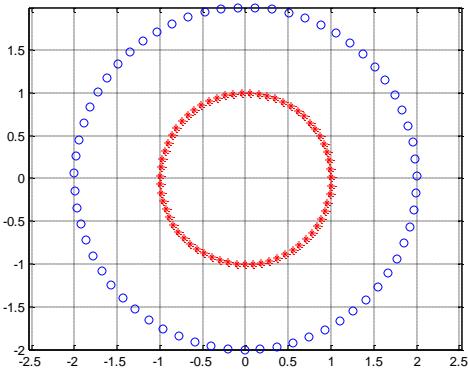
- Linear methods
 - Assume that data are linearly separable.
 - Simple, easy, fundamental, computationally cheap, etc.
 - However, real-world data samples are hardly linearly separable.



- Nonlinear methods
 - Applicable to complex real-world data that are not linearly separable.
 - Provide more classification power.

Linear vs Nonlinear Methods

- Data are not linearly separable -> might be separable in high dimension.

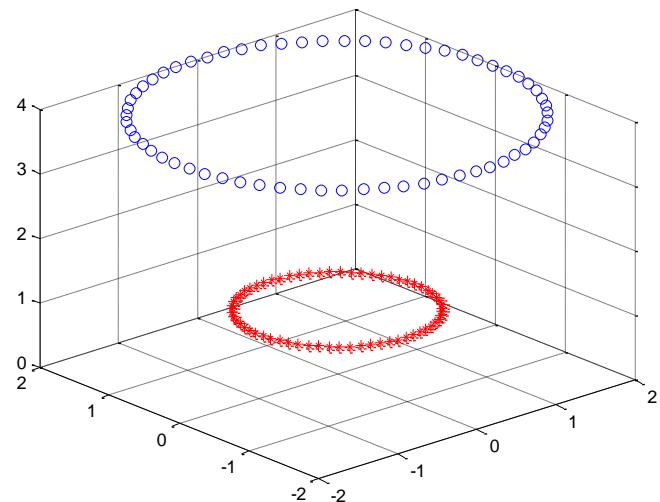


original data space

- lower dim.
- not linearly separable

$$\phi: \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{bmatrix}$$



feature space

- higher dim.
- linearly separable

- Mappings onto a space higher in dimension than the original space might provide greater classification power.
- Data become linearly separable at the expense of dimensionality.

Explicit Mapping Example

- Consider polynomial kernel

– $d = 1$

$$\Phi(\mathbf{x}) = \mathbf{x} \quad \Phi(\mathbf{x}) \bullet \Phi(\mathbf{y}) = (x_1 y_1 + x_2 y_2) = \mathbf{x} \bullet \mathbf{y}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

– $d = 2$

$$\Phi(\mathbf{x}) = \begin{bmatrix} x_1^2 & x_1 x_2 & x_2 x_1 & x_2^2 \end{bmatrix}^T$$

$$\Phi(\mathbf{x}) \bullet \Phi(\mathbf{y}) = \begin{bmatrix} x_1^2 & x_1 x_2 & x_2 x_1 & x_2^2 \end{bmatrix} \begin{bmatrix} y_1^2 \\ y_1 y_2 \\ y_2 y_1 \\ y_2^2 \end{bmatrix} = x_1^2 y_1^2 + 2x_1 x_2 y_1 y_2 + x_2^2 y_2^2 = (\mathbf{x} \bullet \mathbf{y})^2$$

– Any d

$$\Phi(\mathbf{x}) \bullet \Phi(\mathbf{y}) = (\mathbf{x} \bullet \mathbf{y})^d$$

- Define Kernel function

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \bullet \mathbf{y})^d = \Phi(\mathbf{x}) \bullet \Phi(\mathbf{y})$$

Kernel Methods

- Problem: ϕ is unknown.
 - A proper nonlinear mapping function is not defined explicitly.
 - We do not even know the dimensionality of the feature space.
- Kernel-based extension of a linear methods
 - If a certain linear method is formulated with dot products, it can be performed in the higher dimensional feature space based on $\phi(\mathbf{x}) \bullet \phi(\mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y})$.
 - Thus, if $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \bullet \phi(\mathbf{y})$ is defined, the linear method can be applied to the feature space without any explicit usage of ϕ .
- Kernel Trick
 - Data are not represented individually anymore, but only through a set of pairwise comparisons.
 - Instead of using a mapping $\phi: R^d \rightarrow R^{d_{high}}$ to represent each object $\mathbf{x} \in R^d$, a comparison function $k(\mathbf{x}, \mathbf{y})$ is used.
 - Hence, no need to compute mapping
 - No need to compute dot product

The Kernel Trick

- We can take advantage of high dimensional representations without having to work in the high dimension space.
- It is possible to compute dot products in high dimension spaces **without explicitly mapping** into these spaces. We employ dot products of the form

$$K(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \bullet \Phi(\mathbf{y})$$

– Example: Polynomial kernel of degree d

$$K(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \bullet \Phi(\mathbf{y}) = (\mathbf{x} \bullet \mathbf{y})^d$$

- A Kernel function $K()$ can be represented as
 - $K(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x})^T \Phi(\mathbf{y})$ if and only if for a square integrable function $g(\mathbf{x})$ ($\int g(\mathbf{x})^2 d\mathbf{x}$ is finite)

$$\int K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0$$

- This is called **Mercer's condition**.

Common Kernels

- Polynomial of degree d

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \bullet \mathbf{y})^d$$

- Polynomial of degree up to d

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \bullet \mathbf{y} + 1)^d$$

- Gaussian Kernels (Radial Basis Function)

$$K(\mathbf{x}, \mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}}$$

- Hyperbolic Tangent

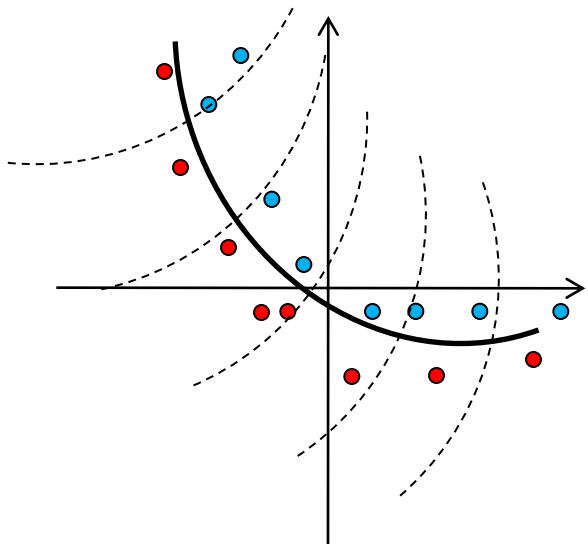
$$K(\mathbf{x}, \mathbf{y}) = \tanh(\eta \mathbf{x} \bullet \mathbf{y} + v)$$

Kernel PCA

- KPCA is designed to perform PCA in the arbitrarily high-dimensional feature space related to the input space through a nonlinear mapping.
- KPCA assumes that the nonlinear mapping makes the mapped data linearly separable.
- Based on the observation that PCA is formulated with dot products, one can apply the kernel trick to PCA and consequently obtain KPCA, the kernel-based extension of PCA.

Kernel PCA

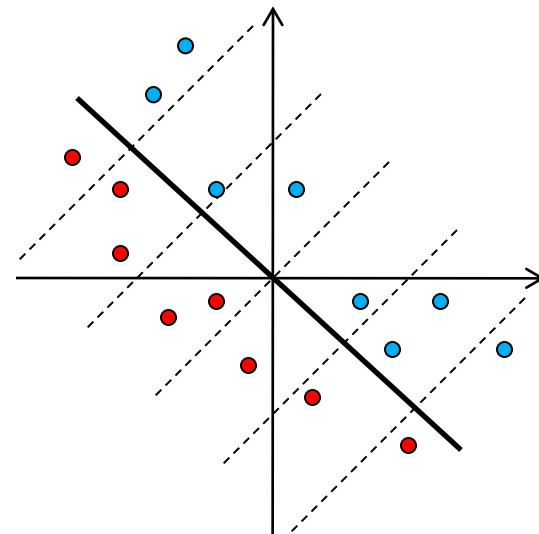
- low dimension: $\mathbf{x} \in R^d$
- nonlinear: $k(\mathbf{x}_i, \mathbf{x}_j)$



$\phi(\cdot)$

$d << d_{high}$

- high dimension: $\phi(\mathbf{x}) \in R^{d_{high}}$
- linear: $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$



Kernel trick: $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$

The subspace achieved by KPCA is nonlinear in the input space, which allows KPCA to be applied to complex real-world data that are not linearly separable.

PCA Refresher

- PCA efficiently represents the data by finding orthonormal axes which maximally de-correlate the data
 - Assumptions: data samples are independent, and Gaussian distributed.
- PCA finds the principal axes by diagonalizing the covariance matrix.

$$\Sigma = \frac{1}{N} \mathbf{X}\mathbf{X}^T = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T$$

- Assume N sample points that have been centered ($m = 0$)

$$\lambda \mathbf{v} = \Sigma \mathbf{v}$$

- Find all eigenvectors, arrange them, project onto them, and use the coefficients
- To be able to use kernels for PCA, we have to rewrite it in terms of dot products

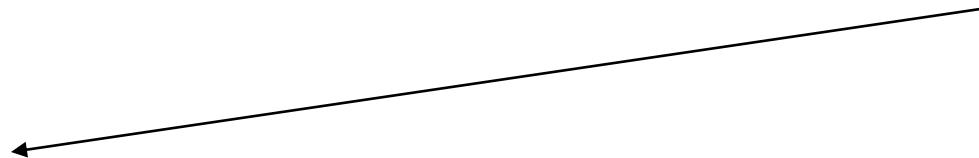
$$\Sigma \mathbf{v} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T \mathbf{v} = \lambda \mathbf{v}$$

- Thus

$$\mathbf{v} = \frac{1}{N\lambda} \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T \mathbf{v} = \frac{1}{N\lambda} \sum_{i=1}^N (\mathbf{x}_i \bullet \mathbf{v}) \mathbf{x}_i$$

Show that $(\mathbf{x}\mathbf{x}^T)\mathbf{v} = (\mathbf{x}^T\mathbf{v})\mathbf{x}$

$$\mathbf{x}\mathbf{x}^T\mathbf{v} = \begin{bmatrix} x_1x_1 & x_1x_2 & \cdots & x_1x_d \\ x_2x_1 & x_2x_2 & & \vdots \\ \vdots & \ddots & & \\ x_dx_1 & & x_dx_d \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{bmatrix} = \begin{bmatrix} x_1x_1v_1 + x_1x_2v_2 + \cdots + x_1x_dv_d \\ x_2x_1v_1 + x_2x_2v_2 + \cdots + x_2x_dv_d \\ \vdots \\ x_cx_1v_1 + x_cx_2v_2 + \cdots + x_cx_dv_d \\ x_dx_1v_1 + x_dx_2v_2 + \cdots + x_dx_dv_d \end{bmatrix}$$



$$\begin{bmatrix} (x_1v_1 + x_2v_2 + \cdots + x_dv_d)x_1 \\ (x_1v_1 + x_2v_2 + \cdots + x_dv_d)x_2 \\ \vdots \\ (x_1v_1 + x_2v_2 + \cdots + x_dv_d)x_d \end{bmatrix} = [x_1v_1 + x_2v_2 + \cdots + x_dv_d] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = (\mathbf{x}^T\mathbf{v})\mathbf{x}$$

KPCA - 1

- We had

$$\mathbf{v} = \frac{1}{N\lambda} \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T \mathbf{v} = \frac{1}{N\lambda} \sum_{i=1}^N (\mathbf{x}_i \bullet \mathbf{v}) \mathbf{x}_i$$

- However $(\mathbf{x}_i \cdot \mathbf{v})$ is just a scalar, which means that all solutions \mathbf{v} with non zero eigenvalue lie in the span of $\mathbf{x}_1, \dots, \mathbf{x}_d$

$$\mathbf{v} = \sum_{i=1}^N \alpha_i \mathbf{x}_i$$

Eigenvectors can be expressed as linear combinations of the training data

- Map into another space,

$$\Phi : X \rightarrow H, \mathbf{x} \rightarrow \Phi(\mathbf{x})$$

- Assuming we can center the data in that space ($\sum_{k=1}^N \Phi(\mathbf{x}_k) = 0$), we can write the covariance matrix as:

$$\Sigma = \frac{1}{N} \sum_{i=1}^N \Phi(\mathbf{x}_i) \Phi(\mathbf{x}_i)^T$$

- We just showed that all solutions \mathbf{v} with non zero eigenvalues must lie in the span of $\Phi(\mathbf{x}_1), \dots, \Phi(\mathbf{x}_N)$, that is

$$\Sigma \mathbf{v} = \lambda \mathbf{v} = \lambda \sum_{i=1}^N \alpha_i \Phi(\mathbf{x}_i)$$

KPCA - 2

- Proof

$$\Phi: X \rightarrow H, \mathbf{x} \rightarrow \Phi(\mathbf{x}) \quad \Sigma = \frac{1}{N} \sum_{i=1}^N \Phi(\mathbf{x}_i) \Phi(\mathbf{x}_i)^T$$

$$\Sigma \mathbf{v} = \lambda \mathbf{v} \rightarrow \left(\frac{1}{N} \sum_{i=1}^N \Phi(\mathbf{x}_i) \Phi(\mathbf{x}_i)^T \right) \mathbf{v} = \lambda \mathbf{v}$$

$$\mathbf{v} = \left(\frac{1}{\lambda N} \sum_{i=1}^N \Phi(\mathbf{x}_i) \Phi(\mathbf{x}_i)^T \right) \mathbf{v} = \sum_{i=1}^N \left(\frac{\Phi(\mathbf{x}_i)^T \mathbf{v}}{\lambda N} \right) \Phi(\mathbf{x}_i) = \sum_{i=1}^N \alpha_i \Phi(\mathbf{x}_i)$$

- To express the relationship entirely in terms of the inner-product kernel, we premultiply both sides by $\Phi(\mathbf{x}_k)^T$

$$\Phi(\mathbf{x}_k)^T \Sigma \mathbf{v} = \lambda \Phi(\mathbf{x}_k)^T \mathbf{v}$$

- Combine everything back into the previous equation

$$\lambda \left[\Phi(\mathbf{x}_k)^T \sum_{i=1}^N \alpha_i \Phi(\mathbf{x}_i) \right] = \Phi(\mathbf{x}_k)^T \left[\frac{1}{N} \sum_{j=1}^N \Phi(\mathbf{x}_j) \Phi(\mathbf{x}_j)^T \right] \left[\sum_{i=1}^N \alpha_i \Phi(\mathbf{x}_i) \right]$$

KPCA - 3

- Regrouping terms, we obtain

$$\lambda \sum_{i=1}^N \alpha_i \left(\Phi(\mathbf{x}_k)^T \Phi(\mathbf{x}_i) \right) = \frac{1}{N} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \Phi(\mathbf{x}_k)^T \Phi(\mathbf{x}_i) \left(\Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j) \right)$$

- Define the $N \times N$ matrix \mathbf{K} , called the kernel matrix, whose ij^{th} element in the inner product kernel $K(\mathbf{x}_i, \mathbf{x}_j)$
- The $N \times 1$ vector $\boldsymbol{\alpha}$ whose j^{th} element is the coefficient α_j

$$\mathbf{K} = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & \cdots & K(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ K(\mathbf{x}_N, \mathbf{x}_1) & \cdots & K(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix} \quad \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{bmatrix} \quad K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$$

- The first equation becomes:

$$N\lambda \mathbf{K}\boldsymbol{\alpha} = \mathbf{K}^2 \boldsymbol{\alpha}$$

- Which can be solved by the following eigenvalue/eigenvector problem:

$$N\lambda \boldsymbol{\alpha} = \mathbf{K}\boldsymbol{\alpha}$$

KPCA - 4

- Normalization
 - We have to make sure than the eigenvectors V are orthonormal, we scale eigenvectors α

$$\left\langle \mathbf{v}^{(k)}, \mathbf{v}^{(k)} \right\rangle = 1 \Rightarrow \left(\sum_{i=1}^N \alpha_i^{(k)} \Phi(\mathbf{x}_i) \right) \left(\sum_{j=1}^N \alpha_j^{(k)} \Phi(\mathbf{x}_j) \right) = 1$$

$$\sum_{i=1}^N \sum_{j=1}^N \alpha_i^{(k)} \alpha_j^{(k)} \Phi(\mathbf{x}_i) \Phi(\mathbf{x}_j) = 1 \Rightarrow$$

$$\sum_{i=1}^N \sum_{j=1}^N \alpha_i^{(k)} \alpha_j^{(k)} \mathbf{K}_{ij} = 1 \Rightarrow \left(\boldsymbol{\alpha}^{(k)} \mathbf{K} (\boldsymbol{\alpha}^{(k)})^T \right) = 1$$

- Projecting a new test sample:
 - We now show how to project a test point onto the eigenvectors in the high-dimensional space H in terms of dot product (so we can still utilize the kernel trick).

