Introduction to Pattern Recognition

MTSC 852, Fall 2015

Sokratis Makrogiannis, Ph.D.

Department of Mathematical Sciences
Delaware State University

smakrogiannis@desu.edu

October 20, 2015

Outline

Pattern Recognition

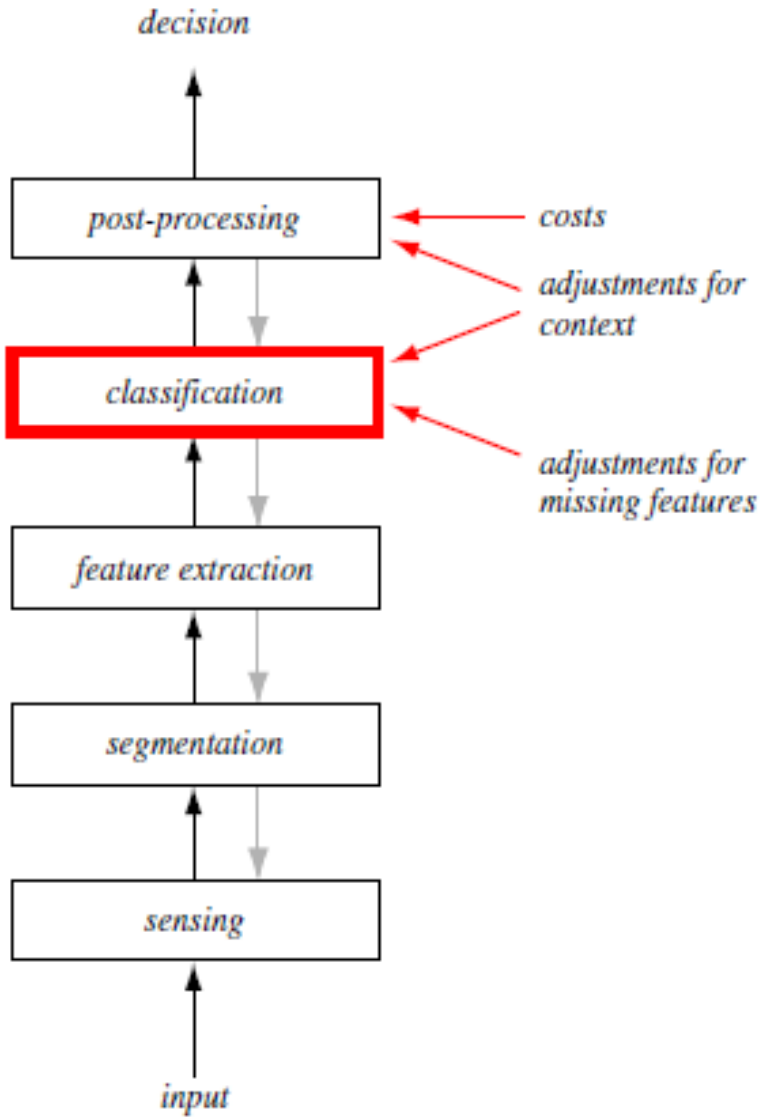
Definitions

- The research field that solves problems for recognition and classification of objects represented by sets of measurements
- "The process of giving names ω to observations $\mathbf{x}$" -Schümann
- "A problem of estimating density functions in a high-dimensional space and dividing the space into regions of catergories or classes" -Fukunaga
- "The assignment of a physical object or event to one of pre-specified categories" -Duda and Hart

Components of a Pattern Recognition System

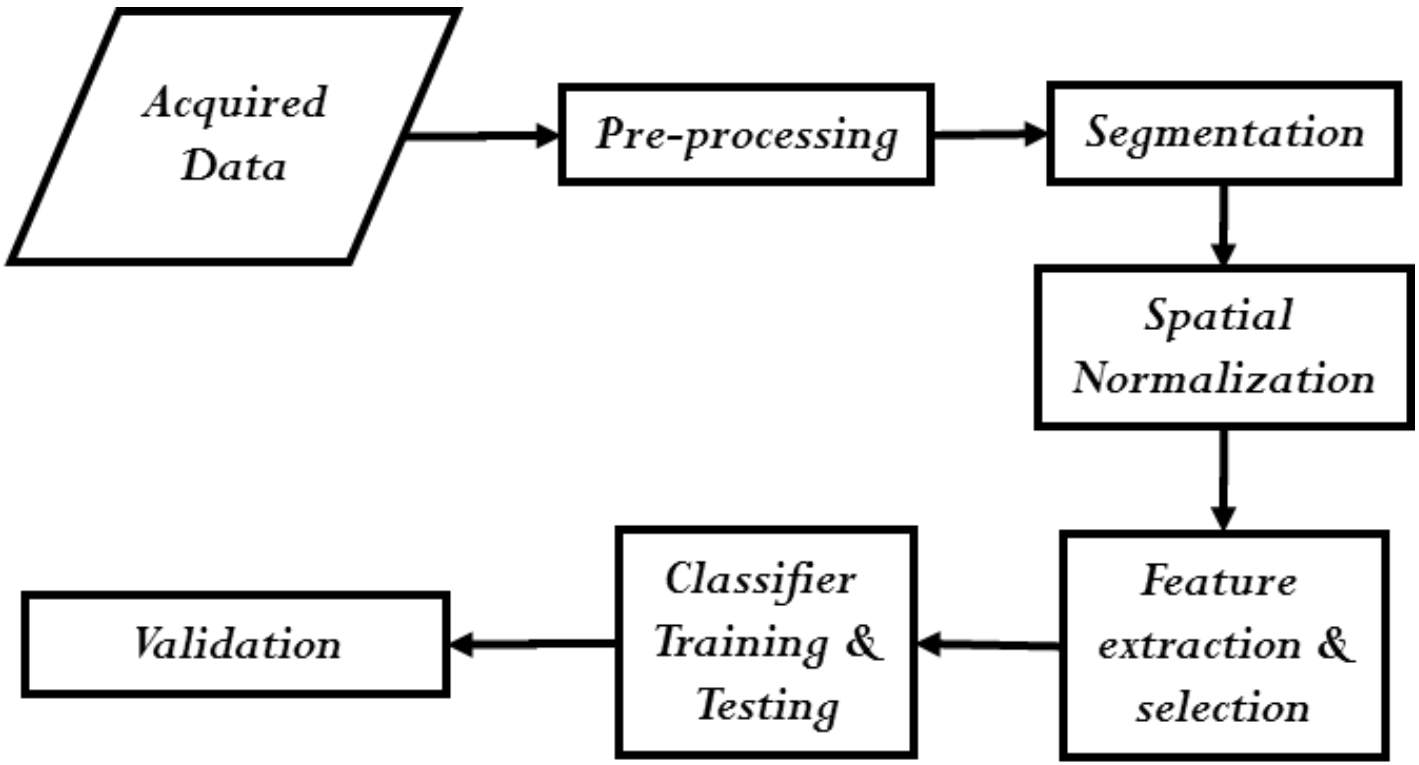
Main components

- Sensors
- Preprocessing
- Feature Extraction
- Classification
- Performance Evaluation



Pattern Recognition Systems

Main components of Brain Atrophy Recognizer

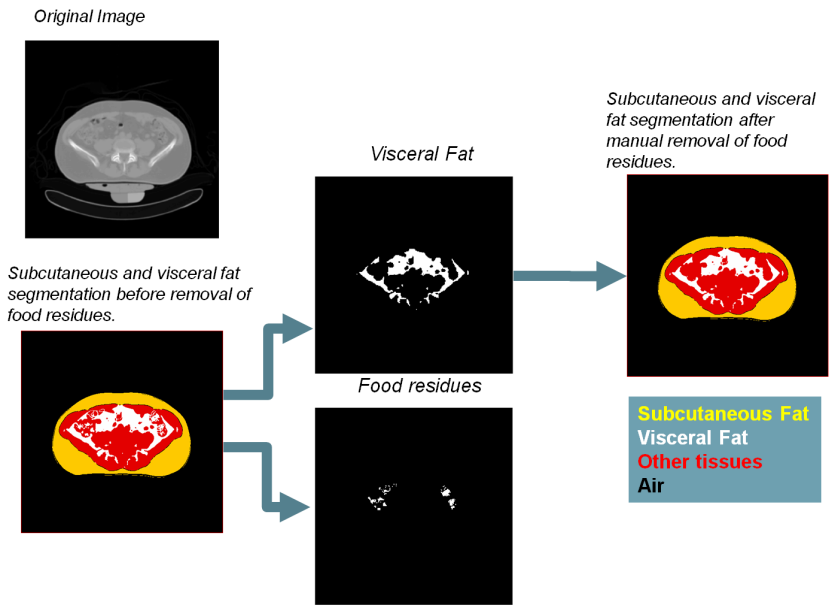


Sensing

- Different sensors mainly transducers can be used such as cameras, microphone arrays, MRI or CT scanners, fingerprint devices, etc
- Characteristics and limitations of transducers may pose challenges to pattern recognition
- Usual characteristics are bandwidth, resolution, quantization, electronic noise, distortions, signal-to-noise ratio, etc

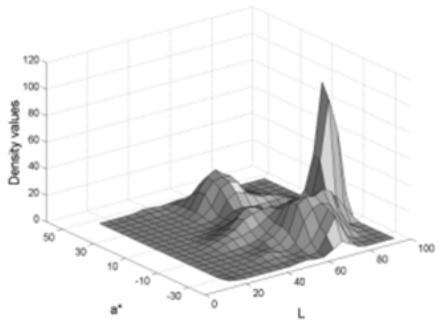
Segmentation and Grouping

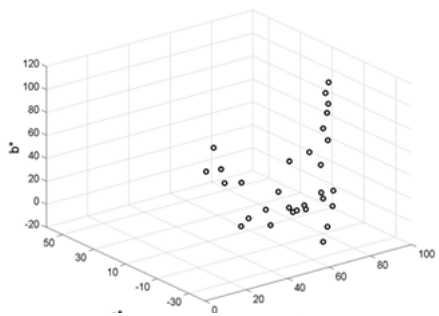
- The goal of this step is to separate the different objects
- That is, to delineate visual objects in images or videos, separate characters in manual scripts, separate phonemes for voice recognition, etc
- Segmentation is a complicated problem that depends on the domain



Feature Extraction/Dimensionality Reduction

- Goal of a feature extractor is to create object representations that are similar for objects of the same class and dissimilar for objects of different classes
- Therefore, we are seeking to compute distinguishing, or discriminant, features



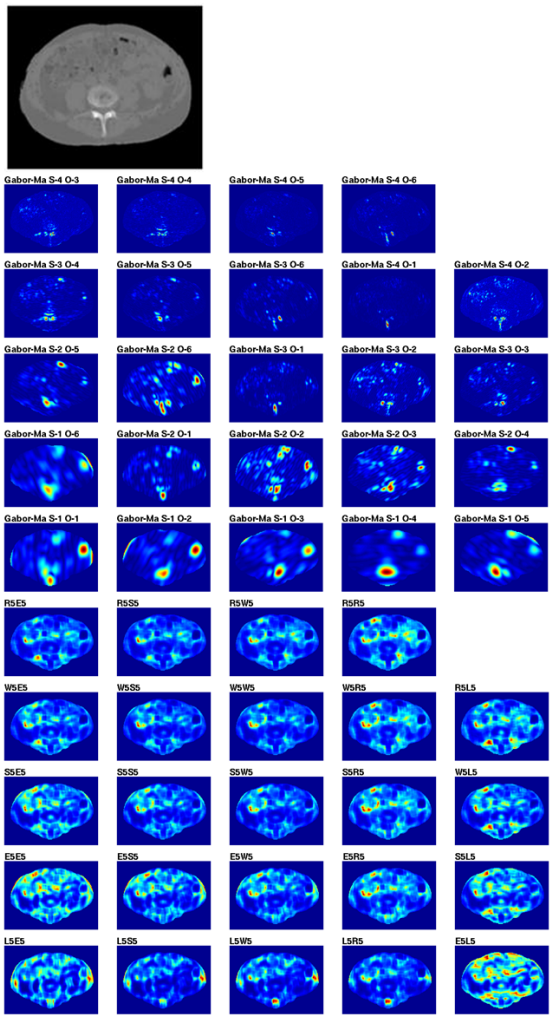


Feature Extraction/Dimensionality Reduction

- Another desired property of features mainly extracted from images is invariance to geometric transformations such as translation, rotation scaling, or nonlinear deformations, and resiliency to occlusions
- In speech recognition we are looking for features that are invariant to time translations and to changes in amplitude
- Feature computation is also domain-specific
- A related research area deals with the selection/computation of the most discriminant features for classification; this is called dimensionality reduction

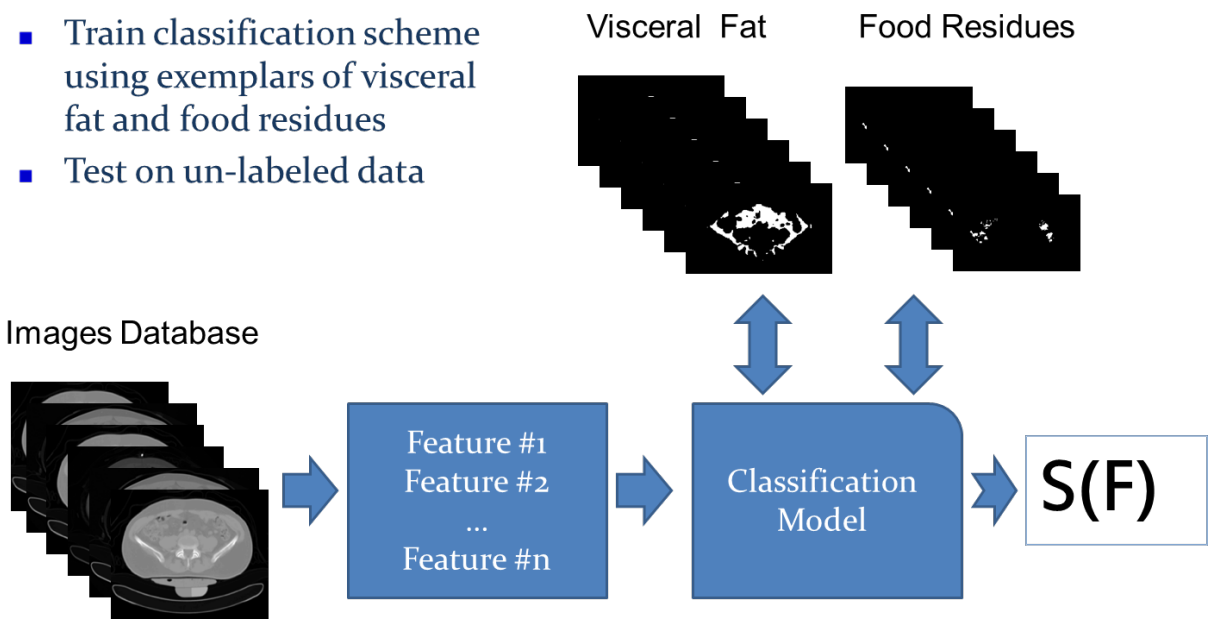
Feature Extraction Example

- Computation of texture features from an abdominal CT scan



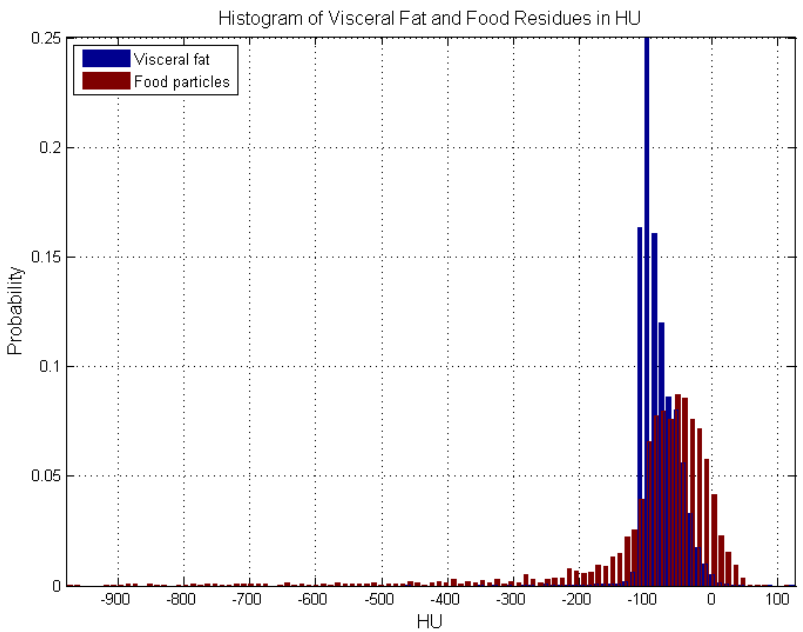
Classification

- The classifier uses as input the feature vector that represents an object and produces a category label for the object
- Classification is a major part of this work



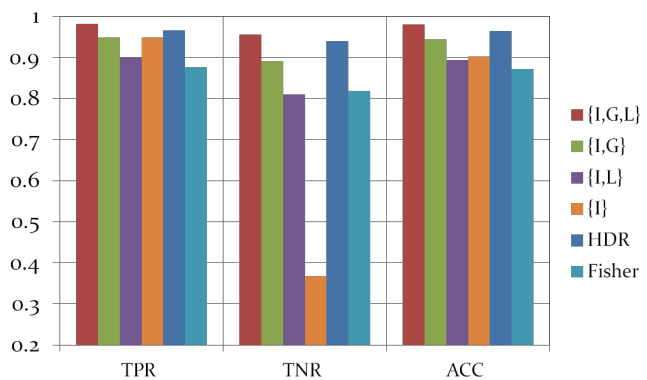
Classification

- One challenge in classification is the variability in feature values that reduce the separation between different categories
- Variability may be caused by complexity or noise
- The term noise refers to any randomness in measurements that our model cannot account for



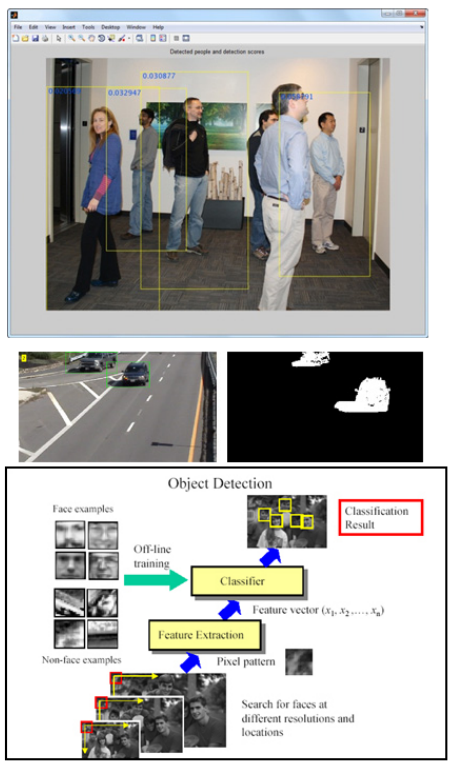
Classification Decision and Performance Evaluation

- This component uses the classifier's output to make a decision
- One criterion for making a classification decision is the error rate. The error rate is the percentage of new patterns assigned to the wrong category. Therefore we seek to minimize the error rate in our decision
- Often times we associate a risk with each classification decision. This can be used to compute a total cost that will help make a decision
- More recent approaches multiple classifiers and combine their decisions to reduce the classification error rates



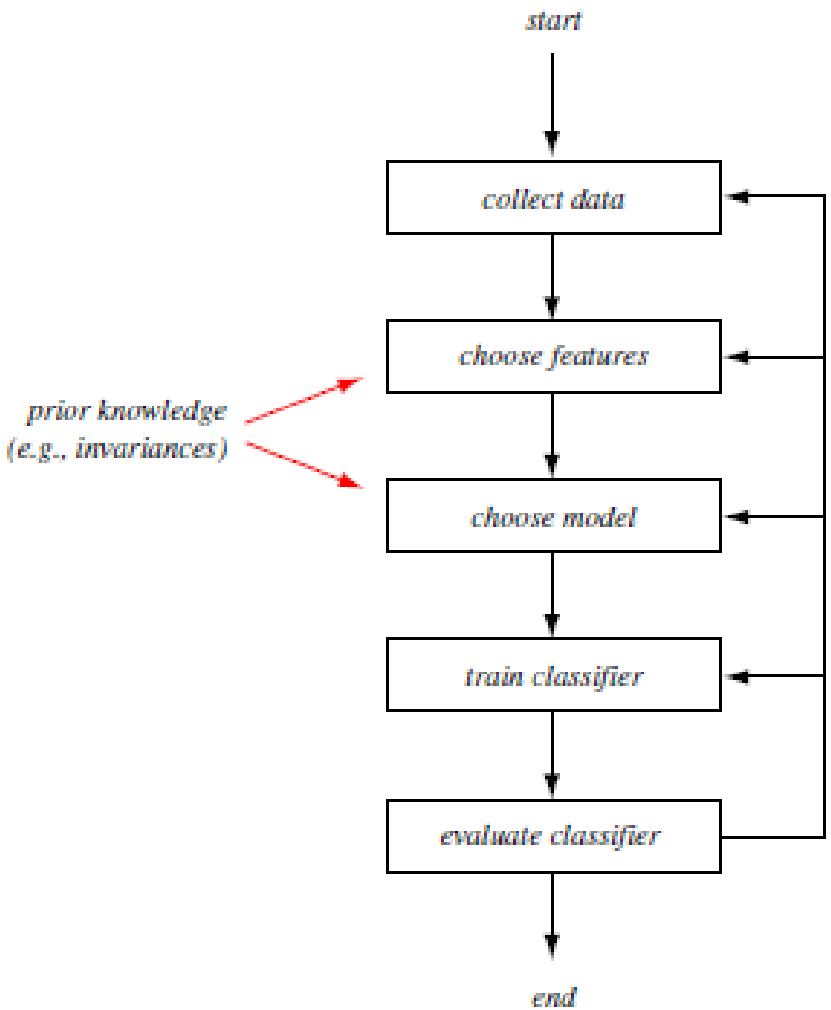
Pattern Recognition Problems

- Machine vision
  Visual inspection
  Security
- Character recognition
  Automated mail sorting
  Signature verification
- Computer-aided diagnosis
  Breast cancer diagnosis
  EEG, ECG signal analysis
- Speech recognition
  Human Computer Interaction systems



Pattern Recognition System Design Cycle

- In this section we outline the procedure that is followed to design a pattern classification system
- It is called a cycle because the design sequence is repeated until we reach the desired outcome



Data Collection

- Data collection has a significant impact on the classifier design in terms of feasibility and costs
- We often conduct a feasibility study with a small set of data to design our system, then perform a full scale study to estimate performance
- One crucial question is, how many training and testing samples are needed to design a system that can be used for real applications?

