

Machine Learning in Robotics

Lecture 2: Regression

Prof. Dongheui Lee

Human-Centered Assistive Robotics
Technical University of Munich



www.hcr.ei.tum.de



Regression problems

- The goal is to make quantitative (real valued) predictions on the basis of a vector of features or attributes
- Examples: house prices, stock values, survival time, fuel efficiency of cars, etc.
- Questions: What can we assume about the problem? How do we formalize the regression problem? How do we evaluate predictions?



Introduction

Least Mean Square

Polynomial Curve Fitting

Application

2

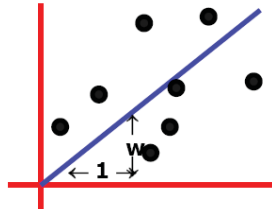


Linear Regression

- Linear regression assumes that expected value of the output given an input is linear.

$$y^{(i)} = wx^{(i)} + \epsilon \quad (1)$$

input x	output y
1	1
2	2
3	2.2
4	3.1
1.5	1.9



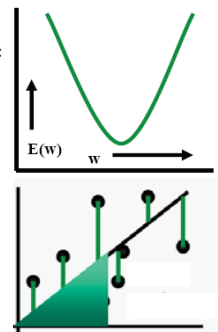
$x^{(i)}$: i-th input
 $y^{(i)}$: i-th output

Linear Least Squares Regression : Single Parameter

- Which value of w makes the output values most likely?
- One that minimizes sum of squares of residuals.

$$E = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - wx^{(i)})^2$$

$$w = \frac{\sum_{i=1}^n x^{(i)} y^{(i)}}{\sum_{i=1}^n x^{(i)2}}$$



- We can use it for prediction.

Linear Least Squares Regression

We need to define a class of functions (types of predictions we will try to make) such as linear predictions:

$$f(x) = w_0 + w_1x \quad (2)$$

where w_0 and w_1 are the parameters we need to set.

We need an estimation criterion so as to be able to select appropriate values for our parameters (w_0 and w_1) based on the training set $\{(x^{(i)}, y^{(i)}), \dots, (x^{(n)}, y^{(n)})\}$

For example, we can use the empirical loss:

$$E = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - f(x^{(i)}))^2 \quad (3)$$



Estimating the parameters

- We minimize the empirical squared loss

$$\begin{aligned} E &= \frac{1}{n} \sum_{i=1}^n (y^{(i)} - f(x^{(i)}))^2 \\ &= \frac{1}{n} \sum_{i=1}^n (y^{(i)} - w_0 - w_1x^{(i)})^2 \end{aligned} \quad (4)$$

- By setting the derivatives with respect to w_0 and w_1 to zero we get necessary conditions for the optimal parameter values

$$\begin{aligned} \frac{\partial}{\partial w_0} E &= 0 \\ \frac{\partial}{\partial w_1} E &= 0 \end{aligned} \quad (5)$$



Estimating the parameters

- By setting the derivatives with respect to w_0 and w_1 to zero

$$\begin{aligned}\frac{\partial}{\partial w_0} E &= 0 \\ \frac{\partial}{\partial w_1} E &= 0\end{aligned}\tag{6}$$

- we get necessary conditions for the optimal parameter values

$$\begin{aligned}w_0 &= \frac{\sum y^{(i)} \sum x^{(i)^2} - \sum x^{(i)} \sum x^{(i)} y^{(i)}}{n \sum x^{(i)^2} - (\sum x^{(i)})^2} \\ w_1 &= \frac{n \sum x^{(i)} y^{(i)} - \sum x^{(i)} \sum y^{(i)}}{n \sum x^{(i)^2} - (\sum x^{(i)})^2}\end{aligned}\tag{7}$$



Linear regression problem with multiple variables

We can express the solution a bit more generally by resorting to a matrix notation

$$X = \begin{bmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \\ \vdots \\ \mathbf{x}^{(n)} \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_m^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_m^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} & \cdots & x_m^{(n)} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_m \end{bmatrix}$$

so that $f(\mathbf{x}) = X\mathbf{w}$.

The result becomes

$$\mathbf{w}^* = (X^T X)^{-1} X^T \mathbf{y}$$



Solving Linear regression in matrix notation

Our empirical loss becomes $E = \frac{1}{n} \|y - Xw\|^2$.

By setting the derivatives of E with respect to w to zero, we get the same optimality conditions as before, now expressed in a matrix form

$$\begin{aligned}\frac{\partial}{\partial w} E &= \frac{1}{n} \frac{\partial}{\partial w} (y - Xw)^T (y - Xw) \\ &= \frac{1}{n} \frac{\partial}{\partial w} (w^T X^T X w - 2y^T X w + y^T y) \\ &= \frac{1}{n} \left(\frac{\partial w^T X^T X w}{\partial w} - 2y^T X \right) \\ &= \frac{1}{n} (2w^T X^T X - 2y^T X) = 0\end{aligned}$$

which yields

$$w^* = (X^T X)^{-1} X^T y$$

Gradient Descent

- Another way to minimize $E(w)$
- Start with an initial value of w , keep changing w to reduce $E(w)$

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} E(w)$$

$$w_j := w_j - 2\alpha(f(x) - y)x_j$$

- Batch Gradient Descent
 - All training data is taken into account

$$w_j := w_j - \frac{\alpha}{n} \sum_{i=1}^n (f(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

- Incremental (Stochastic) Gradient Descent

$$w_j := w_j - \alpha(f(x^{(i)}) - y^{(i)}) x_j^{(i)}, \text{ for } i = 1 \text{ to } n$$

Probabilistic approach

- Assume

$$y^{(i)} = \mathbf{x}^{(i)} \mathbf{w} + \epsilon^{(i)}$$

$$\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$$

$$p(y^{(i)} | \mathbf{x}^{(i)}; \mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \mathbf{x}^{(i)} \mathbf{w})^2}{2\sigma^2}\right)$$

- Likelihood

$$\begin{aligned} L(\mathbf{w}) &= \prod_{i=1}^n p(y^{(i)} | \mathbf{x}^{(i)}; \mathbf{w}) \\ &= \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \mathbf{x}^{(i)} \mathbf{w})^2}{2\sigma^2}\right) \end{aligned}$$

- Choose parameters to maximize the likelihood
= same as minimizing LMS

Beyond linear regression

- The linear regression functions

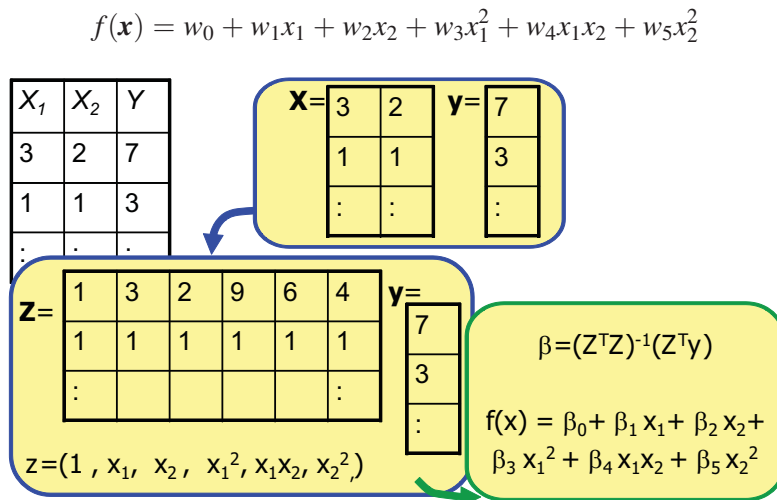
$$f(\mathbf{x}) = w_0 + w_1 x_1 + \dots + w_m x_m$$

are convenient because they are linear in the parameters, not necessarily in the input \mathbf{x}

- We can easily generalize these classes of functions to be non-linear functions of the inputs \mathbf{x} but still linear in the parameters \mathbf{w} . For example: m th order polynomial prediction

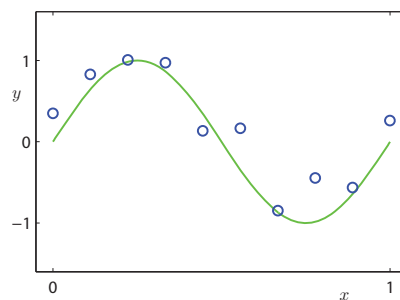
$$f(\mathbf{x}) = w_0 + w_1 x + w_2 x^2 + \dots + w_m x^m$$

Quadratic Regression



Polynomial Curve Fitting

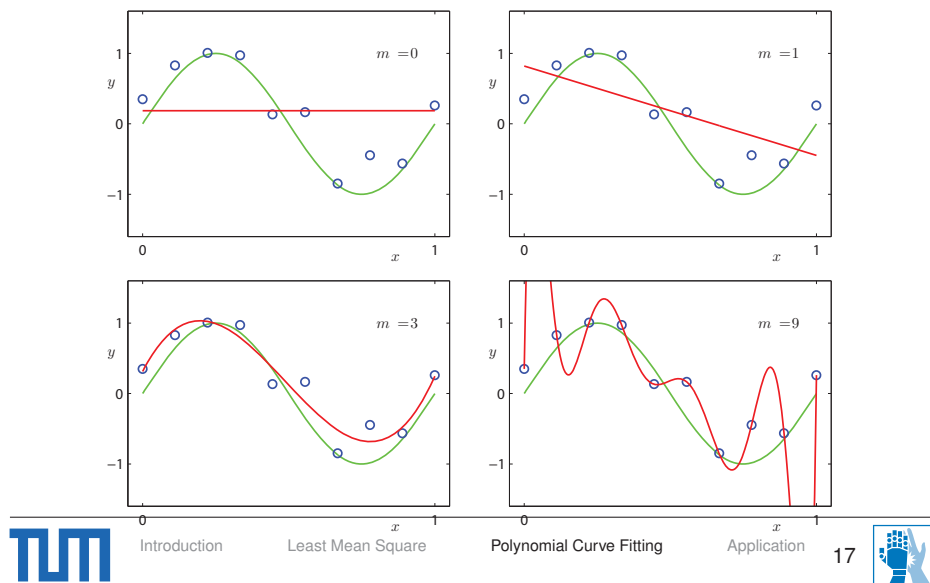
$$f(x) = w_0 + w_1x + w_2x^2 + \dots + w_mx^m$$



Minimize the empirical error

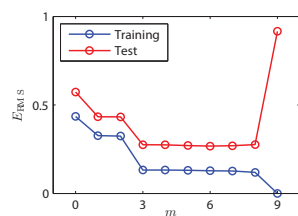
$$E = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - f(x^{(i)}))^2$$

Polynomial Curve Fitting with different orders



Polynomial Curve Fitting

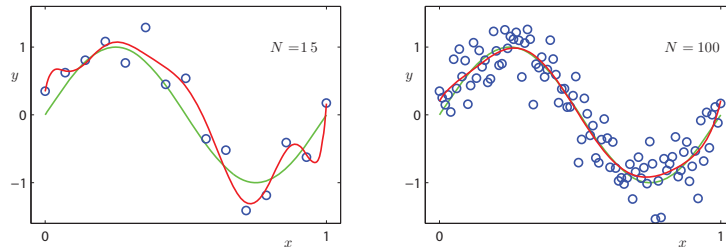
Root-mean-square error & Polynomial coefficients



	$m = 0$	$m = 1$	$m = 3$	$m = 9$
w_0^*	0.19	0.82	0.31	0.35
w_1^*		-1.27	7.99	232.37
w_2^*			-25.43	-5321.83
w_3^*			17.37	48568.31
w_4^*				-231639.30
w_5^*				640042.26
w_6^*				-1061800.52
w_7^*				1042400.18
w_8^*				-557682.99
w_9^*				125201.43

Polynomial Curve Fitting

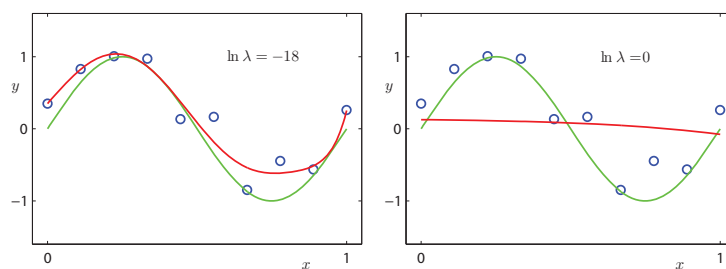
9th order polynomials by increasing the training data, $n = 15$ and $n = 100$



Regularization

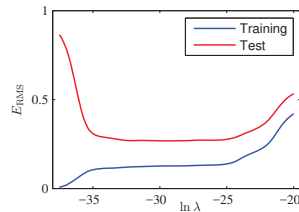
Penalize large coefficient values

$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n (f(x^{(i)}, \mathbf{w}) - y^{(i)})^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$



Regularization

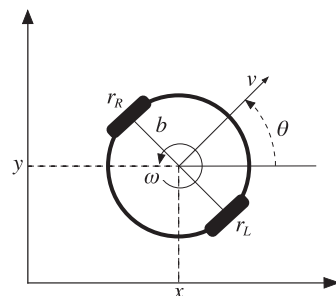
Root-mean-square error & Polynomial coefficients



	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
w_0^*	0.35	0.35	0.13
w_1^*	232.37	4.74	-0.05
w_2^*	-5321.83	-0.77	-0.06
w_3^*	48568.31	-31.97	-0.05
w_4^*	-231639.30	-3.89	-0.03
w_5^*	640042.26	55.28	-0.02
w_6^*	-1061800.52	41.32	-0.01
w_7^*	1042400.18	-45.95	-0.00
w_8^*	-557682.99	-91.53	0.00
w_9^*	125201.43	72.68	0.01

Robotics Applications: Odometry calibration

Estimate the pose (x, y, θ) of a mobile robot given the angular velocity of each wheel (ω_L, ω_R)



$$\dot{x} = v \cos(\theta)$$

$$\dot{y} = v \sin(\theta)$$

$$\dot{\theta} = \omega$$

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \mathbf{W}(r_R, r_L, b) \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix}$$

Odometry calibration consists in estimating \mathbf{W}

Robotics Applications: Odometry calibration

$$\begin{aligned}
 x_{t+1} &= x_t + v \cos(\theta) \Delta T \\
 y_{t+1} &= y_t + v \sin(\theta) \Delta T \\
 \theta_{t+1} &= \theta_t + \omega \Delta T \\
 \begin{bmatrix} v \\ \omega \end{bmatrix} &= \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix}
 \end{aligned}
 \implies
 \begin{aligned}
 \begin{bmatrix} x_N - x_0 \\ y_N - y_0 \end{bmatrix} &= \mathbf{X}(\omega_R, \omega_L) \begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} \\
 \theta_N - \theta_0 &= \mathbf{Z}(\omega_R, \omega_L) \begin{bmatrix} w_{21} \\ w_{22} \end{bmatrix}
 \end{aligned}$$

Executing M trajectories the parameters $(w_{11}, w_{12}, w_{21}, w_{22})$ can be identified using Linear Regression

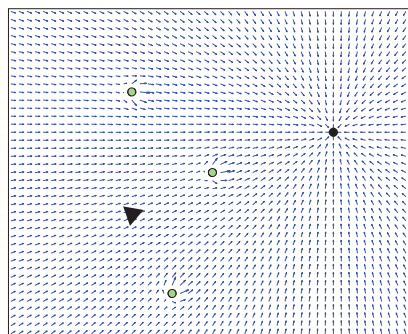
- G. Antonelli, S. Chiaverini and G. Fusco *A Calibration Method for Odometry of Mobile Robots Based on the Least-Squares Technique: Theory and Experimental Validation*. Transactions on Robotics. 2005.



Robotics Applications: Obstacle avoidance

Assign an attractive potential (u_{att}) to the goal position and repulsive potentials (u_{rep}) to the obstacles

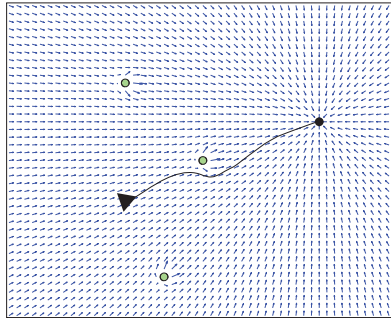
$$u = u_{att} + u_{rep} \rightarrow f_j = \frac{\partial}{\partial p_j} u_{att} + \frac{\partial}{\partial p_j} u_{rep}$$



Robotics Applications: Obstacle avoidance

A collision-free trajectory is generated using Gradient Descent

$$\mathbf{p}^{(i+1)} = \mathbf{p}^{(i)} - \alpha \frac{\mathbf{f}^{(i)}}{\|\mathbf{f}\|}$$



- O. Khatib *Real-time obstacle avoidance for manipulators and mobile robots.* International Journal of Robotics Research. 1986.



Reference

- Bishop, Pattern Recognition and Machine Learning
Chap. 1.1, 3, 6.4.1, 6.4.2
- Mitchell, Machine Learning
Chap. 4.4.3, 8.3



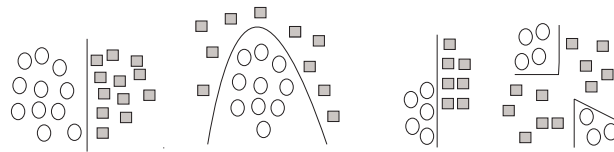
Features and Patterns

- Good Features and Bad Features



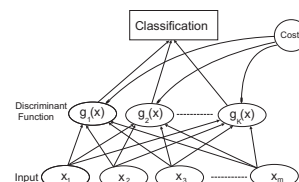
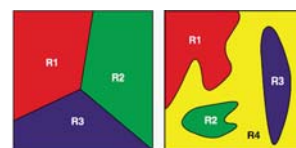
- More Feature Properties

linear separability, nonlinear separability, highly correlated, and multi-modal features



Classifier

- The task of a classifier is to partition feature space into class-labeled decision regions
- **decision boundary** : borders between **decision regions**
- The classification of feature vector x consists of determining which decision region it belongs to and assign x to this class
- A classifier can be represented as a set of discriminant functions
 - The classifier assigns a feature vector x to a class by using discriminant functions



Bayesian Decision Theory

- Bayesian Decision Theory is a fundamental statistical approach to the problem of pattern classification.
- Quantifies the tradeoffs between various classifications using probability and the costs that accompany such decisions/classifications.
- Assumptions
 - Decision problem is posed in probabilistic terms.
 - All relevant probability values are known.

Bayes Theorem



Thomas Bayes (1702-1761)

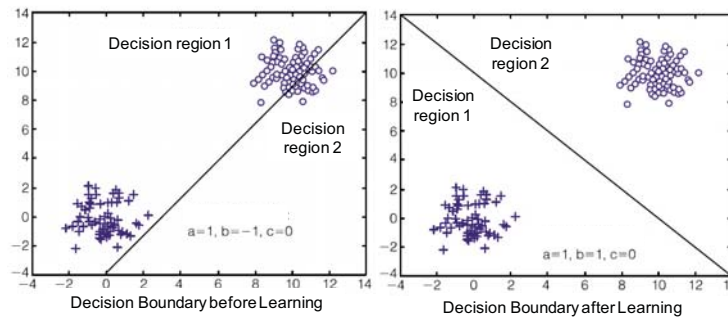
$$p(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)p(\omega_i)}{p(\mathbf{x})}$$

- $p(\omega_i)$: prior probability (of class ω_i)
- $p(\omega_i|\mathbf{x})$: posterior probability (of class ω_i given the observation \mathbf{x})
- $p(\mathbf{x}|\omega_i)$: likelihood (conditional probability of observation \mathbf{x} given class ω_i)
- $p(\mathbf{x})$: a normalized constant that does not affect the decision

Linear Classifier

Linear models for classification

- Decision boundaries are linear functions of the input vectors x and defined by $m - 1$ dimensional hyperplanes within the m dimensional input space. $g(x_1, x_2) = ax_1 + bx_2 + c$
- Data set whose classes can be separated by linear decision surfaces are said to be *linearly separable*.



Linear Classifier

- Linear regression
 - Model prediction is given by a linear function of the parameter w .
- Linear Models for classification
 - To predict discrete class label

$$g(x) = w^T x + w_0$$

- w : weight vector
 - w_0 : bias
- How to find the weight vector?
 - Normally minimization of classification errors



Decision Boundary

- The case of two classes
 - If $g(\mathbf{x}) \geq 0$, the input \mathbf{x} is assigned to class ω_1
 - Else, assign to class ω_2
- The case of multiple classes
 - Single K -class discriminant comprising K linear functions

$$g_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0}$$

- Assign a point \mathbf{x} into class ω_k if $g_k(\mathbf{x}) \geq g_j(\mathbf{x})$ for all $j \neq k$

Likelihood Ratio Test (LRT)

- Given the problem of classifying a given measurement \mathbf{x} , a 'reasonable' heuristic decision rule would be:
 - Choose the class that is more 'likely' given the evidence provided by the measured feature \mathbf{x}
 - Evaluate the posterior probability of each class $p(\omega_i|\mathbf{x})$ and choose the class with the largest $p(\omega_i|\mathbf{x})$
- Let us examine the consequences of this decision rule for a 2-class problem
 - the decision rule becomes

$$\begin{aligned} \text{if } p(\omega_1|\mathbf{x}) > p(\omega_2|\mathbf{x}) & \text{ choose } \omega_1 \\ \text{otherwise} & \text{ choose } \omega_2 \end{aligned}$$

- Likelihood ratio test

$$\frac{p(\mathbf{x}|\omega_1)}{p(\mathbf{x}|\omega_2)} \underset{\omega_2}{\overset{\omega_1}{\gtrless}} \frac{p(\omega_2)}{p(\omega_1)}$$

Likelihood Ratio Test : An Example

Given a classification problem with the following class conditional densities, derive a decision rule based on the Likelihood Ratio Test (assume equal priors)

$$p(x|\omega_1) = \frac{1}{\sqrt{2\pi}} \exp -\frac{(x-4)^2}{2} , \quad p(x|\omega_2) = \frac{1}{\sqrt{2\pi}} \exp -\frac{(x-10)^2}{2}$$

Solution :

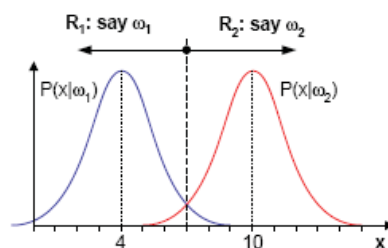
Likelihood Ratio Test : An Example

Given a classification problem with the following class conditional densities, derive a decision rule based on the Likelihood Ratio Test (assume equal priors)

$$p(x|\omega_1) = \frac{1}{\sqrt{2\pi}} \exp -\frac{(x-4)^2}{2} , \quad p(x|\omega_2) = \frac{1}{\sqrt{2\pi}} \exp -\frac{(x-10)^2}{2}$$

Solution :

$$x \begin{matrix} \leq \omega_1 \\ > \omega_2 \end{matrix} 7$$



Probability of error

Probability of error:
$$p(\text{error}) = \sum_{i=1}^K p(\text{error}|\omega_i)p(\omega_i)$$

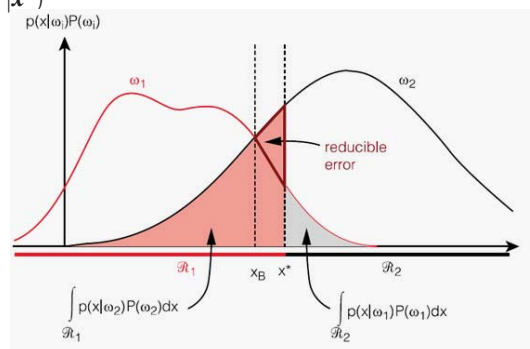
The class conditional probability of error :
 $p(\text{error}|\omega_i) = p(\text{choose } \omega_j|\omega_i) = \int_{R_j} p(\mathbf{x}|\omega_i)d\mathbf{x}$
 for our 2-class problem

$$\begin{aligned} p(\text{error}) &= p(\mathbf{x} \in R_2, \omega_1) + p(\mathbf{x} \in R_1, \omega_2) \\ &= p(\mathbf{x} \in R_2|\omega_1)p(\omega_1) + p(\mathbf{x} \in R_1|\omega_2)p(\omega_2) \\ &= \int_{R_2} p(\mathbf{x}|\omega_1)p(\omega_1)d\mathbf{x} + \int_{R_1} p(\mathbf{x}|\omega_2)p(\omega_2)d\mathbf{x} \\ &= p(\omega_1) \int_{R_2} p(\mathbf{x}|\omega_1)d\mathbf{x} + p(\omega_2) \int_{R_1} p(\mathbf{x}|\omega_2)d\mathbf{x} \end{aligned}$$

Since we assumed equal priors, then $p(\text{error}) = (\epsilon_1 + \epsilon_2)/2$

What is the minimum possible error?