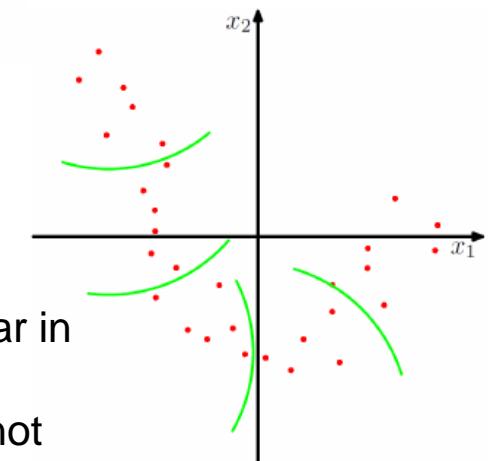
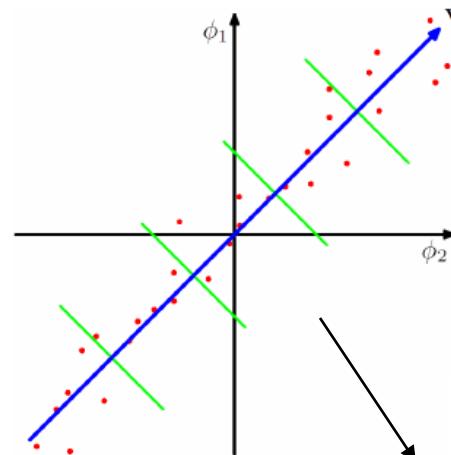
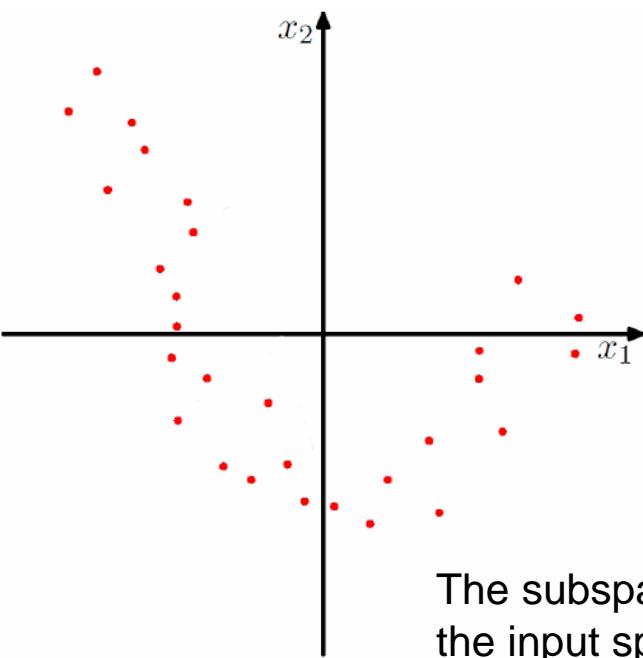
$$\left\langle \mathbf{v}^{(k)}, \Phi(\mathbf{x}) \right\rangle = \left(\sum_{i=1}^N \alpha_i^{(k)} \Phi(\mathbf{x}_i) \right) \bullet \Phi(\mathbf{x}) = \sum_{i=1}^N \alpha_i^{(k)} \mathbf{K}(\mathbf{x}_i, \mathbf{x})$$

KPCA Recap

- The eigenvectors now reside in the high dimensional space H

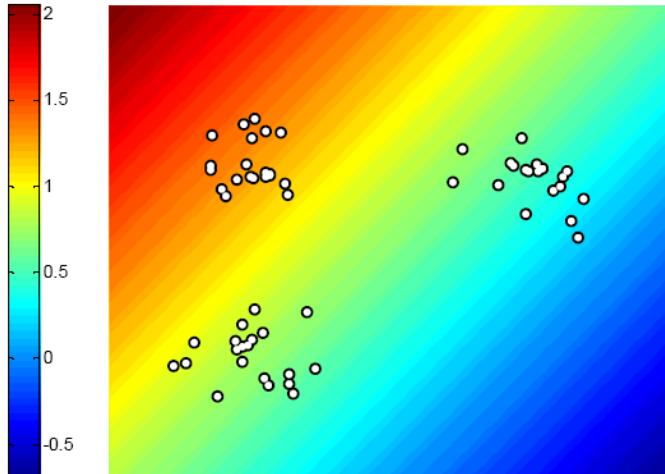
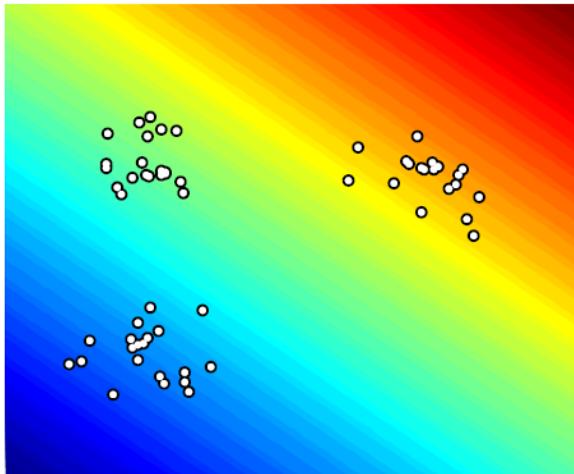
$$\mathbf{v}^{(k)} = \sum_{i=1}^N \alpha_i^{(k)} \Phi(\mathbf{x}_i)$$

- This implies that kernel PCA can be used to feature extraction but CANNOT be used (directly) for reconstruction

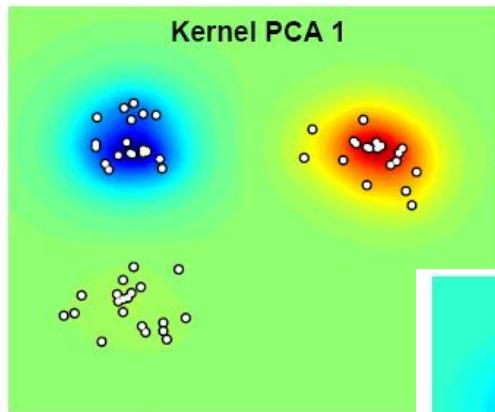


The subspace achieved by KPCA is nonlinear in the input space, which allows KPCA to be applied to complex real-world data that are not linearly separable.

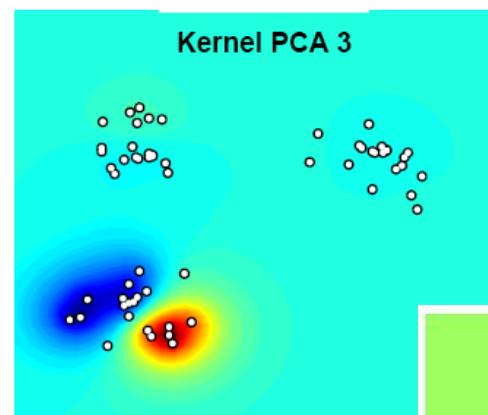
PCA vs Kernel PCA



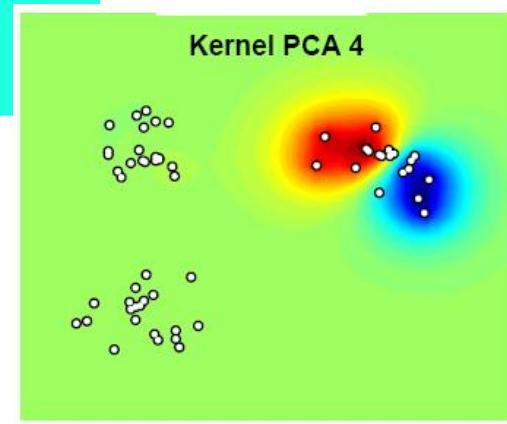
Kernel PCA 1

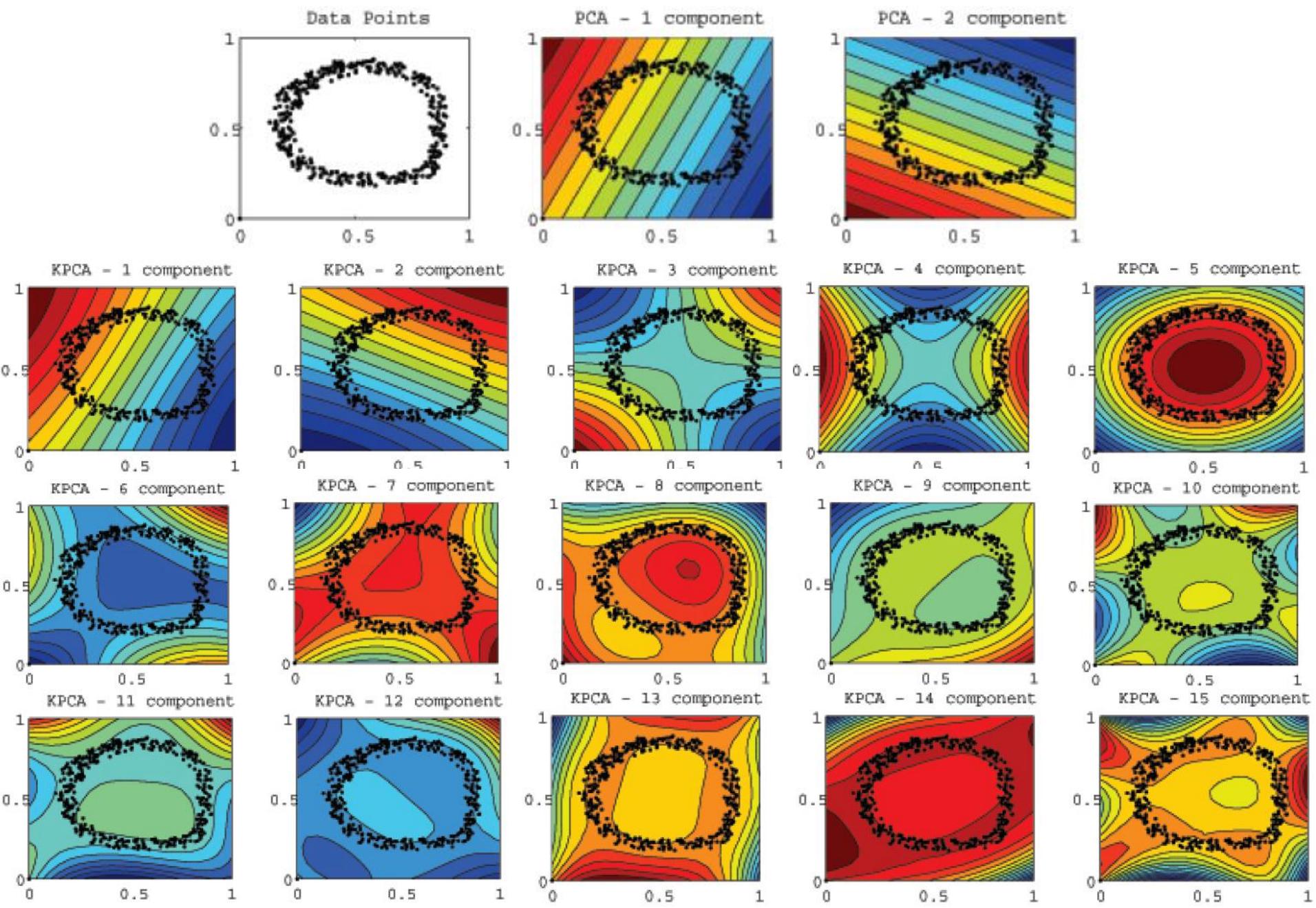


Kernel PCA 3



Kernel PCA 4





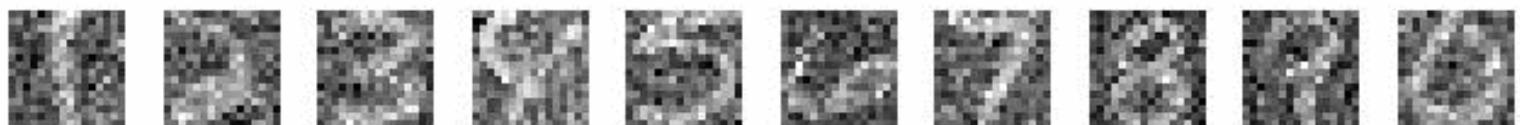
Applications Of KPCA

Denoising of the USPS hand-written numerals corrupted by Gaussian noise

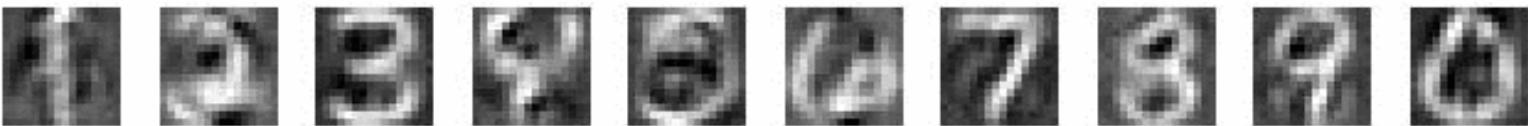
Ground truth



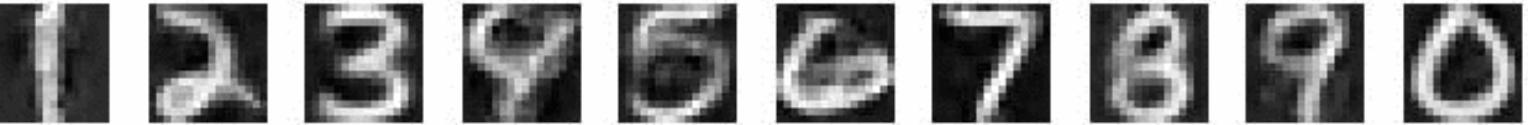
Noisy images



Linear PCA



Kernel PCA



References

- B. Schölkopf and A. Smola, “Learning with Kernels”
- J. Shawe-Taylor and N. Cristianini, “Kernel Methods for Pattern Analysis”