What Features to Compute

- This is a critical and domain-specific problem
- Prior knowledge may help in this stage. For example, when we design a face recognition system, we can use our knowledge of face structure like topological relations between the nose, lips, and eyes

Which Models to Use

- In this stage we develop criteria for the selection of mathematical models to represent each class
- The main question is how can we measure or predict if a hypothesized model approximates well the underlying true model of our patterns?
- Usually the answer depends on the domain; we can study the complexity, dimensionality, variability, and population of patterns

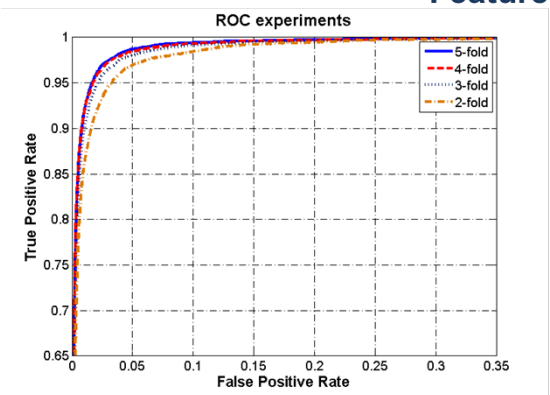
Training Stage

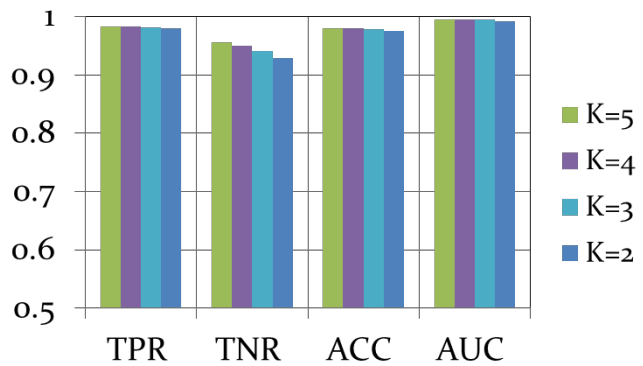
- Training is the process of determining the classifier using data that we call training data
- These are called learning from example techniques

Performance Evaluation

- This is a necessary component for determining the feature set, and classification model.
- An additional objective is to identify the potential for further improvement and to address overfitting

Feature Domain: {I,G,L}





Computational Complexity

- One consideration is how an algorithm scales as a function of the feature dimensionality, number of objects, or number of categories
- Sometimes the performance of a recognizer is excellent but it is not possible to implement
- We are mostly concerned with the complexity for making a decision; not so much for the complexity of learning stage that is executed off-line

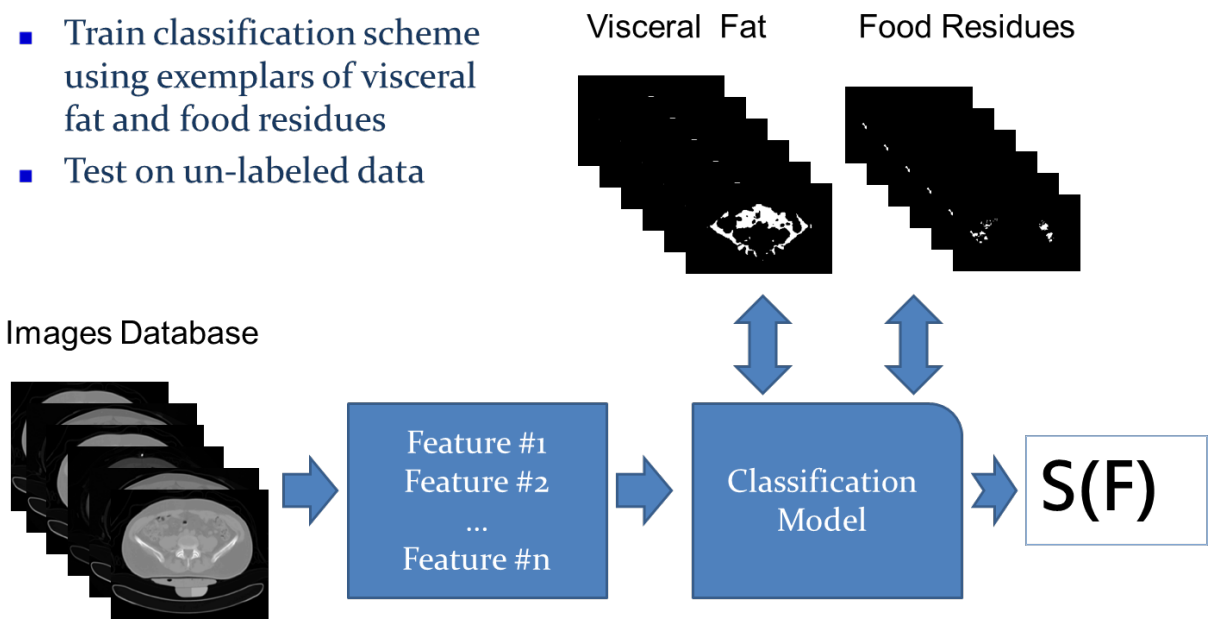
Learning Techniques

Main categories

- Supervised learning
- Unsupervised learning
- Reinforcement learning

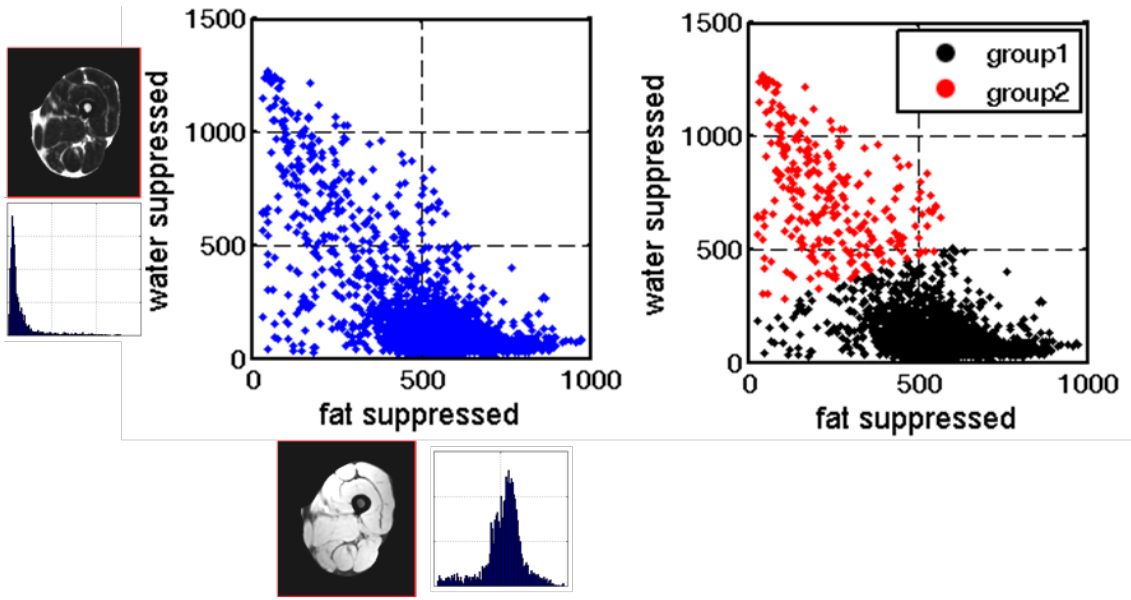
Supervised Learning

- In supervised learning, we use a training data set with patterns of known categories so we can fit our models and/or adapt the classifier parameters



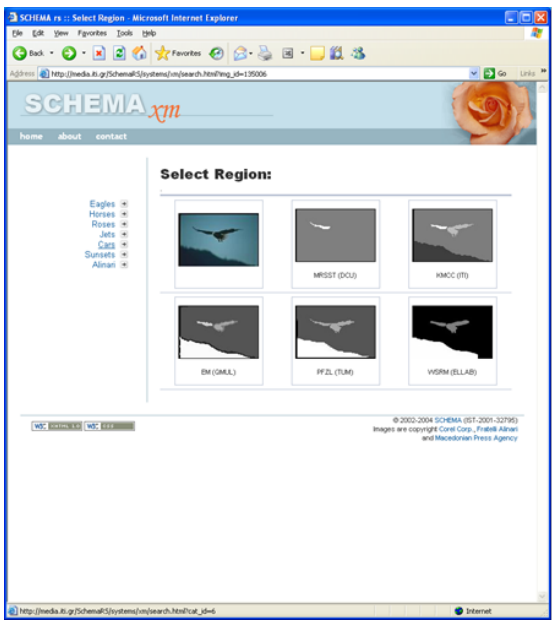
Unsupervised Learning

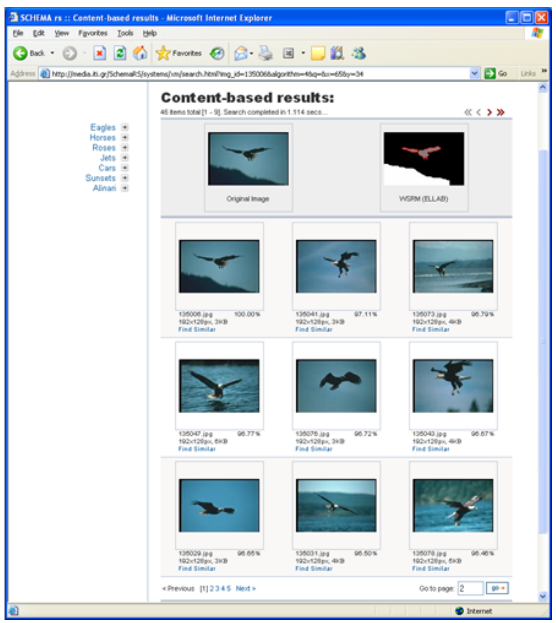
- In unsupervised learning there exists no training set and the system learns forms clusters of the input patterns using optimization algorithms
- Relevant questions:
  How do we select the number of clusters?
  How do we select the cluster models?



Reinforcement Learning

- In this type of learning, the only given feedback is that the tentative category is right or wrong, without further information

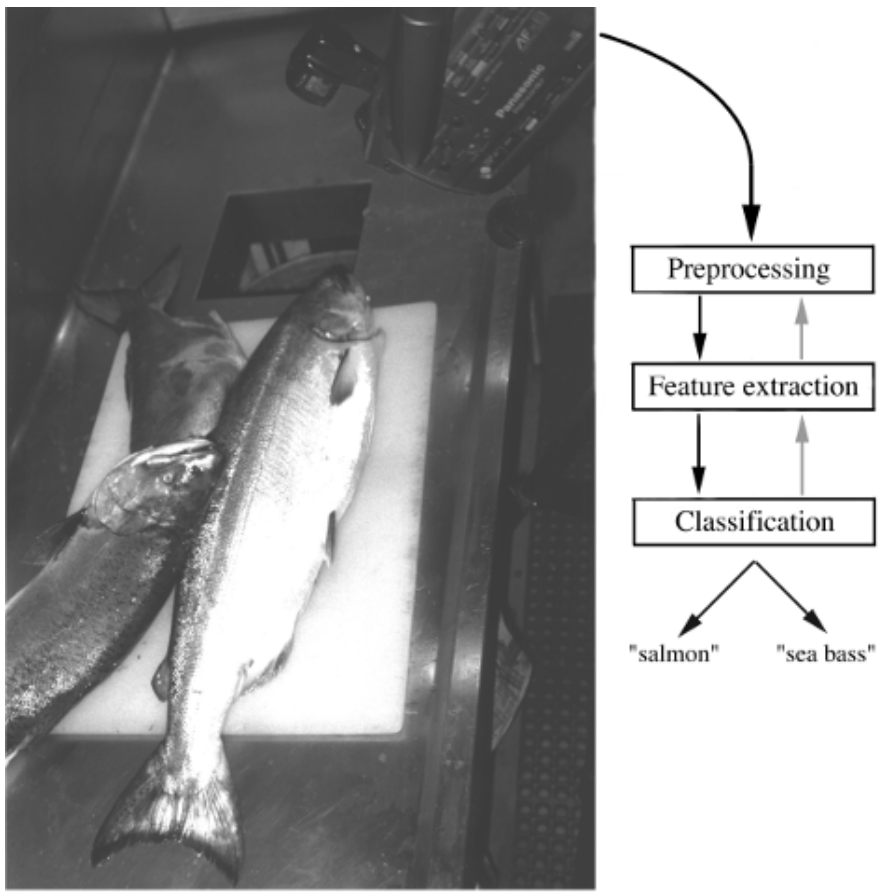




Pattern Recognition Example

Problem Description

- Suppose that we want to build a system for automatic recognition of sea bass and salmon in a fish packing plant
- The fish are placed on a conveyor belt and a fixed camera acquires images of fish on the moving conveyor belt
- We decide to use the physical characteristics such as length, width, brightness, position of the mouth of fish to separate the two species



Pattern Recognition Example

- In pattern recognition terms, we are asked to build a classifier of fish into two categories, that is salmon and sea bass
- The physical characteristics that we want to measure are called features
- To describe each fish type we develop mathematical models
- We find the best matching model for each fish to perform classification

Pattern Recognition Example

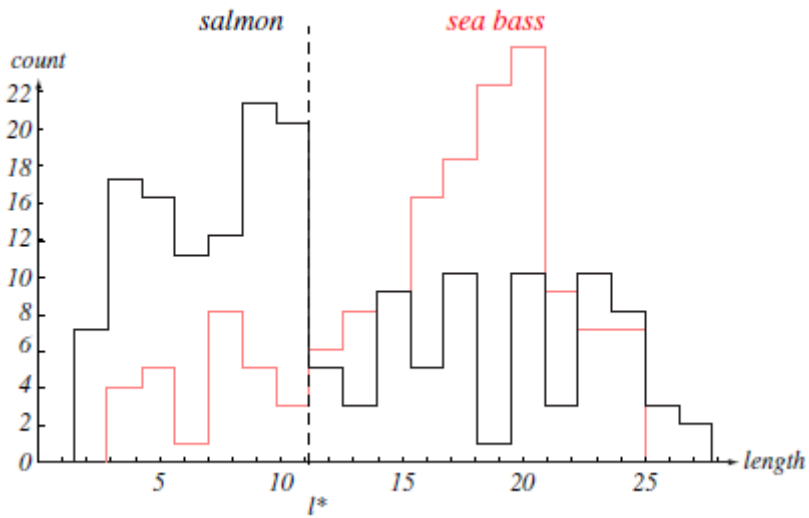
Main stages

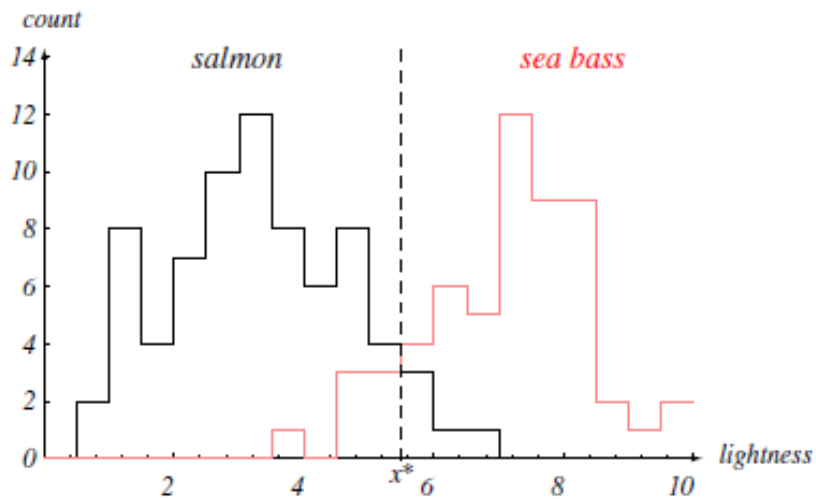
- Pre-processing: this stage can be used to improve image quality, reduce image noise, apply segmentation to separate the fish, so we can compute more accurate features
- Feature computation or extraction is the process of measuring the characteristics of fish from the sensor data, for example lightness, length, width, number of fins, etc
- The model refers to the probability density model for sea bass and for salmon and the type of boundary between these classes, that is linear, quadratic, multi-modal, etc
- Performance evaluation is the process of calculating the probability of erroneous classification decisions
- Generalization studies if the selected recognizer will sustain a good performance under real conditions, i.e., a diverse set of unlabeled patterns

Pattern Recognition Example

Feature Extraction

- Feature computation or extraction is the process of measuring the characteristics of fish from the sensor data, for example lightness, length, width, number of fins, etc

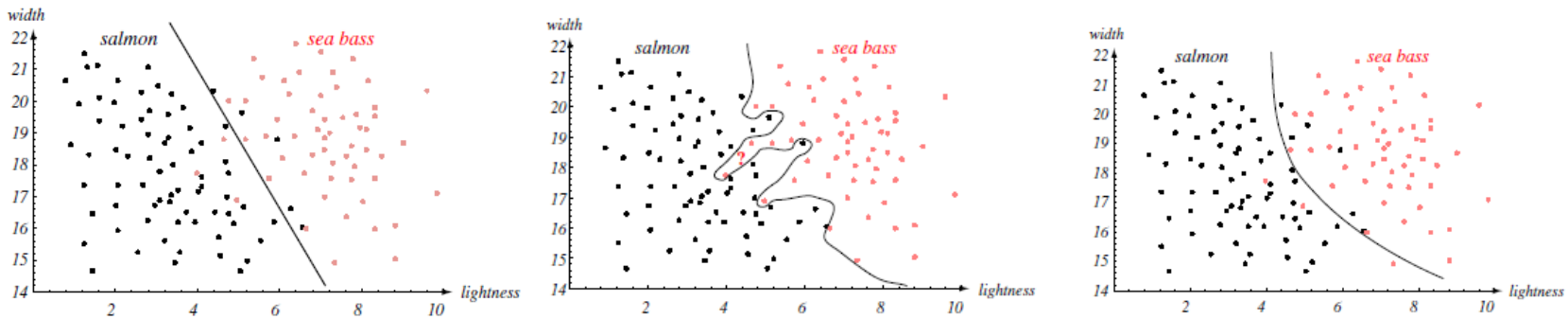




Pattern Recognition Example

Classifier Generalization

- The model refers to the probability density model for sea bass and for salmon and the type of boundary between these classes, that is linear, quadratic, multi-modal, etc
- Generalization studies if the selected recognizer will maintain a good performance level under real conditions, i.e., a diverse set of unlabeled patterns



Probability Theory Review

MTSC 852, Fall 2015

Sokratis Makrogiannis, Ph.D.