- The optimal decision rule will minimize $p(\text{error}|\mathbf{x})$ for every value of \mathbf{x} , so that the integral above is minimized.
- From the figure it becomes clear that, for any value of \mathbf{x}' , the Likelihood Ratio Test decision rule will always have a lower $p(\text{error}|\mathbf{x}')$



Bayes Risk

- Is the penalty of misclassifying a class ω_1 example as class ω_2 the same as the reciprocal?
 - misclassifying a cancer sufferer as a healthy patient
- Bayes Risk
 - the expected value of the overall risk for a 2-class problem

$$R = \int_{R_1} R(\alpha_1|\mathbf{x})p(\mathbf{x})d\mathbf{x} + \int_{R_2} R(\alpha_2|\mathbf{x})p(\mathbf{x})d\mathbf{x}$$

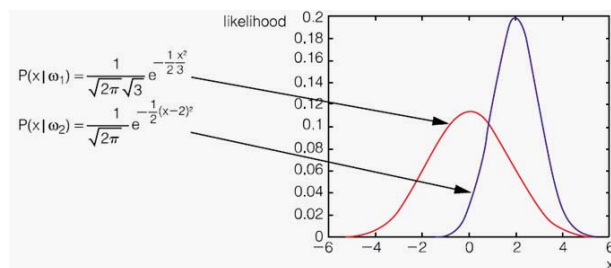
$$= \int_{R_1} (c_{11}p(\omega_1|\mathbf{x}) + c_{12}p(\omega_2|\mathbf{x}))p(\mathbf{x})d\mathbf{x} + \int_{R_2} (c_{21}p(\omega_1|\mathbf{x}) + c_{22}p(\omega_2|\mathbf{x}))p(\mathbf{x})d\mathbf{x}$$

Choose ω_1 if $R(\alpha_1|\mathbf{x}) < R(\alpha_2|\mathbf{x})$.

$$\Lambda(\mathbf{x}) = \frac{p(\mathbf{x}|\omega_1)}{p(\mathbf{x}|\omega_2)} \begin{matrix} \geq_{\omega_1} \\ \leq_{\omega_2} \end{matrix} \frac{(c_{12} - c_{22})P(\omega_2)}{(c_{21} - c_{11})P(\omega_1)}$$

Bayes Risk : An Example

- Consider a classification problem with two classes defined by the following likelihood functions
 - What is the likelihood ratio?
 - When $p(\omega_1) = p(\omega_2) = 0.5$, $c_{11} = c_{22} = 0$, $c_{12} = 1$ and $c_{21} = 3^{1/2}$, determine a decision rule that minimizes the bayes risk.



Solution: If $1.27 < x < 4.73$, x belongs to ω_2 .

Bayesian Decision Criteria

- Bayes Criterion

$$\Lambda(\mathbf{x}) = \frac{p(\mathbf{x}|\omega_1)}{p(\mathbf{x}|\omega_2)} \underset{\omega_2}{\overset{\omega_1}{\geq}} \frac{(c_{12} - c_{22})P(\omega_2)}{(c_{21} - c_{11})P(\omega_1)}$$

- Maximum A Posteriori (MAP) Criterion

$$\Lambda(\mathbf{x}) = \frac{p(\mathbf{x}|\omega_1)}{p(\mathbf{x}|\omega_2)} \underset{\omega_2}{\overset{\omega_1}{\geq}} \frac{P(\omega_2)}{P(\omega_1)} \iff \frac{P(\omega_1|\mathbf{x})}{P(\omega_2|\mathbf{x})} \underset{\omega_2}{\overset{\omega_1}{\geq}} 1$$

- Maximum Likelihood (ML) Criterion

$$\Lambda(\mathbf{x}) = \frac{p(\mathbf{x}|\omega_1)}{p(\mathbf{x}|\omega_2)} \underset{\omega_2}{\overset{\omega_1}{\geq}} 1$$

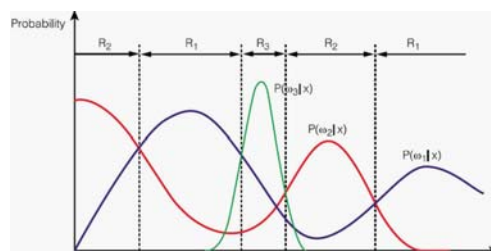
multi-class problems : probability of error

- The decision rule that minimizes $p(\text{error})$ generalizes very easily to multi-class problems

$$p(\text{error}) = 1 - p(\text{correct})$$

- The problem of minimizing $p(\text{error})$ is equivalent to that of maximizing $p(\text{correct})$.

$$p(\text{correct}) = \sum_{i=1}^K p(\omega_i) \int_{R_i} p(\mathbf{x}|\omega_i) d\mathbf{x}$$



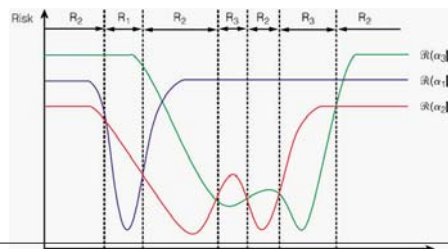
multi-class problems : Bayes Risk

To determine which decision rule yields the minimum Bayes Risk for the multi-class problem

$$R = \int R(\alpha(\mathbf{x})|\mathbf{x})p(\mathbf{x})d\mathbf{x}$$

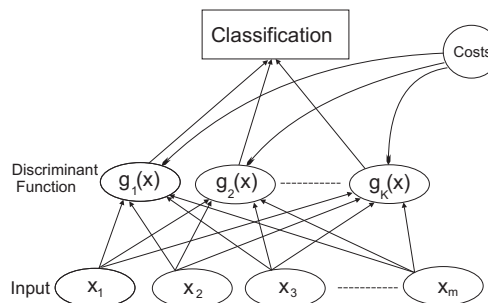
$$R(\alpha_i|\mathbf{x}) = \sum_{j=1}^K c_{ij}p(\omega_j|\mathbf{x})$$

Choose ω_i which has the minimum $R(\alpha_i|\mathbf{x})$



Discriminant functions

Assign x to class ω_i if $g_i(x) > g_j(x), \forall j \neq i$.



Three basic decision rules

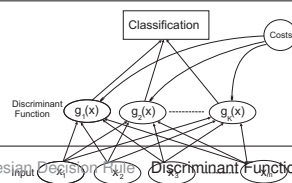
Criterion	Discriminant function
Bayes	$g_i(x) = -R(\alpha_i \mathbf{x})$
MAP	$g_i(x) = p(\omega_i \mathbf{x})$
ML	$g_i(x) = p(\mathbf{x} \omega_i)$

Discriminant Functions for the Normal Density

- Decision rule based on the MAP discriminant function to choose ω_i if $g_i(\mathbf{x}) > g_j(\mathbf{x}), \forall j \neq i$ where $g_i(\mathbf{x}) = p(\omega_i|\mathbf{x})$
- Gaussian Discriminant Analysis, Gaussian Bayes Classifier: to model $p(\mathbf{x}|\mathbf{y})$ as a Gaussian distribution
- Multivariate Gaussian distribution

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{m/2} |\Sigma|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right]$$

$$g_k(\mathbf{x}) = -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^T \Sigma_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) - \frac{1}{2} \ln |\Sigma_k| + \ln p(\omega_k) \quad (1)$$



Case 1 : $\Sigma_k = \sigma^2 I$

- This situation occurs when the features are statistically independent with the same variance for all classes

$$g_k(\mathbf{x}) = -\frac{1}{2\sigma^2} (-2\boldsymbol{\mu}_k^T \mathbf{x} + \boldsymbol{\mu}_k^T \boldsymbol{\mu}_k) + \ln p(\omega_k) = \mathbf{w}_k^T \mathbf{x} + w_{k0}$$

$$\text{where } \begin{cases} \mathbf{w}_k = \frac{1}{\sigma^2} \boldsymbol{\mu}_k \\ w_{k0} = -\frac{1}{2\sigma^2} \boldsymbol{\mu}_k^T \boldsymbol{\mu}_k + \ln p(\omega_k) \end{cases}$$

- If equal priors, a minimum-distance or nearest mean classifier

$$g_k(\mathbf{x}) = -\frac{1}{2\sigma^2} (\mathbf{x} - \boldsymbol{\mu}_k)^T (\mathbf{x} - \boldsymbol{\mu}_k)$$

Case 1 : $\Sigma_k = \sigma^2 I$, An example

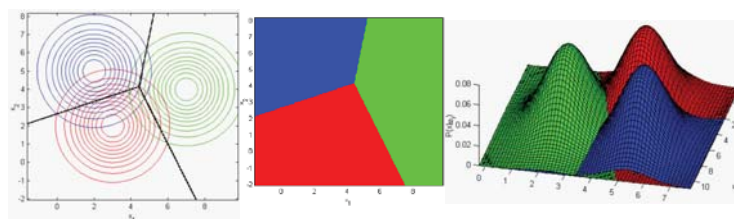
Compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\begin{aligned} \mu_1 &= \begin{bmatrix} 3 & 2 \end{bmatrix}^T & \mu_2 &= \begin{bmatrix} 7 & 4 \end{bmatrix}^T & \mu_3 &= \begin{bmatrix} 2 & 5 \end{bmatrix}^T \\ \Sigma_1 &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} & \Sigma_2 &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} & \Sigma_3 &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \end{aligned}$$

Case 1 : $\Sigma_k = \sigma^2 I$, An example

Compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\begin{aligned} \mu_1 &= \begin{bmatrix} 3 & 2 \end{bmatrix}^T & \mu_2 &= \begin{bmatrix} 7 & 4 \end{bmatrix}^T & \mu_3 &= \begin{bmatrix} 2 & 5 \end{bmatrix}^T \\ \Sigma_1 &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} & \Sigma_2 &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} & \Sigma_3 &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \end{aligned}$$



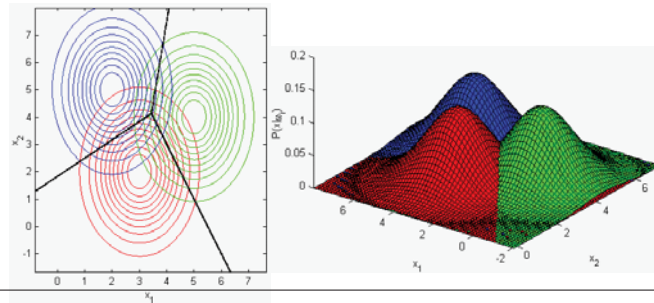
- $$g_k(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) + \ln p(\omega_k)$$

$$\begin{aligned} \mu_1 &= \begin{bmatrix} 3 & 2 \end{bmatrix}^T & \mu_2 &= \begin{bmatrix} 5 & 4 \end{bmatrix}^T & \mu_3 &= \begin{bmatrix} 2 & 5 \end{bmatrix}^T \\ \Sigma_1 &= \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} & \Sigma_2 &= \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} & \Sigma_3 &= \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \end{aligned}$$

Case 2 : $\Sigma_k = \Sigma$ (diagonal), An example

Compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\begin{aligned} \mu_1 &= \begin{bmatrix} 3 & 2 \end{bmatrix}^T & \mu_2 &= \begin{bmatrix} 5 & 4 \end{bmatrix}^T & \mu_3 &= \begin{bmatrix} 2 & 5 \end{bmatrix}^T \\ \Sigma_1 &= \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} & \Sigma_2 &= \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} & \Sigma_3 &= \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \end{aligned}$$



Case 3 : $\Sigma_k = \Sigma$ (non-diagonal)

- The case that all the classes have the same covariance matrix, but this is no longer diagonal
- The quadratic discriminant becomes

$$g_k(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) + \ln p(\omega_k)$$

$$g_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0}$$

$$\text{where } \begin{cases} \mathbf{w}_k = \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k \\ w_{k0} = -\frac{1}{2} \boldsymbol{\mu}_k^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k + \ln p(\omega_k) \end{cases}$$

- If equal priors, a Mahalanobis distance classifier

$$g_k(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)$$

Case 3 : $\Sigma_k = \Sigma$ (non-diagonal), An example

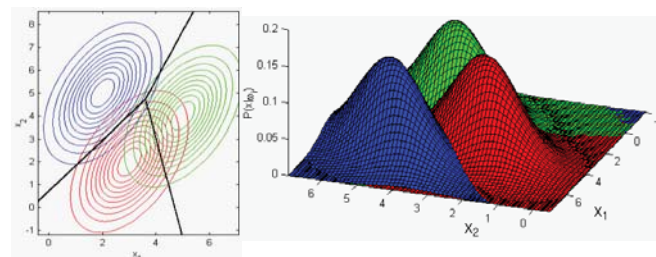
Compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\begin{aligned} \mu_1 &= \begin{bmatrix} 3 & 2 \end{bmatrix}^T & \mu_2 &= \begin{bmatrix} 5 & 4 \end{bmatrix}^T & \mu_3 &= \begin{bmatrix} 2 & 5 \end{bmatrix}^T \\ \Sigma_1 &= \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2 \end{bmatrix} & \Sigma_2 &= \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2 \end{bmatrix} & \Sigma_3 &= \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2 \end{bmatrix} \end{aligned}$$

Case 3 : $\Sigma_k = \Sigma$ (non-diagonal), An example

Compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\begin{aligned} \mu_1 &= \begin{bmatrix} 3 & 2 \end{bmatrix}^T & \mu_2 &= \begin{bmatrix} 5 & 4 \end{bmatrix}^T & \mu_3 &= \begin{bmatrix} 2 & 5 \end{bmatrix}^T \\ \Sigma_1 &= \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2 \end{bmatrix} & \Sigma_2 &= \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2 \end{bmatrix} & \Sigma_3 &= \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2 \end{bmatrix} \end{aligned}$$



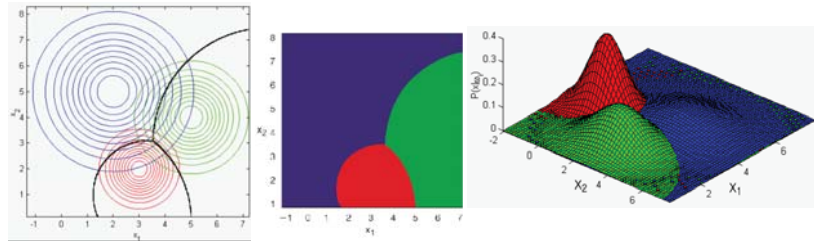
- $$\begin{aligned} g_k(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) - \frac{1}{2} \ln |\boldsymbol{\Sigma}_k| + \ln p(\omega_k) \\ &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\sigma}_k^{-2}(\mathbf{x} - \boldsymbol{\mu}_k) - \frac{m}{2} \ln(\boldsymbol{\sigma}_k^2) + \ln p(\omega_k) \end{aligned}$$

$$\begin{aligned} \mu_1 &= \begin{bmatrix} 3 & 2 \end{bmatrix}^T & \mu_2 &= \begin{bmatrix} 5 & 4 \end{bmatrix}^T & \mu_3 &= \begin{bmatrix} 2 & 5 \end{bmatrix}^T \\ \Sigma_1 &= \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} & \Sigma_2 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \Sigma_3 &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \end{aligned}$$

Case 4 : $\Sigma_k = \sigma_k^2 I$, An example

Compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\begin{aligned} \mu_1 &= \begin{bmatrix} 3 & 2 \end{bmatrix}^T & \mu_2 &= \begin{bmatrix} 5 & 4 \end{bmatrix}^T & \mu_3 &= \begin{bmatrix} 2 & 5 \end{bmatrix}^T \\ \Sigma_1 &= \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} & \Sigma_2 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \Sigma_3 &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \end{aligned}$$



Case 5 : $\Sigma_k = \text{arbitrary}$

$$g_k(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) - \frac{1}{2} \ln |\boldsymbol{\Sigma}_k| + \ln p(\omega_k)$$

Reorganizing terms in a quadratic form yields

$$g_k(\mathbf{x}) = \mathbf{x}^T \mathbf{W}_k \mathbf{x} + \mathbf{w}_k^T \mathbf{x} + w_{k0}$$

$$\text{where } \begin{cases} \mathbf{W}_k = -\frac{1}{2} \boldsymbol{\Sigma}_k^{-1} \\ \mathbf{w}_k = \boldsymbol{\Sigma}_k^{-1} \boldsymbol{\mu}_k \\ w_{k0} = -\frac{1}{2} \boldsymbol{\mu}_k^T \boldsymbol{\Sigma}_k^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \ln |\boldsymbol{\Sigma}_k| + \ln p(\omega_k) \end{cases}$$

Case 5 : $\Sigma_k = \text{arbitrary}$, An example

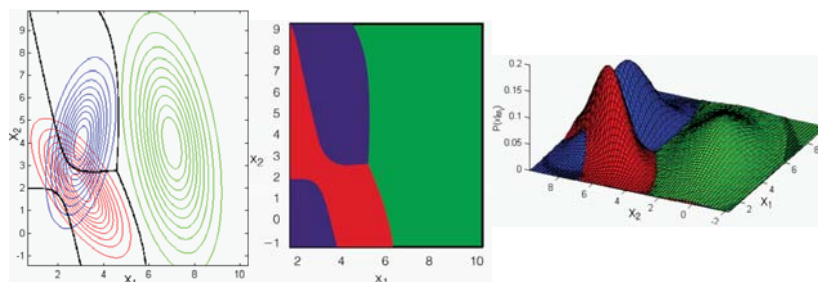
Compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\begin{aligned} \mu_1 &= \begin{bmatrix} 3 & 2 \end{bmatrix}^T & \mu_2 &= \begin{bmatrix} 5 & 4 \end{bmatrix}^T & \mu_3 &= \begin{bmatrix} 2 & 5 \end{bmatrix}^T \\ \Sigma_1 &= \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} & \Sigma_2 &= \begin{bmatrix} 1 & -1 \\ -1 & 7 \end{bmatrix} & \Sigma_3 &= \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 3 \end{bmatrix} \end{aligned}$$

Case 5 : $\Sigma_k = \text{arbitrary}$, An example

Compute the decision boundaries for a 3-class, 2-dimensional problem with the following class mean vectors and covariance matrices and equal priors

$$\begin{aligned} \mu_1 &= \begin{bmatrix} 3 & 2 \end{bmatrix}^T & \mu_2 &= \begin{bmatrix} 5 & 4 \end{bmatrix}^T & \mu_3 &= \begin{bmatrix} 2 & 5 \end{bmatrix}^T \\ \Sigma_1 &= \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} & \Sigma_2 &= \begin{bmatrix} 1 & -1 \\ -1 & 7 \end{bmatrix} & \Sigma_3 &= \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 3 \end{bmatrix} \end{aligned}$$



An example

- Given 3-dimensional 2-classes with following mean vectors, covariance matrices, and priors

$$\begin{aligned} \mu_1 &= [0 \quad 0 \quad 0]^T & \mu_2 &= [1 \quad 1 \quad 1]^T \\ \Sigma_1 &= \Sigma_2 = \begin{bmatrix} 1/4 & 0 & 0 \\ 0 & 1/4 & 0 \\ 0 & 0 & 1/4 \end{bmatrix} & p(\omega_2) &= 2p(\omega_1) \end{aligned}$$

- Which class does this vector $x = [0.1 \ 0.7 \ 0.8]^T$ belong to?

Solution :

An example

- Given 3-dimensional 2-classes with following mean vectors, covariance matrices, and priors

$$\begin{aligned} \mu_1 &= [0 \quad 0 \quad 0]^T & \mu_2 &= [1 \quad 1 \quad 1]^T \\ \Sigma_1 &= \Sigma_2 = \begin{bmatrix} 1/4 & 0 & 0 \\ 0 & 1/4 & 0 \\ 0 & 0 & 1/4 \end{bmatrix} & p(\omega_2) &= 2p(\omega_1) \end{aligned}$$

- Which class does this vector $x = [0.1 \ 0.7 \ 0.8]^T$ belong to?

Solution :

$$x \in \omega_2$$

Least Squares for Classification

Linear model

$$f(\mathbf{x}) = w_0 + w_1x_1 + \dots + w_mx_m$$

The parameters are estimated by minimizing the error

$$E = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - f(\mathbf{x}^{(i)}))^2$$

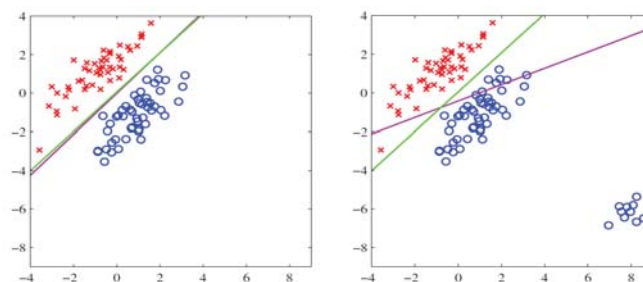
The result becomes

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

In the case of binary classification, $y \in \{0, 1\}$

Least Squares for Classification

For 2D input vector



- Magenta : Linear regression
- Green : Logistic regression

Logistic Regression

Consider learning $x \rightarrow y$, where

- for 2 class problem: y is boolean. $y \in \{0, 1\}$
- x is a vector of real-valued feature. item Model $P(x_i|y = y_k)$ as Gaussian $\mathcal{N}(0, \sigma_i)$.
- Model $P(y)$ as Bernoulli π .

The form $P(y = 1|\mathbf{x})$ becomes

$$f(\mathbf{x}) = P(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

Derive $P(y = 1|\mathbf{x})$

$$\begin{aligned}
 P(y = 1|\mathbf{x}) &= \frac{P(y = 1)P(\mathbf{x}|y = 1)}{P(y = 1)P(\mathbf{x}|y = 1) + P(y = 0)P(\mathbf{x}|y = 0)} \\
 &= \frac{1}{1 + \frac{P(y=0)P(\mathbf{x}|y=0)}{P(y=1)P(\mathbf{x}|y=1)}} = \frac{1}{1 + \exp(\ln(\frac{P(y=0)P(\mathbf{x}|y=0)}{P(y=1)P(\mathbf{x}|y=1)}))} \\
 &= \frac{1}{1 + \exp\left[\ln\frac{1-\pi}{\pi} + \sum_i \ln\frac{P(x_i|y=0)}{P(x_i|y=1)}\right]} \\
 \sum_i \ln\frac{P(x_i|y = 0)}{P(x_i|y = 1)} &= \sum_i \left(\frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2}x_i + \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2}\right) \\
 f(\mathbf{x}) = P(y = 1|\mathbf{x}) &= \frac{1}{1 + \exp(-\mathbf{w}^T\mathbf{x})}
 \end{aligned}$$

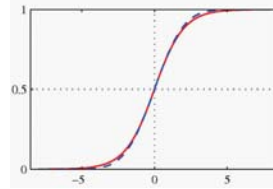
Logistic Regression

$$P(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

$$P(y = 0|\mathbf{x}) = \frac{\exp(-\mathbf{w}^T \mathbf{x})}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

$$\frac{P(y = 0|\mathbf{x})}{P(y = 1|\mathbf{x})} = \exp(-\mathbf{w}^T \mathbf{x})$$

$$\ln \frac{P(y = 0|\mathbf{x})}{P(y = 1|\mathbf{x})} = -\mathbf{w}^T \mathbf{x}$$



logistic function

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

Training Logistic Regression

- How to estimate the parameters?
- Maximum likelihood estimate for parameters \mathbf{w}

$$\mathbf{w} = \operatorname{argmax} \prod_{i=1}^n p(\mathbf{x}^{(i)}, y^{(i)} | \mathbf{w})$$

- Maximum conditional likelihood estimate

$$\mathbf{w} = \operatorname{argmax} \prod_{i=1}^n p(y^{(i)} | \mathbf{x}^{(i)}, \mathbf{w})$$

Training Logistic Regression

- Choose parameter to maximize conditional likelihood

$$L(\mathbf{w}) = p(\mathbf{y}|\mathbf{X}; \mathbf{w}) = \prod_{i=1}^n f(\mathbf{x}^{(i)})^{y^{(i)}} \{1 - f(\mathbf{x}^{(i)})\}^{1-y^{(i)}}$$

$$l(\mathbf{w}) = \ln L(\mathbf{w}) = \sum_{i=1}^n \left\{ y^{(i)} \ln f(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \ln(1 - f(\mathbf{x}^{(i)})) \right\}$$

- Batch gradient ascent

$$w_j := w_j + \frac{\partial}{\partial w_j} l(\mathbf{w})$$

$$w_j := w_j - \frac{\alpha}{n} \sum_{i=1}^n (f(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

Sign Language Recognition

- American Sign Language recognition (10 signs)
- Features: 22 hand joint angles, 3D hand position, 3D hand orientation, 5 finger forces
- Comparison of naive Bayesian Classifier, Decision Tree, Neural Network



C. Shahabi, L. Kaghazian, S. Mehta, A. Ghoting, G. Shanbhag and M. McLaughlin, *Analysis of Haptic Data for Sign Language Recognition*, International conference on Universal Access in Human-Computer Interaction, 2001.

Sign Language Recognition

- The naive Bayesian Classifier has the highest average accuracy with 50 training examples.



C. Shahabi, L. Kaghazian, S. Mehta, A. Ghoting, G. Shanbhag and M. McLaughlin, *Analysis of Haptic Data for Sign Language Recognition*, International conference on Universal Access in Human-Computer Interaction, 2001.



In Door Place Classification

- Determining whether a cleaning robot (with bumper and vision sensors) is in a hall, a room, or a corridor
- From vision sensing an orientation histogram I is created, by grouping straight lines with a similar orientation.




C. Yi, Y. C. Oh, I. H. Suh and B.-U. Choi, *In door Place Classification Using Robot Behavior and Vision Data*, International Journal of Advanced Robotic Systems, 2011.



In Door Place Classification

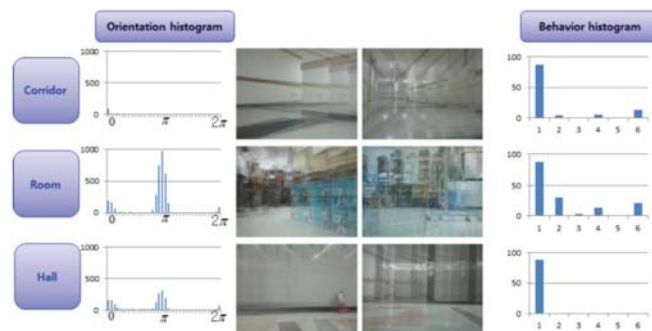
- From robot behavioral data a behavioral histogram B are calculated.
- From visual and behavioral histograms, $p(\omega_i|I, B)$, the probability that a place belongs to specific class, is calculated.

 C. Yi, Y. C. Oh, I. H. Suh and B.-U. Choi, *In door Place Classification Using Robot Behavior and Vision Data*, International Journal of Advanced Robotic Systems, 2011.