Recap

- Linear vs Non-Linear Methods
- Kernel Mapping
- The Kernel Trick
- Common Kernels
- Kernel PCA (KPCA)

Felix Juefei Xu & Dipan K. Pal

Pattern Recognition Theory

Recitation 1: Linear Algebra

The Basics

- Scalar (x)
 - A real number
- Vector (\mathbf{x})

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = [x_1 \quad x_2 \quad \cdots \quad x_n]^T$$

The Basics

- Matrix (\mathbf{X})

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & & x_{2m} \\ \dots & \dots & \dots & \dots \\ x_{n1} & x_{n2} & & x_{nm} \end{bmatrix}$$

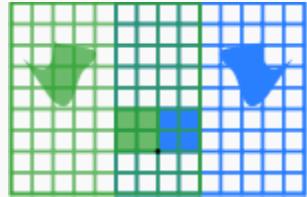
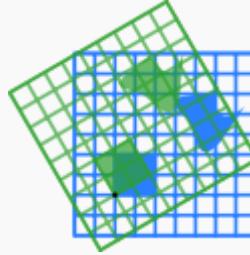
Types of Matrices

- Square & Rectangular matrices
- Upper & Lower Triangular matrices
- Identity matrix
- Symmetric & Skew-symmetric matrices
- Hermitian matrix
- Singular matrix
- Orthogonal matrix

Consider a matrix as...

- ... an operator ...
- ... which linearly transforms a vector ...
- ... to a different vector space!

For example:

Horizontal flip	Scaling by a factor of 3/2	Rotation by $\pi/6R = 30^\circ$
$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 3/2 & 0 \\ 0 & 3/2 \end{bmatrix}$	$\begin{bmatrix} \cos(\pi/6^R) & -\sin(\pi/6^R) \\ \sin(\pi/6^R) & \cos(\pi/6^R) \end{bmatrix}$
		

* Examples from Wikipedia

Vector Spaces & Subspaces

For any $m \times n$ matrix \mathbf{A} with rank r :

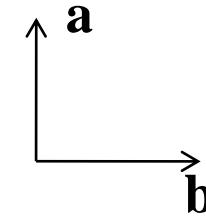
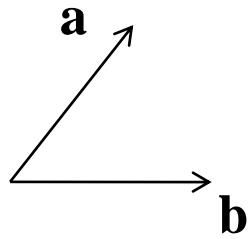
- Column Space
 - Contains all linear combinations of the columns of the matrix
 - Has dimension r
 - $\mathbf{Ax} = \mathbf{b}$ can be solved iff \mathbf{b} is in the column space of \mathbf{A} .
- Null Space
 - Contains all vectors \mathbf{x} such that $\mathbf{Ax} = 0$
 - Has dimension $(n-r)$
- Row Space / Left Null Space (equivalent to Column & Null Space on \mathbf{A}^T)

Inner Products

- Inner product of \mathbf{x} and \mathbf{y} : Sum of element-wise products

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y} = [x_1 \quad x_2 \quad \cdots \quad x_n] \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^n x_i y_i$$

- \mathbf{x} and \mathbf{y} must be equal in length
- Result is a **scalar**
 - Test of similarity of two vectors
 - Don't forget to normalize vectors before comparing!



Outer Products

- Outer product of \mathbf{x} and \mathbf{y} :

$$\mathbf{x} \otimes \mathbf{y} = \mathbf{x}\mathbf{y}^T = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} = \begin{bmatrix} x_1 y_1 & x_1 y_2 & x_1 y_3 \\ x_2 y_1 & x_2 y_2 & x_2 y_3 \\ x_3 y_1 & x_3 y_2 & x_3 y_3 \\ x_4 y_1 & x_4 y_2 & x_4 y_3 \end{bmatrix}$$

- \mathbf{x} and \mathbf{y} can be of different lengths
- Result is a matrix

Linear independence

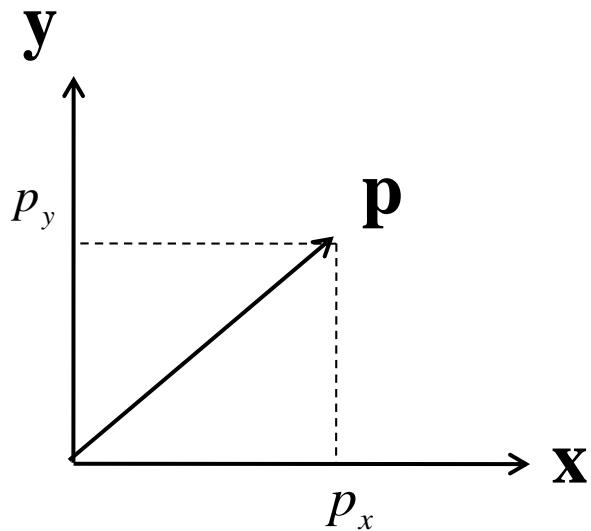
- Non-zero vectors $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ are linearly independent only if

$$a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + a_3\mathbf{x}_3 \neq 0$$

for any non-zero set of constants a_n

- We say a family of vectors is a linearly independent family if none of them can be written as a linear combination of finitely many other vectors in the family.

Linear independence & Orthogonality



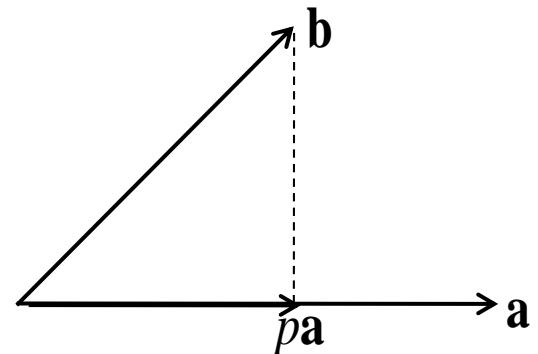
Inner product of orthogonal vectors is 0

$$\mathbf{x}^T \mathbf{y} = 0$$

Projections

- We want to find the value of p which minimizes the error $\|\mathbf{b} - p\mathbf{a}\|$, i.e.

$$\begin{aligned}\hat{p} &= \arg \min_p \|\mathbf{b} - p\mathbf{a}\| \\ &= \arg \min_p (\mathbf{b} - p\mathbf{a})^T (\mathbf{b} - p\mathbf{a}) \\ &= \arg \min_p (\mathbf{b}^T \mathbf{b} + p^2 \mathbf{a}^T \mathbf{a} - 2p\mathbf{a}^T \mathbf{b})\end{aligned}$$



Taking derivative and setting it to 0 :

$$0 = 2\hat{p}\mathbf{a}^T \mathbf{a} - 2\mathbf{a}^T \mathbf{b}$$

$$\hat{p} = \frac{\mathbf{a}^T \mathbf{b}}{\mathbf{a}^T \mathbf{a}}$$

- Hence, if \mathbf{a} is unit norm, the projection coefficient is equivalent to the inner product of \mathbf{a} and \mathbf{b}

Matrix Operations

1. Matrix Transpose, Conjugate Transpose

- $(\mathbf{A}^T)^T = \mathbf{A}$
- $(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$
- $(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$
- $(r\mathbf{A})^T = r\mathbf{A}^T$

2. Matrix Determinant

- Only exists for square matrices
- $\det(a\mathbf{X}) = a^n \det(\mathbf{X})$
- $\det(\mathbf{AB}) = \det(\mathbf{A})\det(\mathbf{B})$
- $\det(\mathbf{A}^T) = \det(\mathbf{A})$
- $\det(\mathbf{A}^{-1}) = (\det(\mathbf{A}))^{-1}$

Matrix Operations

3. Matrix arithmetic:

- Addition & Subtraction: Element-wise operations
- Matrix Multiplication:
 - $\mathbf{AB} \neq \mathbf{BA}$
- Vector Multiplication
- Scalar Multiplication

4. Matrix Inverse:

- $(\mathbf{A}^{-1})^{-1} = \mathbf{A}$
- $(\mathbf{A}^T)^{-1} = (\mathbf{A}^{-1})^T$

5. Matrix Rank

Matrix Operations

6. (Moore-Penrose) Pseudo-Inverse :

- The pseudoinverse A^+ of a matrix A is a generalization of the inverse matrix. The most widely known is the Moore-Penrose pseudoinverse.
- The pseudoinverse is defined and unique for all matrices whose entries are real or complex numbers.
- If the matrix A has dimensions $m \times n$, and is full rank, then use the left inverse if $m > n$, and the right inverse if $m < n$.
- Left inverse: $A_{\text{left}}^{-1} = (A^T A)^{-1} A^T$, i.e. $A_{\text{left}}^{-1} A = I_n$
- Right inverse: $A_{\text{right}}^{-1} = A^T (A A^T)^{-1}$, i.e. $A A_{\text{right}}^{-1} = I_m$

System of Linear Equations

- Suppose we have a set of linear equations such as the following example:

$$\begin{aligned}a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3\end{aligned}$$

In matrix form, we can write this as:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\mathbf{A} \mathbf{x} = \mathbf{b}$$

System of Linear Equations $\mathbf{Ax} = \mathbf{b}$

	Under-determined	Well defined	Over-determined
Equations vs. Unknowns	Less linearly independent equations than unknowns	As many linearly independent equations as unknowns	More linearly independent equations than unknowns
A	A is “fat”	A is square	A is “tall”
# of solutions	Usually infinitely many solutions	Usually one solution	Usually no solution
Typical solution	Minimum Norm solutions $\mathbf{x} = \mathbf{A}^T (\mathbf{A}\mathbf{A}^T)^{-1} \mathbf{b}$	<u>Exact</u> solution: $\mathbf{x} = \mathbf{A}^{-1} \mathbf{b}$	<u>Least squared error</u> solution: $\mathbf{x} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b}$

Eigen decomposition

- Any matrix \mathbf{A} represents a transformation operation on a vector

$$\mathbf{Ax} = \mathbf{x}'$$

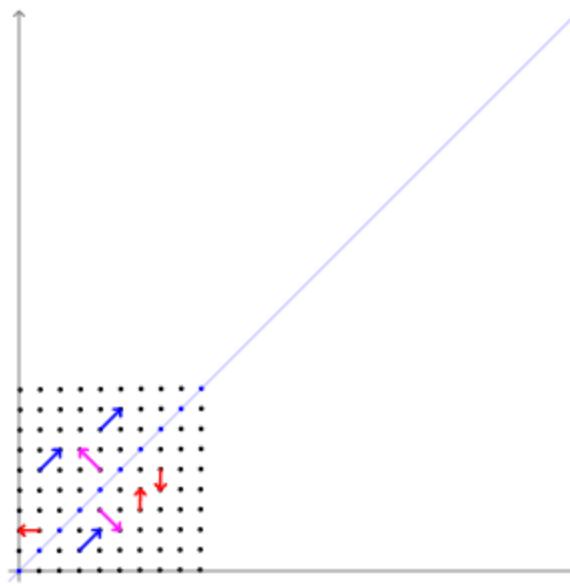
- For certain vectors, the transformation is merely a scale change

$$\mathbf{Ax} = \lambda \mathbf{x}$$

Eigen decomposition

$$\mathbf{Ax} = \lambda \mathbf{x}$$

- \mathbf{x} is the eigen vector of transformation \mathbf{A}
- λ is the corresponding eigen value



Determining the eigen decomposition

$$\mathbf{Ax} = \lambda \mathbf{x}$$

$$(\mathbf{A} - \lambda \mathbf{I})\mathbf{x} = 0$$

For a non-trivial solution

$$|\mathbf{A} - \lambda \mathbf{I}| = 0$$

This equation is called the characteristic equation.
Solve for the eigen values λ and hence obtain 'x's

Some properties of eigen decomposition

- Rank of \mathbf{A} = Number of non-zero eigen values of \mathbf{A} = Number of linearly independent eigen vectors
- Determinant of \mathbf{A} , $|\mathbf{A}| = \prod_{i=1}^{i=n} \lambda_i$
- Trace of $\mathbf{A} = \sum_{i=1}^{i=n} \lambda_i$

Some properties of eigen decomposition

- If \mathbf{A} is symmetric, then the eigen vectors are orthogonal
- If λ is an eigen value of \mathbf{A} ,
 - then $\frac{1}{\lambda}$ is an eigen value of \mathbf{A}^{-1}
 - $\lambda - k$ is an eigen value of $\mathbf{A} - k\mathbf{I}$
 - λ^m is an eigen value of \mathbf{A}^m

Eigen decomposition

- A matrix is
 - positive definite if all its eigen values

$$\lambda_i > 0$$

- positive semi-definite if

$$\lambda_i \geq 0$$

- negative definite if

$$\lambda_i < 0$$

- negative semi-definite if

$$\lambda_i \leq 0$$

Calculus

$$\nabla f(\mathbf{x}) = \text{gradient}(f(\mathbf{x})) = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

$$\frac{\partial \mathbf{x}^T \mathbf{c}}{\partial \mathbf{x}} = \frac{\partial \mathbf{c}^T \mathbf{x}}{\partial \mathbf{x}} = c$$

$$\frac{\partial \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = \mathbf{A}$$

Product Rule: $\frac{\partial(g(\mathbf{x}))^T h(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial(h(\mathbf{x}))^T}{\partial \mathbf{x}} g(\mathbf{x}) + \frac{\partial(g(\mathbf{x}))^T}{\partial \mathbf{x}} h(\mathbf{x})$

e.g.: $\frac{\partial \mathbf{x}^T \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = (\mathbf{A}^T + \mathbf{A})\mathbf{x}$ In this case ($g(\mathbf{x}) = \mathbf{x}, h(\mathbf{x}) = \mathbf{A}\mathbf{x}$)

Chain Rule: $\frac{\partial g(h(\mathbf{x}))}{\partial \mathbf{x}} = \frac{\partial g}{\partial h} \cdot \frac{\partial h}{\partial \mathbf{x}}$

Highly Recommended

- “Linear Algebra and its applications” – 4th Ed., Gilbert Strang
- The Matrix Cookbook
 - <http://matrixcookbook.com/>
- Gilbert Strang MIT video lectures
 - <http://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/video-lectures/>

Lagrange multipliers are used to solve constrained optimization problems. If we have a function $f(x, y)$ and we want to optimize (maximize or minimize) to find the maximum/minimum value. Here we are not allowed to consider all (x, y) while looking for this value. Instead, the (x, y) you can consider are constrained to lie on some curve or surface

Typical, Lagrange multipliers problem can be stated as follows:

Minimize (or maximize) $w = f(x, y, z)$ constrained by $g(x, y, z) = c$.