Department of Mathematical Sciences
Delaware State University

smakrogiannis@desu.edu

October 20, 2015

Outline

Notation and Preliminaries

- d-dimensional vector: $\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$

  Transpose of $\mathbf{x}$: $x^T = (x_1 x_2 \dots x_d)$

- $n \times d$ matrix: $M = \begin{pmatrix} m_{11} & m_{12} & \dots & m_{1d} \\ m_{21} & m_{22} & \dots & m_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \dots & m_{nd} \end{pmatrix}$

  Transpose of M: $M^T = \begin{pmatrix} m_{11} & m_{21} & \dots & m_{n1} \\ m_{12} & m_{22} & \dots & m_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ m_{1d} & m_{2d} & \dots & m_{nd} \end{pmatrix}$

  Symmetric matrix, if $m_{ij} = m_{ji}$

  Antisymmetric matrix, if $m_{ij} = -m_{ji}$

Notation and Preliminaries

- Multiply vector by matrix: $M\mathbf{x} = \mathbf{y}$

$$\begin{pmatrix} m_{11} & m_{12} & \dots & m_{1d} \\ m_{21} & m_{22} & \dots & m_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \dots & m_{nd} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

or,

$$y_i = \sum_{j=1}^{d} m_{ij} x_j$$

Inner Product

- The inner product of two vectors $\mathbf{x}, \mathbf{y}$ is a scalar:

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T \mathbf{y} = \mathbf{y}^T \mathbf{x} = \sum_{i=1}^{d} x_i y_i$$

- Euclidean norm: $\|\mathbf{x}\| = (\mathbf{x}^T \mathbf{x})^{1/2}$
- Angle between $\mathbf{x}$ and $\mathbf{y}$: $\cos\theta = \frac{\mathbf{x}^T \mathbf{y}}{\|x\|\|y\|}$
- $\mathbf{x}^T \mathbf{y} = 0 \rightarrow$ vectors are orthogonal
- $\|\mathbf{x}^T \mathbf{y}\| = \|x\|\|y\| \rightarrow$ vectors are colinear
- Cauchy Schwartz inequality: $\|\mathbf{x}^T \mathbf{y}\| \leq \|x\|\|y\|$
- Vectors $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$ are linearly independent, if no vector in the set can be written as a linear combination of any of the others

Outer Product

- Multiply vector by matrix: $M = \mathbf{x}\mathbf{y}^T$

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} (y_1 y_2 \dots y_n) = \begin{pmatrix} x_1 y_1 & x_1 y_2 & \dots & x_1 y_n \\ x_2 y_1 & x_2 y_2 & \dots & x_2 y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_d y_1 & x_d y_2 & \dots & x_d y_n \end{pmatrix}$$

Derivatives of Matrices

- Let $f(\mathbf{x})$ be a scalar-valued function of d variables $\mathbf{x} = (x_1 x_2 \dots x_d)^T$

  Gradient of f: $\nabla f(\mathbf{x}) = \begin{pmatrix} \partial f/\partial x_1 \\ \vdots \\ \partial f/\partial x_d \end{pmatrix}$
- Let $\mathbf{f}(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^n$. Its Jacobian matrix is:

  $\mathbf{J}(\mathbf{x}) = \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \partial f_1/\partial x_1 & \dots & \partial f_1/\partial x_d \\ \vdots & \vdots & \ddots \\ \partial f_n/\partial x_1 & \dots & \partial f_n/\partial x_d \end{pmatrix}$
- Let matrix M with elements dependent on θ. Its derivative in θ is :

  $\frac{\partial M}{\partial \theta} = \begin{pmatrix} \partial m_{11}/\partial \theta & \dots & \partial m_{1d}/\partial \theta \\ \vdots & \vdots & \ddots \\ \partial m_{n1}/\partial \theta & \dots & \partial m_{nd}/\partial \theta \end{pmatrix}$

  Derivative of inverse: $\frac{\partial}{\partial \theta} M^{-1} = -M^{-1} \frac{\partial M}{\partial \theta} M^{-1}$

  Derivative identities:

  $\frac{\partial}{\partial \mathbf{x}} [M\mathbf{x}] = M, \quad \frac{\partial}{\partial \mathbf{x}} \left[\mathbf{y}^T \mathbf{x}\right] = \frac{\partial}{\partial \mathbf{x}} \left[\mathbf{x}^T \mathbf{y}\right] = \mathbf{y}, \quad \frac{\partial}{\partial \mathbf{x}} \left[\mathbf{x}^T M \mathbf{x}\right] = \left[M + M^T\right] \mathbf{x}$

Taylor Expansions

- Taylor expansion for $f(x)$ about x_0:

$$f(x) = f(x_0) + \left.\frac{df}{dx}\right|_{x=x_0} (x - x_0) + \frac{1}{2!}\left.\frac{d^2 f}{dx^2}\right|_{x=x_0} (x - x_0)^2 + O\left((x - x_0)^3\right)$$

- Taylor expansion for vector $f(\mathbf{x})$ about $\mathbf{x}_0$:

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \left[\frac{df}{d\mathbf{x}}\right]^T_{\mathbf{x}=\mathbf{x}_0} (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2!}(\mathbf{x} - \mathbf{x}_0)^T \left[\frac{d^2 f}{d\mathbf{x}^2}\right]^T_{\mathbf{x}=\mathbf{x}_0} (\mathbf{x} - \mathbf{x}_0) + O\left(\|\mathbf{x} - \mathbf{x}_0\|^3\right)$$

Determinant, Trace, Rank

- Determinant of a $d \times d$ matrix A: $|A| = \sum_{k=1}^{d} a_{ik} |A_{i|k}| (-1)^{k+i}$
  $A_{i|k}$: minor formed by removing the ith row and kth column of A
- Trace is the sum of diagonal elements:

$$tr[A] = \sum_{k=1}^{d} a_{kk}$$

- Rank of a matrix is the number of linearly independent rows or columns
- A square matrix is non-singular $\leftrightarrow$ its rank equals the number of rows (or columns)
- A non-singular matrix has a non-zero determinant

Matrix Inversion

- Suppose square matrix $M_{d\times d}$
- Inverse of M is M^{-1}: $MM^{-1} = M^{-1}M = \mathbb{I}$
  M^{-1} exists $\leftrightarrow$ M is non-singular
  M^{-1} can be also written as $M^{-1} = \frac{Adj[M]}{|M|}$
  $Adj[M]$: matrix whose i,j entry equals to cofactor $C_{i,j} = (-1)^{j+i}M_{j|i}$
- Pseudo-inverse of M, denoted by $M^{\dagger}$, is used when A^{-1} does not exist, or when M is not square
  If M^TM is non-singular, pseudoinverse $M^{\dagger}$ is $M^{\dagger} = [M^TM]^{-1}M^T$
  Observe that $M^{\dagger}M = \mathbb{I}$
  We use pseudo-inverse to solve least squares problem
  Inverse of product of square matrices M, N: $[MN]^{-1} = N^{-1}M^{-1}$

Eigenvectors and Eigenvalues

- Suppose square matrix $M_{d\times d}$
- Frequently, we need to solve linear equations of the form

$$M\mathbf{x} = \lambda\mathbf{x} \Rightarrow (M - \lambda\mathbb{I})\mathbf{x} = 0 \Rightarrow \begin{cases} \mathbf{x} = 0 & \textbf{trivial case} \\ M - \lambda\mathbb{I} = 0 & \textbf{non-trivial case} \end{cases}$$

$$M - \lambda\mathbb{I} = 0 \Rightarrow |M - \lambda\mathbb{I}| = 0 \Rightarrow \lambda^d + a_1\lambda^{d-1} + \ldots + a_{d-1}\lambda + a_d = 0$$

  For each root we solve the set of linear equations
- Solution vectors $\mathbf{e}_i$ are called eigenvectors, while corresponding solution scalars λ_i are called eigenvalues
- M is real and symmetric $\rightarrow$ there exist d solution vectors $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d\}$ and d corresponding eigenvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_d\}$
- When multiplied by M, the eigenvectors change in magnitude but not orientation: $M\mathbf{e}_i = \lambda_i\mathbf{e}_i$

Eigenvectors and Eigenvalues Properties

- Suppose that we have found the eigendecomposition: $M\mathbf{e}_i = \lambda_i \mathbf{e}_i$. The following properties apply
  - $tr[M] = \sum_{i=1}^{d} \lambda_i$
  - $|M| = \prod_{i=1}^{d} \lambda_i$
  - M is non-singular $\rightarrow$ all eigenvalues are non-zero
  - M is real and symmetric $\rightarrow$ all eigenvalues are real and eigenvectors are orthogonal
  - M is positive definite $\rightarrow$ all eigenvalues are positive

Probability Theory Review

MTSC 852, Fall 2015

Sokratis Makrogiannis, Ph.D.

Department of Mathematical Sciences
Delaware State University

smakrogiannis@desu.edu

September 27, 2015

Outline

Outline

Outline

Outline

Outline

Discrete Random Variables

- Let x be a random variable with $x \in \mathscr{X}$ and $\mathscr{X} = \{v_1, v_2, \ldots, v_m\}$
- Then p_i is probability of $x = v_i$

$$p_i = Pr[x = v_i], i = 1, \ldots, m$$

- Axioms

$$p_i \geq 0$$

$$\sum_{i=1}^{m} p_i = 1$$

- Probability Mass Functions

$$P(x) \geq 0$$

$$\sum_{x \in \mathscr{X}} P(x) = 1$$

Expected Values

- Mean of random variable x:

$$\mathrm{E}[x] = \mu = \sum_{x \in \mathscr{X}} xP(x) = \sum_{i=1}^{m} v_i p_i$$

- For a function $f(x)$:

$$\mathrm{E}[f(x)] = \sum_{x \in \mathscr{X}} f(x)P(x)$$

- Variance $\mathrm{E}\left[(x-\mu)^2\right]$

$$\mathrm{E}\left[(x-\mu)^2\right] = \sum_{x \in \mathscr{X}} (x-\mu)^2 P(x) = \mathrm{Var}[x] = \sigma^2$$

- We can show that

$$\mathrm{Var}[x] = \mathrm{E}[x^2] - [\mathrm{E}[x]]^2$$

Pairs of Discrete Random Variables

- Let x, y be random variables such that $x \in \mathscr{X} = \{v_1, v_2, \ldots, v_m\}$ and $y \in \mathscr{Y} = \{w_1, w_2, \ldots, w_n\}$
- Joint probability:

$$p_{ij} = Pr[x = v_i, y = w_j]$$

- Axioms:

$$p_{ij} \geq 0, \quad \sum_{i=1}^{m} \sum_{j=1}^{n} p_{i,j} = 1$$

- Joint probability mass function:

$$P(x, y) \geq 0, \quad \sum_{x \in \mathscr{X}} \sum_{y \in \mathscr{Y}} P(x, y) = 1$$

- Marginal distributions:

$$P_x(x) = \sum_{y \in \mathscr{Y}} P(x, y), \quad P_y(y) = \sum_{x \in \mathscr{X}} P(x, y)$$

Statistical Independence

Definition (Statistically independent variables)

Random variables x, y are statistically independent, if and only if

$$P(x, y) = P_x(x) \cdot P_y(y)$$

Expected Values of Functions of Two Variables

- For a function $f(x,y)$:

$$\mathrm{E}[f(x,y)] = \sum_{x\in\mathscr{X}}\sum_{y\in\mathscr{Y}} f(x,y)P(x,y)$$

- Means:

$$\mathrm{E}[x] = \mu_x = \sum_{x\in\mathscr{X}}\sum_{y\in\mathscr{Y}} xP(x,y), \quad \mathrm{E}[y] = \mu_y = \sum_{x\in\mathscr{X}}\sum_{y\in\mathscr{Y}} yP(x,y)$$

$$\mathrm{E}[\mathbf{x}] = \mu = \sum_{\mathbf{x}\in\mathscr{X}\times\mathscr{Y}} \mathbf{x}P(\mathbf{x})$$

- Variances:

$$\mathrm{Var}[x] = \sigma_x^2 = \mathrm{E}\left[(x-\mu_x)^2\right], \quad \mathrm{Var}[y] = \sigma_y^2 = \mathrm{E}\left[(y-\mu_y)^2\right]$$

- Covariance:

$$\sigma_{xy}^2 = \mathrm{E}[(x-\mu_x)(y-\mu_y)] = \sum_{x\in\mathscr{X}}\sum_{y\in\mathscr{Y}} (x-\mu_x)(y-\mu_y)P(x,y)$$

Expected Values of Functions of Two Variables

Corollary

x, y *stat. independent* $\rightarrow \sigma_{xy} = 0$

Corollary

$\sigma_{xy} = 0 \rightarrow x, y$ *uncorrelated*

Result

x, y *stat. independent* $\rightarrow$
$\mathrm{E}[f(x)g(y)] = \mathrm{E}[f(x)]\mathrm{E}[g(y)]$

- Cauchy-Schwartz inequality:

$$\sigma_{xy}^2 \leq \sigma_x^2 \sigma_y^2, \quad \rho \in [-1, 1]$$

- Correlation Coefficient ρ:

$$\rho = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

- Covariance matrix:

$$\Sigma = \mathrm{E}\left[(\mathbf{x} - \mu_{\mathbf{x}})(\mathbf{y} - \mu_{\mathbf{y}})^T\right]$$

Conditional Probability

- Conditional probability of x given y:

$$Pr[x = v_i | y = w_j] = \frac{Pr[x = v_i, y = w_j]}{Pr[y = w_j]}$$

By use of probability mass:

$$P(x|y) = \frac{P(x,y)}{P_y(y)}$$

Result

x, y *stat. independent* $\rightarrow P(x|y) = P(x)$

Law of Total Probability and Bayes Rule

- Law of total probability: $P(y) = \sum_{x \in \mathscr{X}} P(x, y) = \sum_{x \in \mathscr{X}} P(y|x)P(x)$
- Based on conditional probability definition:

$$\left.\begin{array}{l} P(x, y) = P(y|x)P(x) \\ P(x, y) = P(x|y)P(y) \end{array}\right\} \Rightarrow P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

- Bayes rule:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \quad posterior = \frac{likelihood \times prior}{evidence}$$

Vector Random Variables

- We use vector notation to generalize to more variables, i.e. $\mathbf{x} = \{x_1, x_2, \ldots, x_d\}$
- Joint probability mass function $P(\mathbf{x})$ Axioms

$$P(\mathbf{x}) \geq 0, \quad \sum_{\mathbf{x} \in \mathbb{R}^d} P(\mathbf{x}) = 1$$

- Statistically independent $\mathbf{x_i}$: $P(\mathbf{x}) = \prod_{i=1}^{d} P_{x_i}(x_i)$
- Bayes rule: $P(\mathbf{x_1}|\mathbf{x_2}) = \frac{P(\mathbf{x_2}|\mathbf{x_1})P(\mathbf{x_1})}{P(\mathbf{x_2})} = \frac{P(\mathbf{x_2}|\mathbf{x_1})P(\mathbf{x_1})}{\sum_{\mathbf{x_1} \in \mathscr{X}_1} P(\mathbf{x_2}|\mathbf{x_1})P(\mathbf{x_1})}$

Example

$$P(x_1, x_4) = \sum_{x_2} \sum_{x_3} \sum_{x_5} P(x_1, x_2, x_3, x_4, x_5)$$

$$P(x_1, x_2|x_3) = \frac{P(x_1, x_2, x_3)}{P(x_3)}$$

Continuous Random Variables

Single random variable

- Let x be a continuous random variable
- Probability density function (PDF) $p(x)$: $P_r[x \in (a,b)] = \int_a^b p(x)dx$
- PDF must satisfy: $p(x) \geq 0, \quad \int_{-\infty}^{\infty} p(x)dx = 1$
- Expected value: $\mathrm{E}[f(x)] = \int_{-\infty}^{\infty} f(x)P(x)dx$
- Mean: $\mathrm{E}[x] = \int_{-\infty}^{\infty} xP(x)dx$
- Variance: $\mathrm{E}\left[(x-\mu)^2\right] = \int_{-\infty}^{\infty} (x-\mu)^2 P(x)dx$

Continuous Random Variables

Multivariate case

- Let $\mathbf{x}$ be vector of continuous random variables $\mathbf{x} = \{x_1, x_2, \ldots, x_d\}$
- Probability density function (PDF) $p(\mathbf{x})$:
  $P_r[\mathbf{x} \in \mathscr{X}_1 \times \ldots \times \mathscr{X}_d] = \int_{\mathbf{x} \in \mathscr{X}_1 \times \ldots \times \mathscr{X}_d} p(\mathbf{x}) d\mathbf{x}$
- PDF must satisfy: $p(\mathbf{x}) \geq 0, \quad \int_{-\infty}^{\infty} p(\mathbf{x}) d\mathbf{x} = 1$
- Expected value:
  $\mathrm{E}[f(\mathbf{x})] = \int_{-\infty}^{\infty} f(\mathbf{x}) P(\mathbf{x}) d\mathbf{x} = \int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} f(\mathbf{x}) P(\mathbf{x}) dx_1 \cdots dx_d$
- Mean: $\mathrm{E}[\mathbf{x}] = \int_{-\infty}^{\infty} \mathbf{x} P(\mathbf{x}) d\mathbf{x}$
- Covariance matrix:
  $\Sigma = \mathrm{E}\left[(\mathbf{x} - \mu_{\mathbf{x}})(\mathbf{y} - \mu_{\mathbf{y}})^T\right] = \int_{-\infty}^{\infty} (\mathbf{x} - \mu)(\mathbf{x} - \mu)^T P(\mathbf{x}) d\mathbf{x}$
- Stat. independent variables in $\mathbf{x} \Rightarrow p(\mathbf{x}) = \prod_{i=1}^{d} P(x_i)$, and Σ is diagonal

Normal Distributions

Normal density (1-D)

- Probability density function: $p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} = N(\mu, \sigma)$
- Expectations: $\mathrm{E}[1] = \int_{-\infty}^{\infty} P(x)dx = 1, \quad \mathrm{E}[x] = \int_{-\infty}^{\infty} xP(x)dx = \mu$
  $\mathrm{E}\left[(x-\mu)^2\right] = \int_{-\infty}^{\infty} (x-\mu)^2 P(x)dx = \sigma^2$
- Approximations of cumulative probabilities: $Pr[|x-\mu| \leq \sigma] \simeq 0.68$, $Pr[|x-2\mu| \leq \sigma] \simeq 0.95$, $Pr[|x-3\mu| \leq \sigma] \simeq 0.997$
- Standardized random variable: $u = \frac{x-\mu}{\sigma} \Rightarrow p(u) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{u^2}{2}} = N(0,1)$

Theorem (Central Limit Theorem)

The distribution of the sum of d independent random variables approaches the normal distribution (under various conditions)

Bayesian Decision Theory

MTSC 852, Fall 2015

Sokratis Makrogiannis, Ph.D.

Department of Mathematical Sciences
Delaware State University

smakrogiannis@desu.edu

September 3, 2015

Outline

Outline

Outline

Preliminaries

- The Bayesian Decision Theory is a statistical approach to pattern classification
- This approach assigns a class label to a pattern by quantifying the tradeoffs among the different decisions
- Bayesian Decision Theory poses the decision problem in probabilistic terms
- It assumes that all relevant probability information is known

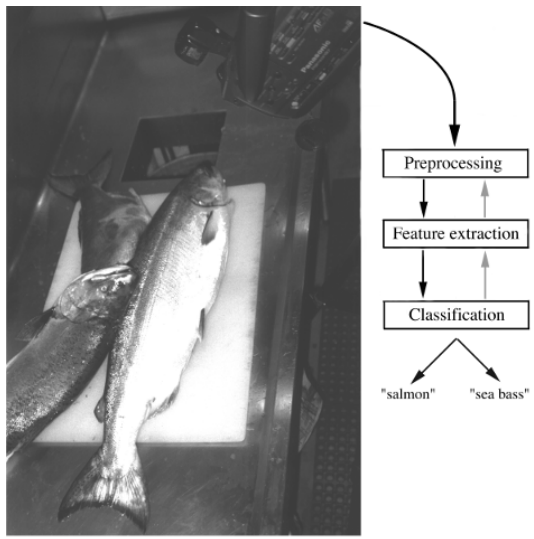
Main concept

- The decisions minimize a total expected "risk"
- Straightforward risk is the classification error
- The risk is associated with the cost of making a specific decision

Terminology

Consider the problem: an observer watches fish arrive on a conveyor belt. There are two kinds of fish: salmon and sea bass. The objective is to predict the type of fish that will appear next, assuming that the sequence is random

- *State of nature* ω is a random variable that describes the class, that is ω_1 for salmon and ω_2 for sea bass
- *Prior probabilities* $P(\omega_j)$: probabilities for states of nature (categories)
- *Feature vector* x: vector consisting of measurements for patterns, e.g. fish lightness



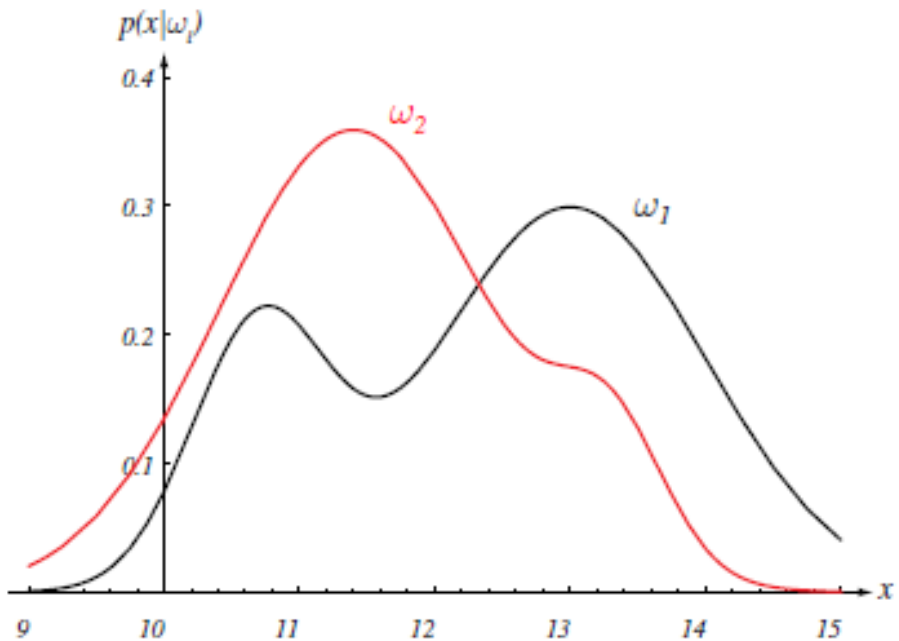
Terminology

Consider the problem: an observer watches fish arrive on a conveyor belt. There are two kinds of fish: salmon and sea bass. The objective is to predict the type of fish that will appear next, assuming that the sequence is random

- *Probability density function* $p(x)$: How frequently a pattern with value x occurs
- *Class-conditional probability density* $p(x|\omega_j)$: Frequency of occurrence of a feature x for a specific class label ω_j. Known as likelihood. Probability of a lightness value for a specific fish type
- *Posterior probability density* $p(\omega_j|x)$: Probability of class ω_j, given a measurement x. Probability of occurrence of a specific fish type, given the lightness value

Class-conditional Densities

A key process in pattern classification is to estimate the class-conditional density for each category, also referred to as likelihood



Decision Based on Priors Only

Decide ω_1 if $P(\omega_1) > P(\omega_2)$, otherwise decide ω_2

- This rule always makes the same decision
- It favors the most likely class
- $P(error) = min[P(\omega_1), P(\omega_2)]$
- It may be applied if we know only the prior probabilities $P(\omega_j)$, where ω_j is the state of nature

Bayes formula

- Suppose that in addition to $P(\omega_j)$, we have obtained measurements x and we know the likelihoods $p(x|\omega_j)$
- Then the Bayes rule yields

$$P(\omega_j|x) = \frac{p(x|\omega_j)P(\omega_j)}{p(x)}, \text{ where } p(x) = \sum_{j=1}^{2} p(x|\omega_j)P(\omega_j)$$

- This formula be expressed as $posterior = \frac{likelihood \times prior}{evidence}$

Bayes Decision Rule

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

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

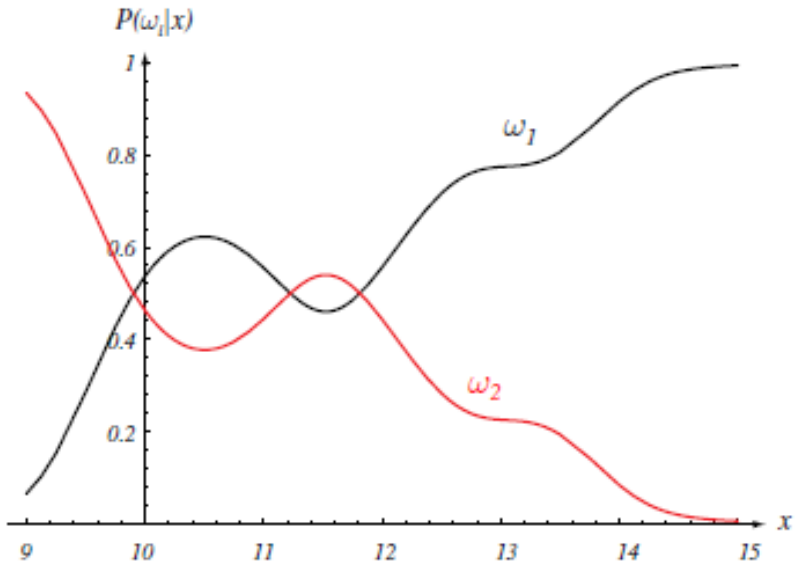


Figure: Posterior probabilities corresponding to class-conditionals of previous figure for $P(\omega_1) = 2/3$ and $P(\omega_2) = 1/3$