Mobile Robot Navigation

$$p(\omega_i|I, B) = \frac{p(\omega_i)p(I, B|\omega_i)}{p(I, B)}$$

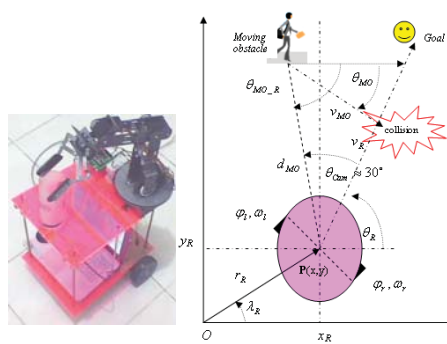
$$p(\omega_i)p(I, B|\omega_i) = p(\omega_i)p(I|\omega_i)p(B|\omega_i)$$



Application : Moving obstacle avoidance

Reference: W. Budiharto, D. Purwanto and A. Jazidie, *A Robust Obstacle Avoidance for Service Robot Using Bayesian Approach*, 2011.

The obstacle avoidance problem is formulated using decision theory, prior and posterior distribution and loss function to determine an optimal response based on inaccurate sensor data.



Moving obstacle avoidance using Bayesian approach

Reference: W. Budiharto, D. Purwanto and A. Jazidie, *A Robust Obstacle Avoidance for Service Robot Using Bayesian Approach*, 2011.

- $\Theta = (\theta_1, \theta_2) = (Obstacle, noObstacle)$ are two possible path states.
- The probability of path state θ given observation z is
$$p(\theta|z) = \frac{p(z|\theta)p(\theta)}{\sum p(z|\theta)p(\theta)}.$$
- Define an action space $A = (a_1, a_2) = (maneuver, stop)$.
- Compute bayes risk for each action $a : \sum_{\theta} C(\theta, a) p(\theta|z)$.
- Take an action which leads to smaller risk

Summary

- Bayesian decision
- Gaussian Bayes classifiers is a quadratic classifiers.
 - The Bayes classifiers for normally distributed classes with equal covariance matrices is a linear classifier
 - The minimum Mahalanobis distance classifier is optimum for normal distributed classes, equal covariance matrices and equal priors.
 - The minimum Euclidean distance classifier is optimum for normal distributed classes, equal covariance matrices proportional to the identity matrix and equal priors.
 - Both Euclidean and Mahalanobis minimum distance classifiers are linear classifiers
- Logistic regression

Summary and Next Lecture

- Reading
 - Duda, Chap. 2.1-7
 - Bishop Chap. 1.5
 - Bishop Chap. 4
- Topics for next lecture : Unsupervised Clustering

Machine Learning in Robotics

Lecture 4: Unsupervised Clustering

Prof. Dongheui Lee

Human-centered Assistive Robotics
Technische Universität München

dhlee@tum.de



<http://www.hcr.ei.tum.de/>



Supervised vs. Unsupervised learning

- Supervised learning
 - A pattern is a pair of variables $\{x, \omega\}$ where x is a collection of observations or features (feature vector) and ω is the concept behind the observation (label)
- Unsupervised learning
 - Use unlabeled data, a collection of feature vectors without the class label ω



Supervised vs. Unsupervised learning

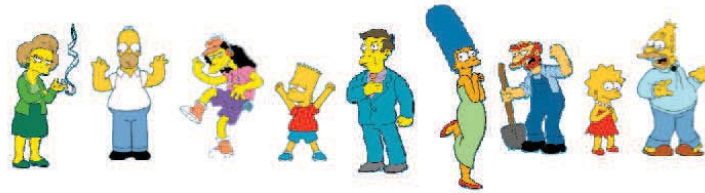
- Supervised learning
 - A pattern is a pair of variables $\{x, \omega\}$ where x is a collection of observations or features (feature vector) and ω is the concept behind the observation (label)
- Unsupervised learning
 - Use unlabeled data, a collection of feature vectors without the class label ω
 - These methods are called unsupervised because they are not provided the correct answer

Supervised vs. Unsupervised learning

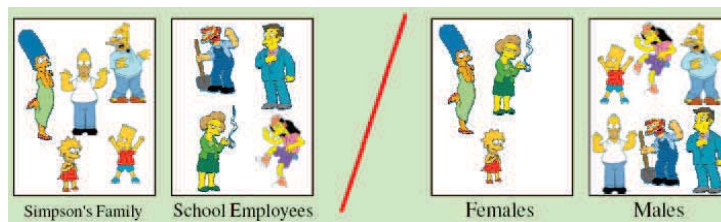
- Supervised learning
 - A pattern is a pair of variables $\{x, \omega\}$ where x is a collection of observations or features (feature vector) and ω is the concept behind the observation (label)
- Unsupervised learning
 - Use unlabeled data, a collection of feature vectors without the class label ω
 - These methods are called unsupervised because they are not provided the correct answer
 - Unsupervised methods may appear to have limited capabilities, but they are useful
 - ▶ Labeling large data sets can be a costly procedure
 - ▶ Class labels may not be known beforehand
 - ▶ Large datasets can be compressed by finding a small set of proto-types

TEST : Unsupervised clustering

What is the natural grouping among those objects?



Many possibilities!!!



Two approaches for unsupervised learning

- Parametric approaches
 - Functional forms for the underlying class-conditional densities are assumed, and we must estimate the parameters

$$p(\mathbf{x}|\theta) = \sum_{i=1}^K p(\mathbf{x}|\omega_i, \theta_i) p(\omega_i)$$

Two approaches for unsupervised learning

- Parametric approaches
 - Functional forms for the underlying class-conditional densities are assumed, and we must estimate the parameters

$$p(\mathbf{x}|\theta) = \sum_{i=1}^K p(\mathbf{x}|\omega_i, \theta_i) p(\omega_i)$$

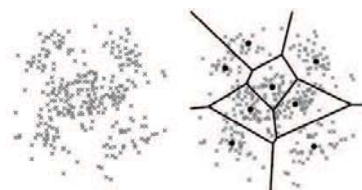
- Non-parametric approaches
 - No assumptions are made about the underlying densities
 - Instead, seek a partition of the data into clusters
 - These methods are typically referred to as clustering

Vector Quantization (VQ)

- Vector Quantization is a lossy data compression method
- Mapping n feature vectors $\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)}\}$ to K classes of feature vectors $\mathbf{Y} = \{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(K)}\}$

$$\mathbf{y}^{(j)} = c(\mathbf{x}^{(i)}), \quad j = 1, \dots, K; \quad i = 1, \dots, n; \quad K < n$$

- Code vector $\mathbf{y}^{(j)}$
 - Cluster centers
 - Points with high density in feature space
- Code book $\mathbf{Y} = \{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(K)}\}$
 - The set of all code vectors



Nonparametric clustering

Nonparametric clustering involves three steps:

- Defining a measure of (dis)similarity between examples
- Defining a criterion (Distortion) function for clustering
- Defining an algorithm to minimize (or maximize) a criterion (distortion) function

Similarity Measures

- A measuring rule of $d(x, y)$ for the distance between two vectors x and y is considered a metric if it satisfies the following properties.
 - non-negativity: $d(x, y) \geq 0$
 - reflexivity: $d(x, y) = 0$ if and only if $x = y$
 - symmetry: $d(x, y) = d(y, x)$
 - triangle inequality: $d(x, y) + d(y, z) \geq d(x, z)$

Similarity Measures

- The most general form of distance metric is the power norm

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^m |x_i - y_i|^q \right)^{1/q}$$

- where $q \geq 1$ is a selectable parameter - general Minkowski metric
- When $q = 2$, Euclidean metric $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\| = \sqrt{(\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y})}$
- When $q = 1$, Manhattan or city block metric - sum of the absolute distances along each of the m coordinate axes.
- Notice that the above distance metrics are measures of dissimilarity.

Similarity Measures

- Inner product

$$s(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^T \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$$

- When the features are binary-valued (0 or 1), the normalized inner product is a measure of the relative possession of common attributes.
- One variation is Tanimoto distance, a ratio of the number of shared attributes to the number possessed by \mathbf{x} or \mathbf{y} .

$$s(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^T \mathbf{y}}{\mathbf{x}^T \mathbf{x} + \mathbf{y}^T \mathbf{y} - \mathbf{x}^T \mathbf{y}}$$

Similarity measure



Criterion (Distortion) function

- Once a (dis)similarity measure has been determined, we need to define a criterion function to be optimized.
- This function measures the clustering quality of any partition of the data.
 - The most widely used criterion function for clustering is the *sum-of-square-error*

$$J = \sum_{i=1}^K \sum_{x \in \omega_i} (x - y^{(i)})^2 \quad \text{where } y^{(i)} = \frac{1}{N_i} \sum_{x \in \omega_i} x$$

Here, N_i is the number of samples in the ω_i

- This criterion measures how well the data set is represented by the cluster centers
- Other criterion functions exist, based on the scatter matrices used in Linear Discriminant Analysis (LDA).
 - Trace criterion $tr[S_w]$
 - Determinant criterion $|S_w|$
 - Invariant criterion $tr[S_T^{-1}S_w]$

Iterative optimization

- Once a criterion function has been defined, we must find a partition of the data set that minimizes the criterion.
- Exhaustive enumeration of all partitions, which guarantees the optimal solution, is unfeasible.
- Common approach is to proceed in an iterative fashion
 - Find reasonable initial partition
 - Move samples from one cluster to another in order to reduce the criterion function
- These iterative methods produce sub-optimal solution but are computationally tractable

Iterative optimization

- Once a criterion function has been defined, we must find a partition of the data set that minimizes the criterion.
- Exhaustive enumeration of all partitions, which guarantees the optimal solution, is unfeasible.
- Common approach is to proceed in an iterative fashion
 - Find reasonable initial partition
 - Move samples from one cluster to another in order to reduce the criterion function
- These iterative methods produce sub-optimal solution but are computationally tractable
- Approaches
 - K-means algorithm
 - LBG algorithm
 - Fuzzy K-means algorithm
 - Basic iterative minimum-squared-error clustering

k-means algorithm (Lloyd, 1982)

Cluster dataset $X = \{x^{(1)}, x^{(2)}, \dots, x^{(n)}\}$, given K

1. Initialization: Choose K random vectors $Y = \{y^{(1)}, y^{(2)}, \dots, y^{(K)}\}$ as an initial mean set
2. E-step: For each data point, find the closest class and label them. Dataset is divided into K classes

$$X_i = \{x^{(j)} \mid d(x^{(j)}, y^{(i)}) \leq d(x^{(j)}, y^{(k)}), j = 1, \dots, n; k = 1, \dots, K\}$$

3. M-step: From the current clusters, their mean vectors are updated

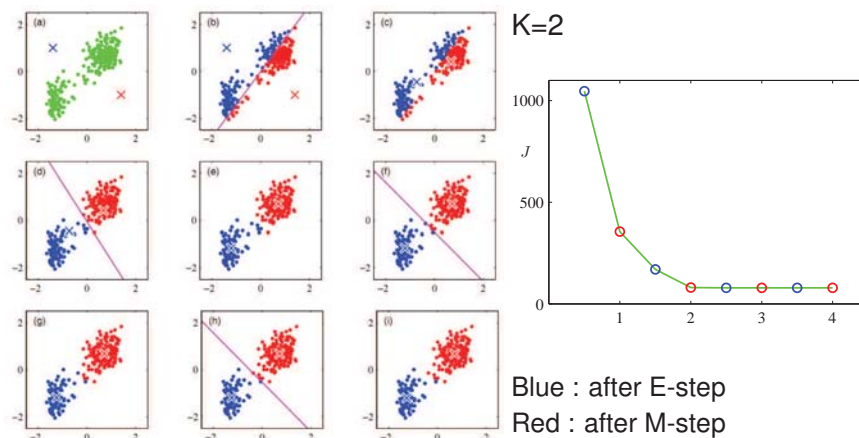
$$y^{(i)} = \frac{1}{N_i} \sum_{x \in \omega_i} x = \mathcal{C}(X_i)$$

4. Calculate the total distortion, the sum of the distance between each datapoint and its closest cluster mean.

$$J = \sum_{i=1}^K \sum_{x \in \omega_i} (x - y^{(i)})^2$$

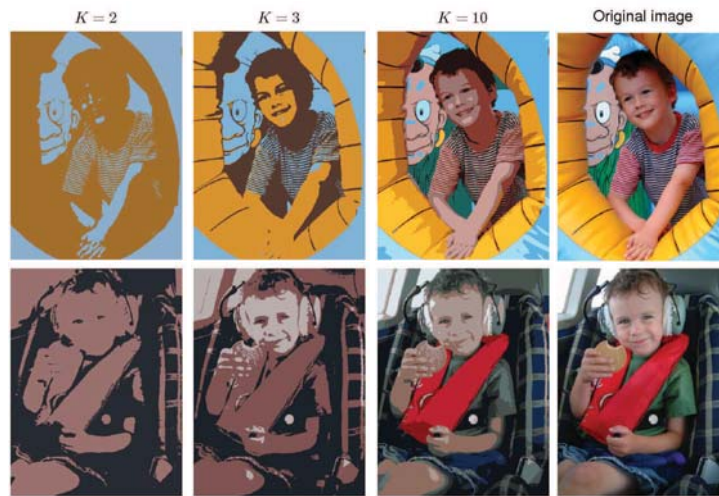
5. Evaluate the convergence. If converged, stop. Else, go to step 2.

Illustration of k-means algorithm



Source: C. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006

k-means clustering applications



Source: C. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006

k-means clustering: Remarks

- The way to initialize the means is not specified. One popular way to start is to randomly choose k of the samples.
- A local optimization algorithm
 - Converge to a local rather than global minimum of J
 - Convergence properties of the k-means algorithm [MacQueen 1967]
 - The results depend on the initial values for the means. The standard solution is to try a number of different starting points.
- Uniform search
- Slow convergence
- It can happen that the set of samples closest to $y^{(i)}$ is empty, so that $y^{(i)}$ cannot be updated.
- The results depend on the value of K .

Basic Iterative Minimum-Squared-Error Clustering

- A sequential version of the k-means clustering algorithm.
 - k-means procedure waits until all n samples have been reclassified before updating.
 - the Basic Iterative Minimum-Squared-Error Clustering updates after each sample is reclassified.
- Disadvantages
 - more susceptible to being trapped in local minima
 - results depend on the order in which the candidates are selected
- Merits
 - at least a stepwise optimal procedure
 - suitable to problems in which samples are acquired sequentially and clustering must be done online

Fuzzy k-means algorithm

- Allows one piece of data to belong to two or more clusters
- Minimizing the below objective function

$$J = \sum_{k=1}^K \sum_{i=1}^n P(\omega_k | \mathbf{x}^{(i)})^m (\mathbf{x}^{(i)} - \mathbf{y}^{(k)})^2$$
 - m is a parameter to adjust the blending of different clusters. If $m = 0$, k-means. For $m \geq 1$, the criterion allows belonging to multiple clusters
 - $P(\omega_k | \mathbf{x}^{(i)})$ is the degree of membership of $\mathbf{x}^{(i)}$ in the cluster k
- an iterative optimization of the objective function shown above, with the update of membership $P(\omega_k | \mathbf{x}^{(i)})$ and the cluster centers $\mathbf{y}^{(k)}$ by:

$$P(\omega_k | \mathbf{x}^{(i)}) = \frac{1}{\sum_{j=1}^K \left(\frac{\|\mathbf{x}^{(i)} - \mathbf{y}^{(k)}\|}{\|\mathbf{x}^{(i)} - \mathbf{y}^{(j)}\|} \right)^{\frac{2}{m-1}}}$$

$$\mathbf{y}^{(k)} = \frac{\sum_{i=1}^n P(\omega_k | \mathbf{x}^{(i)})^m \mathbf{x}^{(i)}}{\sum_{i=1}^n P(\omega_k | \mathbf{x}^{(i)})^m}$$

Fuzzy k-means algorithm : Pseudocode

Cluster dataset, given K

1. Initialization: Initialize K random mean vectors $Y = \{y^{(1)}, y^{(2)}, \dots, y^{(K)}\}$. Initialize $P(\omega_k | x^{(i)})$ for $i = 1..n, k = 1..K$
2. Normalize $P(\omega_k | x^{(i)})$ so that $\sum_k P(\omega_k | x^{(i)}) = 1$ for $i = 1..n$
3. Update mean vectors

$$y^{(k)} = \frac{\sum_{i=1}^n P(\omega_k | x^{(i)})^m x^{(i)}}{\sum_{i=1}^n P(\omega_k | x^{(i)})^m}$$

4. Update the degree of membership of $x^{(i)}$ in the cluster k

$$P(\omega_k | x^{(i)}) = \frac{1}{\sum_{j=1}^K \left(\frac{\|x^{(i)} - y^{(k)}\|}{\|x^{(i)} - y^{(j)}\|} \right)^{\frac{2}{m-1}}}$$

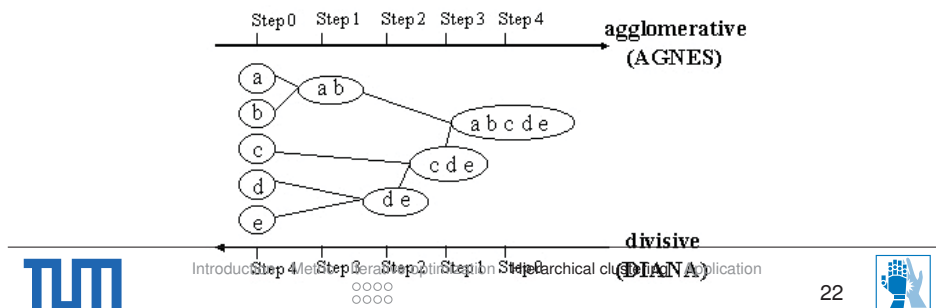
5. If $\max_{i,k} P(\omega_k | x^{(i)})$ is converged, then stop. Else, go to step 2.

Another variation of k-means

- ISODATA, which stands for Iterative Self-Organizing Data Analysis Technique (Algorithm) is an extension to the k-means algorithm with some heuristics to automatically select the number of clusters
- The algorithm works in an iterative fashion
 - (1) Perform k-means clustering
 - (2) Split any clusters whose samples are sufficiently dissimilar
 - (3) Merge any two clusters sufficiently close
 - (4) Go to (1)

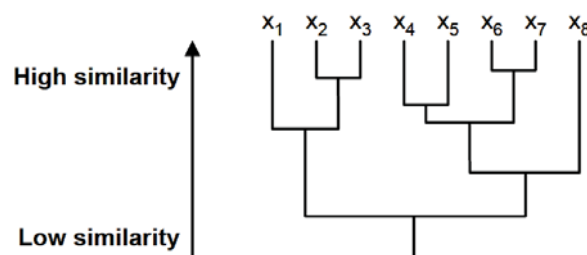
Hierarchical clustering

- k-means and ISODATA create disjoint clusters, resulting in a "flat" data representation
 - ▶ However, sometimes it is desirable to obtain a hierarchical representation of data, with clusters and sub-clusters arranged in a tree-structured fashion (i.e., biological taxonomy)
- Hierarchical clustering methods can be grouped in two general classes
 - ▶ Agglomerative (bottom-up, merging) : Starting with n singleton clusters, successively merge clusters until one cluster is left
 - ▶ Divisive (top-down, splitting) : Starting with a unique cluster, successively split the clusters until n singleton examples are left



Dendrograms

- The preferred representation for hierarchical clusters is the dendrogram
- The dendrogram is a binary tree that shows the structure of the clusters
 - ▶ In addition to the binary tree, the dendrogram provides the similarity measure between clusters (the vertical axis)
- An alternative representation is based on sets
 - ▶ $\{\{x_1, \{x_2, x_3\}\}, \{\{x_4, x_5\}, \{x_6, x_7\}\}, x_8\}$
 - ▶ However, unlike the dendrogram, sets cannot express quantitative information



Divisive clustering

1. Start with one large cluster
2. Find "worst" cluster
3. Split it
4. If $K < n$, go to step 2
 - How to choose the "worst" cluster
 - How to split clusters
 - The computations required by divisive clustering are more intensive than for agglomerative clustering methods.

Non-uniform Binary Split Algorithm

1. Initialization: Calculate a center for all data points. $k := 1$

$$\mathbf{y}^{(1)} = \mathcal{C}(X)$$

2. Choose a class which has the largest distortion among current classes

$$J_i \geq J_j, \quad \forall j = 1, \dots, k$$

3. Split the class into two subclasses by using a small random vector

$$X_a = \left\{ \mathbf{x} \text{ which is closer to } \mathbf{y}^{(i)} + \mathbf{v}_i \right\}$$

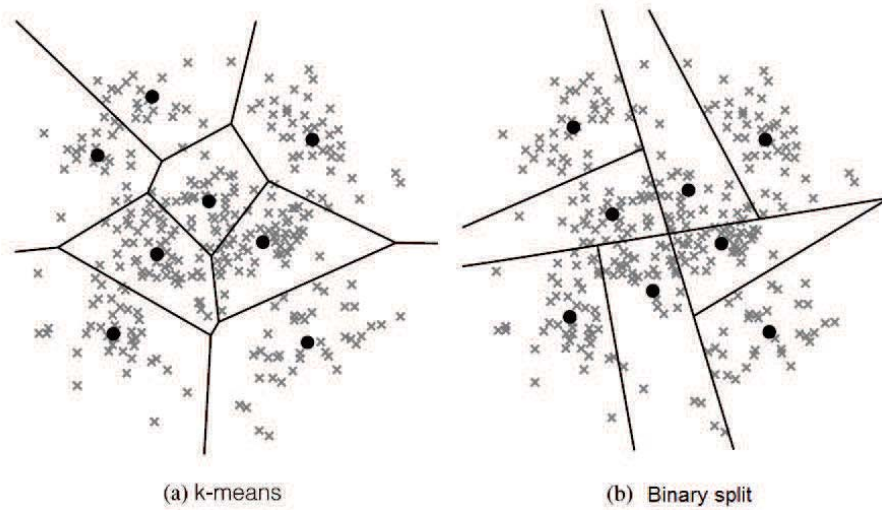
$$X_b = \left\{ \mathbf{x} \text{ which is closer to } \mathbf{y}^{(i)} - \mathbf{v}_i \right\}$$

4. Update the code vectors

$$\mathbf{y}^{(i)} := \mathcal{C}(X_a) \text{ and } \mathbf{y}^{(k+1)} := \mathcal{C}(X_b)$$

5. If $k = K$, stop. Else, $k := k + 1$ and go to step 2.

k-means vs. Non-uniform binary split



Non-uniform binary split: Not good quality of codebook. But very fast

LBG algorithm

- Combination of k-means and Binary split
- Proposed by Linde, Buzo, and Gray (1980)
- Instead of random initial codebook, let's use codebook from binary split
- Faster convergence and better quality codebook than k-means algorithm

Pseudo-code of LBG algorithm

1. Initialization: Calculate a center for all data points. $k := 1$

$$\mathbf{y}^{(1)} = \mathcal{C}(\mathbf{X})$$

2. Splitting: For each cluster, split it into two subclasses. $k := 2k$

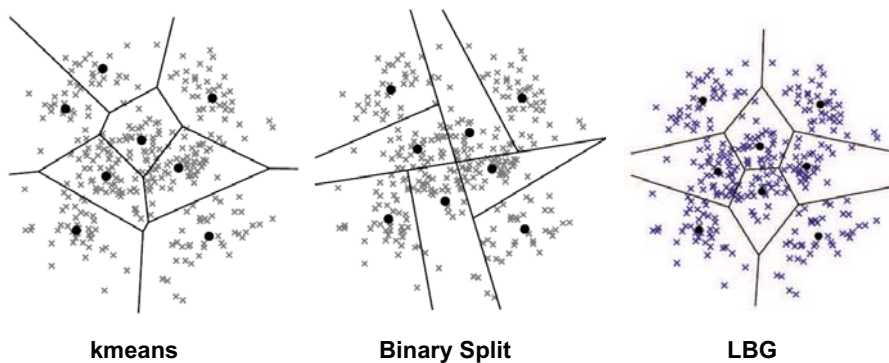
$$\mathbf{y}_a^{(i)} = \mathbf{y}^{(i)} + \mathbf{v}_i \text{ and } \mathbf{y}_b^{(i)} = \mathbf{y}^{(i)} - \mathbf{v}_i$$

- ### 3. Iteration (k-means algorithm)

- a E-step: For each data point, find the closest cluster which achieves the minimum distance measure
- b M-step: From the current cluster, update their mean vectors
 $\mathbf{y}^{(i)} = \mathcal{C}(\mathbf{X}_i)$
- c Iterate E-step and M-step until it converges

4. If the desired number of code vectors is obtained ($k = K$), stop. Else, go to step 2.

Comparison



Agglomerative clustering

1. Start with n singleton cluster
 2. Find nearest clusters
 3. Merge them
 4. If $K > 1$, go to step 2
- How to find the "nearest" pair of clusters
 - ▶ Minimum distance $d_{\min}(\mathbf{X}_i, \mathbf{X}_j) = \min_{\mathbf{x} \in \omega_i, \mathbf{x}' \in \omega_j} \|\mathbf{x} - \mathbf{x}'\|$
 - ▶ Maximum distance $d_{\max}(\mathbf{X}_i, \mathbf{X}_j) = \max_{\mathbf{x} \in \omega_i, \mathbf{x}' \in \omega_j} \|\mathbf{x} - \mathbf{x}'\|$
 - ▶ Average distance $d_{\text{avg}}(\mathbf{X}_i, \mathbf{X}_j) = \frac{1}{N_i N_j} \sum_{\mathbf{x} \in \omega_i} \sum_{\mathbf{x}' \in \omega_j} \|\mathbf{x} - \mathbf{x}'\|$
 - ▶ Mean distance $d_{\text{mean}}(\mathbf{X}_i, \mathbf{X}_j) = \|\mathbf{y}^{(i)} - \mathbf{y}^{(j)}\|$

Agglomerative clustering

Minimum distance

- When d_{min} is used to measure distance between clusters, the algorithm is called the *nearest neighbor* or *single-linkage* clustering algorithm
- If the algorithm is allowed to run until only one cluster remains, the result is a minimum spanning tree (MST)

Maximum distance

- When d_{max} is used to measure distance between clusters, the algorithm is called the *farthest neighbor* or *complete-linkage* clustering algorithm
- From a graph-theoretic point of view, each cluster constitutes a complete sub-graph

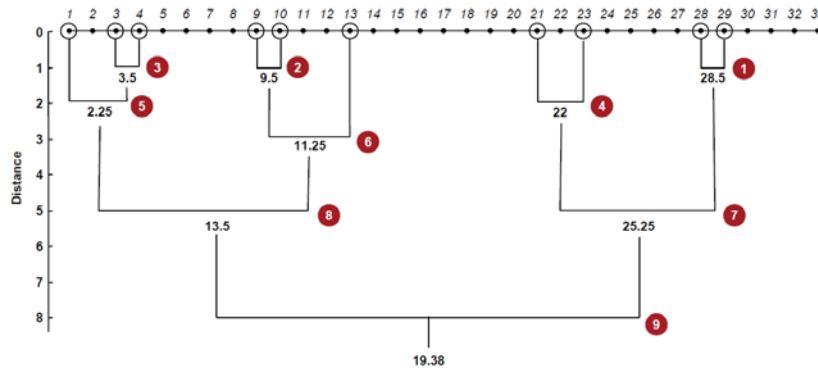
Average and mean distance

- The average and mean distance approaches are more robust to outliers
- Of the two, the mean distance is computationally more attractive, since Notice that the average distance approach involves the computation of $N_i N_j$ distances for each pair of clusters

Agglomerative clustering example

Perform agglomerative clustering on the following dataset using the single-linkage metric

- $X = \{1, 3, 4, 9, 10, 13, 21, 23, 28, 29\}$
- In case of ties, always merge the pair of clusters with the largest mean
- Indicate the order in which the merging operations occur

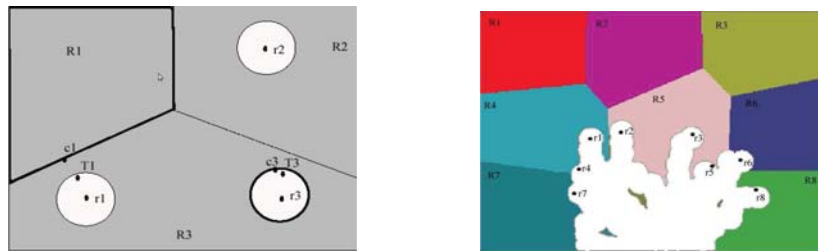


Multi-Robot Exploration


- Task definition
 - Exploration of unknown areas by means of k mobile robots.
 - Application: Search/Rescue robot, planetary exploration, reconnaissance
- Problems
 - Reduce the difference of waiting time among different regions of a workspace.
 - Ensure a balanced exploration of the environment

Wu L., Puig D., and Garcia M. A. *Balanced Multi-Robot Exploration through a Global Optimization Strategy*. Journal of Physical Agents. 2010.