Lagrange multipliers solution: Local minima (or maxima) must occur at a critical point. This is a point where $\nabla f = \lambda \nabla g$, and $g(x, y, z) = c$.

Example Find the point on (called P) on the line $2x + y = 1$ such that the rectangle formed by P and the origin O has maximum area

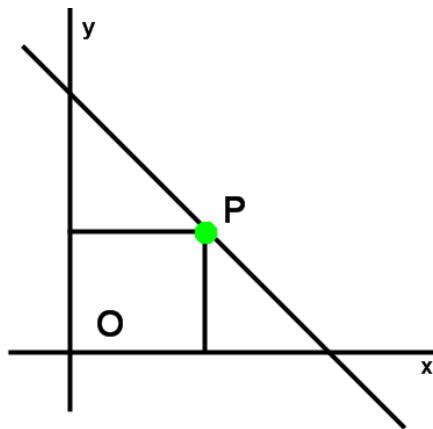


Figure 1:

Answer

We need to solve: $g(x, y) = x + 2y = 1$ and the constraint that maximizes $f(x, y) = xy$ (the area)

The gradients are: $\nabla g = \langle 1, 2 \rangle$ and $\nabla f = \langle y, x \rangle$

Lagrange multipliers: λ

We have 3 equations for 3 variables as follows:

$$y = \lambda$$

$$x = 2\lambda$$

$$x + 2y = 1$$

The solution is: $4\lambda = 1$

Thus, $y = \frac{1}{4}$ and $x = \frac{1}{2}$

We know this is a maximum because the maximum occurs either at a critical point or on the boundary. In this case, the boundary points are on the axes at $(1, 0)$ and $(0, 1/2)$, which gives a rectangle with area = 0.

Reminder No. 1: Uncorrelated vs. Independent

36-402, Advanced Data Analysis*

Last updated: 27 February 2013

A reminder of about the difference between two variables being uncorrelated and their being independent.

Two random variables X and Y are **uncorrelated** when their correlation coefficient is zero:

$$\rho(X, Y) = 0 \quad (1)$$

Since

$$\rho(X, Y) = \frac{\text{Cov}[X, Y]}{\sqrt{\text{Var}[X] \text{Var}[Y]}} \quad (2)$$

being uncorrelated is the same as having zero covariance. Since

$$\text{Cov}[X, Y] = E[XY] - E[X]E[Y] \quad (3)$$

having zero covariance, and so being uncorrelated, is the same as

$$E[XY] = E[X]E[Y] \quad (4)$$

One says that “the expectation of the product factors”. If $\rho(X, Y) \neq 0$, then X and Y are **correlated**.

Two random variables are **independent** when their joint probability distribution is the product of their marginal probability distributions: for all x and y ,

$$p_{X,Y}(x,y) = p_X(x)p_Y(y) \quad (5)$$

Equivalently¹, the conditional distribution is the same as the marginal distribution:

$$p_{Y|X}(y|x) = p_Y(y) \quad (6)$$

If X and Y are not independent, then they are **dependent**. If, in particular, Y is a function of X , then they always dependent²

*Thanks to Prof. Howard Seltman for suggestions.

¹Why is this equivalent?

²For the sake of mathematical quibblers: a *non-constant* function of X .

If X and Y are independent, then they are also uncorrelated. To see this, write the expectation of their product:

$$\mathbb{E}[XY] = \int \int xy p_{X,Y}(x,y) dx dy \quad (7)$$

$$= \int \int xy p_X(x) p_Y(y) dx dy \quad (8)$$

$$= \int x p_X(x) \left(\int y p_Y(y) dy \right) dx \quad (9)$$

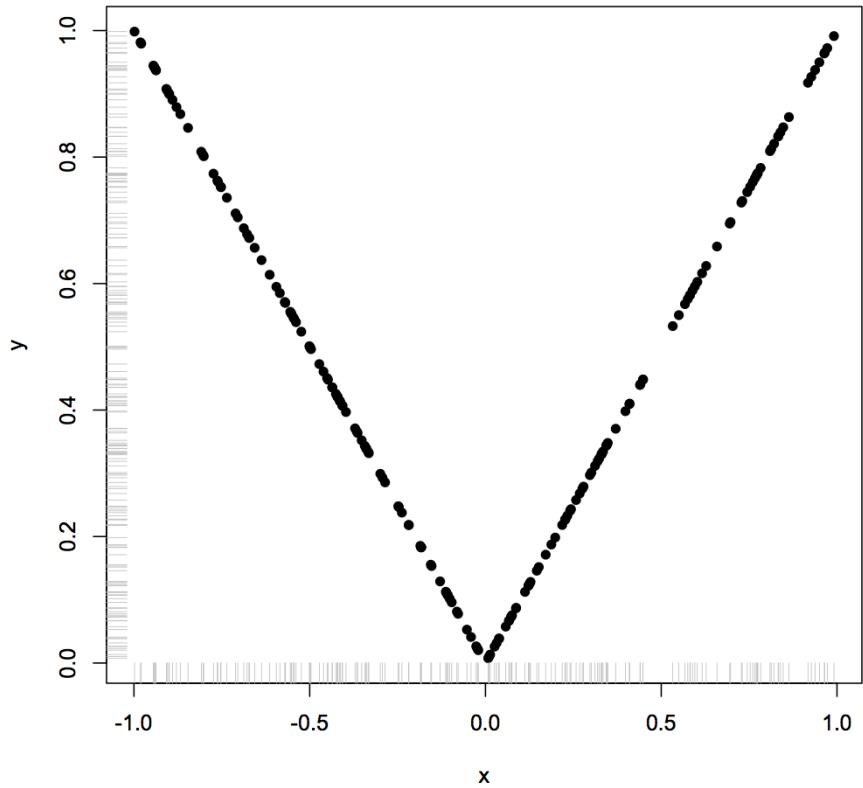
$$= \left(\int x p_X(x) dx \right) \left(\int y p_Y(y) dy \right) \quad (10)$$

$$= \mathbb{E}[X] \mathbb{E}[Y] \quad (11)$$

However, if X and Y are uncorrelated, then they can *still* be dependent. To see an extreme example of this, let X be uniformly distributed on the interval $[-1, 1]$. If $X \leq 0$, then $Y = -X$, while if X is positive, then $Y = X$. You can easily check for yourself that:

- Y is uniformly distributed on $[0, 1]$
- $\mathbb{E}[XY|X \leq 0] = \int_{-1}^0 -x^2 dx = -1/3$
- $\mathbb{E}[XY|X > 0] = \int_0^1 x^2 dx = +1/3$
- $\mathbb{E}[XY] = 0$ (*hint*: law of total expectation).
- The joint distribution of X and Y is not uniform on the rectangle $[-1, 1] \times [0, 1]$, as it would be if X and Y were independent (Figure 1).

The only general case when lack of correlation implies independence is when the joint distribution of X and Y is Gaussian.



```

x <- runif(200,min=-1,max=1)
y <- ifelse(x>0,x,-x)
plot(x,y,pch=16)
rug(x,side=1,col="grey")
rug(y,side=2,col="grey")

```

Figure 1: An example of two random variables which are uncorrelated but strongly dependent. The grey “rug plots” on the axes show the marginal distributions of the samples from X and Y .

Felix Juefei Xu

Pattern Recognition Theory

Recitation 2: Probability and Statistics

Topics To Be Covered

- Basic Probability Theory
 - Elementary Stuff
 - Bayes Rule
- Random Variables (RVs)
 - PDFs and CDFs
 - Mean and Variance
 - Commonly Used PDFs
- Joint Distributions (>1 RV)
- Conditional Probability Revisited
- MATLAB functions

The Basic Stuff

- Define probability of an event as $P(A)$

$$P(A) = \lim_{n \rightarrow \infty} \frac{n_A}{n}$$

- Axioms of probability
 - $0 \leq P(A) \leq 1$
 - $P(\text{Certain Event}) = 1, P(\text{Impossible Event}) = 0$
 - If A and B are **Mutually Exclusive** i.e.

$$P[A \cap B] = 0 \text{ then } P[A \cup B] = P[A] + P[B]$$

- A and B are **Independent Events** if $P(AB) = P(A)P(B)$

Conditional Probability

$$P(A|B) = \frac{P(AB)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

Bayes Rule

$$P(B) = P(B|A_1)P(A_1) + \dots + P(B|A_n)P(A_n)$$

Total Probability Rule

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{P(B)} = \frac{P(B|A_i)P(A_i)}{P(B|A_1)P(A_1) + \dots + P(B|A_n)P(A_n)}$$

Bayes Rule + Total Probability Rule

Random Variable Preliminaries

- An RV represents the probability of different events and hence takes on different values with probabilities that sum up to 1
- An RV can be Continuous, Discrete or Mixed
- Cumulative Distribution Function (CDF) – Non Decreasing Function

$$F_X(x) = P(X \leq x)$$

$$F(+\infty) = 1, F(-\infty) = 0$$

$$F(x_2) - F(x_1) = P(x_2 < x \leq x_1)$$

- Probability Density (Mass) Function (PDF or PMF)

$$f_X(x) = \frac{d}{dx}(F_X(x))$$

$$\int_{-\infty}^{+\infty} f_X(x) dx = 1$$

$$F_X(x) = \int_{-\infty}^x f_X(x) dx$$

$$F_X(x) = \sum_{x \leq x_i} f_X(x_i)$$

Mean and Variance

- Mean is also known as expected value or expectation

$$\mu = E(X) = \int_{-\infty}^{+\infty} x f_X(x) dx$$

Continuous RV

$$\sum_{-\infty}^{+\infty} x f_X(x)$$

Discrete RV

- Variance is second moment about mean

$$\sigma^2 = E[(X - E(X))^2] = E(X^2) - E^2(X)$$



$$\int_{-\infty}^{+\infty} (x - \mu)^2 f_X(x) dx$$

Continuous RV



$$\sum_{-\infty}^{+\infty} (x - \mu)^2 f_X(x)$$

Discrete RV

Properties of Mean and Variance

- Expectation is a linear operator
- $E(X + c) = E(X) + E(c) = E(X) + c$
- $E(cX) = cE(X)$
- $E(X + Y) = E(X) + E(Y)$
- $E(XY) = E(X)E(Y)$ only if X and Y are **uncorrelated** or **independent**
- $\text{var}(aX) = a^2\text{var}(X)$

Discrete Densities

Bernoulli

$$f_X(x) = x^p(1-x)^{(1-p)} \quad X = 0, 1$$

Binomial

$$P(X = k) = \binom{n}{k} p^k q^{n-k} \quad p + q = 1 \quad k = 0, 1, 2 \dots n$$

Geometric

$$P(X = k) = pq^{k-1} \quad k = 1, 2, 3 \dots \infty$$

Poisson

$$P(X = k) = e^{-\lambda} \frac{\lambda^k}{k!} \quad k = 0, 1, 2 \dots \infty$$

Continuous Densities

Uniform

$$\begin{aligned} f_X(x) &= \frac{1}{b-a} & a \leq x \leq b \\ &= 0 & \text{otherwise} \end{aligned}$$

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} \quad -\infty \leq x \leq +\infty$$

Normal

$$F_X(x) = \int_{-\infty}^x \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y-\mu)^2/2\sigma^2} dy \triangleq G\left(\frac{x-\mu}{\sigma}\right)$$

$$G(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-y^2/2} dy$$

Exponential

$$\begin{aligned} f_X(x) &= \lambda e^{-\lambda x} & x \geq 0 \\ &= 0 & \text{otherwise} \end{aligned}$$

Joint Distributions – Bivariate

$$F_{X,Y}(x, y) = P(X \leq x, Y \leq y)$$

CDF

$$f_{X,Y}(x, y) = \frac{\delta^2 F_{X,Y}(x, y)}{\delta x \delta y}$$

PDF

$$f_X(x) = \int_{-\infty}^{\infty} f_{X,Y}(x, y) dy$$

$$f_Y(y) = \int_{-\infty}^{\infty} f_{X,Y}(x, y) dx$$

Marginal PDFs

Joint Distributions – Bivariate

$$\text{cov}(X, Y) = E[(X - E(X))(Y - E(Y))] = E(XY) - E(X)E(Y)$$

Covariance

$$\rho_{X,Y} = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}$$

Correlation Coefficient

$$E(X, Y) = E(X)E(Y)$$

Uncorrelated

$$f_{X,Y}(x, y) = f_X(x)f_Y(y)$$

Independent

Joint Distributions – Multivariate

$$F_{\underline{X}}(\underline{x}) = F_{X_1, X_2 \dots X_n}(x_1, x_2 \dots x_n) = P(X_1 \leq x_1 \dots X_n \leq x_n) = \int_{-\infty}^{x_n} \dots \int_{-\infty}^{x_1} f_{X_1, X_2 \dots X_n}(x_1, x_2 \dots x_n) dx_1 \dots dx_n$$

$$F_{\underline{X}}(-\infty \dots -\infty) = 0 \quad F_{\underline{X}}(\infty \dots \infty) = 1$$

CDF

$$f_{\underline{X}}(\underline{x}) = P(X_1 = x_1 \dots X_n = x_n) = f_{X_1, X_2 \dots X_n}(x_1, x_2 \dots x_n) = \frac{dF_{\underline{X}}(\underline{x})}{d\underline{X}} = \frac{\delta^n F_{X_1, X_2 \dots X_n}(x_1, x_2 \dots x_n)}{\delta x_1 \dots \delta x_n}$$

$$f_{\underline{X}}(\underline{x}) \geq 0$$

$$\int_{-\infty}^{+\infty} \dots \int_{-\infty}^{+\infty} f_{X_1, X_2 \dots X_n}(x_1, x_2 \dots x_n) dx_1 \dots dx_n = 1$$

PDF

Joint Distributions – Multivariate

$$\underline{\mu} = E[\underline{X}] = E[X_1 \dots X_n]^T = [E[X_1] \dots E[X_n]]^T$$

Mean Vector

$$\underline{\Sigma} = \begin{bmatrix} cov(X_1, X_1) & cov(X_1, X_2) & \dots & cov(X_1, X_n) \\ cov(X_2, X_1) & cov(X_2, X_2) & \dots & cov(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ cov(X_n, X_1) & cov(X_n, X_2) & \dots & cov(X_n, X_n) \end{bmatrix} = \begin{bmatrix} \sigma_{X_1}^2 & cov(X_1, X_2) & \dots & cov(X_1, X_n) \\ cov(X_2, X_1) & \sigma_{X_2}^2 & \dots & cov(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ cov(X_n, X_1) & cov(X_n, X_2) & \dots & \sigma_{X_n}^2 \end{bmatrix}$$

$$\underline{\Sigma} = E[(\underline{X} - E(\underline{X}))(\underline{X} - E(\underline{X}))^T] = E[(\underline{X} - \underline{\mu})(\underline{X} - \underline{\mu})^T] = E[\underline{X}\underline{X}^T] - \underline{\mu}\underline{\mu}^T$$