Probability of Error

- The probability of error when we make a decision using the Bayesian rule is

$$P(error|x) = \begin{cases} P(\omega_1|x), & \text{if we decide } \omega_2 \\ P(\omega_2|x), & \text{if we decide } \omega_1 \end{cases}$$

- Hence, $P(error|x) = min[P(\omega_1|x), P(\omega_2|x)]$
- This rule minimizes the average probability for error because

$$P(error) = \int_{-\infty}^{\infty} P(error, x)dx = \int_{-\infty}^{\infty} P(error|x)p(x)dx$$

Bayesian Decision Theory Generalization

- Let $\Omega = \{\omega_1, \omega_2, ..., \omega_c\}$ be the set of c states of nature or classes, and $A = \{a_1, a_2, ..., a_c\}$ be the set of actions. Then $\lambda(\alpha_i|\omega_j)$ is the loss caused by taking action α_i for a true class ω_j.
- Let $\mathbf{x} \in \mathbb{R}^D$ be a D-dimensional feature vector corresponding to measurements
- Then the Bayes formula is defined as

$$P(\omega_j|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_j)P(\omega_j)}{p(\mathbf{x})}, \text{ where } p(\mathbf{x}) = \sum_{j=1}^{c} p(\mathbf{x}|\omega_j)P(\omega_j)$$

Loss Function

- We assume that after an observation $\mathbf{x}$ we take action α_i, when the true state of nature is ω_j. Then the incurred loss for this action is $\lambda(\alpha_i|\omega_j)$
- We define as conditional risk $R(\alpha_i|\mathbf{x})$ the expected loss given by

$$R(\alpha_i|\mathbf{x}) = \sum \lambda(\alpha_i|\omega_j)P(\omega_j|\mathbf{x})$$

Bayes Risk

- General decision rule is a function $\alpha(\mathbf{x})$ with $\alpha : \mathbb{R}^D \rightarrow A$
- The overall risk R is the expected loss incurred by a specific decision estimated by

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

- Hence, to minimize overall risk, we compute the conditional risk $R(\alpha_i|\mathbf{x})$ for $i = 1, ..., \alpha$ and choose the action α_i that minimizes $R(\alpha_i|\mathbf{x})$
- The corresponding overall risk $R*$ is called Bayes risk and denotes the optimal classification performance

Two-category Classification

Assumptions

- Classification problem with two classes ω_1 and ω_2.
- α_i: decide ω_i, for $i = 1, 2$
- $\lambda_{ij} = \lambda(\alpha_i|\omega_j)$

Bayes Decision Rule

- $R(\alpha_1|\mathbf{x}) = \lambda_{11}P(\omega_1|\mathbf{x}) + \lambda_{12}P(\omega_2|\mathbf{x})$
- $R(\alpha_2|\mathbf{x}) = \lambda_{21}P(\omega_1|\mathbf{x}) + \lambda_{22}P(\omega_2|\mathbf{x})$
- Rule: Decide ω_1 if $R(\alpha_1|\mathbf{x}) < R(\alpha_2|\mathbf{x})$; otherwise decide ω_2
- Decide ω_1 if

$$\lambda_{11}P(\omega_1|\mathbf{x}) + \lambda_{12}P(\omega_2|\mathbf{x}) < \lambda_{21}P(\omega_1|\mathbf{x}) + \lambda_{22}P(\omega_2|\mathbf{x})$$

$$(\lambda_{21} - \lambda_{11})P(\omega_1|\mathbf{x}) > (\lambda_{12} - \lambda_{22})P(\omega_2|\mathbf{x})$$

Likelihood Ratio Test (LRT) for Two Categories

Bayes Decision Rule

- Decide ω_1 if

$$\lambda_{11}P(\omega_1|\mathbf{x}) + \lambda_{12}P(\omega_2|\mathbf{x}) < \lambda_{21}P(\omega_1|\mathbf{x}) + \lambda_{22}P(\omega_2|\mathbf{x}) \Leftrightarrow$$

$$(\lambda_{21} - \lambda_{11})P(\omega_1|\mathbf{x}) > (\lambda_{12} - \lambda_{22})P(\omega_2|\mathbf{x}) \Leftrightarrow$$

$$(\lambda_{21} - \lambda_{11})p(\mathbf{x}|\omega_1)P(\omega_1) > (\lambda_{12} - \lambda_{22})p(\mathbf{x}|\omega_2)P(\omega_2) \Leftrightarrow$$

$$\frac{p(\mathbf{x}|\omega_1)}{p(\mathbf{x}|\omega_2)} > \frac{(\lambda_{12} - \lambda_{22})}{(\lambda_{21} - \lambda_{11})}\frac{P(\omega_2)}{P(\omega_1)}$$

Likelihood Ratio Example

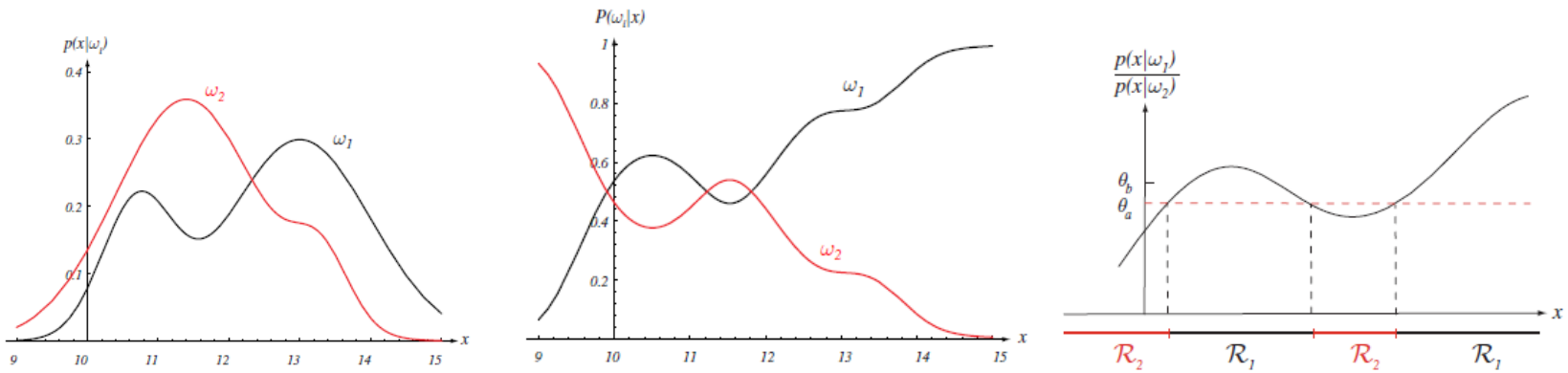


Figure: Bayesian decision theory components: class-conditionals, posterior probabilities, and likelihood ratio (left to right)

Minimum Error-rate Classification

Assumptions

- Classification problem with c classes in $\Omega = \{\omega_1, \omega_2, ..., \omega_c\}$
- α_i: decide ω_i, for $i = 1, .., c$
- $\lambda_{ij} = \lambda(\alpha_i|\omega_j)$, where $\lambda(\alpha_i|\omega_j) = \begin{cases} 0, \text{ if } i = j \\ 1, \text{ if } i \neq j \end{cases}$ for $i, j = 1, ..., c$

Conditional Risk and Bayes Decision Rule

- $R(\alpha_i|\mathbf{x}) = \sum_{j=1}^{c} \lambda_{ij} P(\omega_j|\mathbf{x}) = \sum_{j \neq i}^{c} P(\omega_j|\mathbf{x}) = 1 - P(\omega_i|\mathbf{x})$
- Rule: Decide ω_i if

$$R(\alpha_i|\mathbf{x}) < R(\alpha_j|\mathbf{x}), \forall j \neq i \Leftrightarrow$$

$$1 - P(\omega_i|\mathbf{x}) < 1 - P(\omega_j|\mathbf{x}), \forall j \neq i \Leftrightarrow$$

$$P(\omega_i|\mathbf{x}) > P(\omega_j|\mathbf{x}), \forall j \neq i$$

Maximum A Posteriori (MAP) Criterion

- According to the previous result the decision rule for Minimum Error-rate Classification is:

$$\text{Decide } \omega_i, \text{ if } P(\omega_i|\mathbf{x}) > P(\omega_j|\mathbf{x}), \forall j \neq i \Leftrightarrow$$

$$\text{Decide } \omega_i, \text{ if } \frac{P(\omega_i|\mathbf{x})}{P(\omega_j|\mathbf{x})} > 1, \forall j \neq i$$

- This is also known as Maximum A Posteriori (MAP) criterion as it seeks to maximize the posterior probability $P(\omega_i|\mathbf{x})$

Maximum Likelihood (ML) Criterion

- If in addition to a binary loss function we have equal priors, that is, $P(\omega_i) = P(\omega_j), \forall i \neq j$, then the decision rule becomes

$$\text{Decide } \omega_i, \text{ if } \frac{P(\omega_i|\mathbf{x})}{P(\omega_j|\mathbf{x})} > 1, \forall j \neq i \Leftrightarrow$$

$$\text{Decide } \omega_i, \text{ if } \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(\mathbf{x}|\omega_j)P(\omega_j)} > 1, \forall j \neq i \Leftrightarrow$$

$$\text{Decide } \omega_i, \text{ if } \frac{p(\mathbf{x}|\omega_i)}{p(\mathbf{x}|\omega_j)} > 1, \forall j \neq i$$

- This is also known as Maximum Likelihood (ML) criterion as it seeks to maximize the likelihood $p(\mathbf{x}|\omega_i)$

Bayesian Decision Theory

MTSC 852, Fall 2015

Sokratis Makrogiannis, Ph.D.

Department of Mathematical Sciences
Delaware State University

smakrogiannis@desu.edu

October 6, 2015

Outline

Outline

Outline

Outline

Outline

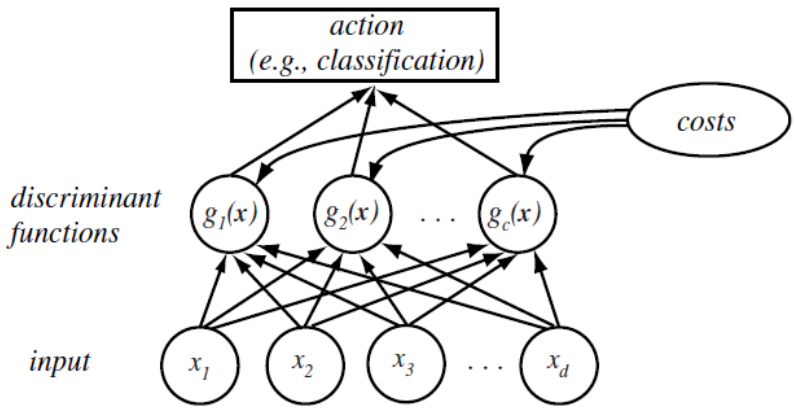
Outline

Discriminant Functions

- A classifier can be represented by a set of discriminant functions $g_i(x), i = 1, ..., c$ corresponding to each state of nature or category.
- Then the decision rule becomes:

$$\text{Assign feature vector } \mathbf{x} \text{ to class } \omega_i, \text{ if } g_i(\mathbf{x}) > g_j(\mathbf{x}) \quad \forall j \neq i$$

- Then the classifier can be considered to be a network or machine that computes c discriminant functions and makes a decision using a maximum operator.



Bayes Classifier with Multiple Categories

- A Bayes classifier can also be described using discriminant functions.
- For the general case that minimizes conditional risks $g_i(\mathbf{x}) = -R(\alpha_i|\mathbf{x})$
- For the MAP –or minimum-error-rate– criterion $g_i(\mathbf{x}) = P(\omega_i|\mathbf{x})$
  With some more operations we can produce other equivalent MAP discriminant functions

$$g_i(\mathbf{x}) = P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{\sum_{j=1}^{c} p(\mathbf{x}|\omega_j)P(\omega_j)}$$

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

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

- For the ML criterion $g_i(\mathbf{x}) = p(\mathbf{x}|\omega_i)$

Two Categories

- Suppose a problem with two categories ω_1 and ω_2.
- Then we can define a single discriminant function by

$$g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x})$$

- The decision rule is:

$$\text{Decide } \omega_1 \text{ if } g(\mathbf{x}) > 0; \text{ otherwise decide } \omega_2$$

- For the MAP –or minimum-error-rate– criterion it follows that

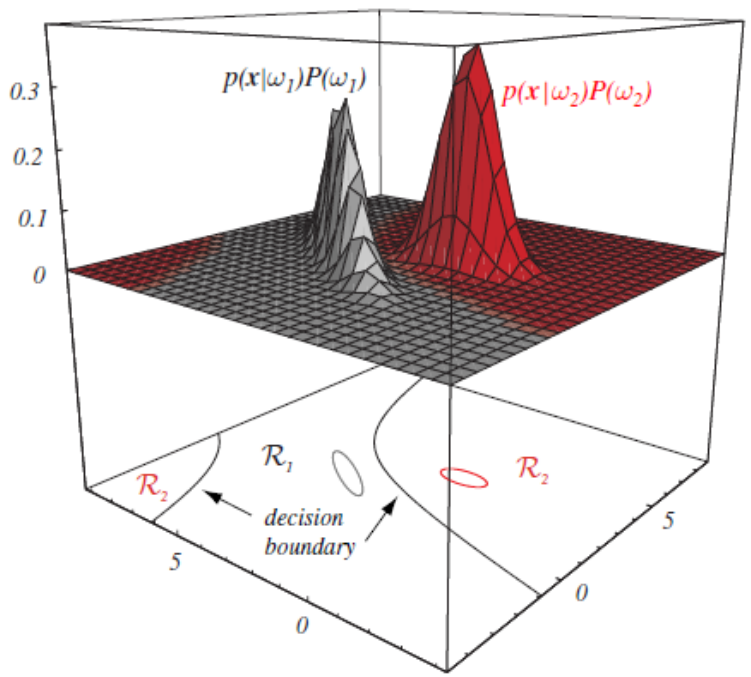
$$g(\mathbf{x}) = P(\omega_1|\mathbf{x}) - P(\omega_2|\mathbf{x}) \Leftrightarrow$$

$$g(\mathbf{x}) = \ln p(\mathbf{x}|\omega_1) + \ln P(\omega_1) - \ln p(\mathbf{x}|\omega_2) - \ln P(\omega_2) \Leftrightarrow$$

$$g(\mathbf{x}) = \ln \frac{p(\mathbf{x}|\omega_1)}{p(\mathbf{x}|\omega_2)} + \ln \frac{P(\omega_1)}{P(\omega_2)}$$

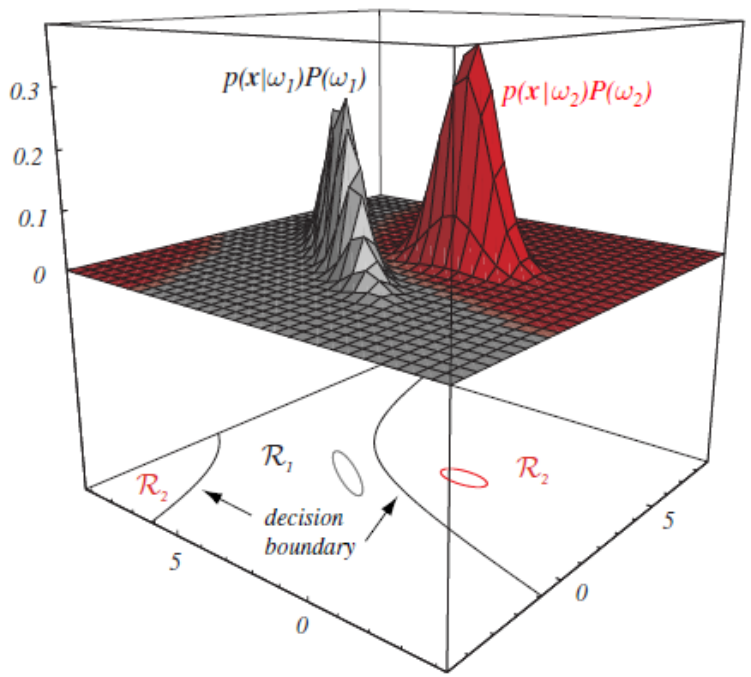
Decision Regions and Boundaries

- Decision rules divide the feature space in Decision Regions $\mathscr{R}_i, i = 1, 2, ..., c$, where c is the number of categories



Decision Regions and Boundaries

- Decision rules divide the feature space in Decision Regions $\mathscr{R}_i, i = 1, 2, ..., c$, where c is the number of categories
- The Decision Regions $(\mathscr{R}_i, \mathscr{R}_j)$ are separated by Decision Boundaries on which $g_i(\mathbf{x}) = g_j(\mathbf{x})$.



Introduction

- As explained in previous sections, we can design a Bayesian classifier if we know the likelihood $p(\mathbf{x}|\omega_i)$ and priors $P(\omega_i)$ for each category i
- Among all density functions the multivariate Gaussian model is very popular
- Normal density models are popular mainly because of the Central Limit Theorem, and the fact that the normal density is analytically tractable

Univariate Normal Density

- Density function also symbolized by $N(\mu, \sigma)$: $p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
- Expected value: $\mu \equiv \mathscr{E}[x] = \int_{-\infty}^{\infty} x p(x) dx$
- Variance: $\sigma \equiv \mathscr{E}[(x-\mu)^2] = \int_{-\infty}^{\infty} (x-\mu)^2 p(x) dx$
- Entropy: $H(p(x)) = -\int_{-\infty}^{\infty} p(x) \ln p(x) dx$

Multivariate Normal Density

- Density function (also symbolized by $N(\mu, \Sigma)$) :
  $p(\mathbf{x}) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} e^{\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)}$
- Expected value: $\mu \equiv \mathscr{E}[\mathbf{x}] = \int_{-\infty}^{\infty} \mathbf{x} p(\mathbf{x}) d\mathbf{x}$
- Covariance matrix:
  $\Sigma \equiv \mathscr{E}[(\mathbf{x}-\mu)(\mathbf{x}-\mu)^T] = \int_{-\infty}^{\infty} (\mathbf{x}-\mu)(\mathbf{x}-\mu)^T p(\mathbf{x}) d\mathbf{x}$

Introduction

- According to previous sections the use of MAP criterion yields the following discriminant functions

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

- For the case of multivariate normal densities for the likelihood, i.e. when $p(\mathbf{x}|\omega_i) = N(\mu, \Sigma)$, it follows that

$$g_i(\mathbf{x}) = -(1/2)(\mathbf{x} - \mu_i)^T \Sigma_i^{-1}(\mathbf{x} - \mu_i) - (d/2)\ln 2\pi - (1/2)\ln|\Sigma_i| + \ln P(\omega_i)$$

- Next, we examine three different Covariance Matrix models

Identical Diagonal Covariance Matrices with $\Sigma_i = \sigma^2 \mathbf{I}$

- We assume that features are statistically independent and have equal standard deviations
- The discriminant function can be simplified as

$$g_i(\mathbf{x}) = -\frac{(\mathbf{x}-\mu_i)^T(\mathbf{x}-\mu_i)}{2\sigma^2} + \ln P(\omega_i) \Leftrightarrow$$

$$g_i(\mathbf{x}) = -\frac{1}{2\sigma^2}[\mathbf{x}^T\mathbf{x} - 2\mu_i^T\mathbf{x} + \mu_i^T\mu_i] + \ln P(\omega_i)$$

- Further simplification –by discarding category-invariant terms – yields a linear discriminant function

$$g_i(\mathbf{x}) = \mathbf{w}_i^T\mathbf{x} + b_{i0}$$

where

$$\mathbf{w}_i = (1/\sigma^2)\mu_i, b_{i0} = -(1/2\sigma^2)\mu_i^T\mu_i + \ln P(\omega_i)$$

Identical Diagonal Covariance Matrices with $\Sigma_i = \sigma^2 \mathbf{I}$

- A classifier defined by linear discriminant functions is called a linear machine.
- The decision surfaces defined by $g_i(\mathbf{x}) - g_j(\mathbf{x})$ are hyperplanes in the feature space
- Here the decision surface is described by the equation

$$\mathbf{w}^T(\mathbf{x} - \mathbf{x}_0) = 0$$

with

$$\mathbf{w} = \mu_i - \mu_j$$

$$\mathbf{x}_0 = (1/2)(\mu_i + \mu_j) - \frac{\sigma^2}{\|\mu_i - \mu_j\|^2} \ln \frac{P(\omega_i)}{P(\omega_j)} (\mu_i - \mu_j)$$

$$\|\mu_i - \mu_j\|^2 = (\mu_i - \mu_j)^T (\mu_i - \mu_j)$$

- Therefore, the decision surface is a hyperplane passing through $\mathbf{x}_0$ and orthogonal to the line linking the category means $(\mathbf{x} - \mathbf{x}_0)$

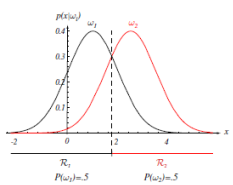
Identical Diagonal Covariance Matrices with $\Sigma_i = \sigma^2 \mathbf{I}$

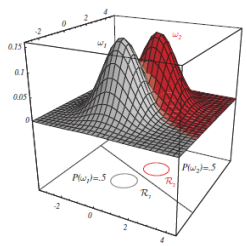
Additional notes

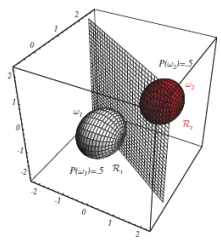
- If all priors $P(\omega_i)$ are equal, then the discriminant function can be simplified to

$$g_i(\mathbf{x}) = \|\mathbf{x} - \mu_i\|$$

- Hence, each pattern will be classified to the category with the closest mean using a Euclidean norm
- This is called a minimum-distance classifier