Multi-Robot Exploration




- K-means is used to partition an unexplored space in regions.
- Each region is assigned to a robot solving an optimization problem.

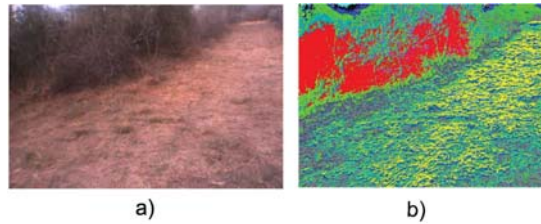
 Wu L., Puig D., and Garcia M. A. *Balanced Multi-Robot Exploration through a Global Optimization Strategy*. Journal of Physical Agents. 2010.

Outdoor Robots Navigation


- Task definition
 - Autonomous navigation for outdoor, unstructured environments.
 - Recognize navigable terrain and avoid obstacles, based on their appearance.
- Problem
 - Do reliable segmentation of outdoor scenes in an efficient manner (online).

 Blas M. R., Agrawal M., Sundaresan A., and Konolige K. *Fast Color/Texture Segmentation For Outdoor Robots*. IEEE International Conference on Intelligent Robots and Systems. 2008.

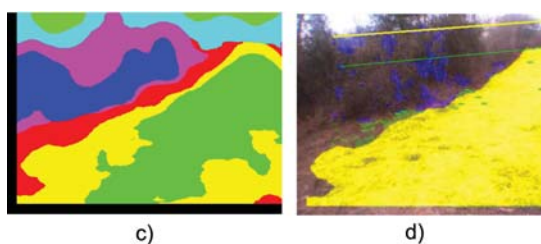
Outdoor Robots Navigation




- Use compact texture/color descriptors (feature vectors) and fast unsupervised clustering algorithms.
- K-means is used to cluster neighborhood feature vectors in a small set of basis vectors (*textons*).
- Each pixel is classified as belonging to one cluster using Euclidean distance (b).

 Blas M. R., Agrawal M., Sundaresan A., and Konolige K. *Fast Color/Texture Segmentation For Outdoor Robots*. IEEE International Conference on Intelligent Robots and Systems. 2008.

Outdoor Robots Navigation



- Extract an histogram counting textons in a neighborhood of each pixel
- K-means to extract a set of histogram profiles ($k = 8$).
- *Earth Movers Distance* is used to merge similar clusters (c).

 Blas M. R., Agrawal M., Sundaresan A., and Konolige K. *Fast Color/Texture Segmentation For Outdoor Robots*. IEEE International Conference on Intelligent Robots and Systems. 2008.

Summary and Next Lecture

- References
 - Duda Chap. 10.4, 10.6-9
 - Bishop Chap. 9.1
 - Michell Chap. 6.12
- Next Lecture
 - Maximum Likelihood Estimation
 - Gaussian Mixture Model



Machine Learning in Robotics Dimensionality Reduction

Prof. Dongheui Lee

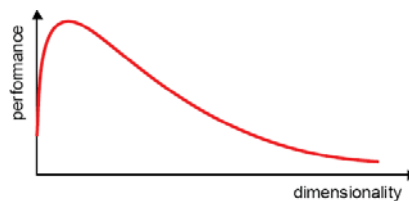
*Human-centered Assistive Robotics
Technische Universität München*

dhlee@tum.de



The curse of dimensionality

- The problems associated with multivariate data analysis as the dimensionality increases
- Toy example : 3-class pattern recognition or an approach which divide the sample space into equally spaced bins
 - 1D \rightarrow 3 bins \rightarrow 3^2 examples
 - 2D \rightarrow 3^2 bins \rightarrow 3^3 examples
 - 3D \rightarrow 3^3 bins \rightarrow 3^4 examples
- For a given sample size, there is a maximum number of features above which the performance of our classifier will degrade rather than improve



The curse of dimensionality

Implication of the curse of dimensionality

- Exponential growth in the number of examples, required to maintain a given sampling density
- Exponential growth in the complexity of the target function (a density estimate) with increasing dimensionality
- Humans have an extraordinary capacity to discern patterns in 1~3D. But these capability degrades drastically for higher dimensions

Why dimensionality reduction?

- Learning a target function from data where some features are irrelevant \rightarrow reduce variance. Improve accuracy
- Wish to visualize high dimensional data
- Intrinsic dimensionality of data is smaller than the number of features used to describe it

Dimensionality Reduction

Two approaches

- Feature extraction: creating a subset of new features by combinations of the existing features $m > k$

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \rightarrow \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} = f \left(\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} \right)$$

- Feature selection: choosing a subset of all the features (the ones more informative)

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \rightarrow \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ \vdots \\ x_{i_k} \end{bmatrix}$$

Feature Extraction Problem

- Given a feature space, to find a mapping $y = f(x)$ such that the reduced feature vector y preserves most of the information or structure of original feature x
- The optimal mapping function $f(x)$ will result in no increase in the minimum probability of error
- In general, the mapping function can be a nonlinear function. But, often feature extraction is limited to linear transforms. $y = W^T x$

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \rightarrow \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1m} \\ u_{21} & u_{22} & \cdots & u_{2m} \\ & & \ddots & \\ u_{k1} & u_{k2} & \cdots & u_{km} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

Two Approaches for Feature Extraction

Unsupervised approaches

- Principal Components Analysis
- Singular Value Decomposition
- Independent Components Analysis

Supervised approaches

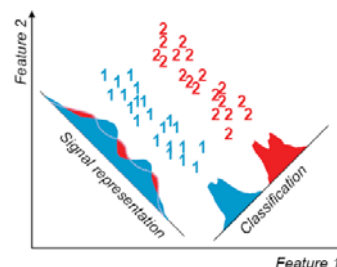
- Fisher Linear Discriminant Analysis (LDA)
- Hidden Layers of Neural Networks

Feature Extraction

The selection of feature extraction mapping $y = f(x)$ is guided by an objective function to maximize (or minimize)

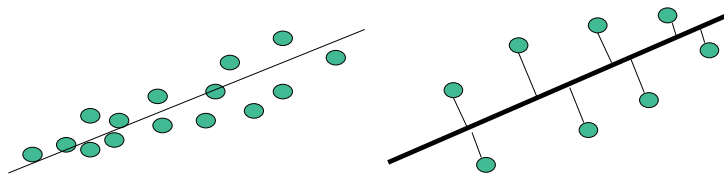
Depending on the criteria measures by objective function. Two categories

- Signal representation
 - Goal : to represent samples accurately
 - Principal Components Analysis (PCA)
- Classification
 - Goal : to enhance the class-discriminatory information
 - Linear Discriminant Analysis (LDA)



Principal Components Analysis (PCA)

- Given data points in m -dimensional space, project into lower dimensional space while preserving as much information as possible \Rightarrow Choose a line that fits the data so the points are spread out well along the line
- Seeks a projection that best represent the data in a least-square sense \Rightarrow minimize sum of squares of distances to the line
- We wish to explain/summarize the underlying variance-covariance structure of a large set of variables through a few linear combinations of these variables



Space Representation

- High-dimensional space representation x
- Low-dimensional space representation y

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix}$$

- PCA allows us to compute a linear transformation that maps data from a high dimensional space to a lower dimensional sub-space

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \rightarrow \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1m} \\ u_{21} & u_{22} & \cdots & u_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ u_{k1} & u_{k2} & \cdots & u_{km} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

Maximum variance formulation

- Goal: to project the data onto a space having dimensionality $k < m$ while maximizing the variance of the projected data
- Begin with $k = 1$
- m -dimensional unit vector $\mathbf{u}_1 = [u_{11} \ u_{12} \ \cdots \ u_{1m}]^T$, $\mathbf{u}_1^T \mathbf{u}_1 = 1$
- Each data point $\mathbf{x}^{(i)}$ is then projected onto a scalar value $y^{(i)} = \mathbf{u}_1^T \mathbf{x}^{(i)}$
- Mean, covariance of the original data
 $\boldsymbol{\mu} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}^{(i)}$, $\mathbf{S} = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}^{(i)} - \boldsymbol{\mu})(\mathbf{x}^{(i)} - \boldsymbol{\mu})^T$
- mean of the projected data is $\mathbf{u}_1^T \boldsymbol{\mu}$
- variance of the projected data $\frac{1}{n} \sum_{i=1}^n \{\mathbf{u}_1^T \mathbf{x}^{(i)} - \mathbf{u}_1^T \boldsymbol{\mu}\}^2 = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$
- Maximize the projected variance $\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$ wrt \mathbf{u}_1

$$\frac{\partial}{\partial \mathbf{u}_1} (\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^T \mathbf{u}_1)) = 0, \quad \mathbf{S} \mathbf{u}_1 = \lambda_1 \mathbf{u}_1$$

- Variance will be a maximum value when \mathbf{u}_1 is equal to the eigenvector having the largest eigenvalue λ_1
- When k increases, choose $\mathbf{u}_1 \dots \mathbf{u}_k$ as corresponding eigenvectors to k largest eigenvalues $\lambda_1, \lambda_2 \dots \lambda_k$

PCA in General

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \rightarrow \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1m} \\ u_{21} & u_{22} & \cdots & u_{2m} \\ & & \ddots & \\ u_{k1} & u_{k2} & \cdots & u_{km} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

- $\{\mathbf{u}_j\}$ are orthogonal basis vectors.
- $\mathbf{u}_1 = [u_{11} \ u_{12} \ \cdots \ u_{1m}]^T$ is 1st eigenvector of covariance matrix, and known as the first principal component. This axis explains as much as possible of original variance in data set
- $\mathbf{u}_2 = [u_{21} \ u_{22} \ \cdots \ u_{2m}]^T$ is 2nd eigenvector of covariance matrix, and the 2nd principal component
- $\mathbf{u}_k = [u_{k1} \ u_{k2} \ \cdots \ u_{km}]^T$ is k th eigenvector of covariance matrix, and the k th principal component

PCA algorithm

1. Compute the mean and covariance matrix

$$\boldsymbol{\mu} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}^{(i)}, \quad \mathbf{S} = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}^{(i)} - \boldsymbol{\mu})(\mathbf{x}^{(i)} - \boldsymbol{\mu})^T$$

2. Find eigenvectors and eigenvalues of \mathbf{S}
3. Choose the k largest eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_k$
4. Choose the corresponding eigenvectors $\mathbf{u}_1, \dots, \mathbf{u}_k$ and make transformation matrix $\mathbf{W} = [\mathbf{u}_1 \mathbf{u}_2 \dots]$
5. Project original data $\mathbf{y} = \mathbf{W}^T \mathbf{x}$

PCA algorithm: Alternate formulation

Often the data is made zero mean as a preprocess step:

1. Compute the mean

$$\boldsymbol{\mu} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}^{(i)}$$

2. Replace each $\mathbf{x}^{(i)}$ by $\mathbf{x}^{(i)} - \boldsymbol{\mu}$ (preprocessing step).
3. Calculate the covariance matrix

$$\mathbf{S} = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}^{(i)})(\mathbf{x}^{(i)})^T$$

4. Find eigenvectors and eigenvalues of \mathbf{S}
5. Choose the k largest eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_k$
6. Choose the corresponding eigenvectors $\mathbf{u}_1, \dots, \mathbf{u}_k$ and make transformation matrix $\mathbf{W} = [\mathbf{u}_1 \mathbf{u}_2 \dots]$
7. Project the zero mean data $\mathbf{y} = \mathbf{W}^T \mathbf{x}$
8. For a new \mathbf{x}' , PCA projection will be $\mathbf{y}' = \mathbf{W}^T (\mathbf{x}' - \boldsymbol{\mu})$

Principal Components Analysis

- Principal Components Analysis is the oldest technique in multivariate analysis
- PCA is also known as the Karhunen-Loève transform
- PCA was first introduced by Pearson in 1901, and generalized by Loève in 1963
- The main limitation of PCA is that it does not consider class separability since it does not take into account the class label of the feature vector
 - PCA simply performs a coordinate rotation that aligns the transformed axes with the directions of maximum variance
 - There is no guarantee that the directions of maximum variance will contain good features for discrimination

PCA - simple example

Compute the principal components for the following two-dimensional dataset

$X = (x_1, x_2) = (1, 2), (3, 3), (3, 5), (5, 4), (5, 6), (6, 5), (8, 7), (9, 8)$

Solution

- The (biased) covariance estimate of the data is

$$S = \begin{bmatrix} 6.25 & 4.25 \\ 4.25 & 3.5 \end{bmatrix}$$

- The eigenvalues are the zeros of the characteristic equation

$$Su = \lambda u \Rightarrow |S - \lambda I| = 0 \Rightarrow \lambda_1 = 9.34, \lambda_2 = 0.41$$

- The eigenvectors are the solutions of the system

$$\begin{bmatrix} 6.25 & 4.25 \\ 4.25 & 3.5 \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{12} \end{bmatrix} = \begin{bmatrix} \lambda_1 u_{11} \\ \lambda_1 u_{12} \end{bmatrix} \Rightarrow \begin{bmatrix} u_{11} \\ u_{12} \end{bmatrix} = \begin{bmatrix} 0.81 \\ 0.59 \end{bmatrix}$$

$$\begin{bmatrix} 6.25 & 4.25 \\ 4.25 & 3.5 \end{bmatrix} \begin{bmatrix} u_{21} \\ u_{22} \end{bmatrix} = \begin{bmatrix} \lambda_2 u_{21} \\ \lambda_2 u_{22} \end{bmatrix} \Rightarrow \begin{bmatrix} u_{21} \\ u_{22} \end{bmatrix} = \begin{bmatrix} -0.59 \\ 0.81 \end{bmatrix}$$

Linear Discriminant Analysis

- Multiple classes and PCA
 - Suppose there are C classes in the training data
 - PCA is based on the sample covariance which characterizes the scatter of the entire data set, irrespective of class-membership
 - The projection axes chosen by PCA might not provide good discrimination power
- What is the goal of LDA?
 - Perform dimensionality reduction while preserving as much of the class discriminatory information as possible
 - Seeks to find directions along which the classes are best separated
 - Takes into consideration the scatter within-classes but also the scatter between-classes
 - More capable of distinguishing image variation due to identity from variation due to other sources such as illumination and expression

Linear Discriminant Analysis

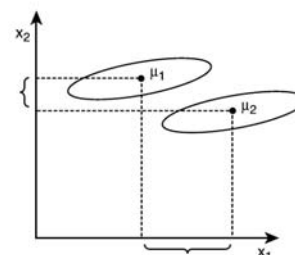
In order to find a good projection vector, we need to define a measure of separation between the projected classes

- Projection $\mathbf{y} = \mathbf{W}^T \mathbf{x}$
- The mean of original and projected dataset

$$\boldsymbol{\mu}_j = \frac{1}{n_j} \sum_{\mathbf{x} \in X_j} \mathbf{x} \quad , \quad \tilde{\boldsymbol{\mu}}_j = \frac{1}{n_j} \sum_{\mathbf{y} \in Y_j} \mathbf{y} = \frac{1}{n_j} \sum_{\mathbf{x} \in X_j} \mathbf{W}^T \mathbf{x} = \mathbf{W}^T \boldsymbol{\mu}_j$$
- The case of two classes, the distance between the projected means

$$|\tilde{\boldsymbol{\mu}}_1 - \tilde{\boldsymbol{\mu}}_2| = |\mathbf{W}^T (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)|$$

⇒ Not so good measure, since it does not take into account the standard deviation within the class.



Fisher Linear Discriminant Analysis

- Proposed by Ronald Aylmer Fisher
- To maximize a function that represents the difference between means normalized by a measure of the within-class scatter

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

- Define the scatter matrix for each class
 - Original space $\mu_i = \frac{1}{n_i} \sum_{x \in X_i} \mathbf{x}$, $S_i = \sum_{x \in X_i} (\mathbf{x} - \mu_i)(\mathbf{x} - \mu_i)^T$
 - projected space $\tilde{\mu}_i = \frac{1}{n_i} \sum_{x \in X_i} \mathbf{w}^T \mathbf{x}$, $\tilde{S}_i = \sum_{x \in X_i} \mathbf{w}^T (\mathbf{x} - \mu_i)(\mathbf{x} - \mu_i)^T \mathbf{w} = \mathbf{w}^T S_i \mathbf{w}$
- Within-class scatter matrix $S_W = S_1 + S_2$, $\tilde{S}_1 + \tilde{S}_2 = \mathbf{w}^T S_W \mathbf{w}$
- Between-class scatter matrix $S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$, $(\tilde{\mu}_1 - \tilde{\mu}_2)^2 = \mathbf{w}^T S_B \mathbf{w}$
- The criterion function $J(\mathbf{w}) = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}}$

Linear Discriminant Analysis

Maximize $J(\mathbf{w})$

$$\frac{\partial}{\partial \mathbf{w}} [J(\mathbf{w})] = \frac{\partial}{\partial \mathbf{w}} \left[\frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}} \right] = 0$$

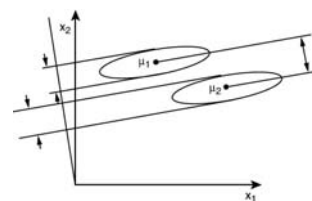
\vdots

$$S_W^{-1} S_B \mathbf{w} = J \mathbf{w}$$

$$\mathbf{w}^* = \arg \max_{\mathbf{w}} \left[\frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}} \right] = S_W^{-1} (\mu_1 - \mu_2)$$

Properties of LDA

- LDA is a parametric method since it assumes unimodal Gaussian likelihoods
- If the distributions are significantly non-Gaussian, the LDA projections will not be able to preserve any complex structure of the data that may be needed for classification



Independent component analysis

Source separation : Cocktail Party Problem



- separate the mixed signal into sources.

Independent component analysis

Source separation : Cocktail Party Problem



- separate the mixed signal into sources.
- Assumption: different sources are **independent**

Demo: http://research.ics.aalto.fi/ica/cocktail/cocktail_en.cgi

Independent component analysis

$$x = As \text{ which implies } s = Wx \text{ where } W = A^{-1}$$

- x : observations
- s : the independent components
- A : the **mixing matrix**
- W : the **unmixing matrix**.
- Goal: to recover the sources $s^{(i)}$ that had generated our data

$$x^{(i)} = As^{(i)}$$

Independent component analysis

- the distribution of each source $s_i : p_s$. By the condition of independence,

$$p(s) = \prod_{i=1}^m p_s(s_i) \quad (1)$$

If $x = As$, the density of x is given as $p_x(x) = p_s(Wx) \cdot |W|$.

$$p(x) = \prod_{i=1}^m p_s(w_i^T x) \cdot |W| \quad (2)$$

- A choice of cdf can be a sigmoid function $g(s) = \frac{1}{1+\exp(-s)}$. $p_s(s) = g'(s)$
- Learn W that maximizes the log-likelihood function for a given dataset $\{x^{(i)}\}$

$$l(W) = \sum_{i=1}^n \left(\sum_{j=1}^m \log g'(w_j^T x^{(i)}) + \log |W| \right) \quad (3)$$

For a training sample $x^{(i)}$, the gradient ascent rule:

$$W := W + \alpha \begin{pmatrix} 1 - 2g(w_1^T x^{(i)}) \\ 1 - 2g(w_2^T x^{(i)}) \\ \vdots \\ 1 - 2g(w_n^T x^{(i)}) \end{pmatrix} x^{(i)} + (W^T)^{-1} \quad (4)$$

PCA application: Face recognition

Eigenfaces for face detection/recognition

- Template matching problem
- Difficult to perform recognition in a high-dimensional space
- Improvements can be achieved by mapping data into a lower dimensionality space

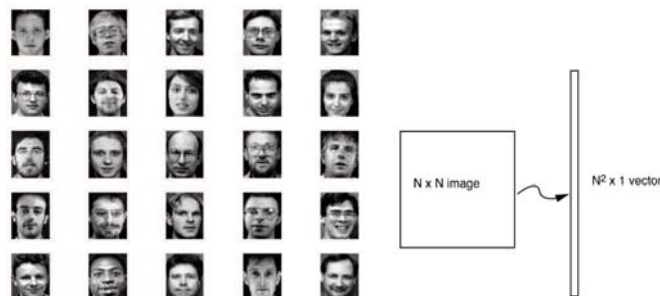
Data is projected into a lower dimensional space using PCA.



 M. Turk, A. Pentland, *Eigenfaces for Recognition*. Journal of Cognitive Neuroscience, 3(1), pp. 71-86, 1991.

PCA application: Face recognition

- Step 1. Obtain face images (training data set)
- Step 2. Preprocess the face images (centered and same size) and represent each image as a vector $I^{(i)} \rightarrow x^{(i)}, \forall i = 1, \dots, n$



PCA application: Face recognition

- Step 3. Compute the average face

$$\mu = \frac{1}{n} \sum_{i=1}^n \mathbf{x}^{(i)}$$



- Step 4. Compute the covariance matrix

$$S = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}^{(i)} - \mu)(\mathbf{x}^{(i)} - \mu)^T$$

- Step 5. Compute eigenvalues of the covariance matrix
- Step 6. Choose the k largest eigenvalues and their corresponding eigenvectors

PCA application: Face recognition

- Now, we can represent each face as a linear combination of the best k eigenvectors
- Eigenfaces: eigenvectors of face dataset



PCA application: Face recognition

- Step 7. For face recognition of a new image x , normalize the input image. Project on the eigenspace.

$$y_i = u_i^T x \quad \forall i = 1, \dots, k \quad y = [y_1 \dots y_k]^T$$

- Step 8. Compare the projected input image y with face classes y_j . Find the class which results in smallest (Euclidean / Mahalanobis) distance. If the minimum Euclidean distance is smaller than a threshold, the image is recognized as the face class.
If $\min \|y - y_j\| < \tau$, y belongs to the j -th face class.

PCA application: Object Grasping

- The PCA analysis of grasping data reveals that 87% of objects can be grasped by the first three synergies (eigen-postures) of the hand, which lead to the design and control of an underactuated gripper.

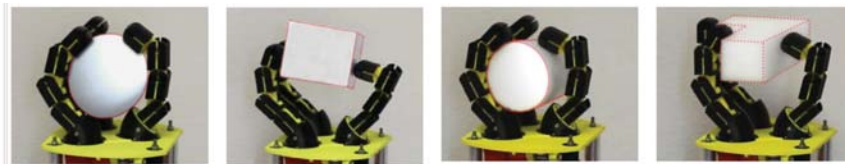


Figure: Hand prototype during the execution of some simple grasps. Only one synergy is used to grasp four test objects. Despite this, depending by the actuation variable, grasped object shape, contact point positions and internal forces, different grasp configurations can be achieved.



Catalano, Manuel G., et al. *Adaptive synergies for the design and control of the Pisa/IIT SoftHand*. The International Journal of Robotics Research 33.5 (2014): 768-782.

PCA application: Object Grasping

- Their approach enables to adopt a model of the hand with a number of independent actuators that is smaller than the number of joints.
- Shape adaptation in underactuated hands using differential transmission e.g. with gears, closed-chain mechanisms or pulleys etc.
- The shape adaptation of the gripper comes from the kinematic model which has non-uniqueness of the shape attained by an underactuated gripper

PCA application: Object Grasping

A total of 107 objects of different shapes were successfully grasped with the 19-joints hand by using only a single actuator.