Covariance Matrix

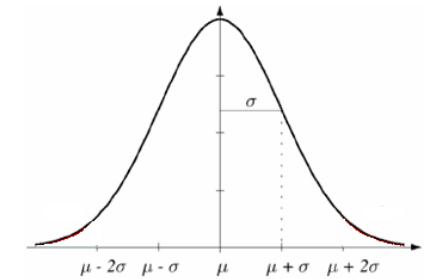
$$f_{X_1 \dots X_n}(x_1 \dots x_n) = \prod_i^n (f_{X_i}(x_i))$$

Independence

Gaussian (Normal) Distribution

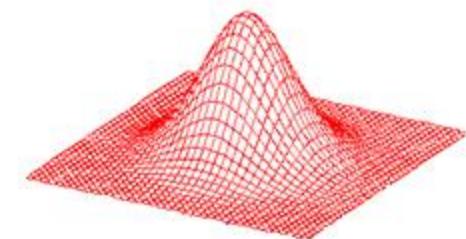
Univariate Normal Distribution

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$$



Multivariate Normal Distribution

$$f_{\underline{X}}(\underline{x}) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left[-\frac{(\underline{x} - \underline{\mu})^T \Sigma^{-1} (\underline{x} - \underline{\mu})}{2} \right]$$



If \underline{X} is $N(\underline{\mu}, \Sigma)$ then $\underline{Y} = \mathbf{A}\underline{X}$ is $N(\mathbf{A}\underline{\mu}, \mathbf{A}\Sigma\mathbf{A}^T)$

Conditional Probability Revisited

$$f_{X|Y}(x|y) = P(X = x|Y = y) = f_{X,Y}(x,y)/f_Y(y) = P(X = x, Y = y)/P(Y = y)$$

Bayes Rule

$$f(y|x) = f(x,y)/f(x)$$

$$f(x|y) = f(x,y)/f(y)$$

$$f(x,y) = f(x|y)f(y) = f(y|x)f(x)$$

Simplified Notation

$$f(x|y) = \frac{f(x,y)}{f(y)} = \frac{f(x|y)f(y)}{f(y)} = \frac{f(y|x)f(x)}{f(y)} = \frac{f(y|x)f(x)}{\int_{-\infty}^{\infty} f(y|x)f(x)dx}$$

The Grand Scheme

Useful MATLAB Functions

- rand/randn
- randperm
- pdf
- normpdf
- mvnpdf
- cdf
- erf, erfc, erfinv, erfcinv

References

- Useful Definitions and Results in Probability Theory - Notes By Prof. Vijaykumar Bhagavatula for Pattern Recognition
- Athanasios Papoulis, S. Unnikrishna Pillai, “Probability, Random Variables and Stochastic Processes,” TMH 4th edition, 2002
- Richard O. Duda, Peter E. Hart, David G. Stork, “Pattern Classification,” Wiley 2nd edition, 2007
- MATLAB Help

18-794 - Pattern Recognition Theory, Fall 2013

Notes On Probability, Statistics & MATLAB

September 16th, 2013

Abstract This is a set of notes that summarizes the concepts outlined in the recitation. These notes are meant to be a quick refresher of fundamental formulae and mathematical principles related to probability and statistics that will come in handy later on in the course. For a more complete understanding of these concepts, please read up on material mentioned in the references section. More notes and links related to these topics will be posted on blackboard.

Terminology: P always represents probability of an event, E represents expectation and A, B, C etc are used to denote events. Please do not get confused between scalars, vectors, matrices and Random Variables (RVs). The convention followed is that RVs are always capitalized (X) and are capitalized, underlined if they are random vectors (\underline{X}). Matrices are capitalized, in bold and have a bar below them ($\bar{\Sigma}$). Vectors are in lower case, not in bold and have a bar below them ($\underline{\mu}$), while scalars are simply represented in lower case, are not in bold and have no bar below them (s).

1 Basic Probability Theory

The probability of an event A can be determined using observations and the relative frequency approach so that we can finally assign the value $P(A) = \lim_{n \rightarrow \infty} \frac{n_A}{n}$ if we make n observations and the event A occurs n_A times. This is one of the oldest definitions of probability but is still worth keeping in mind.

The axioms of probability (which seem quite obvious) are:

1. $0 \leq P(A) \leq 1$
2. The probability of a certain event equals 1 and that of an impossible event equals 0
3. If the events A and B are mutually exclusive i.e. $P[A \cap B] = 0$ then $P[A \cup B] = P[A] + P[B]$

Many other theorems of probability follow from axioms of set theory. For example if A is an event and \bar{A} is its complement event, then using set theory it becomes easy to see that $P(\bar{A}) = 1 - P(A)$. Similarly using a Venn diagram it can be observed that $A \cup B = A + B - A \cap B$ and hence $P(A \cup B) = P(A) + P(B) - P(A \cap B)$ and if A and B are mutually exclusive, then $A \cap B = 0$ and hence we end up with axiom 3.

Conditional Probability and Bayes Theorem The conditional probability of an event A , given that an event B has occurred is given by

$$P(A|B) = \frac{P(AB)}{P(B)} = \frac{P(B|A)P(A)}{P(B)} \quad (1)$$

assuming of course that $P(A)$ and $P(B)$ are non zero. This rule, known as Bayes rule is widely used and almost any course at CMU that involve the use of probability start with a description of it. At this point it becomes necessary to go into the details of two more terms.

Two events A and B are said to be mutually exclusive if they cannot occur together at the same so that $P(AB) = 0$ and hence $P(A \cup B) = P(A) + P(B)$. Two events A and B are statistically independent if the

occurrence of A does not in any way affect the occurrence of B . This is a very powerful concept and makes life very simple when computing complex probabilistic events composed of the occurrence of several small events because it leads to the fact that $P(AB) = P(A)P(B)$. Using this in Bayes rule leads to some more results. If A and B are independent then we can observe that

$$P(A|B) = \frac{P(AB)}{P(B)} = \frac{P(B)P(A)}{P(B)} = P(A) \quad (2)$$

Total Probability Law The total probability law states that if $U = [A_1 \dots A_n]$ is a partition of S into n mutually exclusive events and B is an arbitrary event, then

$$P(B) = P(B|A_1)P(A_1) + \dots + P(B|A_n)P(A_n) \quad (3)$$

and hence using Bayes rule we can also determine that

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{P(B)} = \frac{P(B|A_i)P(A_i)}{P(B|A_1)P(A_1) + \dots + P(B|A_n)P(A_n)} \quad (4)$$

The simplest way of determining the probability of an event is to enumerate all possible ways the event can occur and then determine the total possible outcomes and simply take the ratio of the two. Here are a couple of examples to illustrate this and the use of Bayes rule.

Example 1 The birthday problem: Assume there are n people at a party (including you), what is the probability that (a) Two or more people share the same birthday (b) Someone in the group has the same birthday as you?

Answer 1 (a) To approach this problem it would be easier to determine the probability of the complementary event (A) and determine the probability of no two people sharing a birthday. Let us also use $N = 365$. Among n people, each person could have a birthday on any of the N days, hence the total outcomes possible are N^n . If no two people can share the same birthday, then the first person could have a birthday on any of N days, the second one on $N - 1$ days, the third on $N - 2$ and so on that the last one can have a birthday on only $N - (n - 1)$ days. So the $P(\text{No two people have same birthday}) = P(A) = \frac{N(N-1)\dots(N-n+1)}{N^n} = \prod_{i=1}^{n-1} (1 - \frac{i}{N})$. The probability of the complementary event is hence $P(\bar{A}) = 1 - P(A)$ and is the solution to (a). It can be simplified further using the property that $1 - x \simeq e^{-x}$.

(b) Now to obtain the probability of the event B (the event that someone in the group has the same birthday as you), it is again easier to compute $P(\bar{B})$ and look at the ways that no one in the group has the same birthday as you. For this to happen, each of the n people can have a birthday from among $N - 1$ days (all except yours), so $P(\bar{B}) = \frac{(N-1)(N-1)\dots(N-1)}{N^n} = \frac{(N-1)^n}{N^n} = (1 - \frac{1}{N})^n$, which can be approximated as $e^{-n/N}$. Hence, $P(B) = 1 - e^{-n/N}$.

Example 2 - Courtesy 10-701 Fall 2009 Course - John has to answer a True or False question in an exam. The probability that he knows the correct answer is p , and thus the probability that he does not know the correct answer is $1 - p$. Since there is no negative marking, he decides to guess in case he does not know the answer. With probability q he chooses the true option and $1 - q$ the false option. However, he is ignorant that the professor is biased when he decides which of the two options is the correct one. To be precise, the professor with probability δ assigns the correct answer to the true option and with $1 - \delta$ to the false one. John gives an answer to the question and he answers correctly! What is the chance that he actually knew the answer?

Answer 2 $P(\text{John knew answer}|\text{He answers correctly}) = P(\text{John knew answer and answers correctly})/P(\text{John answers correctly})$. Let the event John knew answer = K , the event John gets answer correct = C and the event John guessed = \bar{K} . Hence what we want is to find $P(K|C) = P(KC)/P(C)$. Now the numerator $P(KC) = p$. The denominator is more complex. $P(C) = P(C|K)P(K) + P(C|\bar{K})P(\bar{K})$. Now again,

$P(C|\bar{K}) = P(T_J, T_P|\bar{K}) + P(\bar{T}_J, \bar{T}_P|\bar{K})$ where T_J is the event that John chooses to guess True and T_P is the event that the professor also chooses to set the correct answer as True.

So finally we obtain $P(K|C) = P(KC)/P(C) = \frac{p}{p+(1-p)[q\delta+(1-q)(1-\delta)]}$.

2 Random Variables, PDFs, CDFs etc

A random variable (RV) takes on different values with probabilities that sum up to 1. More formally, to an experiment specified by S, to each outcome i of this experiment we assign a probability given by $X(i)$ where X is the random variable (always represented by a capital letter) and x represents a scalar value that X can take on. For a vector of RVs we use \underline{X} . An RV can be continuous, discrete or mixed. Two fundamental functions associated with an RV are the CDF and PDF.

The cumulative distribution function (CDF) is defined as

$$F_X(x) = P(X \leq x) \quad (5)$$

Properties of the CDF are:

$$F(+\infty) = 1, F(-\infty) = 0 \quad (6)$$

and

$$F(x_2) - F(x_1) = P(x_2 < x \leq x_1) \quad (7)$$

It must also be noted that the CDF of an RV is a non decreasing function.

The probability density (or mass) function (PDF - for continuous RVs or PMF - for discrete RVs) is the derivative of the CDF and is defined as

$$f_X(x) = \frac{d}{dx}(F_X(x)) \quad (8)$$

Thus, for a continuous RV we have that

$$F_X(x) = \int_{-\infty}^x f_X(x) dx \quad (9)$$

and for a discrete RV

$$F_X(x) = \sum_{x \leq x_i} f_X(x_i) \quad (10)$$

Properties of the PDF are:

$$f_X(x) \geq 0 \quad (11)$$

and

$$\int_{-\infty}^{+\infty} f_X(x) dx = 1 \quad (12)$$

The PDF or PMF must sum up to 1, usually some constant is placed outside the integral or summation sign to ensure this as will be seen later when we go over commonly used PDFs.

Mean and Variance The mean of a PDF is called expectation $E(X)$. It is simply obtained as

$$\mu = E(X) = \int_{-\infty}^{+\infty} x f_X(x) dx \quad (13)$$

for a continuous RV and as

$$\sum_{-\infty}^{+\infty} x f_X(x) \quad (14)$$

for a discrete RV.

The variance σ^2 of an RV is the second moment about the mean and is defined as

$$\sigma^2 = E[(X - E(X))^2] = E(X^2) - E^2(X) = \int_{-\infty}^{+\infty} (x - \mu)^2 f_X(x) dx \quad (15)$$

for a continuous RV and as

$$\sum_{-\infty}^{+\infty} (x - \mu)^2 f_X(x) \quad (16)$$

for a discrete RV. The square root of variance is called std. deviation (σ).

Properties of Mean and Variance Some of the most important properties of the Expectation operator and Variance are:

1. $E(X + c) = E(X) + E(c) = E(X) + c$
2. $E(cX) = cE(X)$
3. $E(X + Y) = E(X) + E(Y)$
4. $E(XY) \neq E(X)E(Y)$ unless X and Y are uncorrelated
5. $var(aX) = a^2 var(X)$

Commonly Used PDFs The following PDFs are the ones you will most frequently encounter and it will be beneficial to remember their forms.

Please note that the mean of an RV X is not the value of X that maximizes its PDF ($f_X(x)$). The value of X which does this is known as the mode and is the most probable value of X .

1. Bernoulli

$$f_X(x) = x^p(1-x)^{(1-p)} \quad X = 0, 1 \quad (17)$$

which is simply $P(X = 0) = p$ and $P(X = 1) = q = (1-p)$. $E(X) = p$ and $var(X) = p(1-p)$. This distribution simply assigns probability of success ($X = 1$) as p and probability of failure ($X = 0$) as $q = (1-p)$.

2. Binomial - Repeated Bernoulli trials result in a binomial distribution who's PDF gives a way of calculating the probability of k successes in n trials.

$$P(X = k) = \binom{n}{k} p^k q^{n-k} \quad p + q = 1 \quad k = 0, 1, 2, \dots, n \quad (18)$$

$E(X) = np$ and $var(X) = npq = np(1-p)$.

3. Geometric - X is said to be a geometric RV if

$$P(X = k) = pq^{k-1} \quad k = 1, 2, 3, \dots, \infty \quad (19)$$

This measures the probability of getting first success on the k -th trial. $E(X) = \frac{1}{p}$ and $var(X) = \frac{1-p}{p^2}$.

4. Poisson - X is a Poisson RV with parameter λ if X takes on the values $0, 1, 2, \dots, \infty$ with

$$P(X = k) = e^{-\lambda} \frac{\lambda^k}{k!} \quad k = 0, 1, 2, \dots, \infty \quad (20)$$

$E(X) = \lambda$ and $var(X) = \lambda$.

5. Uniform - X is said to be uniformly distributed in (a, b) $-\infty < a < b < +\infty$ if

$$\begin{aligned} f_X(x) &= \frac{1}{b-a} \quad a \leq x \leq b \\ &= 0 \quad \text{otherwise} \end{aligned} \tag{21}$$

$$E(X) = \frac{a+b}{2} \text{ and } \text{var}(X) = \frac{(b-a)^2}{12}.$$

6. Normal or Gaussian - The most common distribution you will use. A normal RV X is normally distributed with parameters μ and σ^2 and is represented as $X \sim N(\mu, \sigma^2)$. The PDF of a univariate normal distribution is given as.