Identical Arbitrary Covariance Matrices with $\Sigma_i = \Sigma$

- The discriminant function can be simplified by discarding category-invariant terms

$$g_i(\mathbf{x}) = -(1/2)(\mathbf{x} - \mu_i)^T \Sigma^{-1} (\mathbf{x} - \mu_i) + \ln P(\omega_i)$$

- Let the prior probabilities $P(\omega_i)$ be equal for all classes. Then

$$g_i(\mathbf{x}) = -(1/2)(\mathbf{x} - \mu_i)^T \Sigma^{-1} (\mathbf{x} - \mu_i)$$

- After expansion of Mahalanobis distance and simplification

$$g_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x} + b_{i0}$$

where

$$\mathbf{w}_i = \Sigma^{-1} \mu_i, b_{i0} = -(1/2) \mu_i^T \Sigma^{-1} \mu_i + \ln P(\omega_i)$$

Identical Arbitrary Covariance Matrices with $\Sigma_i = \Sigma$

- The decision surfaces defined by $g_i(\mathbf{x}) - g_j(\mathbf{x})$ are hyperplanes in the feature space
- Here the decision surface is described by the equation

$$\mathbf{w}^T(\mathbf{x} - \mathbf{x}_0) = 0$$

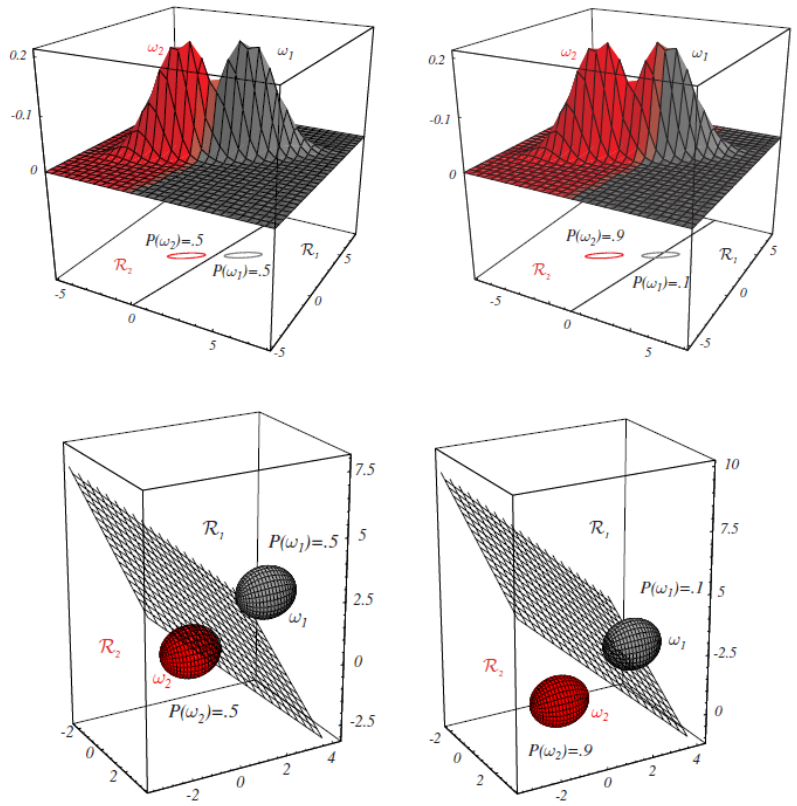
with

$$\mathbf{w} = \Sigma^{-1}(\mu_i - \mu_j)$$

$$\mathbf{x}_0 = (1/2)(\mu_i + \mu_j) - \frac{\ln \frac{P(\omega_i)}{P(\omega_j)}}{(\mu_i - \mu_j)^T \Sigma^{-1} (\mu_i - \mu_j)}(\mu_i - \mu_j)$$

- Therefore, the decision surface is a hyperplane passing through $\mathbf{x}_0$ but it is not necessarily orthogonal to the line linking the category means $(\mathbf{x} - \mathbf{x}_0)$

Identical Arbitrary Covariance Matrices with $\Sigma_i = \Sigma$



Arbitrary Covariance Matrices

- In this case the covariance matrices are different for each category
- The discriminant function takes the form

$$g_i(\mathbf{x}) = \mathbf{x}^T \mathbf{W}_i \mathbf{x} + \mathbf{w}_i^T \mathbf{x} + b_{i0}$$

where

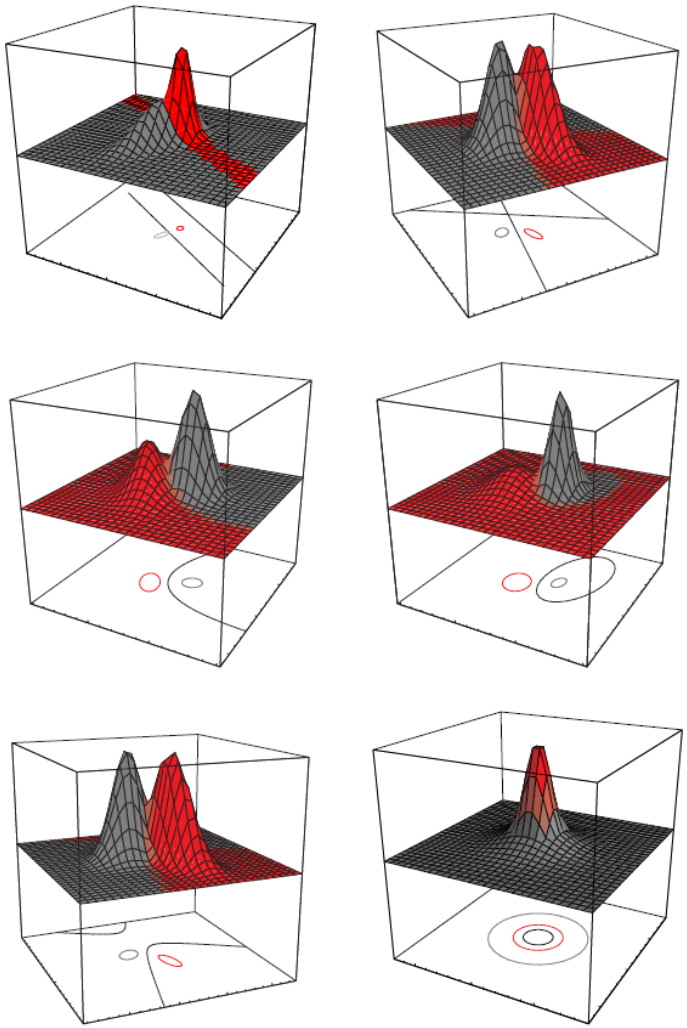
$$\mathbf{W}_i = -(1/2)\Sigma_i^{-1}$$

$$\mathbf{w}_i = \Sigma_i^{-1}\mu_i$$

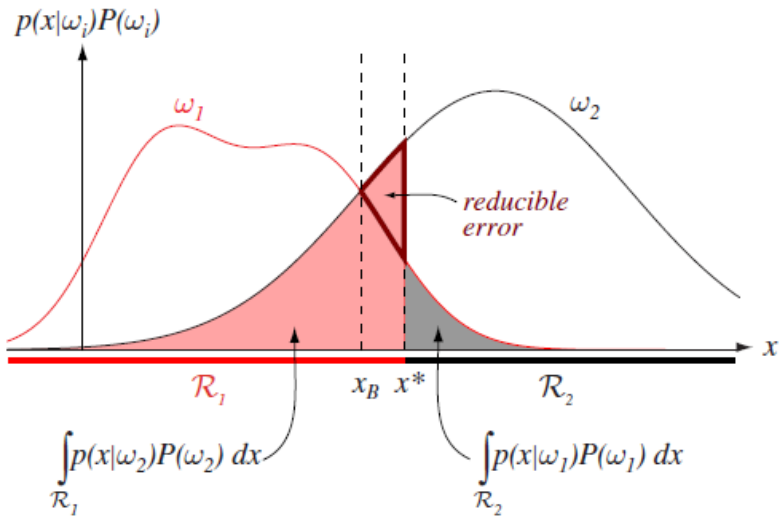
$$b_{i0} = -(1/2)\mu_i^T \Sigma^{-1}\mu_i - (1/2)\ln|\Sigma_i| + \ln P(\omega_i)$$

- This is a quadratic form
- In the two-category case the decision surfaces are hyperquadrics assuming any of the following forms: hyperplanes, hyperspheres, hyperellipsoids, hyperparaboloids, or hyperhyperboloids

Arbitrary Covariance Matrices



Bayes Error Rate

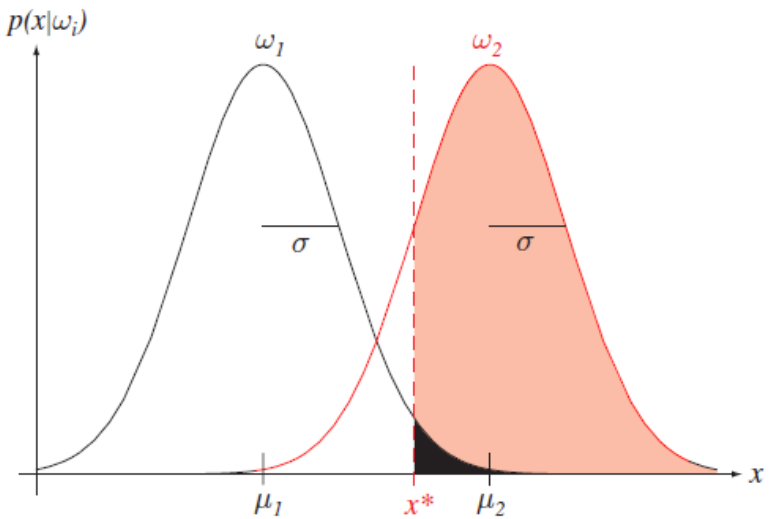


Problem Definition

- Suppose that we want to detect a signal. We measure a feature x -let's say voltage- that has a mean μ_1 when a signal is present, and μ_2 when there is no signal
- We assume two normal distributions with different means but the same variance, that is $N(\mu_1, \sigma)$ and $N(\mu_2, \sigma)$ for classes ω_1 and ω_2 respectively
- For the detection we use a threshold value $x*$ –that is unknown to us– to classify a feature x into signal or no-signal categories
- We seek a measure of *separability*, i.e., how easy it is to separate the two categories and make a prediction
- Suppose we do not know $\mu_1, \mu_2, \sigma, x*$, but we know the sate of nature and the system's decision for each

Probabilities

- $P(x > x* | x \in \omega_2)$: *hit*, we detect a signal and the signal is present
- $P(x > x* | x \in \omega_1)$: *false alarm*, we detect a signal but no signal is present
- $P(x < x* | x \in \omega_2)$: *miss*, we do not detect a signal but the signal is present
- $P(x < x* | x \in \omega_1)$: *correct rejection*, we do not detect a signal and no signal is present
- If we have enough samples we can compute the probabilities experimentally
- If $x*$ changes, then the above probabilities will change

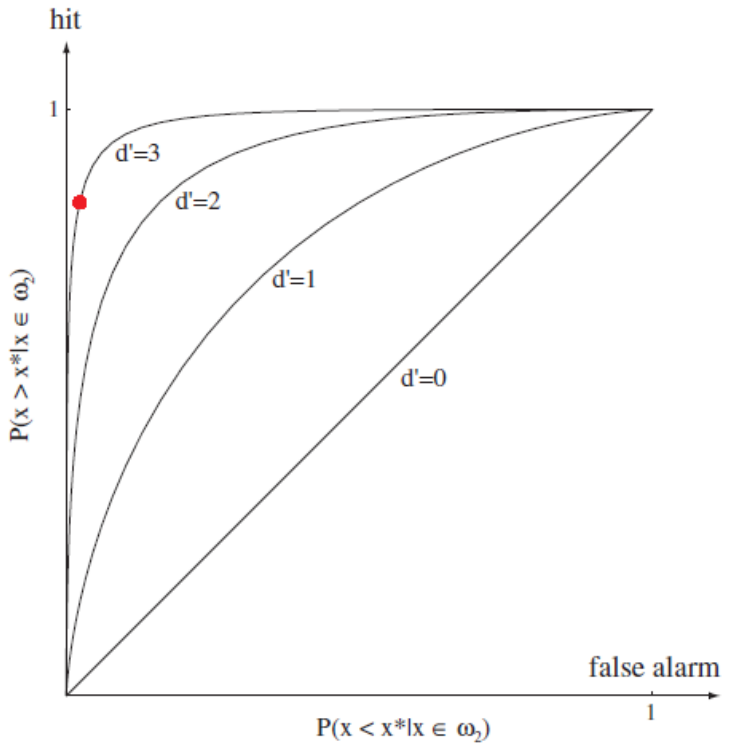


Receiver Operating Characteristics (ROC)

- An ROC graph is a plot of probability of hit vs. probability of false alarm for different values of $x*$
- The curve can be used to measure separability, that is the capacity of our system to detect the signal
- A usual measure of separability is the area under the curve, frequently denoted by $AUC - ROC$
- In this context it is important to choose a measure x whose variations will significantly change the probability values

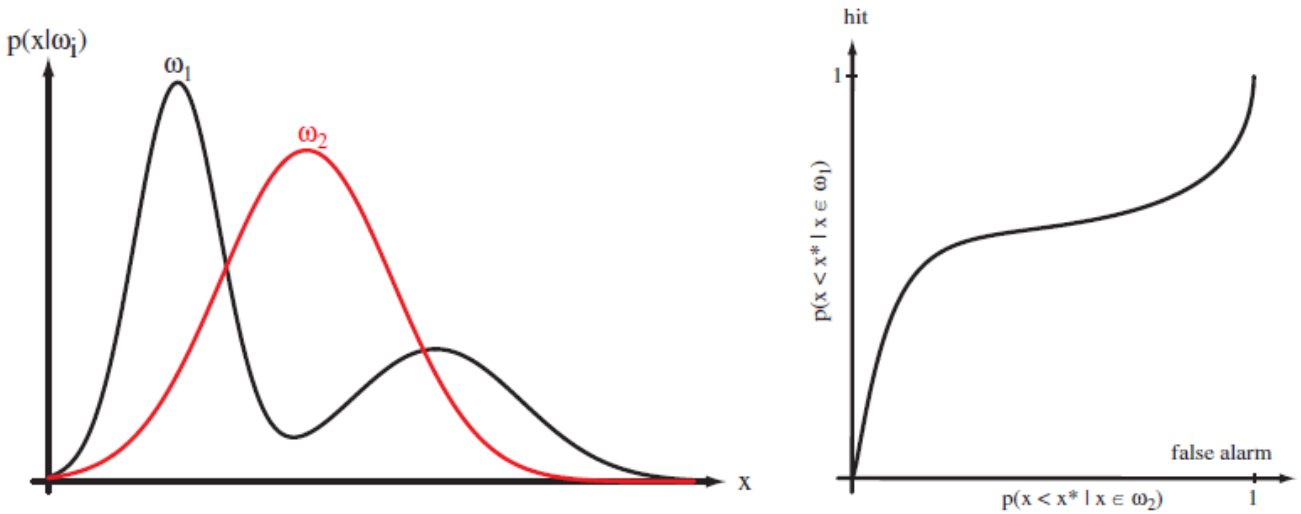
ROC Example

ROC curve for varying discriminability



ROC Example 2

ROC curves are not necessarily symmetric



Bayesian Decision Theory for Discrete Features

- Here we assume that the components of feature vector $\mathbf{x}$ can assume only m discrete values $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_m$
- The Bayes formula involves probabilities only

$$P(\omega_j|x) = \frac{P(\mathbf{x}|\omega_j)P(\omega_j)}{P(\mathbf{x})}$$

with

$$P(x) = \sum_{i=1}^{c} P(\mathbf{x}|\omega_i)P(\omega_i)$$

- Integrals are replaced by sums in definitions of expectation, variance, entropy and other statistical measures
- Fundamental Bayes decision rule is the same To minimize the overall risk, select the action α_i for which the conditional risk $R(\alpha_i|\mathbf{x})$ is minimized

$$\alpha* = arg \min_i R(\alpha_i|\mathbf{x})$$

MTSC 852 - Pattern Recognition
Lab Session
Parametric Estimation

Sokratis Makrogiannis, Ph.D.

October 17, 2015

Contents

1 Bayesian Parameter Estimation for the Gaussian Density

1.1 Univariate Normal Density

Find the class-conditional density $p(x|\mathcal{D})$ using Bayesian estimation assuming that $p(x|\mu) \sim N(\mu, \sigma^2)$, $p(\mu) \sim N(\mu_0, \sigma_0)$ and σ is known.

- Estimate $p(\mu|\mathcal{D})$ using Bayes rule
- Estimate $p(x|\mathcal{D})$ by integration over the parameter space

Find the class-conditional density $p(x|\mathcal{D})$ using Bayesian estimation assuming that $p(x|\mu) \sim N(\mu, \sigma^2)$, $p(\mu) \sim N(\mu_0, \sigma_0)$ and σ is known.

1.2 Estimate $p(\mu|\mathcal{D})$

- According to previous analysis, we seek to estimate $p(\mathbf{x}|\mathcal{D})$ for each class
- This is achieved by integration over the parameter space

$$p(\mathbf{x}|\mathcal{D}) = \int p(\mathbf{x}, \theta|\mathcal{D}) d\theta$$

- From definition of joint probability: $p(\mathbf{x}|\mathcal{D}) = \int p(\mathbf{x}|\theta, \mathcal{D}) p(\theta|\mathcal{D}) d\theta$
- We use Bayes rule to estimate $p(\theta|\mathcal{D})$: $p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)P(\theta)}{\int p(\mathcal{D}|\theta)P(\theta)d\theta}$
- If the samples are independently drawn, then: $p(\mathcal{D}|\theta) = \prod_{i=1}^{n} p(\mathbf{x}_i|\theta)$
- We use Bayes rule to estimate posterior parameter density $p(\mu|\mathcal{D})$:

$$p(\mu|\mathcal{D}) = \frac{p(\mathcal{D}|\mu)p(\mu)}{\int p(\mathcal{D}|\mu)p(\mu)d\mu}$$

- Let samples $\mathcal{D} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}\}$ be independently drawn. Then: $p(\mathcal{D}|\mu) = \prod_{i=1}^{n} p(x_i|\mu)$
- Then: $p(\mu|\mathcal{D}) = \alpha \cdot \prod_{i=1}^{n} p(x_i|\mu)p(\mu)$
- According to assumptions:

$$p(x_i|\mu) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}, \quad p(\mu) = \frac{1}{\sqrt{2\pi}\sigma_0} e^{-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}}$$

- So: $p(\mu|\mathcal{D}) = \alpha \cdot \prod_{i=1}^{n} \left[\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \frac{1}{\sqrt{2\pi}\sigma_0} e^{-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}} \right]$

- After some more manipulations we can show that $p(\mu|\mathcal{D})$ is an exponential function of a quadratic function, hence it has the form $N(\mu_n, \sigma_n)$
- Therefore

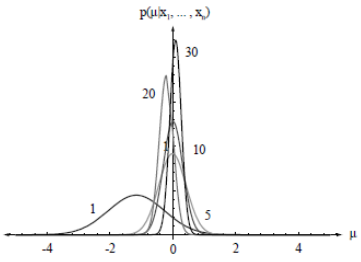
$$p(\mu|\mathcal{D}) = \frac{1}{\sqrt{2\pi}\sigma_n} e^{-\frac{(\mu-\mu_n)^2}{2\sigma_n^2}}$$

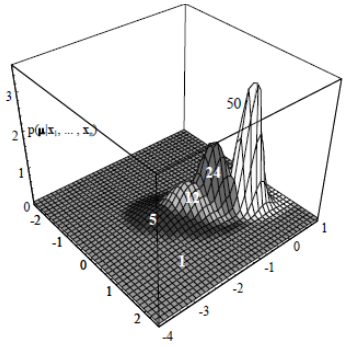
  where,

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

$$\hat{\mu}_n = (1/n)\sum_{i=1}^{n} x_i$$

- σ_n decreases as $n \to \infty$ with $\lim_{n\to\infty} \sigma_n^2 = \frac{\sigma^2}{n}$
- We observe that as the number of training samples increases, $p(\mu|\mathcal{D})$ becomes sharper around μ_n. This process is called *Bayesian learning*
- If $\sigma_0 \neq 0$, then μ_n approaches the sample mean $\lim_{n\to\infty} \mu_n = \hat{\mu}_n$.





1.3 Estimate $p(x|\mathcal{D})$

- According to Bayesian estimation process

$$p(x|\mathcal{D}) = \int p(x|\mu, \mathcal{D})p(\mu|\mathcal{D})d\mu \Leftrightarrow$$

$$p(x|\mathcal{D}) = \int \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \frac{1}{\sqrt{2\pi}\sigma_n} e^{-\frac{(\mu-\mu_n)^2}{2\sigma_n^2}} d\mu \Leftrightarrow$$

$$p(x|\mathcal{D}) = \frac{1}{2\pi\sigma\sigma_n} f(\sigma, \sigma_n) e^{-\frac{(x-\mu_n)^2}{2(\sigma^2+\sigma_n^2)}}$$

where $f(\sigma, \sigma_n)$ has an integral form:

$$f(\sigma, \sigma_n) = \int \exp\left[(-1/2)\frac{\sigma^2+\sigma_n^2}{\sigma^2\sigma_n^2}\left(\mu - \frac{\sigma_n^2 x + \sigma^2\mu_n}{\sigma^2+\sigma_n^2}\right)^2\right] d\mu$$

- Observe that $p(x|\mathcal{D}) \sim N(\mu_n, \sigma^2 + \sigma_n^2)$
- The above result gives the class-conditional density $p(x|\omega_i, \mathcal{D}_i)$ based on the posterior parameter mean estimate μ_n and the posterior parameter variance estimate increased by the uncertainty in x that we assume to be known

Exercise 1. *Consider Bayesian estimation of the mean of a one-dimensional Gaussian. Suppose you are given the prior for the mean is* $p(\mu) \sim N(\mu_0, \sigma_0)$.