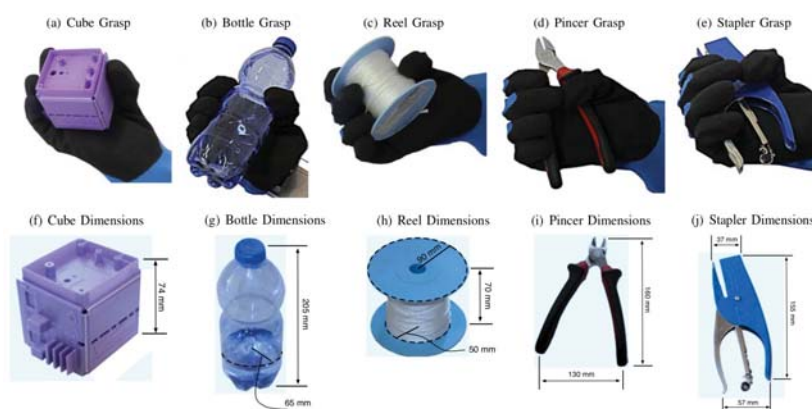


Figure: Some experimental grasps performed with the Pisa/IIT hand, with the object placed in the hand by a human operator.

LDA Application: Emotion recognition

- Biased linear discriminant analysis (BLDA) method that impose large penalties on interclass samples with small differences and small penalties on those samples with large differences

$$S_B = g(i,j)(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$$



Figure: Examples of the original, well-aligned, and misaligned images of one subject from the (upper half) Cohn-Kanade and (lower half) JAFFE database. From left to right are the facial images with anger, disgust, fear, happy, neutral, sad, and surprise expressions, respectively.

Haibin Yan, Marcelo H. Ang Jr, and Aun Neow Poo, *Weighted biased linear discriminant analysis for misalignment-robust facial expression recognition*. ICRA 2011.

LDA Application

- LDA is a supervised subspace learning approach which searches for a set of most discriminative projections to maximize the ratio of between-class variance to within-class variance simultaneously
- When the class information is available, LDA usually outperforms PCA for classification tasks

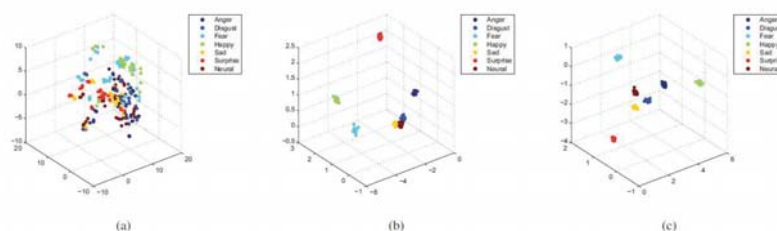


Figure: The projections of the first three components of the original data on the (a) PCA (b) LDA and (c) BLDA feature spaces respectively.

LDA Application

- LDA is a supervised subspace learning approach which searches for a set of most discriminative projections to maximize the ratio of between-class variance to within-class variance simultaneously
- When the class information is available, LDA usually outperforms PCA for classification tasks

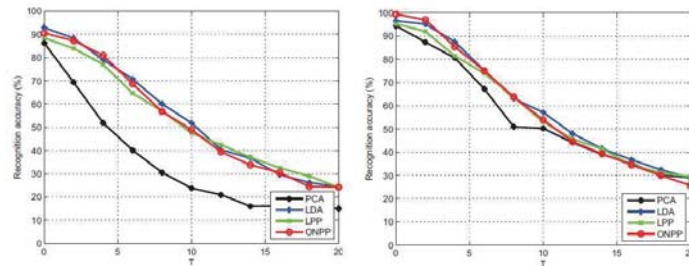


Figure: Recognition accuracy versus different amounts of spatial misalignments. (a) Results obtained on the Cohn-Kanade database. (b) Results obtained on the JAFFE database.

Separating reflections using Independent Components Analysis

A mixed image of a painting and an object is represented as $y_1 = aP + bR$, by photographing through a linear polarizer, the relative strength of the reflections can be adjusted.

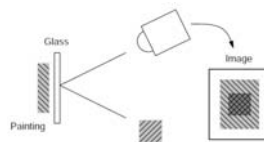


Figure: A photograph of a painting behind glass contains a superposition of the light reflected by the painting and the light reflected directly off the glass.



Farid, Hany, and Edward H. Adelson, *Separating reflections and lighting using independent components analysis*. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol.1, 1999.

Separating reflections using Independent Components Analysis



Figure: Mixture of two images as input for ICA.



Figure: Output produced by ICA.

Announcements

- Further Reading
 - Duda: Chap 3.7, 3.8.1, 3.8.2, 10.13
 - Bishop: Chap. 12.1-12.3, 12.4.1
 - PCA: Tutorial by J. Schlens

Machine Learning in Robotics

Density Estimation - MLE

Prof. Dongheui Lee

*Institute of Automatic Control Engineering
Technische Universität München*

dhlee@tum.de



<http://www.lsr.ei.tum.de/>



Density Estimation - Motivation

- In our previous lecture on decision theory, we saw that the optimal classifier could be expressed as a family of discriminant functions

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

- Decision rule was

choose ω_i if $g_i(\mathbf{x}) > g_j(\mathbf{x}), \forall j \neq i$

- We need to estimate both prior $p(\omega_i)$ and likelihood $p(\mathbf{x}|\omega_i)$
- During next lectures, techniques to estimate the likelihood density function $p(\mathbf{x}|\omega_i)$ will be introduced



Approaches for Density Estimation

Parametric Approach

- Assumes a particular form for the density (e.g., Gaussian) and estimates the parameters of the function (e.g., mean and covariance)
- called Parameter Estimation
 - Maximum Likelihood
 - Bayesian Estimation

Non-Parametric Approach

- No functional form for the density function is assumed. The density estimate is driven entirely by the data
- called Parameter Estimation
 - Kernel Density Estimation
 - Nearest Neighbor Rule

Maximum Likelihood Estimation (MLE)

- Consider a pdf $p(\mathbf{x}|\theta)$ which depends on a set of parameters $\theta = [\theta_1 \dots \theta_p]^T$
- If examples in the data set are drawn independently from $p(\mathbf{x}|\theta)$, the joint probability density of the entire set is given by

$$L(\theta) = p(\mathbf{X}|\theta) = \prod_{i=1} p(\mathbf{x}^{(i)}|\theta)$$

Maximum Likelihood Estimation (MLE)

- Consider a pdf $p(x|\theta)$ which depends on a set of parameters $\theta = [\theta_1 \dots \theta_p]^T$
- If examples in the data set are drawn independently from $p(x|\theta)$, the joint probability density of the entire set is given by

$$L(\theta) = p(X|\theta) = \prod_{i=1} p(x^{(i)}|\theta)$$

- We seek the set of parameters θ_{ML} that maximize the likelihood, and call it the Maximum Likelihood estimate of parameters $\theta_{ML} = \operatorname{argmax}_{\theta} p(X|\theta)$

Maximum Likelihood Estimation (MLE)

- Consider a pdf $p(x|\theta)$ which depends on a set of parameters $\theta = [\theta_1 \dots \theta_p]^T$
- If examples in the data set are drawn independently from $p(x|\theta)$, the joint probability density of the entire set is given by

$$L(\theta) = p(X|\theta) = \prod_{i=1} p(x^{(i)}|\theta)$$

- We seek the set of parameters θ_{ML} that maximize the likelihood, and call it the Maximum Likelihood estimate of parameters $\theta_{ML} = \operatorname{argmax}_{\theta} p(X|\theta)$
- It is easier to work with the logarithm of the likelihood

$$l(\theta) = \ln p(X|\theta) = \sum_{i=1}^n \ln p(x^{(i)}|\theta)$$

$$\nabla_{\theta} l(\theta) = \mathbf{0} \implies \theta_{ML}$$

Maximum Likelihood Estimation (MLE)

- Has good convergence properties as the sample size increases
- Simpler than any other alternative techniques
- Why to use the logarithm of the likelihood?
- Summary
 - We define $l(\theta)$ as the log-likelihood function $l(\theta) = \ln p(X|\theta)$
 - Solve

$$\nabla_{\theta} l(\theta) = \left[\frac{\partial l}{\partial \theta_1}, \dots, \frac{\partial l}{\partial \theta_p} \right]^T = \mathbf{0}$$

MLE Example 1: Gaussian with unknown mean

- The estimation of the parameters of a Gaussian distribution
 $p(\mathbf{x}^{(i)}|\boldsymbol{\mu}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
- Find the ML estimates
 - Parameters: $\theta = \boldsymbol{\mu}$

$$p(\mathbf{x}^{(i)}|\boldsymbol{\mu}) = \frac{1}{(2\pi)^{m/2} |\boldsymbol{\Sigma}|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{x}^{(i)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu}) \right]$$

$$\ln p(\mathbf{x}^{(i)}|\boldsymbol{\mu}) = -\frac{1}{2} \ln((2\pi)^m |\boldsymbol{\Sigma}|) - \frac{1}{2} (\mathbf{x}^{(i)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu})$$

- ML estimates must satisfy

$$\nabla_{\theta} \ln p(X|\theta) = \sum_{i=1}^n \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu}) = \mathbf{0}$$

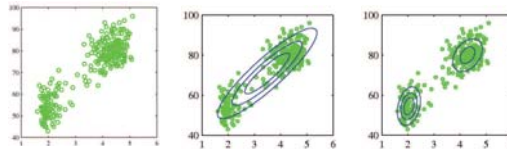
$$\boldsymbol{\theta}_{ML} = \hat{\boldsymbol{\mu}} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}^{(i)}$$

Mixture models

Consider the problem of modeling a pdf given a dataset of examples

$$\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)}\}$$

- If the form of the underlying pdf is known (e.g. Single Gaussian distribution), the problem could be solved using the Maximum Likelihood Estimation method



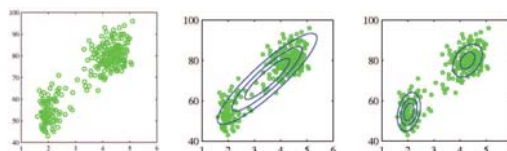
Old Faithful data from Bishop2006

Mixture models

Consider the problem of modeling a pdf given a dataset of examples

$$\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)}\}$$

- If the form of the underlying pdf is known (e.g. Single Gaussian distribution), the problem could be solved using the Maximum Likelihood Estimation method



Old Faithful data from Bishop2006

- Now we will consider an alternative density estimation method which is modeling the pdf with a mixture of parametric densities. In particular, we will focus on mixture models of Gaussian densities

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{j=1}^K p(\mathbf{x}|\boldsymbol{\theta}_j)p(\omega_j) = \sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

Gaussian Mixture Model (GMM)

- Mixture of Gaussians
 - A superposition of K Gaussian densities $p(\mathbf{x}) = \sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$
 - Parameters $\pi_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j$
 - Properties: Asymmetry, multi-modality $0 \leq \pi_j \leq 1, \sum_{j=1}^K \pi_j = 1$
- Previously, we estimated parameters for a single Gaussian distribution by MLE
- Log-likelihood function

$$l(\boldsymbol{\theta}) = \ln p(\mathbf{X} | \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^n \left[\ln \left[\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}^{(i)} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right] \right]$$

Gaussian Mixture Model (GMM)

Log-likelihood function

$$l(\boldsymbol{\theta}) = \ln p(\mathbf{X} | \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^n \left[\ln \left[\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}^{(i)} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right] \right]$$

Find the maximum of this function by differentiation
for $\boldsymbol{\Sigma}_k = \sigma_k^2 \mathbf{I}$

Gaussian Mixture Model (GMM)

Log-likelihood function

$$l(\theta) = \ln p(X|\pi, \mu, \Sigma) = \sum_{i=1}^n \left[\ln \left[\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}^{(i)} | \mu_k, \Sigma_k) \right] \right]$$

Find the maximum of this function by differentiation
for $\Sigma_k = \sigma_k^2 \mathbf{I}$

$$\begin{aligned} \frac{\partial l}{\partial \mu_j} = 0 &\rightarrow \hat{\mu}_j = \frac{\sum_{i=1}^n p(\omega_j | \mathbf{x}^{(i)}, \theta) \mathbf{x}^{(i)}}{\sum_{i=1}^n p(\omega_j | \mathbf{x}^{(i)}, \theta)} \\ \frac{\partial l}{\partial \sigma_j} = 0 &\rightarrow \hat{\sigma}_j^2 = \frac{\sum_{i=1}^n p(\omega_j | \mathbf{x}^{(i)}, \theta) (\mathbf{x}^{(i)} - \mu_j)^2}{\sum_{i=1}^n p(\omega_j | \mathbf{x}^{(i)}, \theta)} \\ \frac{\partial l}{\partial \pi_j} = 0 &\rightarrow \hat{\pi}_j = \frac{\sum_{i=1}^n p(\omega_j | \mathbf{x}^{(i)}, \theta)}{\sum_{k=1}^K \sum_{i=1}^n p(\omega_k | \mathbf{x}^{(i)}, \theta)} \end{aligned}$$

Gaussian Mixture Model (GMM)

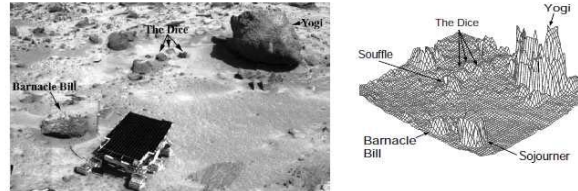
- NOT a closed form analytical solution for GMM parameters
- Due to responsibility depends on the GMM parameters
- Highly non-linear coupled system of equations

⇒ Iterative Numerical Optimization Technique is necessary. **EM algorithm**

Maximum Likelihood technique for a rover localization

Task: To perform rover localization by matching range maps.

Motivation: For greater autonomy in Mars rovers.



(Left) Annotated image, (Right) Terrain map generated from stereo image

- Global Map: panoramic imagery generated at the center of the area from lander.
- Local Map: Occupancy map of local terrain is generated using stereo vision on Sojourner.



Olson, Clark F., and Larry H. Matthies. *Maximum likelihood rover localization by matching range maps*. IEEE International Conference on Robotics and Automation (ICRA). 1998.

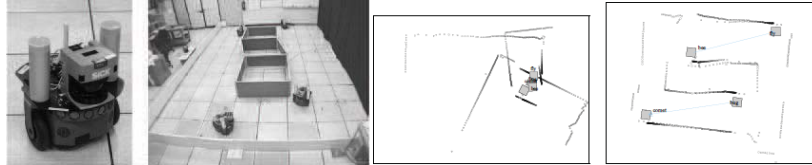
Maximum Likelihood technique for a rover localization


Task: To perform rover localization by matching range maps

- Rover position (x and y coordinates) denoted as X .
- Distances of nearby voxels (n occupied voxels) denoted as D_1, D_2, \dots, D_n .
- The position X yielding the maximum likelihood value i.e. $\ln L(X) = l(X) = \sum_{i=1}^n \ln p(D_i|X)$ is chosen to be the position of the rover.
- $p(D_i|X)$ is a normal distribution with a constant additive term. Normal distribution models difference occupied voxels in btw global map and local map
- Results: Comparison of rover position determined by a human operator and by the proposed localization method. Similar results.

Localization for Mobile Robot Teams

Task: to infer the relative pose of every robot in the team without the use of GPS, external landmarks or instrumentation of the environment

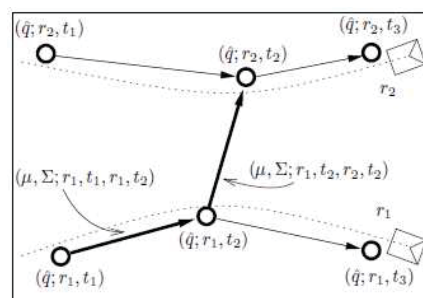


 A. Howard, M. Mataric, G. Sukhatme. *Localization for Mobile Robot Teams Using Maximum Likelihood Estimation*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2002.

Localization for Mobile Robot Teams

Task: to infer the relative pose of every robot in the team

- r_1, r_2 : robots 1 and 2.
- Nodes: robot pose estimates.
- Arcs: observations (motion observation and robot observation).

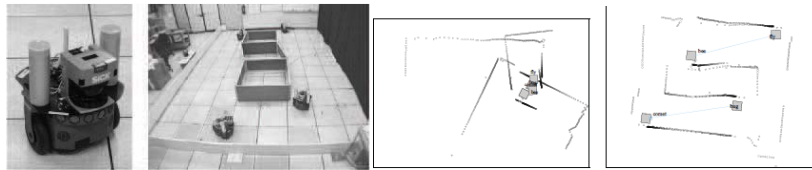


Aim: Maximize $P(O|H)$ using Maximum Likelihood Estimation, O : observation set (motion sensor and robot sensor), H : robot pose estimation set.

Localization for Mobile Robot Teams

Task: to infer the relative pose of every robot in the team

- experimental snapshots
- At $t=1$, the relative robot pose is completely unknown
- By time $t=12$ sec, both robots following the outer wall have observed both robots following inner wall. The two robots on the outer wall can correctly determine each other pose even though they have never seen each other.
- errors: At $t=0$, the localization error is high. By the time $t=20$, the robot performs stable localization, The average range error is about 5.5cm.



Machine Learning in Robotics Gaussian Mixture Model and EM algorithm

Prof. Dongheui Lee

*Institute of Automatic Control Engineering
Technische Universität München*

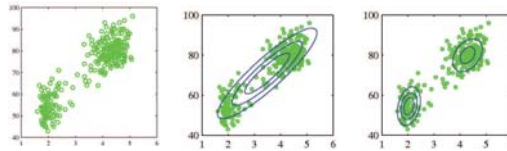
dhlee@tum.de

Mixture models

Consider the problem of modeling a pdf given a dataset of examples

$$X = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)}\}$$

- If the form of the underlying pdf is known (e.g. Single Gaussian distribution), the problem could be solved using the Maximum Likelihood Estimation method



Old Faithful data from Bishop2006

- Now we will consider an alternative density estimation method which is modeling the pdf with a mixture of parametric densities. In particular, we will focus on mixture models of Gaussian densities

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{j=1}^K p(\mathbf{x}|\boldsymbol{\theta}_j)p(\omega_j) = \sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

Gaussian Mixture Model (GMM)

- Mixture of Gaussians
 - A superposition of K Gaussian densities $p(\mathbf{x}) = \sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$
 - Parameters $\pi_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j$
 - Properties: Asymmetry, multi-modality $0 \leq \pi_j \leq 1, \sum_{j=1}^K \pi_j = 1$
- Previously, we estimated parameters for a single Gaussian distribution by MLE
- Log-likelihood function

$$l(\boldsymbol{\theta}) = \ln p(X|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^n \left[\ln \left[\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}^{(i)}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right] \right]$$

Gaussian Mixture Model (GMM)

Log-likelihood function

$$l(\theta) = \ln p(X|\pi, \mu, \Sigma) = \sum_{i=1}^n \left[\ln \left[\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}^{(i)} | \mu_k, \Sigma_k) \right] \right]$$

Find the maximum of this function by differentiation
for $\Sigma_k = \sigma_k^2 \mathbf{I}$

$$\begin{aligned} \frac{\partial l}{\partial \mu_j} = 0 &\rightarrow \hat{\mu}_j = \frac{\sum_{i=1}^n p(\omega_j | \mathbf{x}^{(i)}, \theta) \mathbf{x}^{(i)}}{\sum_{i=1}^n p(\omega_j | \mathbf{x}^{(i)}, \theta)} \\ \frac{\partial l}{\partial \sigma_j} = 0 &\rightarrow \hat{\sigma}_j^2 = \frac{\sum_{i=1}^n p(\omega_j | \mathbf{x}^{(i)}, \theta) (\mathbf{x}^{(i)} - \mu_j)^2}{\sum_{i=1}^n p(\omega_j | \mathbf{x}^{(i)}, \theta)} \\ \frac{\partial l}{\partial \pi_j} = 0 &\rightarrow \hat{\pi}_j = \frac{\sum_{i=1}^n p(\omega_j | \mathbf{x}^{(i)}, \theta)}{\sum_{k=1}^K \sum_{i=1}^n p(\omega_k | \mathbf{x}^{(i)}, \theta)} \end{aligned}$$

Gaussian Mixture Model (GMM)

- No closed form analytical solution for GMM parameters
- Due to responsibility depends on the GMM parameters
- Highly non-linear coupled system of equations

⇒ Iterative numerical optimization technique is necessary **EM algorithm**

EM for GMM

Given a Gaussian Mixture Model, the goal is to maximize the likelihood function with the parameters

□□ Initialize π_j, μ_j, Σ_j

□□ E-step: Evaluate the responsibilities using the current parameters

$$p(\omega_k | \mathbf{x}^{(i)}, \theta) = \frac{\pi_k \mathcal{N}(\mathbf{x}^{(i)} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}^{(i)} | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

□□ M-step: Re-estimate the parameters using the current responsibilities

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{n_k} \sum_{i=1}^n p(\omega_k | \mathbf{x}^{(i)}, \theta) \mathbf{x}^{(i)}$$

$$\hat{\boldsymbol{\Sigma}}_k = \frac{1}{n_k} \sum_{i=1}^n p(\omega_k | \mathbf{x}^{(i)}, \theta) (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_k)(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_k)^T$$

$$\hat{\pi}_k = \frac{n_k}{n} \text{ where } n_k = \sum_{i=1}^n p(\omega_k | \mathbf{x}^{(i)}, \theta)$$

4□ Evaluate the log-likelihood and check for convergence of either the parameters or the log-likelihood □□ not converged, go to step □□

$$l(\theta) = \ln p(\mathbf{x} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{i=1}^n \ln \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}^{(i)} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Example

Probability to get a credit A, B, C, D from a lecture depends on the mean. Assume that the number of students for each credit A, B, C, D is a, b, c, d. Estimate the mean.

$$\omega_1 = \text{A} \quad \mathbb{P}(\text{A}) = 1/4$$

$$\omega_2 = \text{B} \quad \mathbb{P}(\text{B}) = \mu$$

$$\omega_3 = \text{C} \quad \mathbb{P}(\text{C}) = 0.1$$

$$\omega_4 = \text{D} \quad \mathbb{P}(\text{D}) = 1/4 - 0.1$$

$$\text{where } 0 \leq \mu \leq 1/4$$

$$\mathbb{P}(\text{A}) + \mathbb{P}(\text{B}) + \mathbb{P}(\text{C}) + \mathbb{P}(\text{D}) = 1$$

$$\mathbb{P}(\text{A}, \text{B}, \text{C}, \text{D} | \mu) =$$

$$\frac{\partial \ln \mathbb{P}(\text{A}, \text{B}, \text{C}, \text{D} | \mu)}{\partial \mu} =$$

$$\Rightarrow \mu = \frac{a + b}{a + b + c}$$

Example

Probability to get a credit $\omega_1, \omega_2, \omega_3, \omega_4$ from a lecture depends on the mean. Assume that the number of students for each credit $\omega_1, \omega_2, \omega_3, \omega_4$ is a, b, c, d . Estimate the mean.

$$\begin{aligned}\omega_1 &= \mu & \omega_2(\mu) &= 1/\mu \\ \omega_2 &= \mu & \omega_3(\mu) &= \mu \\ \omega_3 &= \mu & \omega_4(\mu) &= 1/\mu \\ \omega_4 &= \mu & \omega_5(\mu) &= 1/\mu - \mu \\ \text{where } 0 &\leq \mu \leq 1/\mu\end{aligned}$$

$$\begin{aligned}\omega_1 + \omega_2 + \omega_3 + \omega_4 &= 1 \\ \omega_1 \omega_2 \omega_3 \omega_4 \omega_5 &= K(1/\mu)^a (\mu)^b (\mu)^c (1/\mu - \mu)^d \\ \frac{\partial \ln(\omega_1 \omega_2 \omega_3 \omega_4 \omega_5)}{\partial \mu} &= \frac{a}{\mu} + \frac{b}{\mu} - \frac{d}{1/\mu - \mu} = 0 \\ \Rightarrow \mu &= \frac{a+b}{a+b+d}\end{aligned}$$



GMM

EM

Application



Example

It is known that $a = 1, b = c = d = 10$, then $\mu = 1/10$. However, now assume that a and b are known, but c and d are unknown.

$$a = \frac{0.1}{0.1 + \mu}, b = \frac{\mu}{0.1 + \mu} \Leftrightarrow \mu = \frac{a+b}{a+b+d}$$

EM algorithm : Start with an initial parameter. Iterate E-step and M-step.