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} \quad -\infty \leq x \leq +\infty \tag{22}$$

The CDF of such a distribution is given by

$$F_X(x) = \int_{-\infty}^x \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y-\mu)^2/2\sigma^2} dy \triangleq G\left(\frac{x-\mu}{\sigma}\right) \tag{23}$$

where the function

$$G(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-y^2/2} dy \tag{24}$$

is available in tabulated form and represents the area under a standard normal curve $X \sim N(0,1)$. To determine the area under a normal variable (Z) with different mean and variance, transform the variable into standard normal form by subtracting its mean and dividing by its std devn. and then use the normal table on it. There are other representations that are used to determine the area under a normal curve such as the error function ($\text{erf}(x)$) and the Q function ($Q(x)$).

The error function is defined as follows

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-y^2} dy \tag{25}$$

Similarly there is a complementary error function $\text{erfc}(x)$ defined as

$$\text{erfc}(x) = 1 - \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-y^2} dy \tag{26}$$

These are the same functions used in MATLAB. It is clearly different from $G(x)$ and the relationship between the two is that

$$G(x) = \frac{1}{2}[1 + \text{erf}(x/\sqrt{2})] \tag{27}$$

The Q function is defined as

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-y^2/2} dy = 1 - G(x) \tag{28}$$

Hence the relationship between $Q(x)$ and $\text{erf}(x)$ is

$$Q(x) = \frac{1}{2}[1 - \text{erf}(x/\sqrt{2})] \tag{29}$$

7. Exponential - An RV X is exponentially distributed with parameter λ if its density function is given by

$$\begin{aligned} f_X(x) &= \lambda e^{-\lambda x} \quad x \geq 0 \\ &= 0 \quad \text{otherwise} \end{aligned} \tag{30}$$

$$E(X) = \frac{1}{\lambda} \text{ and } \text{var}(X) = \frac{1}{\lambda^2}.$$

The first 4 distributions on the list are discrete while the next 3 are continuous.

3 Joint Distributions

Now we need to address the joint PDF of more than 1 RV. We start with the bivariate case (joint PDF of 2 RVs) and can extend this to deal with the multivariate case (many RVs). The joint CDF of 2 RVs X and Y is given by

$$F_{X,Y}(x,y) = P(X \leq x, Y \leq y) \quad (31)$$

The joint PDF of the two RVs is given by

$$f_{X,Y}(x,y) = \frac{\delta^2 F_{X,Y}(x,y)}{\delta x \delta y} \quad (32)$$

The marginal PDFs of X and Y are given by

$$f_X(x) = \int_{-\infty}^{\infty} f_{X,Y}(x,y) dy \quad (33)$$

$$f_Y(y) = \int_{-\infty}^{\infty} f_{X,Y}(x,y) dx \quad (34)$$

These equations were simply obtained by marginalizing out one of the variables by integrating it over its full range of values. The same trick can be applied to understand how to obtain $F_X(x) = P(X \leq x)$ or $F_Y(y) = P(Y \leq y)$ from the joint distribution.

X and Y are SI (Statistically Independent) if

$$f_{X,Y}(x,y) = f_X(x)f_Y(y) \quad (35)$$

This makes life easy when applying Bayes rule (as shown earlier). There is another closely related concept called uncorrelated variables. X and Y are uncorrelated if

$$E(X,Y) = E(X)E(Y) \quad (36)$$

It must be noted that SI does imply uncorrelated variables, but not vice versa i.e. two uncorrelated variables may not be SI. SI is the more powerful concept and is harder to prove. However, if X and Y are both normal RVs, and are uncorrelated, then they will be SI too. This is a property of only normal RVs and can not be applied in general.

Now it becomes important to understand the terms correlation, covariance etc. The covariance between 2 RVs X and Y is defined as

$$\text{cov}(X,Y) = E[(X - E(X))(Y - E(Y))] = E(XY) - E(X)E(Y) \quad (37)$$

For uncorrelated variables, $\text{cov}(X,Y) = 0$. Can you see why? Now the term correlation between 2 RVs X and Y is defined by the correlation coefficient ($\rho_{X,Y}$)

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} \quad (38)$$

$-1 \leq \rho_{X,Y} \leq 1$. A value of 1 implies a strong positive correlation, a value of 0 means no correlation (uncorrelated and maybe SI) and a value of -1 implies strong negative correlation between the RVs X and Y .

If we now have n RVs $X_1, X_2, X_3 \dots X_n$ we represent this multivariate random vector as a column vector $\underline{X} = [X_1 \ X_2 \ X_3 \ \dots \ X_n]^T$. Now the following become clear:

$$F_{\underline{X}}(\underline{x}) = F_{X_1, X_2, \dots, X_n}(x_1, x_2, \dots, x_n) = P(X_1 \leq x_1, \dots, X_n \leq x_n) = \int_{-\infty}^{x_n} \dots \int_{-\infty}^{x_1} f_{X_1, X_2, \dots, X_n}(x_1, x_2, \dots, x_n) dx_1 \dots dx_n \quad (39)$$

$$F_{\underline{X}}(-\infty \dots -\infty) = 0 \quad F_{\underline{X}}(\infty \dots \infty) = 1 \quad (40)$$

$$f_{\underline{X}}(\underline{x}) = P(X_1 = x_1 \dots X_n = x_n) = f_{X_1, X_2 \dots X_n}(x_1, x_2 \dots x_n) = \frac{dF_{\underline{X}}(\underline{x})}{d\underline{X}} = \frac{\delta^n F_{X_1, X_2 \dots X_n}(x_1, x_2 \dots x_n)}{\delta x_1 \dots \delta x_n} \quad (41)$$

$$f_{\underline{X}}(\underline{x}) \geq 0 \quad (42)$$

$$\int_{-\infty}^{+\infty} \dots \int_{-\infty}^{+\infty} f_{X_1, X_2 \dots X_n}(x_1, x_2 \dots x_n) dx_1 \dots dx_n = 1 \quad (43)$$

$$\underline{\mu} = E[\underline{X}] = E[X_1 \dots X_n]^T = [E[X_1] \dots E[X_n]]^T \quad (44)$$

Thus, the bivariate case is a simple case of the multivariate case with $\underline{X} = [X_1 \ X_2]^T$. In the univariate case the mean is a scalar, now it is a vector ($\underline{\mu}$), similarly we now need to define a covariance matrix which is analogous to the variance term in the univariate case. This covariance matrix (Σ) stores the covariance between each of the n RVs and is defined as follows.

$$\Sigma = \begin{bmatrix} cov(X_1, X_1) & cov(X_1, X_2) & \dots & cov(X_1, X_n) \\ cov(X_2, X_1) & cov(X_2, X_2) & \dots & cov(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ cov(X_n, X_1) & cov(X_n, X_2) & \dots & cov(X_n, X_n) \end{bmatrix} = \begin{bmatrix} \sigma_{X_1}^2 & cov(X_1, X_2) & \dots & cov(X_1, X_n) \\ cov(X_2, X_1) & \sigma_{X_2}^2 & \dots & cov(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ cov(X_n, X_1) & cov(X_n, X_2) & \dots & \sigma_{X_n}^2 \end{bmatrix} \quad (45)$$

which can be expressed in matrix form as

$$\Sigma = E[(\underline{X} - E(\underline{X}))(\underline{X} - E(\underline{X}))^T] = E[(\underline{X} - \underline{\mu})(\underline{X} - \underline{\mu})^T] = E[\underline{X}\underline{X}^T] - \underline{\mu}\underline{\mu}^T \quad (46)$$

This covariance matrix is going to be a recurring topic in this course. It is an $n \times n$ square matrix and is positive-semidefinite. Its diagonal entries are simply the variances of the variables X_1, X_2, \dots, X_n respectively while its non-diagonal entries are the covariance terms. It is clear that it is also a symmetric matrix as $cov(X_1, X_2) = cov(X_2, X_1)$.

As with the bivariate case it can be shown that n RVs are SI if

$$f_{X_1 \dots X_n}(x_1 \dots x_n) = \prod_i^n (f_{X_i}(x_i)) \quad (47)$$

It becomes important to see what the multivariate normal distribution looks like. If \underline{X} is a multivariate normal RV it is represented as $\underline{X} \sim N(\underline{\mu}, \Sigma)$ and its PDF is now given by the equation below.

$$f_{\underline{X}}(\underline{x}) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left[-\frac{-(\underline{x} - \underline{\mu})^T \Sigma^{-1} (\underline{x} - \underline{\mu})}{2}\right] \quad (48)$$

Another property is worth mentioning at this stage. If \underline{X} is $N(\underline{\mu}, \Sigma)$ then $\underline{Y} = \underline{A}\underline{X}$ is $N(\underline{A}\underline{\mu}, \underline{A}\Sigma\underline{A}^T)$.

This property becomes useful when trying to diagonalize a covariance matrix. The process is called whitening and will be covered again in class. Let us assume we have a random vector \underline{X} that is $N(\underline{\mu}, \Sigma)$ i.e. its covariance matrix is Σ . We obtain the eigenvectors of the covariance matrix and stack them columnwise in matrix \underline{V} and store the eigenvalues along the diagonal of matrix $\underline{\Lambda}$ so $\Sigma = \underline{V}\underline{\Lambda}\underline{V}^T$, $\underline{\Lambda}^T = \underline{\Lambda}$ and $\underline{V}^T \underline{V} = \underline{I}$.

Now we generate a new random vector $\underline{Y} = \underline{A}_w^T \underline{X}$ where $\underline{A}_w = \underline{V}\underline{\Lambda}^{-1/2}$. This will lead to \underline{Y} being $N(\underline{A}_w^T \underline{\mu}, \underline{A}_w^T \Sigma \underline{A}_w) = N(\underline{A}_w^T \underline{\mu}, \underline{\Lambda}^{-1/2} \underline{V}^T \underline{V} \underline{\Lambda} \underline{V}^T \underline{V} \underline{\Lambda}^{-1/2}) = N(\underline{A}_w^T \underline{\mu}, \underline{\Lambda}^{-1/2} \underline{\Lambda} \underline{\Lambda}^{-1/2}) = N(\underline{A}_w^T \underline{\mu}, \underline{I})$.

4 Conditional Probability Revisited

We now revisit Bayes theorem and apply it to PDFs and CDFs. Let X and Y be two continuous RVs, then the conditional PDF $f_{X|Y}(x|y)$ is given by

$$f_{X|Y}(x|y) = P(X = x|Y = y) = f_{X,Y}(x,y)/f_Y(y) = P(X = x, Y = y)/P(Y = y) \quad (49)$$

Suppressing the subscripts for convenience, we can express this above rule in many ways

$$f(x|y) = f(x,y)/f(y) \quad (50)$$

$$f(y|x) = f(x,y)/f(x) \quad (51)$$

$$f(x,y) = f(x|y)f(y) = f(y|x)f(x) \quad (52)$$

and hence

$$f(x|y) = \frac{f(x,y)}{f(y)} = \frac{f(x|y)f(y)}{f(y)} = \frac{f(y|x)f(x)}{f(y)} = \frac{f(y|x)f(x)}{\int_{-\infty}^{\infty} f(y|x)f(x)dx} \quad (53)$$

The last part of the above formula was obtained by marginalizing the joint PDF.

If RV Y is being determined by observing RV X then we use the notation:

$f_Y(y)$ - a priori probability (or prior)

$f_{X|Y}(x|y)$ - Likelihood

$f_{Y|X}(y|x)$ - a posteriori probability (or posterior)

5 Useful MATLAB Functions

The following are useful MATLAB functions when dealing with PDFs, CDFs etc:

1. rand/randn - The rand/randn function generates arrays of random numbers whose elements are uniformly/normally distributed in the interval $(0,1)/N(0,1)$.
2. randperm - p = randperm(n) returns a random permutation of the integers 1:n.
3. pdf - pdf('name',X,A1,A2,A3) returns a matrix of densities, where 'name' is a string containing the name of the distribution. X is a matrix of values, and A1, A2, and A3 are matrices of distribution parameters. Depending on the distribution, some of the parameters may not be necessary. Look up all the different PDFs that can be generated using MATLAB help. One such example is p = pdf('Normal',-2:2,0,1).
4. normpdf - normpdf(X,MU,SIGMA) computes the normal pdf at each of the values in X using the corresponding parameters in MU and SIGMA. X, MU, and SIGMA can be vectors, matrices, or multidimensional arrays that all have the same size. A scalar input is expanded to a constant array with the same dimensions as the other inputs. The parameters in SIGMA must be positive.
5. mvnpdf - y=mvnpdf(X) returns the n-by-1 vector y, containing the probability density of the multivariate normal distribution with zero mean and identity covariance matrix, evaluated at each row of the n-by-d matrix X. Rows of X correspond to observations and columns correspond to variables or coordinates.

y = mvnpdf(X,mu) returns the density of the multivariate normal distribution with mean mu and identity covariance matrix, evaluated at each row of X. mu is a 1-by-d vector, or an n-by-d matrix. If mu is a matrix,

the density is evaluated for each row of X with the corresponding row of mu. mu can also be a scalar value, which mvnpdf replicates to match the size of X.

y = mvnpdf(X,mu,SIGMA) returns the density of the multivariate normal distribution with mean mu and covariance SIGMA, evaluated at each row of X. SIGMA is a d-by-d matrix, or an d-by-d-by-n array, in which case the density is evaluated for each row of X with the corresponding page of SIGMA, i.e., mvnpdf computes y(i) using X(i,:) and SIGMA(:,:,i). Specify [] for mu to use its default value when you want to specify only SIGMA. If X is a 1-by-d vector, mvnpdf replicates it to match the leading dimension of mu or the trailing dimension of SIGMA.