1. *Write a program that plots the density* $p(x|\mathcal{D})$ *given,* μ_0, σ_0, σ *and training set* $\mathcal{D} = \{x_1, x_2, \ldots, x_n\}$.

2. *Estimate* σ *for the* x_2 *component of* ω_3 *in Table 1 and in file ch3_dhs_samples.dat. Now assume* $\mu_0 = -1$ *and plot your estimated densities* $p(x|\mathcal{D})$ *for each of the following values of the dogmatism* $\sigma^2/\sigma_0^2 : 0.1, 1, 10, 100$.

3. *Repeat above process but this time generate a dense sample set with the same mean and standard deviation as in the real dataset.*

	ω_1			ω_2			ω_3		
point	x_1	x_2	x_3	x_1	x_2	x_3	x_1	x_2	x_3
1	0.42	-0.087	0.58	-0.4	0.58	0.089	0.83	1.6	-0.014
2	-0.2	-3.3	-3.4	-0.31	0.27	-0.04	1.1	1.6	0.48
3	1.3	-0.32	1.7	0.38	0.055	-0.035	-0.44	-0.41	0.32
4	0.39	0.71	0.23	-0.15	0.53	0.011	0.047	-0.45	1.4
5	-1.6	-5.3	-0.15	-0.35	0.47	0.034	0.28	0.35	3.1
6	-0.029	0.89	-4.7	0.17	0.69	0.1	-0.39	-0.48	0.11
7	-0.23	1.9	2.2	-0.011	0.55	-0.18	0.34	-0.079	0.14
8	0.27	-0.3	-0.87	-0.27	0.61	0.12	-0.3	-0.22	2.2
9	-1.9	0.76	-2.1	-0.065	0.49	0.0012	1.1	1.2	-0.46
10	0.87	-1.0	-2.6	-0.12	0.054	-0.063	0.18	-0.11	-0.49

Table 1: Three-dimensional data sampled from three categories.

```
function x_density_given_d = ch3_bayesian_estimation_1d(X, sigma,
    mu_0, sigma_0)
% Bayesian parameter estimation for a univariate normal
    distribution.
% S. Makrogiannis, Delaware State Univ, 10/2015.

% Initial parameters and calculations.
X = sort(X);
n = numel(X);

hat_mu_n = sum(X) / n;
```

```
normal_density = @(x, mu, sigma) ( (1/(sqrt(2*pi)*sigma)) * exp(
    (-0.5) * ( (x - mu) / sigma )^2 ) );

% For a range of values of our random variable x:
for i=1:n
    x_density_given_d(i) = 0;

    for mu = mu_0-4*sigma_0:mu_0+4*sigma_0
        % Estimate p(mu|D) using Bayesian technique.
        [mu_density_given_d(i), mu_n, sigma_n] = ...
            bayesian_parameter_density(mu, sigma, mu_0, sigma_0,
                hat_mu_n, n);

        % Compute p(x|mu)
        x_density_given_mu(i) = normal_density(X(i), mu, sigma);

        % Compute p(x|mu) * p(mu|D)
        % Add up to approximate integral.
        x_density_given_d(i) = x_density_given_d(i) +
            (x_density_given_mu(i) * mu_density_given_d(i));
    end

end

figure, plot(X, x_density_given_d); title('p(X|D)')

end

%-------------------------------------------------------------------------%

function [mu_density, mu_n, sigma_n] = ...
    bayesian_parameter_density(mu, sigma, mu_0, sigma_0, hat_mu_n,
        n)

% Compute mu_n and sigma_n
normal_density = @(x, mu, sigma) ( (1/(sqrt(2*pi)*sigma)) * exp(
    (-0.5) * ( (x - mu) / sigma )^2 ) );
```

```
mu_n = ( n * sigma_0^2 / ( n * sigma_0^2 + sigma^2 ) ) * hat_mu_n
    + ...
    ( sigma^2 / ( n * sigma_0^2 + sigma^2 ) ) * mu_0;

var_n = (sigma_0^2 * sigma^2) / (( n * sigma_0^2 + sigma^2 ));

sigma_n = sqrt(var_n);

mu_density = normal_density(mu, mu_n, sigma_n);

end
```

```
% Bayesian estimation for 1-D Gaussian distributions.

% Load data.
A = load('ch3_dhs_samples.dat');

% Initialize parameters and compute sigma.
dogmatism = [0.1, 1, 10, 100];
n_runs = numel(dogmatism);
sigma = std(A(:,8));
mu_0 = -1;

% Perform density estimation.
for i=1:n_runs
sigma_0 = sqrt(sigma^2/dogmatism(i));
x_density_given_d = ch3_bayesian_estimation_1d(A(:,8), sigma,
    mu_0, sigma_0);
end
```

2 Fisher Linear Discriminant

2.1 Discriminant Analysis

- PCA finds optimal data representations in the least square sense, however this does not imply that the transformed features will produce increased class separability

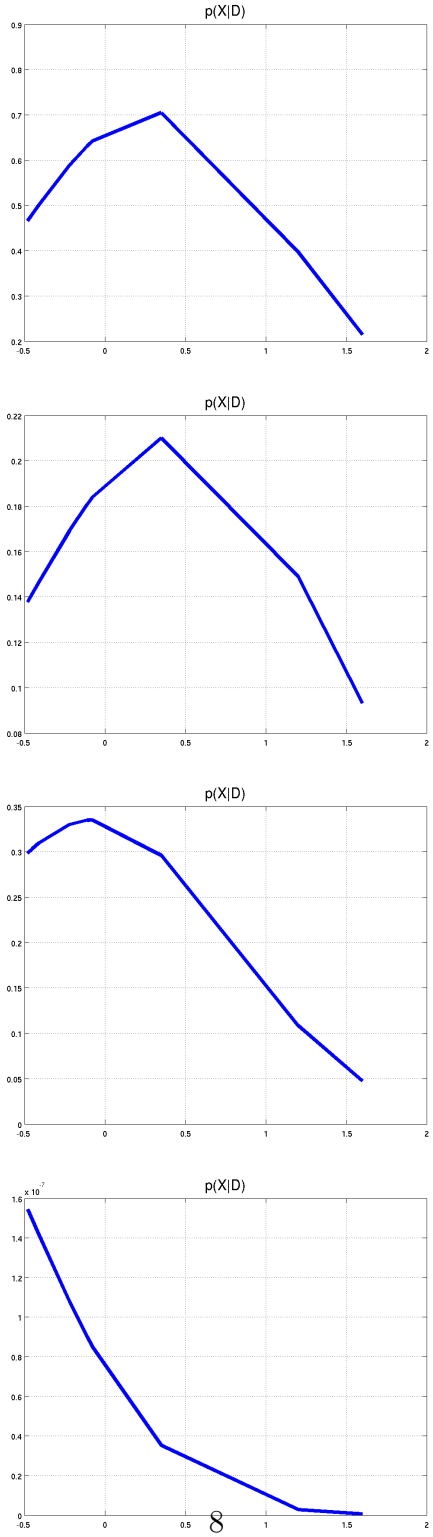


Figure 1: Bayesian parameter estimation example.

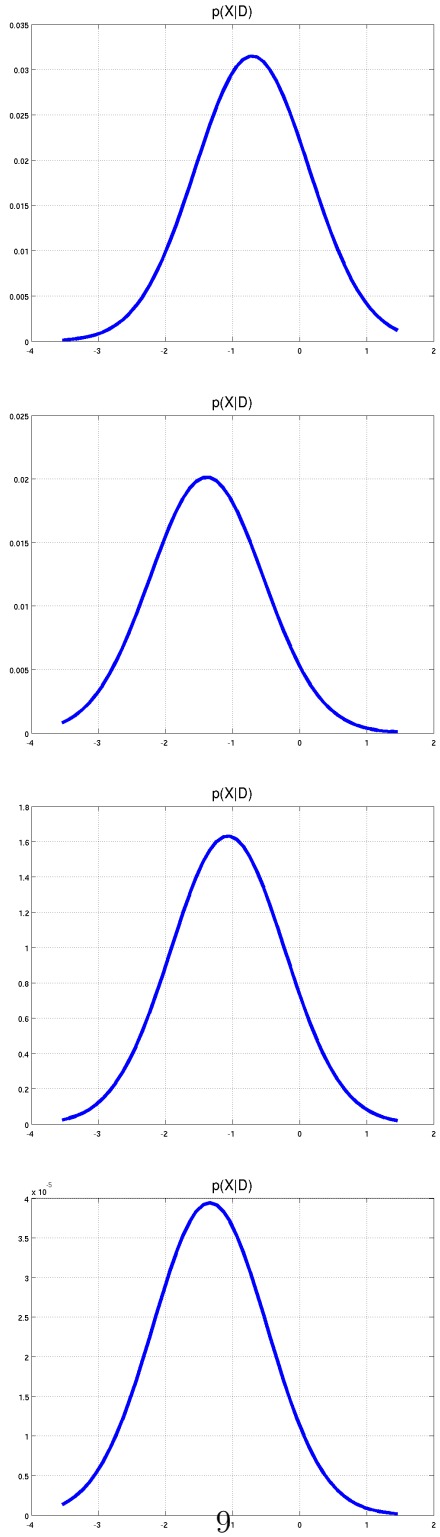


Figure 2: Bayesian parameter estimation example over a densely sampled space.

- On the other hand discriminant analysis techniques look for directions that distinguish between classes

2.2 Problem Definition

- Let's consider the problem of projecting data from d dimensions onto a line
- Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ be the set of n points in a d dimensional space divided into subsets $\mathcal{D}_i$ belonging to categories ω_i with cardinalities n_i, where $i = 1, 2$.
- Then the projections on to the direction determined by $\boldsymbol{w}$ with $|\boldsymbol{w}| = 1$ are

$$y = \boldsymbol{w}^T \boldsymbol{x}$$

- The projections produce a set of n samples y_i with $i = 1, \ldots, n$ divided into subsets $\mathcal{Y}_1$ and $\mathcal{Y}_2$
- Our problem is to find the direction of $\boldsymbol{w}$ that will maximize the separation between the projected points in $\mathcal{Y}_1$ and $\mathcal{Y}_2$

2.3 Class Separability

2.4 Criterion Function

- Fisher Linear Discriminant seeks maximization of $J(\boldsymbol{w})$ defined as

$$J(\boldsymbol{w}) = \frac{|m_{y1} - m_{y2}|^2}{s_{y1}^2 + s_{y2}^2}$$

where

m_{yi} is the sample mean of ω_i in the projected space:

$$m_{yi} = (1/n_i) \sum_{y \in \mathcal{Y}_i} y = (1/n_i) \sum_{x \in \mathcal{D}_i} \boldsymbol{w}^T \boldsymbol{x} = \boldsymbol{w}^T (1/n_i) \sum_{x \in \mathcal{D}_i} \boldsymbol{x} = \boldsymbol{w}^T m_{xi}$$

s_{yi}^2 is the scatter for projected samples of ω_i:

$$s_{yi}^2 = \sum_{y \in \mathcal{Y}_i} (y - m_{yi})^2$$

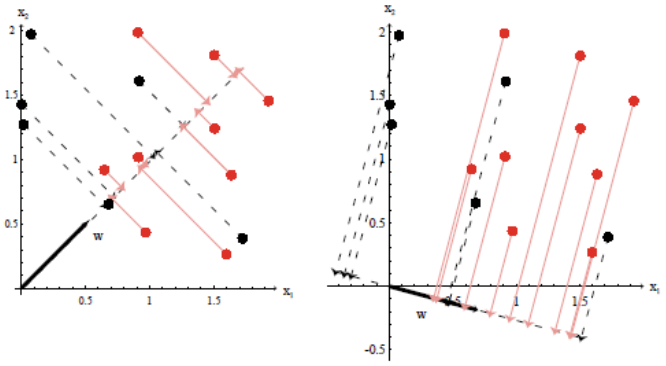


Figure 3: Projection of data onto different directions defined by $\boldsymbol{w}$. Observe that projection displayed in the right figure produces greater separability than the projection displayed in the left figure

2.5 Scatter Matrices

Further we define

- Scatter matrices: $S_i = \sum_{x \in \mathcal{D}_i} (\boldsymbol{x} - \boldsymbol{m}_{xi})(\boldsymbol{x} - \boldsymbol{m}_{xi})^T$, $i = 1, 2$
- Within-class scatter matrix: $S_W = S_1 + S_2$
- Because

$$s_{yi}^2 = \sum_{y \in \mathcal{Y}_i} (y - m_{yi})^2 = \sum_{y \in \mathcal{Y}_i} \left(\boldsymbol{w}^T \boldsymbol{x} - \boldsymbol{w}^T \boldsymbol{m_{xi}}\right)^2$$

$$= \boldsymbol{w}^T \sum_{y \in \mathcal{Y}_i} \left[(\boldsymbol{x} - \boldsymbol{m_{xi}})(\boldsymbol{x} - \boldsymbol{m_{xi}})^T \right] \boldsymbol{w} = \boldsymbol{w}^T S_i \boldsymbol{w},$$

$$s_{y1}^2 + s_{y2}^2 = \boldsymbol{w}^T S_W \boldsymbol{w}$$

- Consider the numerator of $J(\boldsymbol{w})$:

$$|m_{y1} - m_{y2}|^2 = (m_{y1} - m_{y2})^2 = \left(\boldsymbol{w}^T \boldsymbol{m_{x1}} - \boldsymbol{w}^T \boldsymbol{m_{x2}}\right)^2$$

$$= \boldsymbol{w}^T (\boldsymbol{m_{x1}} - \boldsymbol{m_{x2}})^2 = \boldsymbol{w}^T (\boldsymbol{m_{x1}} - \boldsymbol{m_{x2}})(\boldsymbol{m_{x1}} - \boldsymbol{m_{x2}})^T \boldsymbol{w}$$

$$= \boldsymbol{w}^T S_B \boldsymbol{w},$$

where S_B is the between-class scatter matrix

$$S_B = (\boldsymbol{m_{x1}} - \boldsymbol{m_{x2}})(\boldsymbol{m_{x1}} - \boldsymbol{m_{x2}})^T$$

- Proportional to the sample covariance matrix
- Symmetric and positive-semidefinite
- Nonsingular if $n > d$
- Outer product of two vectors
- Symmetric and positive-semidefinite
- Its rank is at most 1

2.6 Optimizing the Criterion Function

- We use the scatter matrix definitions it follow that the criterion function is:

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T S_B \boldsymbol{w}}{\boldsymbol{w}^T S_W \boldsymbol{w}}$$

- This is a Rayleigh quotient
- The $\boldsymbol{w}$ that maximizes $J(\boldsymbol{w})$ must satisfy $S_B\boldsymbol{w} = \lambda S_W \boldsymbol{w}$ (generalized eigenvalue problem)
- If S_W is nonsingular, we have the conventional eigenvalue problem

$$S_W^{-1} S_B \boldsymbol{w} = \lambda \boldsymbol{w}$$

- We do not need to solve

$$S_W^{-1} S_B \boldsymbol{w} = \lambda \boldsymbol{w}$$

- Recall that $S_B\boldsymbol{w}$ is at the direction of $\boldsymbol{m}_1 - \boldsymbol{m}_2$
- Hence the solution is:

$$\boldsymbol{w} = S_W^{-1}(\boldsymbol{m}_1 - \boldsymbol{m}_2)$$

- After the projection, we make a decision in the unidimensional space

2.7 Classification Rule

- Assuming multivariate normal class-conditional densities $p(\boldsymbol{x}|\omega_i)$ with equal covariance matrices Σ, we recall from Ch. 2 that at the decision boundary

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = 0,$$

  where

$$\boldsymbol{w} = \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

- When we use the sample means and sample covariance matrix it follows that $\boldsymbol{w}$ is the one that maximizes the Fisher linear discriminant

- In this case, to classify we apply a threshold to Fisher's linear discriminant

Exercise 2. *Consider the Fisher linear discriminant method*

1. *Write a general program to calculate the optimal direction* $\boldsymbol{w}$ *for a Fisher linear discriminant based on three-dimensional data.*

2. *Find the optimal* $\boldsymbol{w}$ *for categories* ω_2 *and* ω_3 *in Table 1.*

3. *Plot a line representing your optimal direction* $\boldsymbol{w}$ *and mark on it the positions of the projected points.*

4. *In this subspace, fit each distribution with a univariate Gaussian, and find the resulting decision boundary.*

5. *What is the training error (the error on training points themselves) in the optimal subspace you found in part (2)?*

6. *For comparison, repeat parts (4) and (5) using instead the nonoptimal direction* $\boldsymbol{w} = (1.0, 2.0, -1.5)^T$. *What is the training error in this nonoptimal subspace?*

```
function [Y_class, w, X_class] = ch3_fisher_linear_discriminant(X,
    total_classes, class_numbers)
% Compute discriminant and project data to it.
% S. Makrogiannis, Delaware State Univ, 10/2015.

% Get number of classes
c = total_classes;
[n, c_times_d] = size(X);
d = c_times_d / c;
class_numbers_length = numel(class_numbers);

% Compute Sw and its inverse.
Sw = zeros(d, d);
X_class = cell(c, 1);
for i=class_numbers(1):class_numbers(class_numbers_length)
    % Compute scatter matrix for each class.
    first_column = d*(i-1)+1;
    last_column = d*i;
    X_class{i} = X(:, first_column:last_column)';
    mean_vector(:, i) = mean(X_class{i}, 2);
    M = repmat(mean_vector(:, i), 1, n);
```

```
    S{i} = (X_class{i} - M) * (X_class{i} - M)';
    % Add scatter matrices.
    Sw = Sw + S{i};
end

% Compute vector w
Sw_Inv = inv(Sw);
w = Sw_Inv * ...
    ( mean_vector( :, class_numbers(1) ) - ...
    mean_vector( :, class_numbers(class_numbers_length) ) );

% Project data to w.
for i=class_numbers(1):class_numbers(class_numbers_length)
    Y_class{i} = w' * X_class{i};
end

end
```

```
% Fisher linear discriminant.

% Load data.
A = load('ch3_dhs_samples.dat');
c = 3;
[n, c_times_d] = size(A);
d = c_times_d / c;

% Find optimal w for categories omega1 and omega2.
class_numbers = [2, 3];
[Y_class, w, X_class] = ch3_fisher_linear_discriminant(A, 3,
    class_numbers);

% Plot a line representing w and the positions of the plotted
    points.
figure, plot3(X_class{2}(1,:), X_class{2}(2,:), X_class{2}(3,:),
    'bo', 'linewidth', 4); hold on;
plot3(X_class{3}(1,:), X_class{3}(2,:), X_class{3}(3,:), 'gx',
    'linewidth', 4);
% plot3([0, w(1)], [0, w(2)], [0, w(3)], 'k-', 'linewidth', 4);
    grid on;
```

```
title('Points and discriminant vector', 'fontsize', 18);
saveas(gcf, 'Fisher_Linear_Discriminant_Lab.png')

Projection_Vector = cell(3, 1);
for i=1:n
    Projection_Vector{2}(1:c,i) = Y_class{2}(i) * w(1);
    Projection_Vector{3}(1:c,i) = Y_class{3}(i) * w(1);
end

figure, plot3(Projection_Vector{2}(1,:),
    Projection_Vector{2}(2,:), Projection_Vector{2}(3,:), ...
    'bo', 'linewidth', 4); hold on;
plot3(Projection_Vector{3}(1,:), Projection_Vector{3}(2,:),
    Projection_Vector{3}(3,:), ...
    'gx', 'linewidth', 4);
% plot3([0, w(1)], [0, w(2)], [0, w(3)], 'k-', 'linewidth', 4);
    grid on;
title('Projection onto line', 'fontsize', 18);
saveas(gcf, 'Fisher_Linear_Discriminant_Lab_02.png')

figure, plot(Y_class{2}, ones(n, 1),'bo'); hold on;
plot(Y_class{3}, ones(n, 1), 'gx', 'linewidth', 4); grid on;
title('Points in 1-d space', 'fontsize', 18);
saveas(gcf, 'Fisher_Linear_Discriminant_Lab_03.png')

% Fit each distribution with a univariate Gaussian.
mu_2 = mean(Y_class{2});
sigma_2 = std(Y_class{2});
mu_3 = mean(Y_class{3});
sigma_3 = std(Y_class{3});

% Find decision boundary.
y_0 = 0.06;

% Calculate training error.

Y_Data = [Y_class{2}, Y_class{3}];
L_Data = [2* ones(1,n),3* ones(1,n)];

Decision = Y_Data < y_0;
```

```
Decision = Decision + 2;

classification_rate = 100 * (sum( Decision == L_Data) /
    numel(L_Data));
fprintf('Overall classification rate = %f\n', classification_rate);

% Repeat above process for w = [1, 2, -1.5]' and compute the
    training
% error.
```

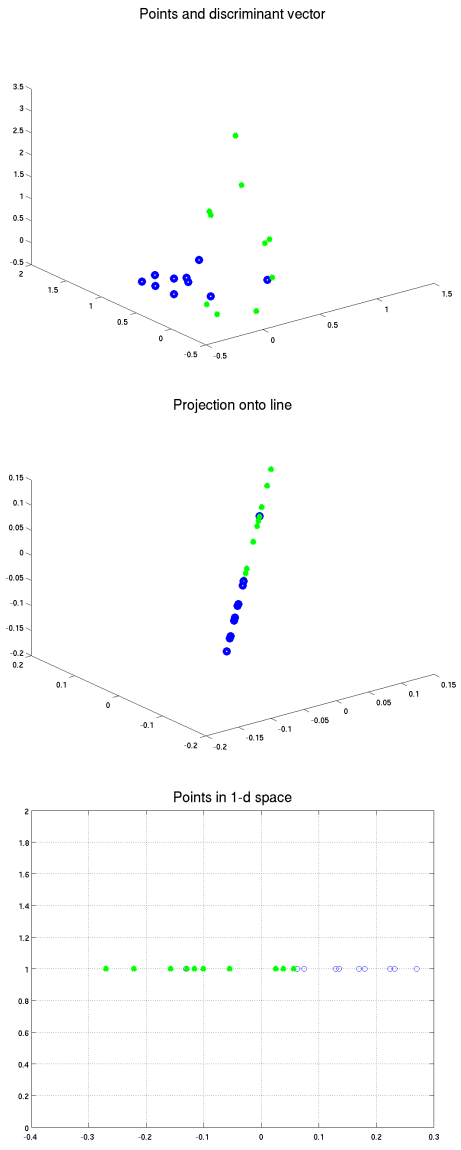


Figure 4: Fisher linear discriminant example.