- Initialization : $\mu(0)$
- E-step : $a = \frac{\mu^{(t)}}{1/\mu^{(t)} + \mu^{(t)}} = \mathbb{E}[a|\mu^{(t)}]$
- M-step : $\mu^{(t+1)} = \frac{a+b}{a+b+d}$



GMM

EM

Application



Example

Given $\sigma^2 = 1$, $\mu = 0$, $N = 10$, estimate μ with an initial guess $\mu(0) = 0$ by the EM algorithm

t	$\mu(t)$	$\sigma^2(t)$
0	0	1
1	0.4	1
2	0.4	1
3	0.4	1
4	0.4	1
5	0.4	1
6	0.4	1
7	0.4	1
8	0.4	1
9	0.4	1

The Expectation-Maximization algorithm

- The EM is a general method for finding the ML estimate of the parameters of a pdf when the data has missing values
- Assume a dataset containing two types of features
 - A set of features X whose value is known we call these the *incomplete* data
 - A set of features Z whose value is unknown we call these the *missing* data
 - θ : model parameters
- We now define a joint pdf $p(X, Z; \theta)$ called the complete-data likelihood
- As suggested by its name, the EM algorithm operates by performing two basic operations over and over:
 - An Expectation step
 - A Maximization step

The Expectation-Maximization algorithm

- **Expectation step** : Find the expected value of the log-likelihood $\ln p(X, \mathbf{z} | \theta)$ with respect to the unknown data \mathbf{z} , given the data X and the current parameter estimates θ

$$Q(\theta, \theta^{old}) = \sum_{\mathbf{z}} p(\mathbf{z} | X, \theta^{old}) \ln p(X, \mathbf{z} | \theta)$$

- **Maximization step** : Find the argument θ that maximizes the expected value defined by $Q(\theta, \theta^{old})$

$$\theta^{new} = \underset{\theta}{\operatorname{argmax}} Q(\theta, \theta^{old})$$

- **Convergence properties**
 - Each iteration of EM is guaranteed to increase the log-likelihood
 - EM algorithm is guaranteed to converge to a local maximum of the likelihood function

EM algorithm to find MAP solution

- EM can be used to find MAP (maximum a posterior) solutions for models in which a prior $p(\theta)$ is defined over parameters
- E-step : same as EM for ML
- M-step : Find the argument θ that maximizes the expected value defined by $Q(\theta, \theta^{old}) + \ln p(\theta)$ instead of $Q(\theta, \theta^{old})$

GMM revisited

- Consider the problem of maximizing the likelihood for the complete data set $\{X, Z\}$
- If complete data set $\{X, Z\}$ is given, the complete-data log likelihood function can be maximized trivially in closed form
- In practice, the latent variables are not given. Therefore, we consider the expectation of the complete-data log-likelihood, wrt the posterior distribution of the latent variables

$$\mathbb{E}[z_k^{(j)}] = \frac{\pi_k \mathcal{N}(\mathbf{x}^{(i)} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}^{(i)} | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} = \gamma(z_k^{(j)})$$

$$\mathbb{E}[\ln \pi(X, Z | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi})] = \sum_{i=1}^n \sum_{k=1}^K \gamma(z_k^{(i)}) \left\{ \ln \pi_k + \ln \mathcal{N}(\mathbf{x}^{(i)} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

Relation to k-means

- EM for GMM
 - Soft assignment of data points to clusters
- k-means
 - Hard assignment of data points to clusters
 - A special case of EM for GMM

$$\mathbb{E}[z_k^{(j)}] = \gamma(z_k^{(j)}) = \begin{cases} 1 & \text{if } |\mathbf{x}^{(i)} - \boldsymbol{\mu}_k| < |\mathbf{x}^{(i)} - \boldsymbol{\mu}_j|, \forall j \\ 0 & \text{else} \end{cases}$$

EM algorithm

- For learning from partly unobserved data
- Maximum Likelihood estimate (MLE) vs EM estimate
 - ML estimate

$$\theta = \arg\max_{\theta} p(X|\theta)$$

- EM estimate

$$\theta = \arg\max_{\theta} E_{\phi}[p(X, \phi|\theta)]$$



GMM

EM

Application



Detection of target and arrow using GMM

Task: To perform archery by a humanoid robot iCub

Proposed approach: reinforcement learning algorithms for learning the skill of archery

- EM based Reinforcement Learning (PoRE)
- chained vector regression (ACVR)

Subproblem: Image processing

- To detect where the target is
- To get the relative position of arrow from the target



Dormushev, Petar, et al. *Learning the skill of archery by a humanoid robot iCub*. IEEE-RAE International Conference on Humanoid Robots, 2014.



GMM

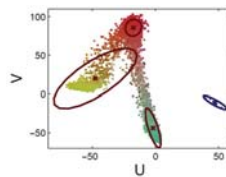
EM

Application



Detection of target and arrow using GMM

- The color detection is done in RGB color space
- Only R and G components are used to ensure robustness against luminosity
- A three component GMM for target, a single component GMM for arrow tip
- Bayesian Information Criterion (BIC) is used for optimizing the number of components in each GMM



Detection of target and arrow using GMM

After learning the likelihood value of each pixel in a new image can be used for classification of pixels. The classified image can be used to detect the center of target (red cross) and arrow (blue circle) in below figure.



Ground Plane Detection

Task : In mobile navigation, detection of ground and non-ground is useful in various application such as: object recognition, obstacle avoidance during autonomous navigation. This paper uses it for object tracking and following.



Conrad and Deoua G. Homography-based Ground Plane Detection for Mobile Robot Navigation Using a Modified EM Algorithm. IEEE International Conference on Robotics and Automation.



GMM

EM

Application



Ground Plane Detection



- From two images, a large number of pixel correspondences are found by RANSAC algorithm.
- EM used to classify pixel correspondences from two images into 2 classes: *Ground plane* and *Non-Ground plane* in order to segment out the ground.
- Robot control: The robot uses pixels on the target object to follow. Obstacle avoidance during autonomous navigation. It keeps the target object in the center of image view.



GMM

EM

Application



Ground Plane Detection

- Homography : a transformation matrix that relates the pixel coordinates of planar points as seen from two different viewing angles

$$\hat{p}_i = H p_i$$

- Homography H is defined as $H = \hat{C}(C + \frac{g g^T}{g^T g})C^{-1}$ with \hat{C} and C containing the intrinsic parameters of the cameras. The parameter Homography H is $\theta = \{R, n, d\}$, which is rotation matrix, translation vector, normal vector of the plane, distance between the camera and plane
- These parameters will be updated via EM
- The pair of corresponding pixels \hat{p}_i, p_i is referred to as pixel correspondence i

Ground Plane Detection

- Expectation Maximization Algorithm

$$p(X|\theta) = \frac{\exp(-\frac{\|X - \frac{1}{n} \sum p_i\|^2}{2\sigma^2})}{\sum_i \exp(-\frac{\|X - \frac{1}{n} \sum p_i\|^2}{2\sigma^2})} \quad \text{where } \sigma^2 = \frac{1}{n} \sum \|p_i - \frac{1}{n} \sum p_i\|^2$$

- where $\sigma^2 = \frac{1}{n} \sum \|p_i - \frac{1}{n} \sum p_i\|^2$, $\theta = (\sigma^2, \mu)$, $X = x$ and $\mu = \frac{1}{n} \sum p_i$
- $p(X|\sigma^2, \mu) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{\|X - \mu\|^2}{2\sigma^2})$
- After computing all posterior probabilities [E-step] the new model parameters are updated [M-step]
- In the M-step, an optimization algorithm [simple method] is used because it does not require an explicit gradient and shows faster convergence
- Ground detection rate : 95%

Announcements

- Further Reading
 - MLE: Uda hap, Bishop hap
 - GMM: Uda hap, Bishop hap
 - EM: Mitchell hap, Bishop hap
- Next Lecture
 - Nonparametric density estimation
 - Kernel Density Estimation, k-NN, Parzen window



GMM

EM

Application



Machine Learning in Robotics Nonparametric Density Estimation

Prof. Dongheui Lee

*Institute of Automatic Control Engineering
Technische Universität München*

dhlee@tum.de



<http://www.lsr.ei.tum.de/>



Density Estimation - Motivation

- In our previous lecture on decision theory, we saw that the optimal classifier could be expressed as a family of discriminant functions

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

- Decision rule was

choose ω_i if $g_i(\mathbf{x}) > g_j(\mathbf{x}), \forall j \neq i$

- We need to estimate both prior $p(\omega_i)$ and likelihood $p(\mathbf{x}|\omega_i)$
- During next lectures, techniques to estimate the likelihood density function $p(\mathbf{x}|\omega_i)$ will be introduced

Approaches for Density Estimation

Parametric Approach

- A given form for the density function is assumed (i.e., Gaussian) and the parameters of the function (i.e., mean and variance) are optimized by fitting the model to the data set
- Parametric density estimation is often referred to as Parameter Estimation

Non-Parametric Approach

- No functional form for the density function is assumed. The density estimate is driven entirely by the data
- called Parameter Estimation
 - Kernel Density Estimation
 - Nearest Neighbor Rule

Histogram Density Model

- The simplest form of non-parametric density estimation is the familiar *histogram*
- Standard histogram : Divide the sample space into distinct bins of width Δ_i and approximate the density at each bin by the fraction of points in the training data that fall into the corresponding bin i .

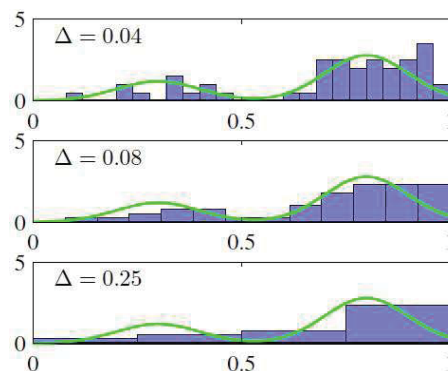
$$p_i = \frac{n_i}{n\Delta_i} \quad \int p(\mathbf{x})d\mathbf{x} = 1$$

where n_i is the number of observations of x falling in bin i and n is the total number of observations.

Often, $\Delta_i = \Delta$.

Properties of Histogram Density Model

A histogram density model is dependent on the choice of histogram bin-width Δ .



- If Δ is very small, the resulting density model is very spiky
- If very large, the model is too smooth

Properties of Histogram Density Model

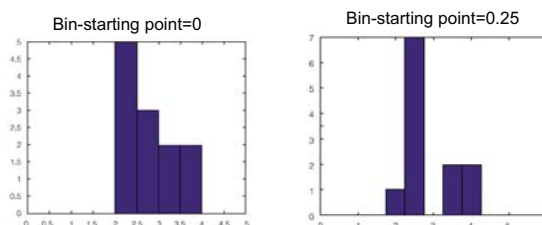
A histogram density model is dependent on the choice of edge location for the bins

- Dataset $X = [2.3 \ 2.4 \ 2.34 \ 2.41 \ 2.71 \ 2.65 \ 3.34 \ 3.73]$
- Bin-width $\Delta = 0.5$

Properties of Histogram Density Model

A histogram density model is dependent on the choice of edge location for the bins

- Dataset $X = [2.3 \ 2.4 \ 2.34 \ 2.41 \ 2.71 \ 2.65 \ 3.34 \ 3.73]$
- Bin-width $\Delta = 0.5$



Properties of Histogram Density Model

- A very simple form of density estimation
- The density estimate depends on the starting position of the bins and bin-width Δ
- The discontinuities of the estimate are not due to the underlying density, they are only an artifact of the chosen bin locations
- A much more serious problem is the curse of dimensionality, since the number of bins grows exponentially with the number of dimensions

General Formulation of Non-parametric density estimation

- The probability that a vector \mathbf{x} , drawn from a distribution $p(\mathbf{x})$, will fall in a region \mathfrak{R} of the sample space is $P = \int_{\mathfrak{R}} p(\mathbf{x}) d\mathbf{x}$
- Suppose that n independently and identically distributed samples $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)})$ are drawn from the probability $p(\mathbf{x})$. The probability that K of these n vectors fall in \mathfrak{R} is given by the binomial law

$$\text{Bin}(K|n, P) = \frac{n!}{K!(n-K)!} P^K (1-P)^{n-K}$$

- If n is very large, $P = \frac{K}{n}$
- If the region \mathfrak{R} is very small, $P = \int_{\mathfrak{R}} p(\mathbf{x}) d(\mathbf{x}) \simeq p(\mathbf{x})V$

General Formulation of Non-parametric density estimation

- The probability that a vector \mathbf{x} , drawn from a distribution $p(\mathbf{x})$, will fall in a region \mathfrak{R} of the sample space is $P = \int_{\mathfrak{R}} p(\mathbf{x}) d\mathbf{x}$
- Suppose that n independently and identically distributed samples $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)})$ are drawn from the probability $p(\mathbf{x})$. The probability that K of these n vectors fall in \mathfrak{R} is given by the binomial law

$$\text{Bin}(K|n, P) = \frac{n!}{K!(n-K)!} P^K (1-P)^{n-K}$$

- If n is very large, $P = \frac{K}{n}$
- If the region \mathfrak{R} is very small, $P = \int_{\mathfrak{R}} p(\mathbf{x}) d(\mathbf{x}) \simeq p(\mathbf{x})V$

$$p(\mathbf{x}) \simeq \frac{K}{nV}$$

General Formulation of Non-parametric density estimation

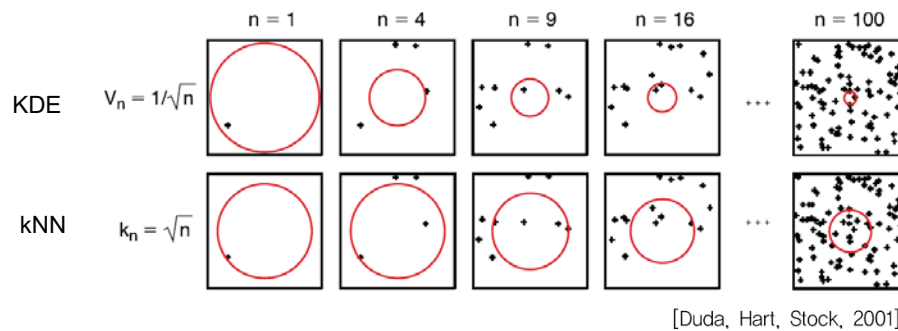
Discussion on underlying assumptions

- In practice the value of n is fixed (the total number of examples)
- In order to improve the accuracy of the estimate $p(\mathbf{x})$ we could let V to approach zero, but then the region \mathfrak{R} would become so small that it would enclose no examples
- This means that in practice we will have to find a compromise value of the volume V
 - Large enough to include enough examples within \mathfrak{R}
 - Small enough to support the assumption that $p(\mathbf{x})$ is constant within \mathfrak{R}

Two approaches

- Fix V and determine K from the data: Kernel Density Estimation (KDE)
- Fix K and determine V from the data: k Nearest Neighbor (kNN)
- As $n \rightarrow \infty$, both approaches become close to the true probability density

General Formulation of Non-parametric density estimation



KDE using a Parzen Window

- Nonparametric density estimation general formula $p(\mathbf{x}) \simeq \frac{K}{nV}$
- Region \mathcal{R} : a small hypercube centered on the estimation point \mathbf{x} , $V = h^m$
- Kernel function

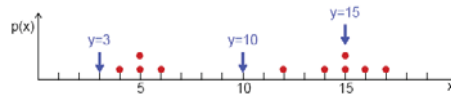
$$k(\mathbf{u}) = \begin{cases} 1 & \text{if } |u_{(j)}| \leq 0.5, \forall j = 1, \dots, m \\ 0 & \text{otherwise} \end{cases}$$

- For the dataset $\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)}\}$, the total number of points inside the hypercube is $K = \sum_{i=1}^n k\left(\frac{\mathbf{x} - \mathbf{x}^{(i)}}{h}\right)$
- Density estimate

$$p(\mathbf{x}) = \frac{1}{nh^m} \sum_{i=1}^n k\left(\frac{\mathbf{x} - \mathbf{x}^{(i)}}{h}\right)$$

Parzen Estimator Simple Example

- Given the dataset below, use Parzen windows to estimate the density $p(x)$ at $x = 3, 10, 15$. Use a bandwidth of $h = 4$.
 $X = \{x^{(1)}, x^{(2)}, \dots, x^{(n)}\} = \{4, 5, 5, 6, 12, 14, 15, 15, 16, 17\}$



- Estimate $p(x = 3), p(x = 10), p(x = 15)$

KDE using a Smooth Kernel

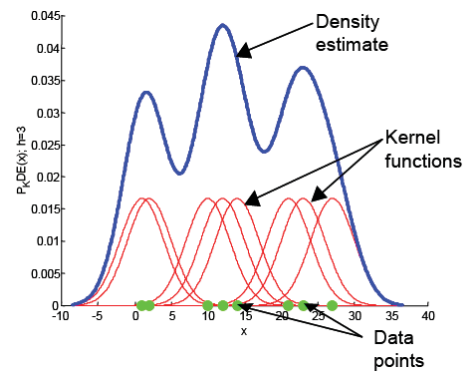
- KDE using the parzen window
 - Discontinuity
 - Equal weights for all data points
- If using smooth Kernel function which $k(u) > 0$, $\int k(u)du = 1$
- For example, a Gaussian

$$p(x) = \frac{1}{n} \sum_{i=1}^n \frac{1}{\sqrt{2\pi}h} \exp\left(-\frac{|x - x^{(i)}|^2}{2h^2}\right)$$



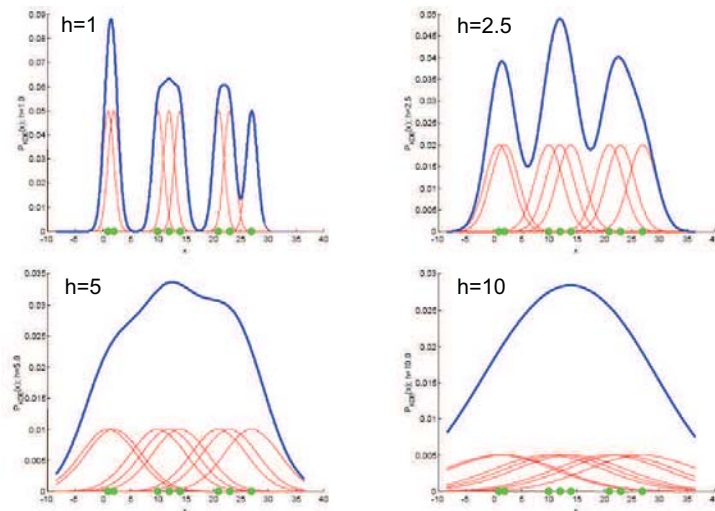
KDE using a Smooth Kernel

- Similar to the Parzen window, estimator is a sum of bumps placed at the data points
- The kernel function determines the shape of the bumps
- The parameter h , also called the *smoothing parameter* or *bandwidth*, determines their width

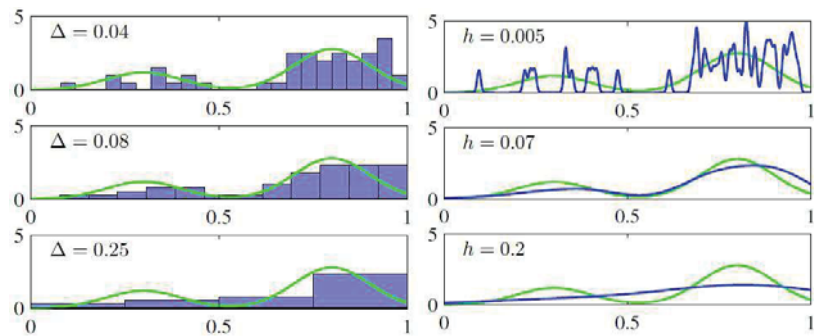


Choosing the bandwidth

- Large h : over-smoothing
- Small h : sensitive to noise



Histogram vs. KDE using a smooth Kernel

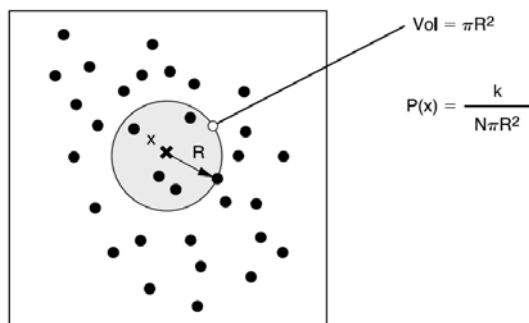


kNN Density Estimation

- In the kNN method we grow the volume surrounding the estimation point x so that it encloses a total of K points
- The density estimate then becomes

$$p(x) \simeq \frac{K}{nV}$$

V is the volume that contains K points



kNN Density Estimation

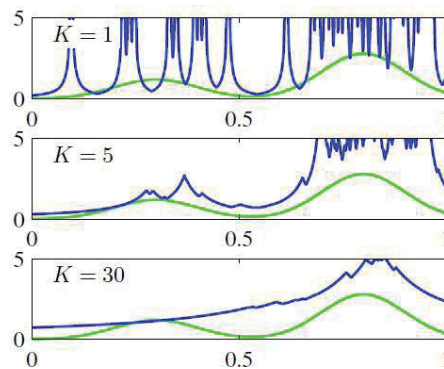


Illustration of K nearest neighbor density using the same data set as in previous examples. We see that the parameter K governs the degree of smoothing, so that a small value of K leads to a very noisy density model (top panel), whereas a large value (bottom panel) smooths out the bimodal nature of the true distribution (shown by the green curve) from which the data set was generated.

K Nearest Neighbor Rule (k-NNR)

An intuitive classification method

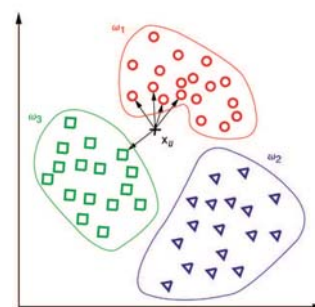
- to classify unlabeled examples based on their similarity with examples in the training set
- To find K closest labeled examples and assign the query data to the class that appears most within the K examples

k-NNR

- An integer K
- A set of labeled examples (training data)
- A metric to measure "closeness"

Example

- A query point x_u
- $K = 5$
- Classification result: ω_1



K Nearest Neighbor Rule (k-NNR)

- k-NNR classification
 - K-nearest neighbor (kNN) density estimation technique
 - Use Bayes theorem
- Problem setting
 - n datapoints
 - For each cluster ω_i , n_i datapoints are included
 - Classify a new point \mathbf{x} . Find the cluster which has the maximum $p(\omega_i|\mathbf{x})$
- Solution

$$\sum_{\forall i} K_i = K \quad , \quad p(\mathbf{x}|\omega_i) = \frac{K_i}{n_i V}$$

$$p(\mathbf{x}) = \frac{K}{nV} \quad , \quad p(\omega_i) = \frac{n_i}{n}$$

$$p(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)p(\omega_i)}{p(\mathbf{x})} = \frac{K_i}{K}$$


Nonparametric Density Estimation

- Big storage requirements
 - Have to save entire training dataset
- Requires large dataset for realistic density estimation
- Expensive computational cost on recall in the case of a large dataset

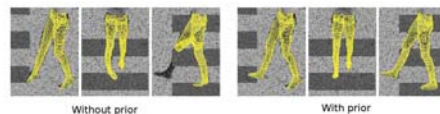
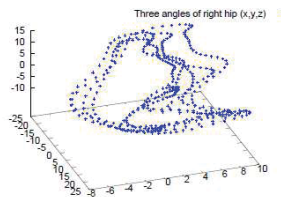
Nonparametric Density Estimation for Human Pose Tracking

- An object model is assumed.
- The pose parameters of the model are learned so that the model optimally explains object's image data.
- The joint probability of a pose x and an image feature C is given by:

$$p(x, C|I) = \frac{p(I|C, x)p(C|x)p(x)}{p(I)}, \quad I : \text{input image.}$$
- Non-parametric density estimation is realized to capture the complex configuration of human pose.

 T. Brox, T. Rosenhahn, U. Kersting and D. Cremers. *Nonparametric Density Estimation for Human Pose Tracking*. Springer-Verlag, 2006.

Nonparametric Density Estimation



- The prior probability for the joint angle Θ is learned through non-parametric density model; Parzen-Rosenblatt estimator:

$$p(\Theta) = \frac{1}{\sqrt{2\pi\sigma N}} \sum_{i=1}^N \exp\left(-\frac{(\Theta_i - \Theta)^2}{2\sigma^2}\right).$$
- N : number of training samples Θ_i .
- σ : tuning parameter.

Segmentation and appearance model building from an image sequence



- Segment a human given multiple video frames; select features which do not change over time.
- Non-parametric kernel-based PDF estimator is used for segmentation of the human.



L. Zhao and L. S. Davis. *Segmentation and appearance model building from an image sequence*. ICIP 2005.

Appearance model building: steps

- Select a constant appearance human body model.
- Estimate the probability of a pixel x belonging to the foreground f :

$$P_{fg} = \sum_i P_{fg}(x_i) \prod_{j=1}^m K\left(\frac{y_j - x_{ij}}{\sigma_j}\right).$$
- Estimate the probability of a pixel x belonging to the background b :

$$P_{bg} = \sum_i P_{bg}(x_i) \prod_{j=1}^m K\left(\frac{y_j - x_{ij}}{\sigma_j}\right).$$

Announcements

- Further Reading
 - Duda, Chapter 4.1-4.5
 - Bishop, Chapter 2.5, 3.2
 - Mitchell, Chapter 8



Introduction

Histogram

NPDE

KDE

kNN

Application

28



Machine Learning in Robotics Lecture 9-10: Hidden Markov models

Prof. Dongheui Lee

*Institute of Automatic Control Engineering
Technische Universität München*

dhlee@tum.de



<http://www.lsr.ei.tum.de/>



Discrete Markov Processes

Consider a system described by the following process

- At any given time, the system can be in one of N possible states $\{s_1, s_2, \dots, s_N\}$
- At regularly spaced times, the system undergoes a transition to a new state
- Transition between states can be described probabilistically

In general, the probability that the system is in state $q_t = s_j$ will be a function of the complete history of the system

- To simplify the analysis, however, it is common to assume that the state of the system at time t depends only on the state at time $t - 1$

$$p(q_t = s_j | q_{t-1} = s_i, q_{t-2} = s_k, \dots) = p(q_t = s_j | q_{t-1} = s_i)$$

- This is known as a first-order Markov Process
- We assume that the transition probability between any two states is independent of time

$$a_{ij} = p(q_t = s_j | q_{t-1} = s_i)$$

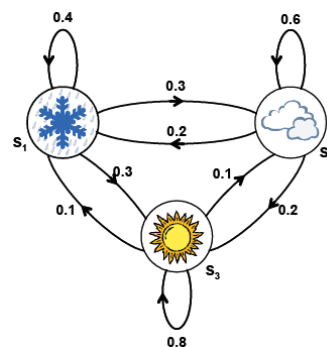
An example

Consider a simple three-state Markov model of the weather.