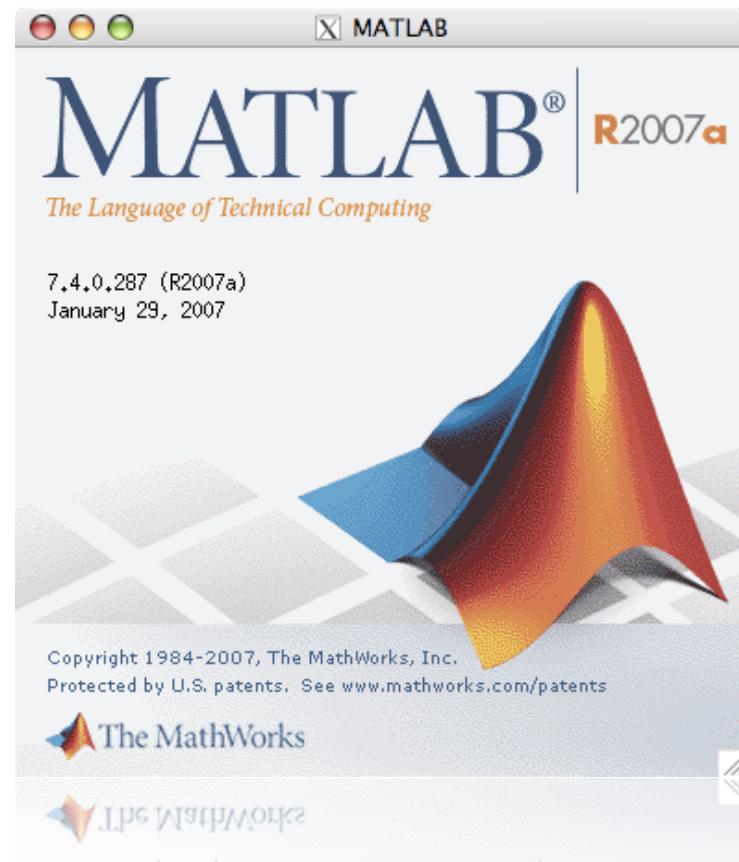
6. cdf - P = cdf('name',X,A1,A2,A3) returns a matrix of probabilities, where name is a string containing the name of the distribution, X is a matrix of values, and A, A2, and A3 are matrices of distribution parameters. Depending on the distribution, some of these parameters may not be necessary. Vector or matrix inputs for X, A1, A2, and A3 must have the same size, which is also the size of P. A scalar input for X, A1, A2, or A3 is expanded to a constant matrix with the same dimensions as the other inputs. Look up all the different PDFs that can be generated using MATLAB help. One such example is p = cdf('Normal',-2:2,0,1).

7. erf, erfc, erfinv, erfcinv - All related to computing area under the normal curve. See MATLAB help for syntax and how to use these functions.

Hopefully these notes served as a quick revision of concepts related to probability. More details can be found in the references mentioned.

References

- [1] Useful Definitions and Results in Probability Theory - Notes By Prof. Vijaykumar Bhagavatula for Pattern Recognition
- [2] Athanasios Papoulis, S. Unnikrishna Pillai, "Probability, Random Variables and Stochastic Processes," *TMH 4th edition*, 2002
- [3] Richard O. Duda, Peter E. Hart, David G. Stork, "Pattern Classification," *Wiley 2nd edition*, 2007
- [4] MATLAB Help



Matlab Tutorial

Joseph E. Gonzalez

What Is Matlab?

- MATrix LABoratory
 - Interactive Environment
 - Programming Language
- Invented in Late 1970s
 - Cleve Moler chairman CSD Univ New Mexico

Why we use it?

- Fast Development
- Debugging
- Mathematical Libraries
- Documentation
- Tradition
- Alternatives: Mathematica, R, Java? ML?...

Details

- Language
 - Like C and Fortran
 - Garbage Collected
- Interface
 - Interactive
 - Apple, Windows, Linux (Andrew)
 - Expensive (“Free” for you)

Matlab^ZLanguage

Nap Time

- This is a comment
 - $((1+2)^3 - 2^2 - 1)/2$
 - 2
- Use ; to suppress output (scripts and functions)
 - $((1+2)^3 - 2^2 - 1)/2;$

- Assignment with equality
 - `a = 5;`
 - No Output
- Logical test like `>`, `<`, `>=`, `<=`, `~=`
- `a == 6`

- Short Circuited Logic
 - `true || (slow_function)`
 - `I % Evaluates Quickly`
 - `true | (slow_function)`
 - `I % Evaluate slowly`
- Matrix logic

- A simple array
 - [1 2 3 4 5]
 - 1 2 3 4 5
- [1,2,3,4,5]
 - 1 2 3 4 5

- All the following are equivalent
 - [1 2 3; 4 5 6; 7 8 9]
 - [1,2,3; 4,5,6; 7,8,9]
 - [[1 2 3; 4 5 6] [3; 6; 9]]
 - [[1 2 3; 4 5 6]; [7 8 9]]

- Creating all ones, zeros, or identity matrices
 - `zeros(rows, cols)`
 - `ones(rows, cols)`
 - `eye(rows)`
- Creating Random matrices

- Make a matrix

- $A = [1 \ 2 \ 3; 4 \ 5 \ 6; 7 \ 8 \ 9]$

- | | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

- Access Individual Elements

Array and Matrix
Indices Start at 1
not 0.
(Fortran)

- Make a matrix
 - $A = [1 \ 2 \ 3; 4 \ 5 \ 6; 7 \ 8 \ 9]$
 -
 - 1 2 3
 - 4 5 6
 - 7 8 9
- Access Individual Elements

- Make a matrix
 - $A = [1 \ 2 \ 3; 4 \ 5 \ 6; 7 \ 8 \ 9]$
 - | | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
- Access Individual Elements
 - $A(1, \text{logical}([1,0,1]))$
 - 1 3
 - $A(\text{mod}(A, 2) == 0)'$
 - 4 2 8 6

- Make a matrix

- $A = [1 \ 2 \ 3; 4 \ 5 \ 6; 7 \ 8 \ 9]$

- $$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

- $A + 2 * (A / 4)$

- Make a matrix

- $A = [1 \ 2 \ 3; 4 \ 5 \ 6; 7 \ 8 \ 9]$

- $$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

- Transpose

- A'

- Matrix Multiplication
 - $A * A$ % Equivalent to A^2
 - $\begin{matrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{matrix}$
 - Element by Element Multiplication
 - $A .* A$ % equivalent to $A.^2$

- Matrix Multiplication

- $\text{inv}(A) \% A^{-1}$

- $1.0e+16 * \begin{matrix} 0.3153 & -0.6305 & 0.3153 \\ -0.6305 & 1.2610 & -0.6305 \\ 0.3153 & -0.6305 & 0.3153 \end{matrix}$

- Solving Systems

- Define some variables and store a function in f
 - $c = 4;$
 - $f = @x \ x + c;$
 - $f(3)$
 - 7

- Like arrays but can have different types
 - `x = {'hello', 2, 3};`
 - `x{1}`
 - ‘hello’
 - `x{2}`
 - 2

- Provide a convenient tool to organize variables
- Create Structs on the fly
 - point.x = 3;
 - point.y = 4;
 - point

Objects

- You can make objects but ...
 - you won't need them.
 - I don't know how to make them.
 - most people don't use them

- If Statements

- ```
c = rand();
```

```
if (c > .5) %% conditional
```

```
 disp('Greater than');
```

```
elseif (c < .5)
```

```
 disp('Less Than');
```

```
else
```

```
 disp('Equal to');
```

```
end
```

- If Statements
- count = 0;
- for i = 1:length(data)  
    count = count + ...  
        (data(i,1) == 4 && data(i,3) == 2);  
    end
- Avoid using for loops

# Scripts vs Functions

- Scripts
  - List of commands that operate on the current workspace
- Functions
  - List of commands that operate in a separate workspace
  - Takes in values from current workspace and returns values
  - Function name = filename
  - Can have additional (hidden) functions

**my\_script.m**

```
disp(["x^2", ...
 num2str(x^2)]);
y = x^2
```

**my\_fun.m**

```
function [y,x] =
my_fun(x)
disp(["x^2", ...
 num2str(x^2)]);
y=x^2
```

Functions must have  
same name as file.

### my\_script.m

```
y = x^2;
x = x + 3;
```

- $x=2$ ; my\_script;

- $x$

- 5

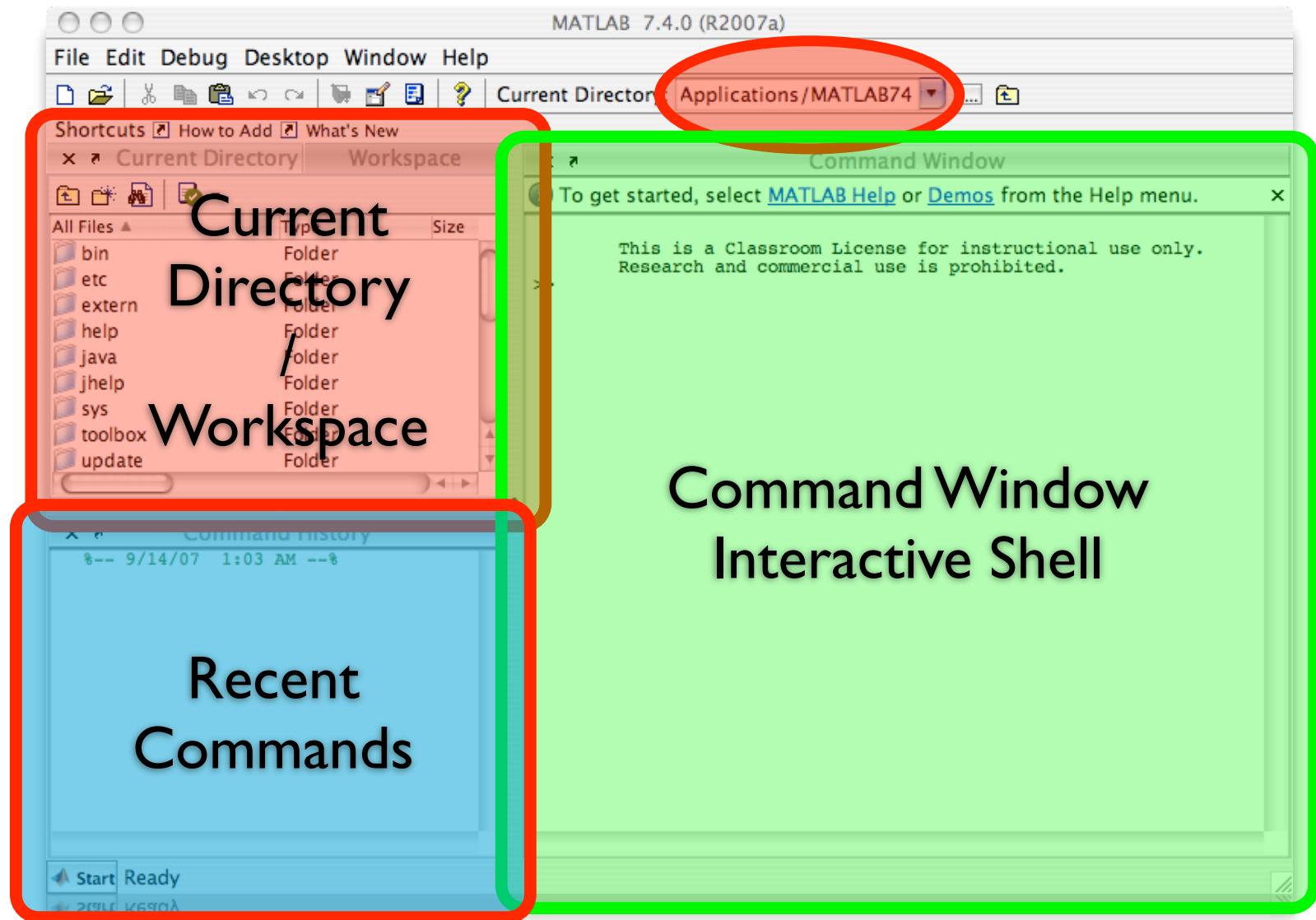
### my\_fun.m

```
function [y, x] =
my_fun(x)
y=x^2;
x = x + 3;
```

```
>> x=2; [y, xp] = my_fun(x);
>> x
ans: 2
>> y
ans: 4
>> xp
ans: 5
```

# Things to Know

- Useful operators
  - `>`, `<`, `>=`, `<=`, `==`, `&`, `|`, `&&`, `||`, `+`, `-`, `/`, `*`, `^`, `...`, `./`,  
`'`, `.*`, `.^`, `\`
- Useful Functions
  - `sum`, `mean`, `var`, `not`, `min`, `max`, `find`, `exists`,  
`clear`, `clc`, `pause`, `exp`, `sqrt`, `sin`, `cos`, `reshape`,  
`sort`, `sortrows`, `length`, `size`, `length`, `setdiff`,  
`ismember`, `isempty`, `intersect`, `plot`, `hist`, `title`,  
`xlabel`, `ylabel`, `legend`, `rand`, `randn`, `zeros`,  
`ones`, `eye`, `inv`, `diag`, `ind2sub`, `sub2ind`, `find`,



# Command Console

- Like a linux shell
  - Folder Based
  - Native Directories
  - ls, cd, pwd
- Use tab key to auto complete
- Use up arrow for last command

## ls : List Directory Contents

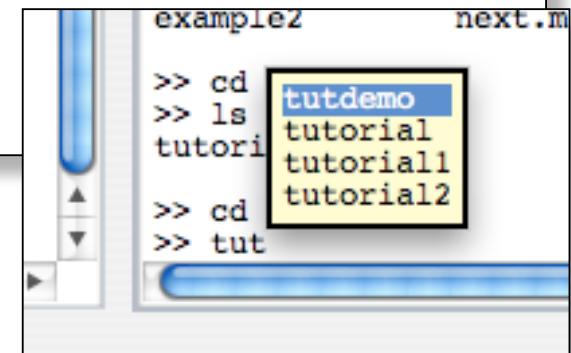
```
> ls
README.txt example3 tutorial.m
example1 my_function.m tutorial1.m
example2 next.m tutorial2.m
```

## pwd : View Current directory

```
ans =
/Users/jegonzal/tutorial
```

## cd : Change Directory

```
>> pwd
ans =
/Users/jegonzal
```



- Get help on a function
  - `help <function name>`
- List names of variables in the environment
  - `whos`
- Clear the environment
  - `clear`

# Folders

- Help organize your programs
- Can only call functions and scripts in:
  - The present working directory (`pwd`)
  - The Matlab path (`path`)
- Call function

```
>> my_script
>> y = my_function(x)
```

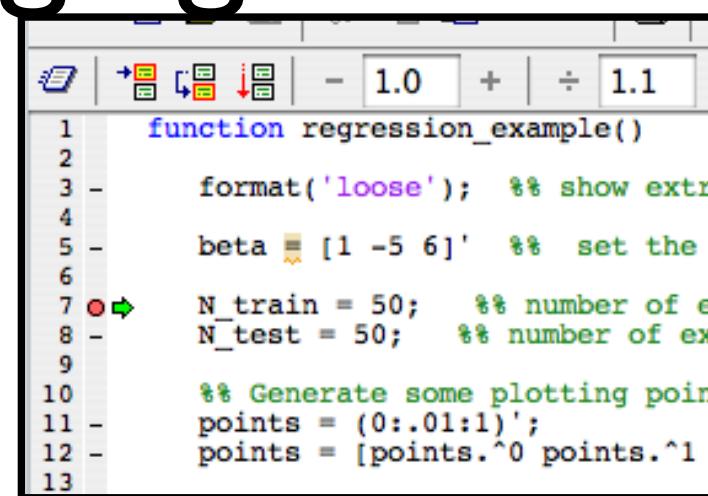
Typing name



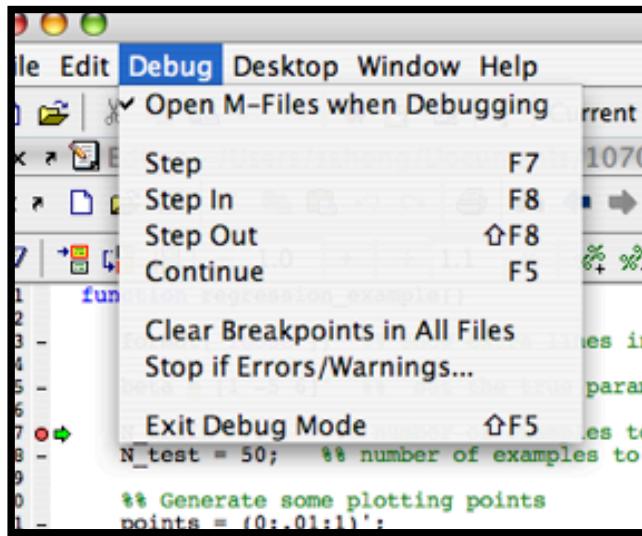


# Debugging

- Insert break points
  - Click to the left of the line (Red Circle)
- Use interactive shell



```
function regression_example()
 format('loose'); % show extra digits
 beta = [1 -5 6]'; % set the true beta
 N_train = 50; % number of examples to train on
 N_test = 50; % number of examples to test on
 % Generate some plotting points
 points = (0:.01:1)';
 points = [points.^0 points.^1]
```



```
K>>
K>> beta
beta =
1
-5
6
```