Parameter Estimation

MTSC 852, Fall 2015

Sokratis Makrogiannis, Ph.D.

Department of Mathematical Sciences
Delaware State University

smakrogiannis@desu.edu

September 22, 2015

Outline

Challenges

- According to Bayesian Decision Theory we can build a classifier when the prior probability $P(\omega)$ and likelihood $p(\mathbf{x}|\omega_i)$ are known
- In the real-world these probabilities and densities are unknown and must be estimated from data
- In specific, the estimation of likelihood (density) may be challenging especially, when the number of samples is limited and the dimensionality is high
- There are two main approaches to this density estimation: parametric and nonparametric

Solutions for Density Estimation

Parameter Estimation

- We assume a model for the density function, for example, Gaussian, Rician, etc, and we need to estimate the parameters of the function, for example, mean and variance.
- Main techniques for parameter estimation
  - Maximum Likelihood
  - Bayesian Estimation

Nonparametric Estimation

- We make no assumptions about the form of the density
- Main techniques for nonparametric estimation
  - Parzen Kernels
  - Nearest Neighbor Rule

The Parameter Estimation Problem

Assumptions

- A set of n training samples/patterns $\mathscr{D} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$
- Samples are drawn from a distribution $p(\mathbf{x}|\omega_i)$ with a known parametric form, e.g. $p(\mathbf{x}|\omega_i) \sim N(\mu_i, \Sigma_i)$
- The parameters are collectively represented by θ, $\theta = (\mu_i, \Sigma_i)$
- Therefore, density can be written as $p(\mathbf{x}|\theta)$

Objective

Given a set of n training samples/patterns $\mathscr{D}$, estimate θ

Maximum Likelihood vs Bayesian Estimation

Maximum Likelihood (ML) Estimation

- Parameters θ are assumed to be fixed
- The solution is found as set that yields the best fitting model

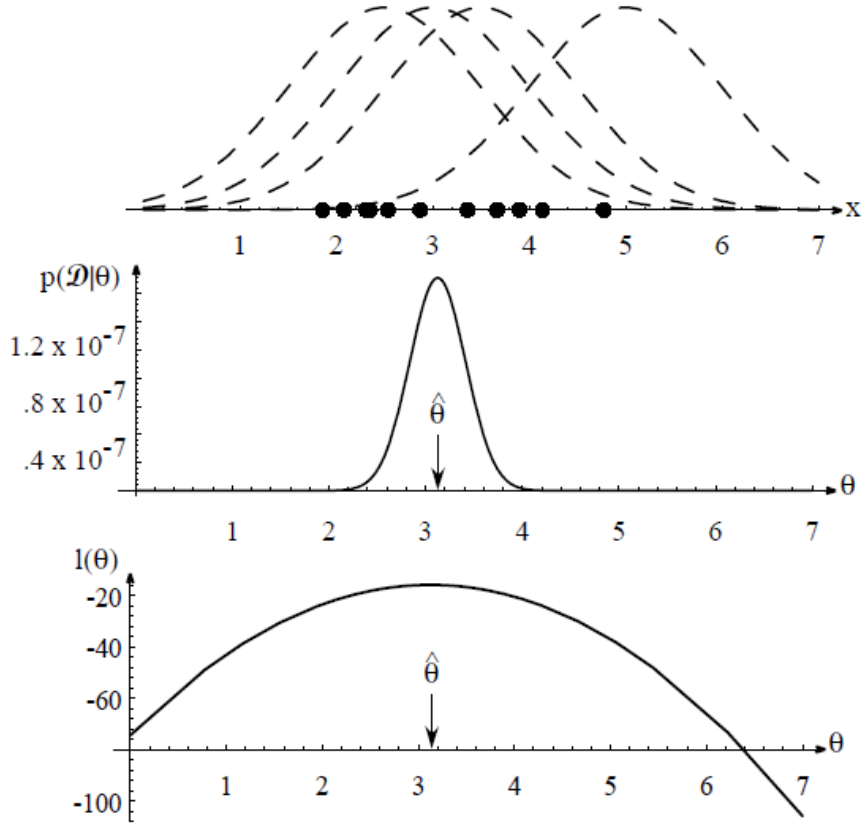
$$\hat{\theta} = arg\max p(\mathscr{D}|\theta)$$

Bayesian Estimation

- Parameters θ are considered to be random variables with known prior distribution
- Given the observations $\mathscr{D}$ we estimate the posterior $p(\theta|\mathscr{D})$

Maximum Likelihood Estimation

- As explained before, we seek to estimate $p(\mathbf{x}|\omega_i, \theta)$
- To achieve this we look for the parameters $\hat{\theta}$ that best describe the n samples $\mathscr{D} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$
- This is equivalent to finding the value $\hat{\theta}$, such that $\hat{\theta} = arg\max p(\mathscr{D}|\theta)$



Maximum Likelihood Estimation

- Assuming that samples in $\mathscr{D}$ are drawn independently,

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

- If $f(\theta) = p(\mathscr{D}|\theta)$ is a differentiable function, we can use differential calculus to find the maximizer from

$$\nabla_\theta f(\theta) = 0$$

- Let $\theta = (\theta_1, \theta_2, ..., \theta_p)$. Then $\nabla_\theta = [\frac{\partial}{\partial \theta_1} \quad \frac{\partial}{\partial \theta_2} \quad ... \quad \frac{\partial}{\partial \theta_p}]^T$

Maximum Likelihood Estimation

- For analytical tractability reasons let us optimize the logarithm of f. Then

$$\hat{\theta} = arg\max \ln f(\theta) = arg\max \ln \prod_{k=1}^{n} p(\mathbf{x_k}|\theta) = arg\max \sum_{k=1}^{n} \ln p(\mathbf{x_k}|\theta)$$

- According to previous treatment we obtain solution from a set of p equations

$$\nabla_\theta \sum_{k=1}^{n} \ln p(\mathbf{x_k}|\theta) = 0$$

Case 1: Gaussian with unknown μ

Problem statement

- We assume a multivariate normal population with unknown mean μ and known covariance matrix Σ
- We seek to estimate μ

- Log likelihood for each sample:
  $\ln p_k(\mathbf{x_k}|\mu) = (-1/2)\ln\left[(2\pi)^d|\Sigma|\right] - (1/2)(\mathbf{x}_k-\mu)^T\Sigma^{-1}(\mathbf{x}_k-\mu)$
- To optimize we solve:

$$\sum_{k=1}^{n}\nabla_\mu \ln p_k(\mathbf{x}_k|\hat{\mu}) = 0 \Leftrightarrow \sum_{k=1}^{n}\Sigma^{-1}(\mathbf{x}_k-\hat{\mu}) = 0$$

- Hence:

$$\hat{\mu} = (1/n)\sum_{k=1}^{n}\mathbf{x}_k$$

Case 2: Gaussian with unknown μ and unknown σ

Problem statement

- We assume a univariate normal population with unknown mean μ and unknown covariance matrix σ
- We seek to estimate $\theta = (\mu, \sigma)$

- Log likelihood for each sample:
  $\ln p_k(x_k|\theta) = (-1/2)\ln\left(2\pi\sigma^2\right) - (1/2\sigma^2)(\mathbf{x}_k - \mu)^2$
- Derivative is $\nabla_\theta \ln p_k(x_k|\theta) = \left[(1/\sigma^2)(x_k - \mu), \quad \frac{-1}{2\sigma^2} + \frac{(x_k-\mu)^2}{2\sigma^4}\right]^T$
- We solve: $\sum_{k=1}^{n}(1/\hat{\sigma}^2)(x_k - \hat{\mu}) = 0$ and $\sum_{k=1}^{n}\left[\frac{-1}{2\hat{\sigma}^2} + \frac{(x_k-\hat{\mu})^2}{2\hat{\sigma}^4}\right] = 0$
- After some rearranging we get:
  $\hat{\mu} = (1/n)\sum_{k=1}^{n} x_k, \quad \hat{\sigma}^2 = (1/n)\sum_{k=1}^{n}(x_k - \hat{\mu})^2$

Case 3: Gaussian with unknown μ and unknown Σ

Problem statement

- We assume a multivariate normal population with unknown mean μ and unknown covariance matrix Σ
- We seek to estimate $\theta = (\mu, \Sigma)$

- Log likelihood for each sample:
  $\ln p_k(\mathbf{x_k}|\mu) = (-1/2)\ln\left[(2\pi)^d|\Sigma|\right] - (1/2)(\mathbf{x}_k - \mu)^T\Sigma^{-1}(\mathbf{x}_k - \mu)$
- Next, we have to solve a system similar to Case 2:
  $\sum_{k=1}^{n} \nabla_\theta \ln p(\mathbf{x}_k|\theta) = 0$
- The maximum-likelihood solution is:
  $\hat{\mu} = (1/n)\sum_{k=1}^{n}\mathbf{x_k}, \quad \hat{\Sigma} = (1/n)\sum_{k=1}^{n}(\mathbf{x_k} - \hat{\mu})(\mathbf{x_k} - \hat{\mu})^T$

Bias of Maximum-Likelihood Estimation Technique

- Maximum likelihood estimates for a Gaussian with unknown μ and unknown Σ:

$$\hat{\mu} = (1/n) \sum_{k=1}^{n} \mathbf{x_k}, \quad \hat{\Sigma} = (1/n) \sum_{k=1}^{n} (\mathbf{x_k} - \hat{\mu})(\mathbf{x_k} - \hat{\mu})^T$$

- Sample mean and sample covariance matrix:

$$\mu = (1/n) \sum_{k=1}^{n} \mathbf{x_k}, \quad C = \frac{1}{n-1} \sum_{k=1}^{n} (\mathbf{x_k} - \hat{\mu})(\mathbf{x_k} - \hat{\mu})^T$$

- Hence $\hat{\mu} = \mu$, $\hat{\Sigma} = \frac{n-1}{n} C$
- Therefore $\hat{\mu}$ is an unbiased estimate of the mean, but $\hat{\Sigma}$ is biased
- $\hat{\Sigma} \to C$ when $n \to \infty$, therefore $\hat{\Sigma}$ is called asymptotically unbiased

Further Comments on Maximum-Likelihood Estimation

- Maximum likelihood estimation is usually simpler than other estimation methods
- The estimates become more accurate as the number of samples increases
- If the likelihood model $p(x|\theta)$ is accurate, it yields very good results
- Model selection is a key process that is studied by the field of Algorithm-Independent Machine Learning

Parameter Estimation

MTSC 852, Fall 2015

Sokratis Makrogiannis, Ph.D.

Department of Mathematical Sciences
Delaware State University

smakrogiannis@desu.edu

September 29, 2015

Outline

Bayesian Estimation

- We consider the parameter vector θ to be a random variable with prior $p(\theta)$
- We use training data $\mathscr{D}$ to estimate the posterior probability density $p(\theta|\mathscr{D})$
- Finally we integrate over the parameter space to estimate the class-conditional density $p(\mathbf{x}|\omega_i, \mathscr{D})$

Class-conditional Densities

- In Bayesian classification we face the problem of making a decision using posterior probabilities $P(\omega_i|\mathbf{x})$
- We need to know the likelihood $p(\mathbf{x}|\omega_i)$ and priors $P(\omega_i)$ to calculate the posteriors
- We have seen that the estimation of class-conditional density (or likelihood) is non-trivial in real-world applications
- To solve this problem we first assume a parametric functional form for the likelihood and then use training samples to estimate the likelihood for each class

Class-conditional Densities

- Given a training sample set $\mathscr{D} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}\}$, the Bayesian rule becomes

$$P(\omega_i|\mathbf{x}, \mathscr{D}) = \frac{p(\mathbf{x}|\omega_i, \mathscr{D})P(\omega_i|\mathscr{D})}{\sum_{j=1}^{c} p(\mathbf{x}|\omega_j, \mathscr{D})P(\omega_j|\mathscr{D})}$$

- If we assume that the class-conditionals of different classes are statistically independent it follows that

$$P(\omega_i|\mathbf{x}, \mathscr{D}_i) = \frac{p(\mathbf{x}|\omega_i, \mathscr{D}_i)P(\omega_i)}{\sum_{j=1}^{c} p(\mathbf{x}|\omega_j, \mathscr{D}_i)P(\omega_j)}$$

- Now we need to solve c likelihood estimation problems

Bayesian Estimation Steps

- According to previous analysis, we seek to estimate $p(\mathbf{x}|\mathscr{D})$ for each class
- This is achieved by integration over the parameter space

$$p(\mathbf{x}|\mathscr{D}) = \int p(\mathbf{x}, \theta|\mathscr{D}) d\theta$$

- Using definition of joint probability:

$$p(\mathbf{x}|\mathscr{D}) = \int p(\mathbf{x}|\theta, \mathscr{D}) p(\theta|\mathscr{D}) d\theta$$

- Bayes rule to estimate $p(\theta|\mathscr{D})$: $p(\theta|\mathscr{D}) = \frac{p(\mathscr{D}|\theta)P(\theta)}{\int p(\mathscr{D}|\theta)P(\theta)d\theta}$
- If samples are independently drawn, then:

$$p(\mathscr{D}|\theta) = \prod_{i=1}^{n} p(\mathbf{x}_i|\theta)$$

Univariate Normal Density

Problem

Find the class-conditional density $p(x|\mathscr{D})$ using Bayesian estimation assuming that $p(x|\mu) \sim N(\mu, \sigma^2)$, $p(\mu) \sim N(\mu_0, \sigma_0)$ and σ is known.

Solution steps

- Estimate $p(\mu|\mathscr{D})$ using Bayes rule
- Estimate $p(x|\mathscr{D})$ by integration over the parameter space

Univariate Normal Density

Problem

Find the class-conditional density $p(x|\mathscr{D})$ using Bayesian estimation assuming that $p(x|\mu) \sim N(\mu, \sigma^2)$, $p(\mu) \sim N(\mu_0, \sigma_0)$ and σ is known.

Estimate $p(\mu|\mathscr{D})$

- According to previous analysis, we seek to estimate $p(\mathbf{x}|\mathscr{D})$ for each class
- This is achieved by integration over the parameter space

$$p(\mathbf{x}|\mathscr{D}) = \int p(\mathbf{x}, \theta|\mathscr{D}) d\theta$$

- From definition of joint probability: $p(\mathbf{x}|\mathscr{D}) = \int p(\mathbf{x}|\theta, \mathscr{D}) p(\theta|\mathscr{D}) d\theta$
- We use Bayes rule to estimate $p(\theta|\mathscr{D})$: $p(\theta|\mathscr{D}) = \frac{p(\mathscr{D}|\theta)P(\theta)}{\int p(\mathscr{D}|\theta)P(\theta)d\theta}$
- If the samples are independently drawn, then: $p(\mathscr{D}|\theta) = \prod_{i=1}^{n} p(\mathbf{x}_i|\theta)$

Estimate $p(\mu|\mathscr{D})$

- We use Bayes rule to estimate posterior parameter density $p(\mu|\mathscr{D})$:

$$p(\mu|\mathscr{D}) = \frac{p(\mathscr{D}|\mu)p(\mu)}{\int p(\mathscr{D}|\mu)p(\mu)d\mu}$$

- Let samples $\mathscr{D} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}\}$ be independently drawn. Then: $p(\mathscr{D}|\mu) = \prod_{i=1}^{n} p(x_i|\mu)$
- Then: $p(\mu|\mathscr{D}) = \alpha \cdot \prod_{i=1}^{n} p(x_i|\mu)p(\mu)$
- According to assumptions:

$$p(x_i|\mu) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}, \quad p(\mu) = \frac{1}{\sqrt{2\pi}\sigma_0} e^{-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}}$$

- So: $p(\mu|\mathscr{D}) = \alpha \cdot \prod_{i=1}^{n} \left[\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \frac{1}{\sqrt{2\pi}\sigma_0} e^{-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}} \right]$

Estimate $p(\mu|\mathscr{D})$

- After some more manipulations we can show that $p(\mu|\mathscr{D})$ is an exponential function of a quadratic function, hence it has the form $N(\mu_n, \sigma_n)$
- Therefore

$$p(\mu|\mathscr{D}) = \frac{1}{\sqrt{2\pi}\sigma_n} e^{-\frac{(\mu-\mu_n)^2}{2\sigma_n^2}}$$

where,

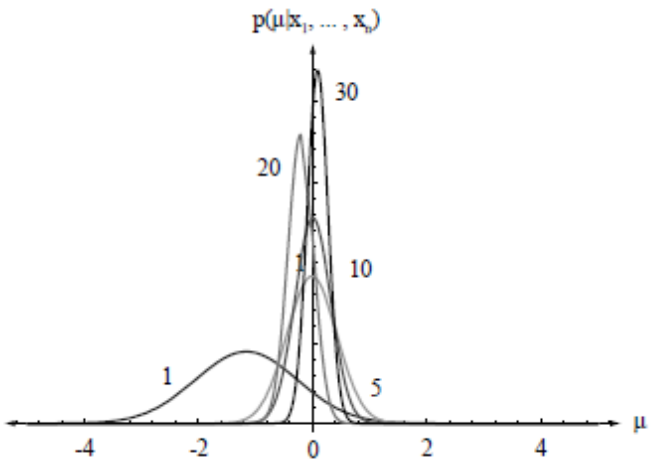
$$\mu_n = \left(\frac{n\sigma_0^2}{n\sigma_0^2+\sigma^2}\right)\hat{\mu}_n + \frac{\sigma^2}{n\sigma_0^2+\sigma^2}\mu_0, \quad \sigma_n^2 = \frac{\sigma_0^2\sigma^2}{n\sigma_0^2+\sigma^2}$$

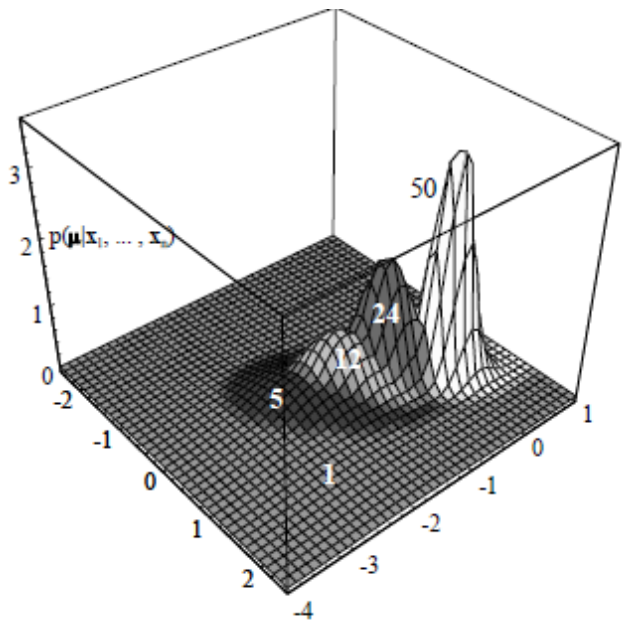
$$\hat{\mu}_n = (1/n)\sum_{i=1}^{n} x_i$$

Estimate $p(\mu|\mathscr{D})$

Bayesian learning

- σ_n decreases as $n \to \infty$ with $\lim_{n\to\infty} \sigma_n^2 = \frac{\sigma^2}{n}$
- We observe that as the number of training samples increases, $p(\mu|\mathscr{D})$ becomes sharper around μ_n. This process is called *Bayesian learning*
- If $\sigma_0 \neq 0$, then μ_n approaches the sample mean $\lim_{n\to\infty} \mu_n = \hat{\mu}_n$.





Estimate $p(x|\mathscr{D})$

- According to Bayesian estimation process

$$p(x|\mathscr{D}) = \int p(x|\mu, \mathscr{D}) p(\mu|\mathscr{D}) d\mu \Leftrightarrow$$

$$p(x|\mathscr{D}) = \int \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \frac{1}{\sqrt{2\pi}\sigma_n} e^{-\frac{(\mu-\mu_n)^2}{2\sigma_n^2}} d\mu \Leftrightarrow$$

$$p(x|\mathscr{D}) = \frac{1}{2\pi\sigma\sigma_n} f(\sigma, \sigma_n) e^{-\frac{(x-\mu_n)^2}{2(\sigma^2+\sigma_n^2)}}$$

where $f(\sigma, \sigma_n)$ has an integral form:

$$f(\sigma, \sigma_n) = \int \exp\left[(-1/2)\frac{\sigma^2+\sigma_n^2}{\sigma^2\sigma_n^2}\left(\mu - \frac{\sigma_n^2 x + \sigma^2\mu_n}{\sigma^2+\sigma_n^2}\right)^2\right] d\mu$$

Estimate $p(x|\mathscr{D})$

- Observe that $p(x|\mathscr{D}) \sim N(\mu_n, \sigma^2 + \sigma_n^2)$
- The above result gives the class-conditional density $p(x|\omega_i, \mathscr{D}_i)$ based on the posterior parameter mean estimate μ_n and the posterior parameter variance estimate increased by the uncertainty in x that we assume to be known

Multivariate Normal Density

Problem

Find the class-conditional density $p(\mathbf{x}|\mathscr{D})$ using Bayesian estimation assuming that $p(\mathbf{x}|\mu) \sim N(\mu, \Sigma)$, $p(\mu) \sim N(\mu_{\mathbf{0}}, \Sigma_0)$, and that Σ, μ_0, Σ_0 are known.