- Any given day, the weather can be described as being
 - State 1: Rain or Snow
 - State 2: Cloudy
 - State 3: Sunny

- Transitions between states are described by the transition matrix

$$A = \{a_{ij}\} = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$$



- This model can be described by the following directed graph

An example

- Q1. Given that the weather on day $t=1$ is sunny, what is the probability that the weather for the next 7 days will be "sun, sun, rain, rain, sun, clouds, sun"?

$$\begin{aligned}
 & p(q_2 = s, q_3 = s, q_4 = r, q_5 = r, q_6 = s, q_7 = c, q_8 = s | q_1 = s) \\
 &= p(q_2 = s | q_1 = s) p(q_3 = s | q_2 = s) p(q_4 = r | q_3 = s) p(q_5 = r | q_4 = r) \\
 & p(q_6 = s | q_5 = r) p(q_7 = c | q_6 = s) p(q_8 = s | q_7 = c) \\
 &= (0.8)(0.8)(0.1)(0.4)(0.3)(0.1)(0.2)
 \end{aligned}$$

- Q2. What is the probability that the weather stays in the same known state s_i for exactly T consecutive days?

$$a_{ii}^{T-1}(1 - a_{ii})$$

Discrete Markov Processes

The previous model assumes that each state can be uniquely associated with an observable event

- Once an observation is made, the state of the system is then trivially retrieved
- This model, however, is too restrictive to be of practical use for most realistic problems

To make the model more flexible, we will assume that the outcomes or observations of the model are a probabilistic function of each state

- Each state can produce a number of outputs according to a unique probability distribution, and each distinct output can potentially be generated at any state
- These are known as Hidden Markov Models (HMM), because the state sequence is not directly observable, it can only be approximated from the sequence of observations produced by the system

An Example: Weather

Markov model → Hidden Markov model:

- Hide the weather!!
- Assume that one cannot see the weather directly. But he has only limited information to guess the weather, which is observing an umbrella of a person.

weather	Probability to have an umbrella
sun	0.1
rain	0.8
cloud	0.3

In Markov process, direct observation of the weather is possible. Therefore, the probability to observe a sequence of weather can be expressed as

$$p(q_1, q_2, \dots, q_T) = p(q_1) \prod_{t=2}^T p(q_t | q_{t-1})$$

But, in HMM, the weather is hidden. The probability of the weather becomes

$$p(q_i | o_i) = \frac{p(o_i | q_i) p(q_i)}{p(o_i)}$$

Given a sequence of observations (umbrella), the conditional probability of the weather sequence is

$$p(q_1, \dots, q_t | o_1, \dots, o_t) \propto \prod_{i=1}^t [p(o_i | q_i)] \prod_{i=2}^t [p(q_i | q_{i-1})] p(q_1)$$

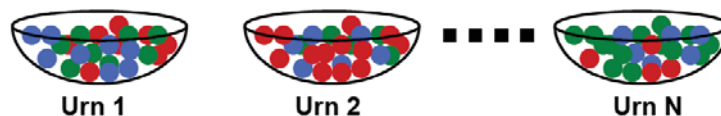
Example: The urn-ball problem

To further illustrate the concept of an HMM, consider this scenario

- You are placed in the same room with a curtain
- Behind the curtain there are N urns, each containing a large number of balls with M different colors
- The person behind the curtain selects an urn according to an internal random process, then randomly grabs a ball from the selected urn
- He shows you the ball, and places it back in the urn
- This process is repeated over and over

Questions?

- How would you represent this experiment with an HMM?
- What are the states? Why are the states hidden?
- What are the observations?



Elements of a Hidden Markov Model

A Hidden Markov Model is characterized by the following

- N , the number of states in the model $\{s_1, s_2, \dots, s_N\}$
- M , The number of discrete observation $\{v_1, v_2, \dots, v_M\}$
- $A = \{a_{ij}\}$, the transition probability

$$a_{ij} = p(q_t = s_j | q_{t-1} = s_i) \quad \sum_j a_{ij} = 1, \forall i$$

- $B = b_j(k)$, the observation probability distribution

$$b_j(k) = p(o_t = v_k | q_t = s_j) \quad \sum_k b_j(k) = 1, \forall j$$

- π_i , the initial state distribution $\pi_i = p(q_1 = s_i)$, $\sum_i \pi_i = 1$

Therefore, an HMM is specified by two scalars (M, N) and three probability distributions (A, B, π) . we will represent an HMM by the compact notation $\lambda = (A, B, \pi)$

HMM generation of observation sequences

Given a completely specified HMM $\lambda = (A, B, \pi)$, how can an observation sequence $\mathcal{O} = \{o_1, o_2, \dots, o_T\}$ be generated?

1. Choose an initial state according to the initial state distribution:
 $q_1 = \arg \max_i \pi_i$
2. Set $t = 1$
3. Generate and observation o_t according to the observation distribution $b_j(k)$: $o_t = \arg \max_k b_{q_t}(k)$
4. Move to a new state according to the state transition distribution a_{ij}
5. Set $t = t + 1$ and return to step 3 until $t \geq T$

The three basic HMM problems

- **Probability Evaluation problem**
Determine the probability $p(\mathcal{O}|\lambda)$ that a particular sequence of visible states $\mathcal{O} = \{o_1, o_2 \dots o_T\}$ was generated by the model $\lambda = \{A, B, \pi\}$.
 \Rightarrow The solution is given by the Forward and Backward procedures.
- **Optimal State Sequence**
Given a model λ and a sequence of observations, determine the most likely sequence of hidden states that led to the observation sequence.
 \Rightarrow The solution is provided by the Viterbi algorithm.
- **Parameter Estimation**
Given a set of observations (training dataset), determine the parameters of the HMM $\lambda = \{A, B, \pi\}$ to maximize the likelihood $p(\mathcal{O}|\lambda)$.
 \Rightarrow Baum-Welch re-estimation procedure

Problem 1: Probability Evaluation

Our goal is to compute the likelihood of an observation sequence $\mathcal{O} = \{o_1, o_2 \dots o_T\}$ given a particular HMM model defined by $\lambda = (A, B, \pi)$

- Computation of this probability involves enumerating every possible state sequence and evaluating the corresponding probability

$$p(\mathcal{O}|\lambda) = \sum_{\forall \mathcal{Q}} p(\mathcal{O}, \mathcal{Q}|\lambda) = \sum_{\forall \mathcal{Q}} p(\mathcal{O}|\mathcal{Q}, \lambda) p(\mathcal{Q}|\lambda)$$

- For a particular state sequence $\mathcal{Q} = \{q_1, q_2 \dots\}$ the probability $p(\mathcal{O}|\mathcal{Q}, \lambda)$ is

$$p(\mathcal{O}|\mathcal{Q}, \lambda) = \prod_{t=1}^T p(o_t|q_t, \lambda) = \prod_{t=1}^T b_{q_t}(o_t)$$

- The probability of the state sequence \mathcal{Q} is

$$p(\mathcal{Q}|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \dots a_{q_{T-1} q_T}$$

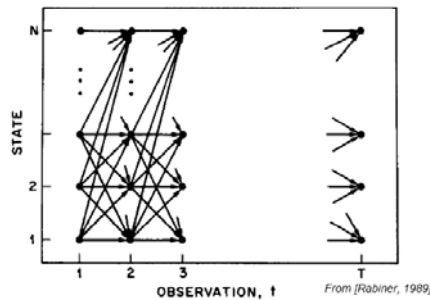
- Merging these results, we obtain

$$P(\mathcal{O}|\lambda) = \sum_{q_1, q_2, \dots, q_T} \pi_{q_1} b_{q_1}(o_{q_1}) a_{q_1 q_2} b_{q_2}(o_{q_2}) \dots a_{q_{T-1} q_T} b_{q_T}(o_{q_T})$$

Problem 1: Probability Evaluation

Computational complexity

- With N^T possible state sequences, this approach becomes unfeasible even for small problems
 - For $N = 5$ and $T = 100$, the method would require the order of 10^{72} computations !!
- Fortunately, the computation of $p(\mathcal{O}|\lambda)$ has the lattice (or trellis) structure shown below, which lends itself to a very efficient implementation known as the *Forward procedure*



Problem 1: The Forward procedure

Consider the following variable $\alpha_t(i)$ defined as *Forward Variable*

$\alpha_t(i) = p(o_1, o_2, \dots, o_t, q_t = s_i | \lambda)$ which represents the probability of the observation sequence up to time t and the state at time t to be s_i , given the model λ

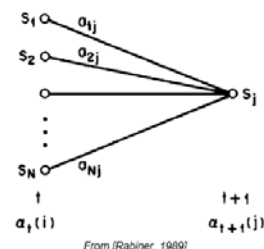
Computation of this variable can be efficiently performed by induction

- Initialization: $\alpha_1(i) = \pi_i b_i(o_1)$

- Induction:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^N \alpha_t(i) a_{ij} \right] b_j(o_{t+1}) \quad \begin{cases} 1 \leq t \leq T-1 \\ 1 \leq j \leq N \end{cases}$$

- Termination: $p(\mathcal{O}|\lambda) = \sum_{i=1}^N \alpha_T(i)$



As a result, computation of $p(\mathcal{O}|\lambda)$ can be reduced from $2T \times N^T$ down to $N^2 \times T$ operations (from 10^{72} to 3000 for $N = 5, T = 100$)

Problem 1: The Backward procedure

In analogy to the forward procedure, consider the *backward variable* $\beta_t(i)$ defined as $\beta_t(i) = p(o_{t+1}, o_{t+2}, \dots, o_T | q_t = s_i, \lambda)$.

$\beta_t(i)$ represents the probability of the partial observation sequence from $t + 1$ to the end, given state s_i at time t and the model λ

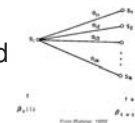
Computation of $\beta_t(i)$ can be done through induction

- Initialization $\beta_T(i) = 1$ (arbitrary)
- Induction

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \quad \begin{cases} t = T-1 \dots 1 \\ 1 \leq i \leq N \end{cases}$$

- Termination: $p(\mathcal{O}|\lambda) = \sum_{i=1}^N b_i(o_1) \beta_1(i) \pi_i$

As a result, computation can be effectively performed in the order of $N^2 \times T$ operations



Problem 2: Optimal State Sequence

Two approaches to find the optimal state sequence

1. finding the states q_t that are individually more likely at each time t
2. finding the single best state sequence path (i.e., maximize the posterior $P(\mathcal{Q}|\mathcal{O}, \lambda)$)

We define *Forward-Backward Variable* $\gamma_t(i)$.

$$\gamma_t(i) = p(q_t = s_i | \mathcal{O}, \lambda)$$

$\gamma_t(i)$ represents the probability of being in state s_i at time t , given the observation sequence \mathcal{O} and the model λ .

$$\gamma_t(i) = p(q_t = s_i | \mathcal{O}, \lambda) = \frac{p(q_t = s_i, \mathcal{O} | \lambda)}{p(\mathcal{O} | \lambda)} = \frac{p(q_t = s_i, \mathcal{O} | \lambda)}{\sum_{i=1}^N p(q_t = s_i, \mathcal{O} | \lambda)}$$

Problem 2: Optimal State Sequence 1

The numerator of $\gamma_t(i)$ is equal to the product of $\alpha_t(i)$ and $\beta_t(i)$

$$\gamma_t(i) = \frac{p(q_t = s_i, \mathcal{O} | \lambda)}{\sum_{i=1}^N p(q_t = s_i, \mathcal{O} | \lambda)} = \frac{\alpha_t(i) \beta_t(i)}{\sum_{i=1}^N \alpha_t(i) \beta_t(i)}$$

The individually most likely state q_t^* at each time is then

$$q_t^* = \arg \max_{1 \leq i \leq N} [\gamma_t(i)] \quad \forall t = 1, \dots, T$$

The problem with choosing the individually most likely states is that the overall state sequence may not be valid

- Consider a situation where the individually most likely states are $q_t = s_i$ and $q_{t+1} = s_j$ but the transition probability $a_{ij} = 0$

To avoid this and other problems, it is common to look for the single best state sequence, at the expense of having sub-optimal individual states

- This is accomplished with the Viterbi algorithm

Problem 2: The Viterbi algorithm

To find the single best state sequence, we define another variable

$$\delta_t(i) = \max_{q_1, \dots, q_{t-1}} p[q_1, \dots, q_{t-1}, q_t = s_i, o_1, o_2, \dots, o_t | \lambda]$$

which represents the highest probability along a single path that accounts for the first t observations and ends at state s_i

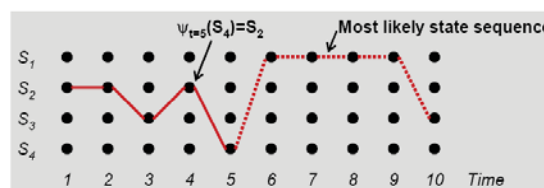
- By induction, $\delta_{t+1}(j)$ can be computed as

$$\delta_{t+1}(j) = \max_i [\delta_t(i) a_{ij}] b_j(o_{t+1})$$

- To retrieve the state sequence, we also need to keep track of the state that maximizes $\delta_t(i)$ at each time t , which is done by constructing an array

$$\psi_{t+1}(j) = \arg \max_{1 \leq i \leq N} [\delta_t(i) a_{ij}]$$

- $\psi_{t+1}(j)$, which is the state at time t that maximizes the probability $\delta_{t+1}(j)$ when transiting to state s_j



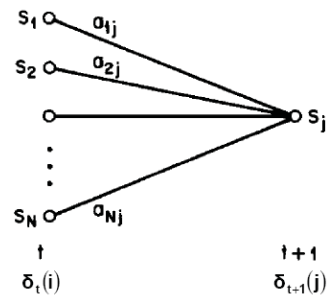
Problem 2: The Viterbi algorithm

The Viterbi algorithm for finding the optimal state sequence

- Initialization
 - $\delta_1(i) = \pi_i b_i(o_1), 1 \leq i \leq N$
 - $\psi_1(i) = 0$ (no previous states)
- Recursion ($t = 2 \dots T$) for $j = 1 \dots N$

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] b_j(o_t)$$

$$\psi_t(j) = \arg \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}]$$



From [Rabiner, 1989]

- Termination
 - $p^* = \max_{1 \leq i \leq N} [\delta_T(i)]$
 - $q_T^* = \arg \max_{1 \leq i \leq N} [\delta_T(i)]$

And the optimal state sequence can be retrieved by backtracking

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \quad t = T - 1 \dots 1$$

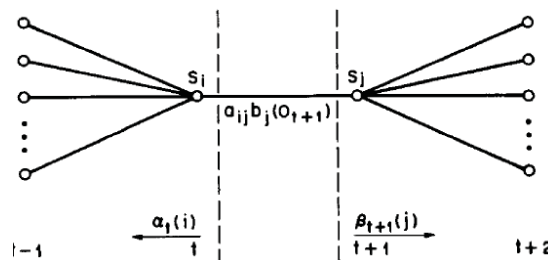
Notice that the Viterbi algorithm is similar to the Forward procedure, except that it uses a maximization over previous states instead of a summation

Problem 3: Parameter estimation

The most important and difficult problem in HMMs is to find the model parameters $\lambda = (A, B, \pi)$ from data

- HMMs are trained with the Maximum Likelihood criterion: seek model parameters that best explain the observations, as measured by $p(\mathcal{O}|\lambda)$
- This problem is solved with an iterative procedure known as Baum-Welch, which is an implementation of the EM algorithm

We define a variable $\xi_t(i, j) = p(q_t = s_i, q_{t+1} = s_j | \mathcal{O}, \lambda)$, the probability of being at state s_i at time t and state s_j at time $t + 1$.



Problem 3: Parameter estimation

From the definition of $\alpha_t(i)$, and $\beta_t(j)$ and conditional probability we can rewrite the equation as follows:

$$\xi_t(i, j) = \frac{p(q_t = s_i, q_{t+1} = s_j, \mathcal{O} | \lambda)}{p(\mathcal{O} | \lambda)} = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{p(\mathcal{O} | \lambda)} = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}$$

Intuitive interpretation of $\gamma_t(i)$ and $\xi_t(i, j)$

- $\gamma_t(i)$ is the probability of being at state s_i at time t given the observation sequence \mathcal{O} and the model λ . It can be related to $\xi_t(i, j)$ by $\gamma_t(i) = \sum_{j=1}^N \xi_t(i, j)$
- The summation of $\gamma_t(i)$ over time is the expected number of times that state s_i is visited or the number of transitions from s_i .

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{expected number of transitions from state } s_i \text{ in } \mathcal{O}$$

- Similarly, the sum of $\xi_t(i, j)$ from $t = 1$ to $t = T - 1$ is the expected number of transitions from state s_i to state s_j .

$$\sum_{t=1}^{T-1} \xi_t(i, j) = \text{expected number of transitions from state } s_i \text{ to state } s_j$$

Problem 3: Parameter estimation

Using this line of reasoning, we can produce a method to iteratively update the parameters of an HMM by simply "counting events"

$$\begin{aligned} \hat{\pi}_i &= \gamma_1(i), \text{ expected frequency (number of times) in state } s_i \text{ at time } (t = 1) \\ \hat{a}_{ij} &= \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\ \hat{b}_j(k) &= \frac{\sum_{t=1, s.t. o_t=v_k}^{T-1} \gamma_t(j)}{\sum_{t=1}^{T-1} \gamma_t(j)} \end{aligned}$$

- where the right-hand side of the equations is computed from the "old" parameter values, and the left-hand side are the re-estimated (new) parameters
- It can be shown that each iteration of this procedure increases the likelihood of the data until a local minimum is found $p(\mathcal{O} | \lambda^{new}) \geq p(\mathcal{O} | \lambda^{old})$
- This property is due to the fact that Baum-Welch is just an implementation of the Expectation-Maximization algorithm

Problem 3: Parameter estimation

Baum-Welch is an implementation of the EM algorithm where

- The observation sequence $\mathcal{O} = \{o_1, o_2, o_3, \dots\}$ is the observed data.
- The underlying state sequence $\mathcal{Q} = \{q_1, q_2, q_3, \dots\}$ is the missing or hidden data
- The incomplete data likelihood is $p(\mathcal{O}|\lambda)$
- The complete data likelihood is $p(\mathcal{O}, \mathcal{Q}|\lambda)$

Therefore, the auxiliary Q function from EM becomes

$$Q(\theta|\theta^{i-1}) = \mathbb{E}_{\mathcal{Z}}[\ln p(X, \mathcal{Z}|\theta)|X, \theta^{i-1}]$$

from which the expected value $\mathbb{E}_{\mathcal{Q}}$ is computed by averaging over all the state sequences.

$$Q(\lambda|\lambda^{i-1}) = \mathbb{E}_{\mathcal{Q}}[\ln p(\mathcal{O}, \mathcal{Q}|\lambda)|\mathcal{O}, \lambda^{i-1}] = \sum_{\forall \mathcal{Q}} \ln p(\mathcal{O}, \mathcal{Q}|\lambda) p(\mathcal{Q}|\mathcal{O}, \lambda)$$

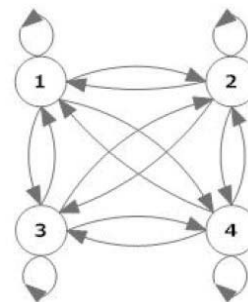
In re-estimation procedure the E-step (expectation) is calculation of the auxiliary function Q , and the M step is the maximization over λ^{i-1} .

Details on this derivation can be found in [Rabiner and Juang, 1993; Bilmes, 1998]

Types of HMM structure

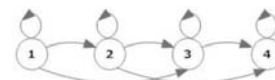
An ergodic HMM is a fully connected model, where each state can be reached in one step from every other state

- This is the most general type of HMM, and the one that has been implicitly assumed in the previous derivations



A left-right model is one where no transitions are allowed to states whose indices are lower than the current state: $a_{ij} = 0 \quad \forall j < i$

- Left-right models are best suited to model signals whose properties change over time, such as speech



Implementation issues for HMMs

Scaling

- Since $\alpha_t(i)$ involves the product of a large number of terms that are less than one, the machine precision is likely to be exceeded at some point in the computation
- To solve this problem, the $\alpha_t(i)$ are re-scaled periodically (e.g., every iteration t) to avoid underflow. A similar scaling is done to the $\beta_t(i)$

Multiple observation sequences

- The HMM derivation in these lectures is based on a single observation sequence. This becomes a problem in left-right models, since the transient nature of the states only allows a few observations to be used for each state
- For this reason, one has to use multiple observation sequences. Re-estimation formulas for multiple sequences can be found in [Rabiner and Juang, 1993]

Initial parameter estimates

- How are the initial HMM parameters chosen so that the local maximum to which Baum-Welch converges to is actually the global maximum?
- Random or uniform initial values for p and A have experimentally been found to work well in most cases
- Careful selection of initial values for B , however, has been found to be helpful in the discrete case and essential in the continuous case. These initial estimates may be found by segmenting the sequences with k-means clustering

Continuous HMMs

The discussion thus far has focused on discrete HMMs

- Discrete HMMs assume that the observations are defined by a set of discrete symbols from a finite alphabet
- In most pattern recognition applications, however, observations are inherently multidimensional and having continuous features

Continuous HMMs

The discussion thus far has focused on discrete HMMs

- Discrete HMMs assume that the observations are defined by a set of discrete symbols from a finite alphabet
- In most pattern recognition applications, however, observations are inherently multidimensional and having continuous features

Two alternatives to handle continuous vectors with HMMs

- Convert the continuous multivariate observations into discrete univariate observations via a codebook (e.g., cluster the observations with k-means). This approach, however, may lead to degraded performance as a result of the discretization of the continuous signals

Continuous HMMs

The discussion thus far has focused on discrete HMMs

- Discrete HMMs assume that the observations are defined by a set of discrete symbols from a finite alphabet
- In most pattern recognition applications, however, observations are inherently multidimensional and having continuous features

Two alternatives to handle continuous vectors with HMMs

- Convert the continuous multivariate observations into discrete univariate observations via a codebook (e.g., cluster the observations with k-means). This approach, however, may lead to degraded performance as a result of the discretization of the continuous signals
- Employ HMM states that have continuous observation densities $b_j(\cdot)$

Continuous HMMs

Continuous HMMs model the observation probabilities with a continuous density function, as opposed to a multinomial

- The most common form is the Gaussian mixture model

$$b_j(\mathbf{o}) = \sum_{k=1}^M c_{jk} \mathcal{N}(\mathbf{o}, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk})$$

where \mathbf{o} is the observation vector, c_{jk} is the mixture coefficient, $\boldsymbol{\mu}_{jk}$ is the mean vector, $\boldsymbol{\Sigma}_{jk}$ is the covariance matrix for the k^{th} Gaussian component in state s_j , and M is the number of Gaussians

Continuous HMMs

Continuous HMMs model the observation probabilities with a continuous density function, as opposed to a multinomial

- The most common form is the Gaussian mixture model

$$b_j(\mathbf{o}) = \sum_{k=1}^M c_{jk} \mathcal{N}(\mathbf{o}, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk})$$

where \mathbf{o} is the observation vector, c_{jk} is the mixture coefficient, $\boldsymbol{\mu}_{jk}$ is the mean vector, $\boldsymbol{\Sigma}_{jk}$ is the covariance matrix for the k^{th} Gaussian component in state s_j , and M is the number of Gaussians

The re-estimation formulas for the continuous case generalize very gracefully from the discrete HMM

- $\gamma_t(j)$ generalizes to $\gamma_t(j, k)$ which is the probability of being at state j at time t with the k^{th} mixture component accounting for the observation \mathbf{o}_t

$$\gamma_t(j, k) = \left[\frac{\alpha_t(i) \beta_t(j)}{\sum_{j=1}^N \alpha_t(i) \beta_t(j)} \right] \left[\frac{c_{jk} \mathcal{N}(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk})}{\sum_{m=1}^M c_{jk} \mathcal{N}(\mathbf{o}_t, \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})} \right]$$

Continuous HMMs

The re-estimation formulas for the continuous HMM become

$$\hat{c}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k)}{\sum_{t=1}^T \sum_{k=1}^M \gamma_t(j, k)} \quad \hat{\mu}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k) \mathbf{o}_t}{\sum_{t=1}^T \gamma_t(j, k)} \quad \hat{\Sigma}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k) (\mathbf{o}_t - \mu_{jk})(\mathbf{o}_t - \mu_{jk})^T}{\sum_{t=1}^T \gamma_t(j, k)}$$

Continuous HMMs

The re-estimation formulas for the continuous HMM become

$$\hat{c}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k)}{\sum_{t=1}^T \sum_{k=1}^M \gamma_t(j, k)} \quad \hat{\mu}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k) \mathbf{o}_t}{\sum_{t=1}^T \gamma_t(j, k)} \quad \hat{\Sigma}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k) (\mathbf{o}_t - \mu_{jk})(\mathbf{o}_t - \mu_{jk})^T}{\sum_{t=1}^T \gamma_t(j, k)}$$

- The re-estimation formula c_{jk} is the ratio between the expected number of times the system is in state j using the k^{th} mixture component, and the expected number of times the system, is in state j .

Continuous HMMs

The re-estimation formulas for the continuous HMM become

$$\hat{c}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k)}{\sum_{t=1}^T \sum_{k=1}^M \gamma_t(j, k)} \quad \hat{\mu}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k) \mathbf{o}_t}{\sum_{t=1}^T \gamma_t(j, k)} \quad \hat{\Sigma}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k) (\mathbf{o}_t - \mu_{jk})(\mathbf{o}_t - \mu_{jk})^T}{\sum_{t=1}^T \gamma_t(j, k)}$$

- The re-estimation formula c_{jk} is the ratio between the expected number of times the system is in state j using the k^{th} mixture component, and the expected number of times the system, is in state j .
- The mean vector μ_{jk} re-estimation formula weights the numerator in the equation c_{jk} by the observation, to produce the portion of the observation that can be accounted by mixture component.