# Walk Through Interface

Felix Juefei Xu

# Pattern Recognition Theory

**Recitation 4: PCA- Recap**

# Topics To Be Covered

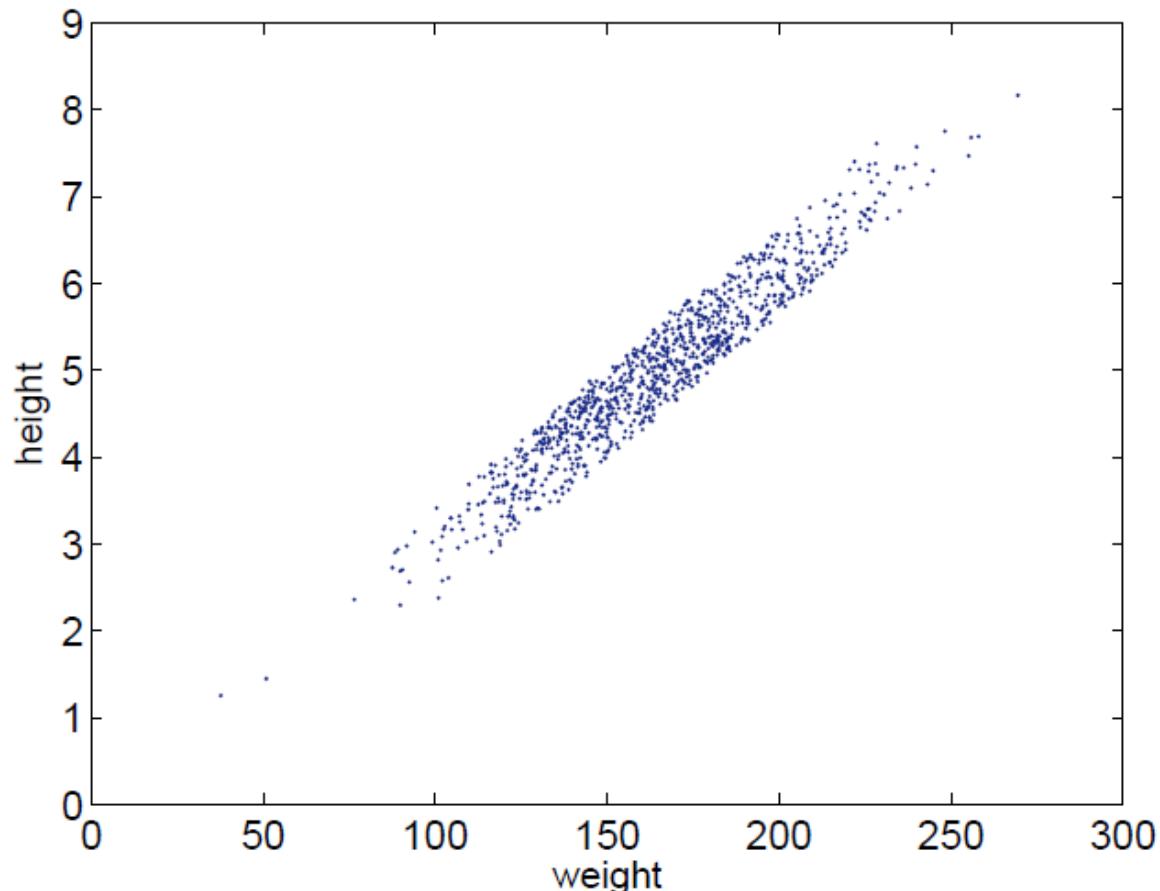
- PCA Basics
  - Motivation
- PCA Derivation
  - Eigen-decomposition on  $\mathbf{X}\mathbf{X}^T$
  - Singular value decomposition (SVD) on  $\mathbf{X}$
- Application and Caveat
- Further Reading
  - More component analysis (CA) methods

# The PCA Basics

- PCA is a linear projection operation that maps a variable of interest to a new coordinate system where the axes represent maximal variability.
- Input data matrix  $\mathbf{X}$  (D by N), D is the dimension, and N is the number of samples. Usually  $D \gg N$
- Output  $\mathbf{Y}$  (D' by N),  $D' \leq D$
- $\mathbf{Y} = \mathbf{P}^T \mathbf{X}$ 
  - Where  $\mathbf{P}$  is the projection matrix (D by D') of which each column is a principal component (PC)

# Motivation

- Remove redundancy
  - If some dimensions are highly correlated (or perfectly correlated, a line), we could discard some dimensions while still capturing the full distribution.



# Derivation

- Most commonly used technique is to apply eigen-decomposition on the covariance matrix as shown in class.
- Covariance matrix of  $\mathbf{X}$  is:  $\mathbf{C} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \boldsymbol{\mu})(\mathbf{x}_n - \boldsymbol{\mu})^T$
- Optimization framework:

$$\mathbf{p}_1 \leftarrow \max F = \mathbf{p}_1^T \mathbf{C} \mathbf{p}_1 + \lambda_1 (1 - \mathbf{p}_1^T \mathbf{p}_1)$$

- The unit norm constraint ensures that the projection is purely rotational without any scaling.
- Eigen-decomposition:  $\mathbf{C} \mathbf{p}_1 = \lambda_1 \mathbf{p}_1$

# Derivation

- $\mathbf{P}_2, \dots, \mathbf{P}_{D'}$  can be found by repeating the aforementioned process.
- A good exercise, manually eigen-decompose a simple 2-by-2 matrix.
- MATLAB: `eigs(C,D')` / `eig(C)`
- Full projection matrix  $\mathbf{P}$  ( $D'=D$ )
  - Covariance matrix is diagonalized as follows:
$$\mathbf{C} = \mathbf{P} \boldsymbol{\Lambda} \mathbf{P}^T$$
- The  $i$ -th eigenvalue indicates the variance explained by projecting the data onto the  $i$ -th PC

# Detour - SVD

- Singular value decomposition of an  $m \times n$  real or complex matrix  $M$  is a factorization of the form:

$$M = U\Sigma V^*$$

- $U$  is a  $m \times m$  real or complex unitary matrix.
- $\Sigma$  is a  $m \times n$  rectangular diagonal matrix with nonnegative real numbers on the diagonal.
- $V^*$  is the conjugate transpose of  $V$ , which is an  $n \times n$  real or complex unitary matrix.
- The left-singular vectors of  $M$  are eigenvectors of  $MM^*$ .
- The right-singular vectors of  $M$  are eigenvectors of  $M^*M$ .

# Detour – Truncated SVD

- Approximating  $\mathbf{M}$  using  $\hat{\mathbf{M}}$  by considering only  $t$  largest singular values. The rest of the matrix is discarded.

$$\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^*$$

$$\hat{\mathbf{M}} = \mathbf{U}_t\Sigma_t\mathbf{V}_t^*$$

- Only the  $t$  column vectors of  $\mathbf{U}$  and  $t$  row vectors of  $\mathbf{V}^*$  corresponding to the  $t$  largest singular values  $\Sigma_t$  .
- Of course, the truncated SVD is no longer an exact decomposition of the original matrix  $\mathbf{M}$  ,  $\hat{\mathbf{M}}$  is a low rank approximation.
- $\mathbf{U}_t$  is thus  $m \times t$ ,  $\Sigma_t$  is  $t \times t$ , and  $\mathbf{V}_t^*$  is  $t \times n$

# Derivation

- Second most commonly used technique is to apply singular value decomposition (SVD) on data matrix  $\mathbf{X}$ .
- SVD of  $\mathbf{X}$  is:  $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ 
  - Where  $\mathbf{U}$  and  $\mathbf{V}$  are orthogonal bases for the column and row spaces of  $\mathbf{X}$ , and  $\Sigma$  is a diagonal matrix of the singular values.
  - Singular values are “stretch factors” that help to match  $\mathbf{u}$ 's with  $\mathbf{v}$ 's
- Now let's derive the covariance matrix of  $\mathbf{X}$  (magic begins!)

$$\begin{aligned}\mathbf{XX}^T &= \mathbf{U}\Sigma\mathbf{V}^T(\mathbf{U}\Sigma\mathbf{V}^T)^T \\ &= \mathbf{U}\Sigma\mathbf{V}^T\mathbf{V}\Sigma\mathbf{U}^T \\ &= \mathbf{U}\Sigma^2\mathbf{U}^T\end{aligned}$$

# Derivation

- Covariance matrix of  $\mathbf{X}$  now becomes:

$$\begin{aligned}\mathbf{XX}^T &= \mathbf{U}\Sigma\mathbf{V}^T(\mathbf{U}\Sigma\mathbf{V}^T)^T \\ &= \mathbf{U}\Sigma\mathbf{V}^T\mathbf{V}\Sigma\mathbf{U}^T \\ &= \mathbf{U}\Sigma^2\mathbf{U}^T\end{aligned}$$

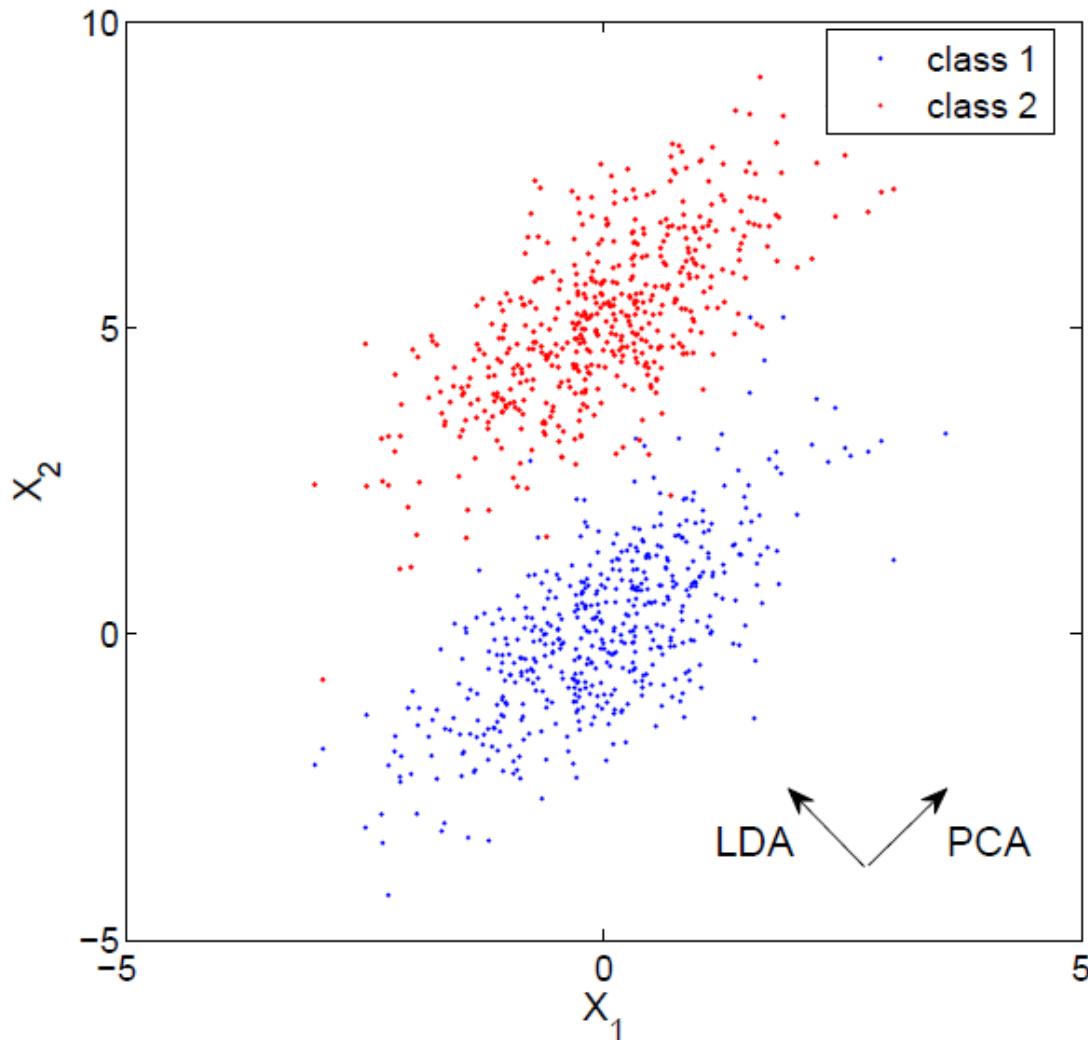
- This is identical to:  $\mathbf{C} = \mathbf{P}\Lambda\mathbf{P}^T$ 
  - Only that **(singular value)<sup>2</sup> = eigenvalue**
- In other words, performing SVD on  $\mathbf{X}$  is equivalent to performing eigen-decomposition on  $\mathbf{XX}^T$

# Application and Caveat

- PCA is very useful if one has limited amount of data
  - e.g., face image ( $128 \times 128 = 16384$ , huge dimension)
- After dimensionality reduction, the dataset becomes more applicable. (easier to handle in regression, classification, etc)
- Eigenface
- The truncated covariance matrix  $\mathbf{C}' = \mathbf{P}\Lambda'\mathbf{P}^T$  is a low-rank approximation of  $\mathbf{C}$ .

# Application and Caveat

- Maximal variability **does not** imply maximal discriminability



# Further Reading

- More component analysis (CA) methods
  - Linear discriminant analysis (LDA)
  - Canonical correlation analysis (CCA)
  - Laplacian eigenmaps (LE)
  - Spectral clustering (SC)
  - Independent component analysis (ICA)

# References

- Jonathon Shlens, “A Tutorial on Principal Component Analysis,” <http://www.snl.salk.edu/~shlens/pca.pdf>
- Chris Bishop, “Pattern Recognition and Machine Learning”, Chapter 12.1