Continuous HMMs

The re-estimation formulas for the continuous HMM become

$$\hat{c}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k)}{\sum_{t=1}^T \sum_{k=1}^M \gamma_t(j, k)} \quad \hat{\mu}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k) \mathbf{o}_t}{\sum_{t=1}^T \gamma_t(j, k)} \quad \hat{\Sigma}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k) (\mathbf{o}_t - \mu_{jk})(\mathbf{o}_t - \mu_{jk})^T}{\sum_{t=1}^T \gamma_t(j, k)}$$

- The re-estimation formula c_{jk} is the ratio between the expected number of times the system is in state j using the k^{th} mixture component, and the expected number of times the system, is in state j .
- The mean vector μ_{jk} re-estimation formula weights the numerator in the equation c_{jk} by the observation, to produce the portion of the observation that can be accounted by mixture component.
- The re-estimation formula for the transition probabilities is the same as in the discrete HMM

Motion learning and recognition

- Coordinate-free representations of a rigid body motion.
- A person perform 20 demonstrations of 9 different class of motions.
- Invariant movements are learned in the form of HMMs.
- HMM parameters are trained by Expectation-Maximization.
- The likelihood for all the HMMs is computed by Forward-Backward procedure.
- The motion is assigned to the class with higher likelihood if this likelihood is over a certain threshold.



De Schutter et al., *Recognition of 6 DOF Rigid Body Trajectories using a Coordinate-Free Representation*, ICRA, 2011.



Continuous motion recognition

- Each gesture consists of a time series of hand positions (features).
- Gestures are quantized using K -means to generate K observable symbols.
- *Set Median String (SMS)* as gesture prototype.
 - ▶ SMS: the string, belonging to the set, that minimizes the sum of the distances above all the strings of a given motion class.
 - ▶ Levenshtein distance as metric

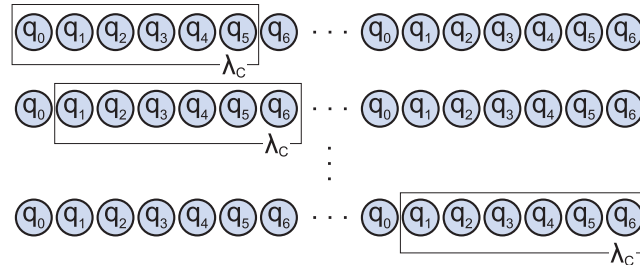



S. Iengo, S. Rossi, M. Staffa and A. Finzi, *Continuous Gesture Recognition for Flexible Human-Robot Interaction*, International Conference on Robotics and Automation, 2014.



Continuous motion recognition

- Prototype of each class learned using a DHMM.
- Temporal sliding method for continuous gesture recognition.



 S. Ingo, S. Rossi, M. Staffa and A. Finzi, *Continuous Gesture Recognition for Flexible Human-Robot Interaction*, International Conference on Robotics and Automation, 2014.

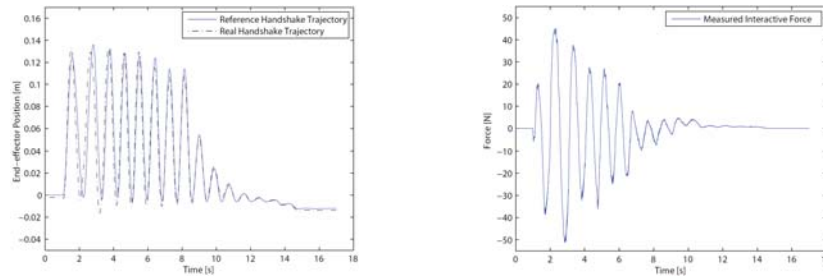
Human robot physical interaction

- Task: ensure the correct behaviour (stiff or compliant) of the robot during the handshake.
- An impedance control with time varying parameters to ensure a compliant behaviour when needed.
- Human impedance parameters estimated by Recursive Least Square.
- Human intentions estimation using discrete HMM.

 Z. Wang, A. Peer and M. Buss, *An HMM approach to realistic haptic human-robot interaction*, World Haptics Conference, 2009.

Understand human intentions from physical interaction

- Symbols Abstractions: Human impedance parameters (Mass-Damping-Stiffness) are abstracted in 8 binary (low - high) observable symbols.
- A discrete HMM with 1 hidden state, representing the Active/Passive behaviour during the handshake is learned.



Z. Wang, A. Peer and M. Buss, *An HMM approach to realistic haptic human-robot interaction*, World Haptics Conference, 2009.

Announcements

- Reading
 - Duda, Chapter 3.10
 - Bishop, Chapter 13.1-13.2

Machine Learning in Robotics

Lecture 11: Introduction to Reinforcement Learning

Prof. Dongheui Lee

*Institute of Automatic Control Engineering
Technische Universität München*

dhlee@tum.de



<http://www.lsr.ei.tum.de/>



Reinforcement Learning

Learning of a behavior without explicit information about correct actions

- Between supervised and unsupervised learning
- No training patterns, but **rewards**
- Inspired by principles of human and animal learning
- Mild assumptions on the process to be controlled
- A control strategy can be learned from scratch



RL

Model based RL

Model free RL

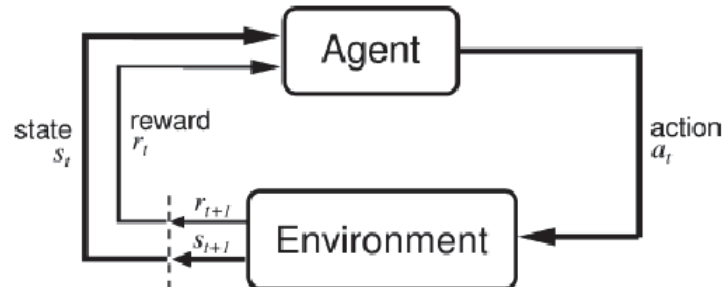
Application

2



Architecture

The agent-environment interaction in reinforcement learning



The Environment

- The environment contains the process to be controlled
- Markov Decision Process (MDP): The environment is modeled by an MDP which is tuple $(S, A, \{P_{sa}\}, \gamma, R)$
 - ▶ S is a set of **states**
 - ▶ A is a set of **actions**
 - ▶ P_{sa} are the **state transition probabilities**.
 - ▶ $\gamma \in [0, 1)$ is the **discount factor**.
 - ▶ $R : S \times A \mapsto \mathbb{R}$ is the **reward function** (Rewards can also be a function of state S only and in that case $R : S \mapsto \mathbb{R}$).

Markov Process, HMM, Markov Decision Process

Task for the Agent

Find a behavior which maximizes the expected total reward

For how long should we consider?

Finite Horizon

$$\max \left[\sum_{t=0}^T r_t \right]$$

Infinite Horizon

$$\max \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

γ is a discount factor ($0 \leq \gamma < 1$)

Reward function

The reward function controls which task should be solved

- Game (Checkers, chess)
Reward only at end: +1 when winning, -1 when loosing
- Avoiding mistakes (pole balancing)
Reward -1 at the end (when falling)
- Find a fast/short/cheap path to a goal
Reward -1 at each step

Simplifying assumptions

- Discrete time
- Finite number of actions $a_i \in a_1, a_2, a_3, \dots, a_n$
- Finite number of states $s_i \in s_1, s_2, s_3, \dots, s_m$
- Environment is a stationary markov decision process
- Reward r only depends on s

Policy and Value function

- Policy
Policy provides a mapping from states to action.

$$\pi(s) \mapsto a$$

- Value Function
Expected total future reward when starting from s and following policy π

$$\begin{aligned} V^\pi(s) &= E[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots | s_0 = s, \pi] \\ &= R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s') \quad (\text{Bellman's equation}) \end{aligned}$$

Optimal Policy

An optimal policy is the the one which maximizes the value function

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$$

$$\pi^*(s) = \arg \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$$

Classical problem: Grid world

- 11 states. Each **state** is represented by a position in the grid world.
- The agent **acts** deterministically by moving to other position.
 $A=\{N,S,E,W\}$
- reward: $R(4,3) = 1$, $R(4,2) = -1$, $R(s) = -0.02$ for all other states
- transition probability: 0.8 for a planned state and 0.1 for the other adjacent two states.

Value Iteration

For each state s , initialize $V(s) := 0$.
 Repeat until convergence
 {
 For every state, update $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s') V(s')$
 }

$V(s)$ can be updated in synchronous and asynchronous manner.

Policy Iteration

Initialize π randomly.
Repeat until convergence
{
(a) Let $V := V^\pi$
(b) For each state s , let $\pi(s) := \arg \max_{a \in A} \sum_{s'} P_{sa}(s') V(s')$
}

Step (a) can be calculated by solving linear equations (with equal number of equations and unknowns).

Monte-Carlo Method

Start at some random state.
Follow π , store the rewards and s_t .
When the goal is reached, update $V^\pi(s)$ estimation for all visited states with the future reward we actually received.

- Monte-Carlo method is suitable only for episodic tasks
- Learns incrementally from episode-by-episode but not step-by-step

Temporal Difference Learning

There are two estimates of the value of a state:

- Before: $V^\pi(s_t)$
- After: $R_{t+1} + \gamma V^\pi(s_{t+1})$

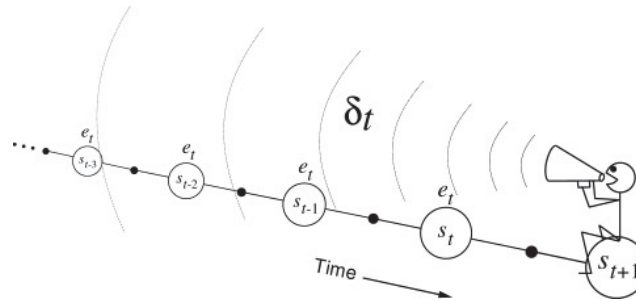
Temporal Difference Learning

Idea: The second estimate is better!

$$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha(R_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t))$$

- Learns considerably faster than the Monte-Carlo method
- Step by step learning.

Eligibility Trace



Q-Learning

Whenever reward r or next state s' cannot be predicted, we cannot calculate π even with a good estimate for V

$Q^\pi(s, a)$, is the expected infinite-horizon discounted return for executing a in state s and thereafter following π

$$Q^\pi(s, a) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t = s, a_t = a, \pi]$$

Q-Learning

Whenever reward r or next state s' cannot be predicted, we cannot calculate π even with a good estimate for V

$Q^\pi(s, a)$, is the expected infinite-horizon discounted return for executing a in state s and thereafter following π

$$Q^\pi(s, a) = E [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t = s, a_t = a, \pi]$$

$$\pi(s) = \arg \max_a Q(s, a)$$

$$V^*(s) = \max_a Q^*(s, a)$$

Q-Learning

Whenever reward r or next state s' cannot be predicted, we cannot calculate π even with a good estimate for V

$Q^\pi(s, a)$, is the expected infinite-horizon discounted return for executing a in state s and thereafter following π

$$Q^\pi(s, a) = E [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t = s, a_t = a, \pi]$$

$$\pi(s) = \arg \max_a Q(s, a)$$


$$V^*(s) = \max_a Q^*(s, a)$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

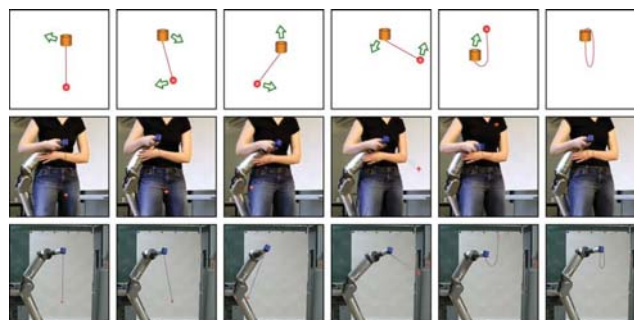
Autonomous helicopter flight via RL



Value of each policy is calculated through Monte-Carlo method *PEGASUS* method uses the observation that almost all computer simulations sample $s' \sim P_{sa}(\cdot)$ by first calling a random number generator to get one (or more) random numbers p , and then calculating s' as some deterministic function of the input s, a and the random p Since the helicopters model is stochastic, random number were fixed in advanced to evaluate different policies


 HJ Kim, Michael I Jordan, Shankar Sastry, and Andrew Y Ng., *Autonomous helicopter flight via reinforcement learning*, In Advances in neural information processing systems, 2003.

Learning Motor Primitives using Reinforcement Learning



POWER is an Expectation Maximization based RL algorithm which does not require learning rate as a parameter:

$$\theta' = \theta + \frac{E\{\sum_{t=1}^T \varepsilon_t Q^\pi(s_t, a_t, t)\}}{E\{\sum_{t=1}^T Q^\pi(s_t, a_t, t)\}} \quad \text{where } \varepsilon_t \text{ is exploration term}$$

 Jens Kober and Jan Peters, *Learning motor primitives for robotics*, pp. 2112 - 2118, ICRA, 2009.

Reading Material

- Mitchell, Chapter 13
- Russell and Norvig, Artificial Intelligence: A Modern Approach, Chapter 21
- Sutton and Barto, Reinforcement Learning: An Introduction



RL

Model based RL

Model free RL

Application

21



Machine Learning in Robotics Gaussian Processes

Prof. Dongheui Lee

*Institute of Automatic Control Engineering
Technische Universität München*

dhlee@tum.de



<http://www.lsr.ei.tum.de/>





Recall: Linear Regression



Gaussian Processes: definition

Gaussian is widely used to describe a **distribution over vector**:

$$y \sim \mathcal{N}(\mu, \Sigma)$$

Similarly, a Gaussian Process defines a **distribution over functions** $p(f(\mathbf{x}))$, where $f(\mathbf{x})$ is a scalar function mapping some input space \mathcal{X} to \mathbb{R} .

$$f : \mathcal{X} \rightarrow \mathbb{R}$$

where $f(\mathbf{x})$ is a random variable and that it can be an infinite-dimensional quantity.

$$f \sim \mathcal{GP}(m, K)$$

Definition: A stochastic process is a Gaussian process if for any finite subset $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\} \subset \mathcal{X}$, the marginal distribution $p(f(\mathbf{x}))$ over that finite subset has a multivariate Gaussian distribution



Gaussian Processes

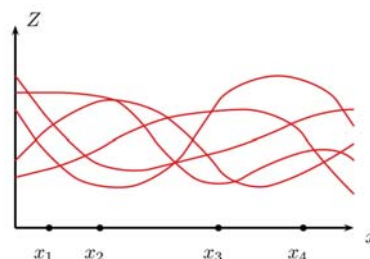
A GP is a Gaussian distribution over functions:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

A GP is parameterized by a mean function $m(\mathbf{x})$, and a covariance function (kernel function $k(\mathbf{x}, \mathbf{x}')$)

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))^T]$$



Gaussian Processes in Practice

A training dataset is given $(\mathbf{x}^{(i)}, f(\mathbf{x}^{(i)}))$ for $i = 1, \dots, n$.

$$f(\mathbf{X}) \sim \mathcal{N}(m(\mathbf{X}), K(\mathbf{X}, \mathbf{X}))$$

$$m(\mathbf{X}) = (m(\mathbf{x}^{(1)}), \dots, m(\mathbf{x}^{(n)}))^T$$

$$K(\mathbf{X}, \mathbf{X}) = \begin{bmatrix} k(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & \dots & k(\mathbf{x}^{(1)}, \mathbf{x}^{(n)}) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}^{(n)}, \mathbf{x}^{(1)}) & \dots & k(\mathbf{x}^{(n)}, \mathbf{x}^{(n)}) \end{bmatrix}$$

Kernel Functions

GP use kernels to lift input data in a higher dimensional feature space (*kernel trick*)

- The kernel function $k(\mathbf{x}, \mathbf{x}')$ is a measure of “closeness” between inputs: Close points \rightarrow similar predictions
- The kernel function has to generate a symmetric positive semi-definite covariance matrix for any set of points $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$

Common Kernel Functions

- Linear: $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$.
- Squared exponential (SE): $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sigma_f^2 \exp^{-\frac{1}{2l} \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2}$.
- Radial basis function (RBF):
 $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sigma_f^2 \exp^{-\frac{1}{2} \sum_{l=1}^d w_l (x_l^{(i)} - x_l^{(j)})^2}$, where $x_l^{(i)}$ is the l -th component of $\mathbf{x}^{(i)}$.

Gaussian Process Regression

We have a training dataset $(X$ and corresponding observations $f = f(X)$. Given a test set X_* , we want to predict the function output f_* .

The GP defines a joint distribution for $p(\mathbf{f}, \mathbf{f}_* | \mathbf{X}, \mathbf{X}_*)$.

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mu \\ \mu_* \end{bmatrix}, \begin{bmatrix} K & K_* \\ K_*^T & K_{**} \end{bmatrix} \right)$$

where $\mu = m(X)$, $\mu_* = m(X_*)$, $K = K(X, X)$, $K_* = K(X, X_*)$, $K_{**} = K(X_*, X_*)$.

To infer f_* (or $p(f_*|X, X_*, f)$), the rule for conditional multivariate Gaussians is applied.

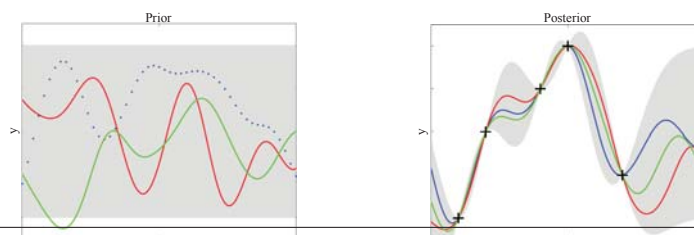
Noiseless GP regression

- **Training Data:** $\mathcal{D} = \{\mathbf{x}^{(i)}, f^{(i)} = f(\mathbf{x}^{(i)})\}_{i=1}^n$
- **Prior:**

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mu \\ \mu_* \end{bmatrix}, \begin{bmatrix} K & K_* \\ K_*^T & K_{**} \end{bmatrix} \right)$$

- **Prediction:** Given a test input x_t ,

$$\begin{aligned} p(\mathbf{f}_* | X, X_*, \mathbf{f}) &= \mathcal{N}(\mathbf{f}_* | \boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*) \\ \boldsymbol{\mu}^* &= \boldsymbol{\mu}_* + \mathbf{K}_*^T \mathbf{K}^{-1} (\mathbf{f} - \boldsymbol{\mu}) \\ \boldsymbol{\Sigma}^* &= \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_* \end{aligned}$$



Noisy GP regression

- **Training Data:** $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)} = f(\mathbf{x}^{(i)}) + \epsilon\}_{i=1}^n = (\mathbf{X}, \mathbf{y})$

$$f \sim \mathcal{GP}(0, k(\cdot, \cdot))$$

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2)$$

- **Prior:** Given a test input \mathbf{x}_t

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} + \sigma_n^2 \mathbf{I} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{bmatrix}\right)$$

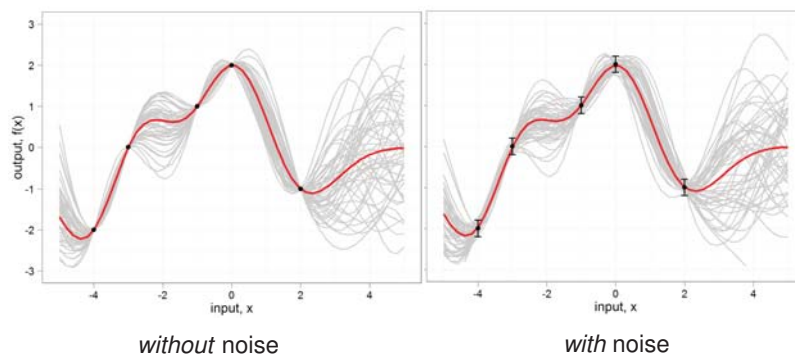
- **Predictive distribution:**

$$p(\mathbf{f}_* | \mathbf{X}, \mathbf{X}_*, \mathbf{y}) = \mathcal{N}(\mathbf{f}_* | \boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)$$

$$\boldsymbol{\mu}^* = \boldsymbol{\mu}_* + \mathbf{K}_*^T [\mathbf{K} + \sigma_n^2 \mathbf{I}]^{-1} (\mathbf{y} - \boldsymbol{\mu})$$

$$\boldsymbol{\Sigma}^* = \mathbf{K}_{**} - \mathbf{K}_*^T [\mathbf{K} + \sigma_n^2 \mathbf{I}]^{-1} \mathbf{K}_*$$

Regression Examples



The role of the Hyper-parameters

- Kernels are functions of a set of *hyper-parameters* θ
 - Example: Squared Exponential kernel
$$k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sigma_f^2 \exp^{-\frac{1}{2l} \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2}$$

$$\theta = [\sigma_f, l]$$
- GP regression results strongly depends on the used hyper-parameters

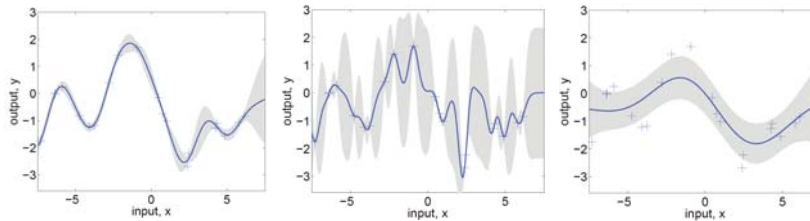


Figure: GP with different hyperparameters. (left) $l = 1$, (middle) $l = 0.3$, (right) $l = 3$

Learning the Hyper-parameters

Hyper-parameters can be learned from observations, which makes GP a parameter-free approach

HOW: Maximize the marginal log-likelihood conditioned on the hyper-parameters θ

- $p(\mathbf{y}|\mathbf{X}, \theta) = \mathcal{N}(\mathbf{0}, \mathbf{K}_y) = \mathcal{N}(\mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I})$
- $\log p(\mathbf{y}|\mathbf{X}, \theta) = -\frac{1}{2} \mathbf{y}^T \mathbf{K}_y^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K}_y| - \frac{n}{2} \log 2\pi$
- $\frac{\partial}{\partial \theta_j} \log p(\mathbf{y}|\mathbf{X}, \theta) = 0 \leftarrow$ Gradient descent

The terms of the marginal likelihood have a clear meaning:

- $-\mathbf{y}^T \mathbf{K}_y^{-1} \mathbf{y} / 2$ represents the data-fit
- $-\log |\mathbf{K}_y| / 2$ is the complexity penalty
- $-n \log 2\pi / 2$ is a normalization constant

Properties of GPs

- GP is a nonparametric model.
- GPs are universal approximators. (GP can fit arbitrary functions).
- Probabilistic model: it returns measure of uncertainties
- Minimal assumptions over a distribution of functions (i.e., joint Normal)
- GP for regression and classification.
- Kernel functions: Design of kernel functions for application, Hyperparameters can be learned.

Robotics Applications: PILCO

- Model-free reinforcement learning (RL) algorithms usually requires several rollouts to learn a suitable control policy
 - ▶ Extremely time consuming on real robots
- PILCO (Probabilistic Inference for Learning Control)
 - ▶ Model-based RL
 - ▶ GP for long-term predictions and policy evaluation
 - ▶ Reduced “real” experience to learn the control policy
- PILCO - Working principle
 1. Perform a rollout (real robot) and learn a model of the robot (GP)
 2. Find an optimal policy for the learned model
 3. Apply the policy to the robot
 4. Finish if task learned, otherwise repeat from 2.

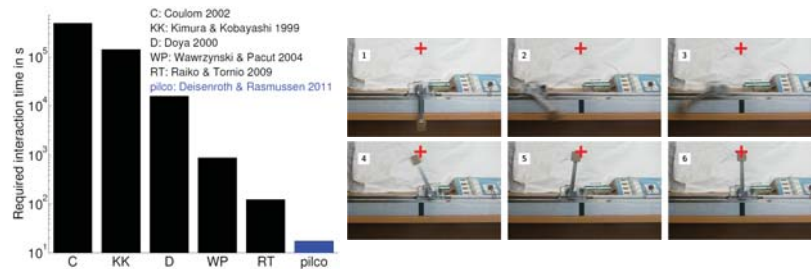


Deisenroth et. al., *Gaussian Processes for Data-Efficient Learning in Robotics and Control*, TPAMI, 2015.

Robotics Applications: PILCO

Task: Swing up a pendulum attached to a cart (cart-pole system, i.e. a cart running on a track and a freely swinging pendulum attached to the cart)


Result: PILCO is able to learn the task significantly faster than state-of-the-art model-based RL approaches



Robotics Applications: LMDS

Locally Modulated Dynamical Systems (LMDS): incrementally refine a set of predefined robotic tasks

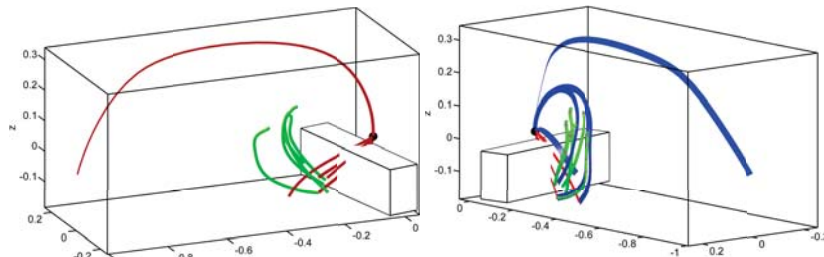
- Predefined tasks are stable dynamical systems $\dot{x} = f(x)$, where x is the robot position
- The user can provide kinesthetic demonstrations to modify the task execution
 - ▶ Learn (GP) a matrix $M(x) = (1 - \alpha(x))R(x)$ from user demonstrations
 - ▶ Modify the original task $\dot{x} = M(x)f(x)$ (do not affect the equilibrium points)

 Kronander et. al., *Incremental Motion Learning with Locally Modulated Dynamical Systems*, RAS, 2015.

Robotics Applications: LMDS

Task: insert plates into slots in a dish rack.

- The original dynamical achieve the task from different starting positions
- The user provides kinesthetic demonstrations to prevent the robot to hit the rack and the task is correctly executed



Introduction

Regression

Hyper-parameters

*Applications

18



Reference

- Rasmussen and Williams, Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning) - Chap. 1, 2, 3



Introduction

Regression

Hyper-parameters

Applications

19