Solution steps

- Estimate $p(\mu|\mathscr{D})$ using Bayes rule
- Estimate $p(\mathbf{x}|\mathscr{D})$ by integration over the parameter space

Estimate $p(\mu|\mathscr{D})$

- Similarly to the unidimensional case we use the Bayes rule to estimate the posterior parametric density, and the assumption for idependent samples $\mathbf{x}_i \in \mathscr{D}$ with $|\mathscr{D}| = n$

$$p(\mu|\mathscr{D}) = \alpha \prod_{i=1}^{n} p(\mathbf{x}_i|\mathscr{D}) p(\mu) \Leftrightarrow$$

$$p(\mu|\mathscr{D}) = \alpha'' e^{(-1/2)(\mu-\mu_n)^T \Sigma_n^{-1} (\mu-\mu_n)}$$

- Hence: $p(\mu|\mathscr{D}) \sim N(\mu_n, \Sigma_n)$
- By equating coefficients we get:

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

where $\hat{\mu}_n = (1/n)\sum_{k=1}^{n} \mathbf{x}_k$

Estimate $p(\mu|\mathscr{D})$

- Finally after some manipulations it follows that

$$\mu_n = \Sigma_0[\Sigma_0 + (1/n)\Sigma]^{-1}\hat{\mu}_n + (1/n)\Sigma[\Sigma_0 + (1/n)\Sigma]^{-1}\mu_0$$

$$\Sigma_n = \Sigma_0[\Sigma_0 + (1/n)\Sigma]^{-1}(1/n)\Sigma$$

$$\hat{\mu}_n = (1/n)\sum_{i=1}^{n}\mathbf{x}_i$$

Estimate $p(\mathbf{x}|\mathscr{D})$

- To estimate the class-conditional density we must compute the integral:

$$p(\mathbf{x}|\mathscr{D}) = \int p(\mathbf{x}|\mu)p(\mu|\mathscr{D})d\mu$$

- Finally, we can show that $p(\mathbf{x}|\mathscr{D}) \sim N(\mu_n, \Sigma + \Sigma_n)$, either
(a) by performing integration, or
(b) by using the central limit theorem and the observation that $\mathbf{x}$ can be considered to be the sum of random variables μ with $p(\mu|\mathscr{D}) \sim N(\mu_n, \Sigma_n)$ and $\mathbf{y}$ with $p(\mathbf{y}) \sim N(0, \Sigma)$

Bayesian Parameter Estimation for Arbitrary Density Models

We can generalize the Bayesian technique to arbitrary density models $p(\mathbf{x}|\mathscr{D})$ with parameters θ

Main assumptions

- The form of $p(\mathbf{x}|\theta)$ is known but the parameter vector θ is unknown
- We know prior density $p(\theta)$
- We can learn more about θ from a set $\mathscr{D}$ of statistically independent n samples $\mathbf{x}_k$, for $k = 1, ..., n$, following the unknown $p(\mathbf{x})$

Bayesian Parameter Estimation for Arbitrary Density Models

How to compute the class-conditional

- Using definition of joint probability: $p(\mathbf{x}|\mathscr{D}) = \int p(\mathbf{x}|\theta, \mathscr{D}) p(\theta|\mathscr{D}) d\theta$
- Bayes rule to estimate $p(\theta|\mathscr{D})$: $p(\theta|\mathscr{D}) = \frac{p(\mathscr{D}|\theta)P(\theta)}{\int p(\mathscr{D}|\theta)P(\theta)d\theta}$
- If samples are independently drawn, then: $p(\mathscr{D}|\theta) = \prod_{i=1}^{n} p(\mathbf{x}_i|\theta)$

Comments

- If $p(\mathscr{D}|\theta)$ reaches a sharp peak at $\theta = \hat{\theta}$ and $p(\theta)$ is not zero at $\theta = \hat{\theta}$, then $p(\theta|\mathscr{D})$ also peaks at $\theta = \hat{\theta}$
- Then $p(\mathbf{x}|\mathscr{D}) \simeq p(\mathbf{x}|\hat{\theta})$, hence the ML and Bayesian estimates will be close
- Still, Bayesian estimation uses more information for the class-conditional density estimate than the ML technique

Recursive Bayes Incremental Learning

- Suppose that we are given a set of training samples $\mathscr{D}^n = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}\}$, where $n > 1$
- Then $p(\mathscr{D}^n|\theta) = p(\mathbf{x}_n|\theta)p(\mathscr{D}^{n-1}|\theta)$
- Using the Bayes rule:
  $p(\theta|\mathscr{D}^n) = \frac{p(\mathbf{x}_n|\theta)p(\mathscr{D}^{n-1}|\theta)P(\theta)}{\int p(\mathbf{x}_n|\theta)p(\mathscr{D}^{n-1}|\theta)P(\theta)d\theta} = \frac{p(\mathbf{x}_n|\theta)p(\theta|\mathscr{D}^{n-1})}{\int p(\mathbf{x}_n|\theta)p(\theta|\mathscr{D}^{n-1})d\theta}$
- Starting from $p(\theta|\mathscr{D}^0) = p(\theta)$ we can recursively compute $p(\theta|\mathbf{x}_1), p(\theta|\mathbf{x}_1, \mathbf{x}_2), ..., p(\theta|\mathbf{x}_1, ..., \mathbf{x}_n)$
- This is an on-line learning method that updates the model as more training data are collected and is called *recursive Bayes approach to parameter estimation*
- When the computation converges to a Dirac delta function centered at the true parameter value, we have achieved *Bayesian Learning*

Identifiability in Recursive Bayes Incremental Learning

- For most typical density models $p(\mathbf{x}|\theta)$, the sequence of posterior densities converges to a delta function, and a true value for θ can be found. Then $p(\mathbf{x}|\theta)$ is called *identifiable*
- However, in some cases there are multiple values of θ that explain the data. Fortunately, the integration operation for estimating $p(\mathbf{x}|\mathscr{D}^n)$ will still converge to $p(\mathbf{x})$ because all optimizing values of θ will yield the same $p(\mathbf{x}|\theta)$
- Non-identifiability may become an issue in unsupervised learning techniques

Comparing Maximum-Likelihood and Bayesian Parameter Estimation

- ML methods are computationally simpler than Bayesian estimation techniques because they employ differential calculus or gradient search techniques to find $\hat{\theta}$. On the other hand, Bayesian estimation involves complex multidimensional integration
- ML solutions correspond to single optimal models, hence they are easier to interpret and understand than Bayesian solutions that are produced by weighted sums of models
- Bayesian estimation techniques use more information and can produce better results for non-uniform and non-symmetric $p(\theta|\mathscr{D})$ distributions
- Bayesian techniques show the problem of bias and variance that depends on the number of training samples
- Overall Bayesian techniques are more theoretically sound, but ML techniques are simpler to implement and are nearly as accurate

Sources of Classification Error

When developing a classifier, we first estimate the posterior densities for each category, then classify a test sample using a maximum decision rule. The sources of error in such a system are:

- *Bayes or Indistinguishability Error* - happens because of overlapping class-conditional densities
- *Model Error* - caused by selection of the wrong model. This is not affected by the parameter estimation technique
- *Estimation Error* - due to the finite number of training samples. It decreases with increasing cardinality of the data set

Parameter Estimation

MTSC 852, Fall 2015

Sokratis Makrogiannis, Ph.D.

Department of Mathematical Sciences
Delaware State University

smakrogiannis@desu.edu

October 12, 2015

Outline

Difficulties Caused by Dimensionality

- Many classification problems involve samples in very high dimensional spaces, i.e. with hundreds, or thousands of features
- In this section we discuss the following considerations:
  1. relation between dimensionality, number of training samples and classification accuracy
  2. computational complexity of classifier
  3. overfitting

Classification Accuracy vs. Number of Features

- We are usually looking for statistically independent features based on theoretical grounds
- Suppose that we have 2-classes of multivariate normal densities with equal covariance matrices, that is $p(x|\omega_j) \sim N(\boldsymbol{\mu}_j, \Sigma)$ with all $P(\omega_j)$ equal for $j = 1, 2$
- Then the Bayes error is: $P(e) = \frac{1}{\sqrt{2\pi}} \int_{r/2}^{\infty} e^{-u^2/2} du$, where $r^2 = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \Sigma^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$
- For conditionally independent features: $\Sigma = diag(\sigma_1^2, ..., \sigma_D^2)$, therefore $r^2 = \sum_{i=1}^{D} \left(\frac{\mu_{i1} - \mu_{i2}}{\sigma_i} \right)^2$
- Observe that the addition of features is expected to increase r^2, therefore reducing the Bayes error

Classification Accuracy vs. Number of Features

- Based on the previous result, the addition of features with unequal class-conditional means will increase the separability of the data
- So it reasonable to add new features if the classification performance with a given feature set is not good enough

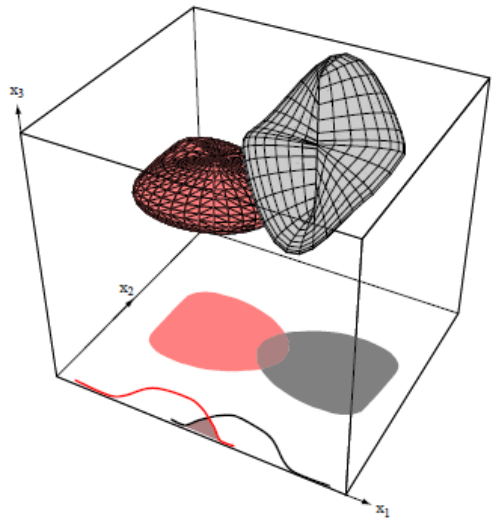


Figure: Projection of data to spaces of reduced dimensionality increases the Bayes error.

Classification Accuracy vs. Number of Features and Number of Training Samples

- However, in practice, the addition of features may reduce the classification performance
- This may happen because (1) the learning model is wrong, or (2) the number of samples is insufficient for the estimation of the class-conditional densities
- This is a significant consideration in classifier design that we will revisit in Ch. 9

Computational Complexity

- The computational load is a factor to be considered in classifier design
- To express the computational load we use the order of function $f(x)$

Definition

Let f and h be two functions of x. We say that $f(x)$ is of the order of $h(x)$, denoted by $f(x) = O(h(x))$ and read as "big oh of $h(x)$", if there exist constants c and x_0 such that $|f(x)| \leq c|h(x)|$, $\forall x > x_0$

Example

We assume that $f(x) = a_2x^2 + a_1x + a_0$. Then $f(x) = O(x^2)$ because for sufficiently large x, we can choose c and x_0 such that $|f(x)| \leq c|x^2|$, $\forall x > x_0$

Computational Complexity

- When estimating the computational complexity of an algorithm we are interested in the number of additions, multiplications and divisions, or in the time and memory requirements
- Let's see the example of computational complexity for a Bayes classifier

Computational Complexity

ML estimation (learning stage)

- $\hat{\mu} = (1/n)\sum_{k=1}^{n} \mathbf{x_k} = O(dn)$
- $\hat{\Sigma} = (1/n)\sum_{k=1}^{n} (\mathbf{x_k} - \hat{\mu})(\mathbf{x_k} - \hat{\mu})^T = O(d^2 n)$
- $g_i(\mathbf{x}) =$
  $-(1/2)(\mathbf{x} - \mu_i)^T \Sigma_i^{-1}(\mathbf{x} - \mu_i) - (d/2)\ln 2\pi - (1/2)\ln|\Sigma_i| + \ln P(\omega_i)$
  - $-(1/2)(\mathbf{x} - \mu_i)^T \Sigma_i^{-1}(\mathbf{x} - \mu_i) = O(dn) + O(d^3)$
  - $(d/2)\ln 2\pi = O(1)$
  - $(1/2)\ln|\Sigma_i| = O(d^2 n)$
  - $\ln P(\omega_i) = O(n)$
- Computations are repeated c times, hence the learning complexity is

$$O(cd^2 n) \simeq O(d^2 n),$$

assuming that $c << d$ or n and $n > d$

Classification Complexity

Decision Rule

- $g_i(\mathbf{x}) = -(1/2)(\mathbf{x}-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}-\mu_i) - (d/2)\ln 2\pi - (1/2)\ln|\Sigma_i| + \ln P(\omega_i)$
  - $g_i(\mathbf{x}) = -(1/2)(\mathbf{x}-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}-\mu_i) : O(d^2)$
  - $-(d/2)\ln 2\pi - (1/2)\ln|\Sigma_i| + \ln P(\omega_i) : O(1)$
- Computations are repeated c times, hence the classification complexity is

$$O(cd^2) \simeq O(d^2),$$

  assuming that $c << d$
- Hence, classification stage is computationally simpler than learning stage

Overfitting

Problem

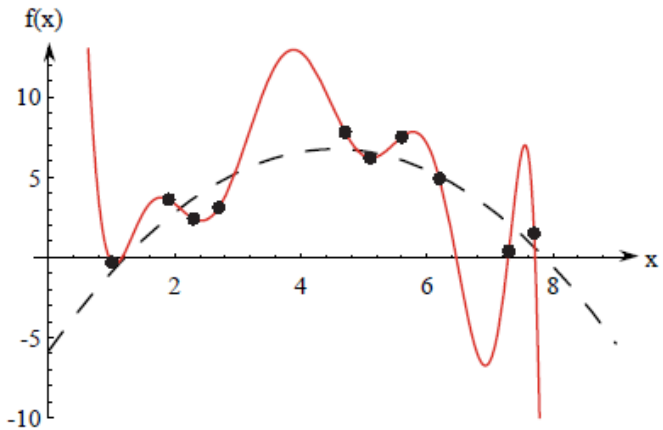
- When the number of samples is too small for the dimensionality, the class models and decision surfaces may not be optimal

Possible Solutions

1. We can reduce the dimensionality by (1) selecting a subset of our original features, or (2) computing features from the existing set
2. We can assume that all samples come from the same covariance matrix and pool the training data
3. We can find a better estimate of Σ (1) by using prior estimate $\boldsymbol{\Sigma}_0$ to get a Bayesian type of estimate $\lambda\Sigma_0 + (1-\lambda)\hat{\Sigma}$ or (2) by applying a threshold to covariances or even assuming statistical independence (heuristic solution)

Overfitting Example

- We have 10 data points obtained by adding zero-mean, Gaussian noise to a parabola
- The two curves correspond to a parabola and a 10-th degree polynomial
- While the 10-th degree polynomial produces the best fit for the training data, the parabola is the closest model



Assume the Same Covariance Matrix for All Classes

Problem

- We are asked to design a classifier with insufficient data for distributions $N(\boldsymbol{\mu}_1, \Sigma_1)$ and $N(\boldsymbol{\mu}_1, \Sigma_2)$

Solution

- We make the simplification that $\Sigma_1 = \Sigma_2 = \Sigma$, where Σ is estimated over both classes
- We normalize the data before estimating Σ

The Shrinkage Technique

- This technique uses a weighted sum of individual covariances that "shrink" toward a common estimate

$$\Sigma_i(\alpha) = \frac{(1-\alpha)n_i\Sigma_i + \alpha n\Sigma}{(1-\alpha)n_i + \alpha n},$$

where $0 < \alpha < 1$ is the regularizing parameter, n_i is the number of samples for each class, n is the total number of samples, $i = 1, ..., c$ and c is the number of classes

- We can also shrink the common covariance matrix to an identity matrix

$$\Sigma(\beta) = (1-\beta)\Sigma + \beta\mathbf{I},$$

where $0 < \beta < 1$

Parameter Estimation

MTSC 852, Fall 2015

Sokratis Makrogiannis, Ph.D.

Department of Mathematical Sciences
Delaware State University

smakrogiannis@desu.edu

October 22, 2015

Outline

Component Analysis for Dimensionality Reduction

- As seen before, a large number of features can improve class separability, but also complicate density estimation, increase computational complexity, and the system may be more susceptible to overfitting
- One solution is to reduce the original dimensionality by linear combinations of features
- Linear techniques are very useful because they are analytically tractable and computationally efficient

Component Analysis for Dimensionality Reduction

- Two popular techniques are (1) Principal Component Analysis, and (2) Fisher Linear Discriminant along with its multidimensional generalization called Multiple Discriminant Analysis
- Both techniques project data to lower dimensional spaces
- Principal Component Analysis seeks the projections that produce the more accurate representation of the data in a least-squares sense
- Fisher Linear Discriminant seeks the projections that best separate the data into the different classes in a least-squares sense

Zero-dimensional Representation

- First stage: representing Data Set by a Single Vector $\boldsymbol{x}_0$
- Suppose a set of n d-dimensional samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$
- We want to find a vector $\boldsymbol{x}_0$ to represent the data
- We require that $\boldsymbol{x}_0$ minimizes the squared error criterion function

$$J_0(\boldsymbol{x}_0) = \sum_{k=1}^{n} \|\boldsymbol{x}_0 - \boldsymbol{x}_k\|^2$$

Zero-dimensional Representation

- We express $J_0(\boldsymbol{x}_0)$ in terms of sample mean $m = (1/n)\sum_{k=1}^{n} \boldsymbol{x}_k$ and it follows that

$$J_0(\boldsymbol{x}_0) = \sum_{k=1}^{n} \|(\boldsymbol{x}_0 - \boldsymbol{m}) - (\boldsymbol{x}_k - \boldsymbol{m})\|^2 = \ldots$$

$$= \sum_{k=1}^{n} \|(\boldsymbol{x}_0 - \boldsymbol{m})\|^2 - \sum_{k=1}^{n} \|(\boldsymbol{x}_k - \boldsymbol{m})\|^2$$

- Hence $J_0(\boldsymbol{x}_0)$ is minimized when $\boldsymbol{x}_0 = \boldsymbol{m}$

One-dimensional Representation

- To obtain a representation of variability of data we project data onto a line running through the sample mean
- Assuming a unit vector $\boldsymbol{e}$, line equation is: $\boldsymbol{x} = \boldsymbol{m} + \alpha \boldsymbol{e}$, where α is scalar distance from sample mean
- Next, use $\boldsymbol{x}_k = \boldsymbol{m} + \alpha_k \boldsymbol{e}, \quad k = 1, \ldots, n$, and find the optimal set of α_k by minimizing function

$$J_1(\alpha_1, \ldots, \alpha_n, \boldsymbol{e}) = \sum_{k=1}^{n} \|(\boldsymbol{m} + \alpha_k \boldsymbol{e}) - \boldsymbol{x}_k\|^2$$

- We set $\frac{\partial J_1}{\partial \alpha_k} = 0$, and obtain

$$\alpha_k = \boldsymbol{e}^T(\boldsymbol{x}_k - \boldsymbol{m})$$

- This is the projection of $\boldsymbol{x}_k - \boldsymbol{m}$ on to the line that passes through the mean with the direction of $\boldsymbol{e}$

Finding the Best Line Direction

- Next step is to find the direction $\boldsymbol{e}$ that produces the best representation
- To do this we first substitute the α_k in J_1

$$J_1(\boldsymbol{e}) = \sum_{k=1}^{n} \alpha_k^2 - 2\sum_{k=1}^{n} \alpha_k^2 + \sum_{k=1}^{n} \|(\boldsymbol{x}_k - \boldsymbol{m})\|^2,$$

and use the definition for scatter matrix $S = \sum_{k=1}^{n}(\boldsymbol{x}_k - \boldsymbol{m})(\boldsymbol{x}_k - \boldsymbol{m})^T$ to obtain

$$J_1(\boldsymbol{e}) = -\boldsymbol{e}^T S \boldsymbol{e} + \sum_{k=1}^{n} \|\boldsymbol{x}_k - \boldsymbol{m}\|^2$$

Finding the Best Line Direction

- Previously we obtained:

$$J_1(\boldsymbol{e}) = -\boldsymbol{e}^T S \boldsymbol{e} + \sum_{k=1}^{n} \|\boldsymbol{x}_k - \boldsymbol{m}\|^2$$

- Hence, we look to maximize $\boldsymbol{e}^T S \boldsymbol{e}$, subject to the constraint $\|\boldsymbol{e}\| = 1$
- By the Langrangian multiplier optimization technique it follows that we should optimize

$$u = \boldsymbol{e}^T S \boldsymbol{e} - \lambda(\boldsymbol{e}^T \boldsymbol{e} - 1)$$

- Then solve

$$\frac{\partial u}{\partial \boldsymbol{e}} = 0 \Rightarrow 2S\boldsymbol{e} - 2\lambda \boldsymbol{e} = 0 \Rightarrow S\boldsymbol{e} = \lambda \boldsymbol{e}$$

Finding the Best Line Direction

- From previous result: $S\boldsymbol{e} = \lambda \boldsymbol{e}$
- We observe that: $\boldsymbol{e}^T S\boldsymbol{e} = \lambda \boldsymbol{e}^T \boldsymbol{e} = \lambda$
- Hence, we are looking for the eigenvector $\boldsymbol{e}_{max}$ with maximum eigenvalue λ_{max} in order to find the best vector direction
- To perform the analysis we project the data onto a line through sample mean with the orientation defined by $\boldsymbol{e}_{max}$

Multi-dimensional Case

- For a projection to a d'-dimensional space the hyperplane is defined as:

$$\boldsymbol{x} = \boldsymbol{m} + \sum_{i=1}^{d'} \alpha_i \boldsymbol{e}_i, \text{ where } d' < d$$

- The criterion function is:

$$J_{d'} = \sum_{k=1}^{n} \|(\boldsymbol{m} + \sum_{i=1}^{d'} \alpha_{ki} \boldsymbol{e}_i) - \boldsymbol{x}_k\|^2$$

- $J_{d'}$ is minimized when $\boldsymbol{e}_i$ for the eigenvectors of scatter matrix $S_{d'}$ with the d' largest eigenvalues
- Because $S_{d'}$ is real and symmetric the eigenvectors are orthogonal
- Coefficients α_i are called principal components

Discriminant Analysis

- PCA finds optimal data representations in the least square sense, however this does not imply that the transformed features will produce increased class separability
- On the other hand discriminant analysis techniques look for directions that distinguish between classes

Problem Definition

- Let's consider the problem of projecting data from d dimensions onto a line
- Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ be the set of n points in a d dimensional space divided into subsets $\mathscr{D}_i$ belonging to categories ω_i with cardinalities n_i, where $i = 1, 2$.
- Then the projections on to the direction determined by $\boldsymbol{w}$ with $|\boldsymbol{w}| = 1$ are

$$y = \boldsymbol{w}^T \boldsymbol{x}$$

- The projections produce a set of n samples y_i with $i = 1, \ldots, n$ divided into subsets $\mathscr{Y}_1$ and $\mathscr{Y}_2$
- Our problem is to find the direction of $\boldsymbol{w}$ that will maximize the separation between the projected points in $\mathscr{Y}_1$ and $\mathscr{Y}_2$

Class Separability

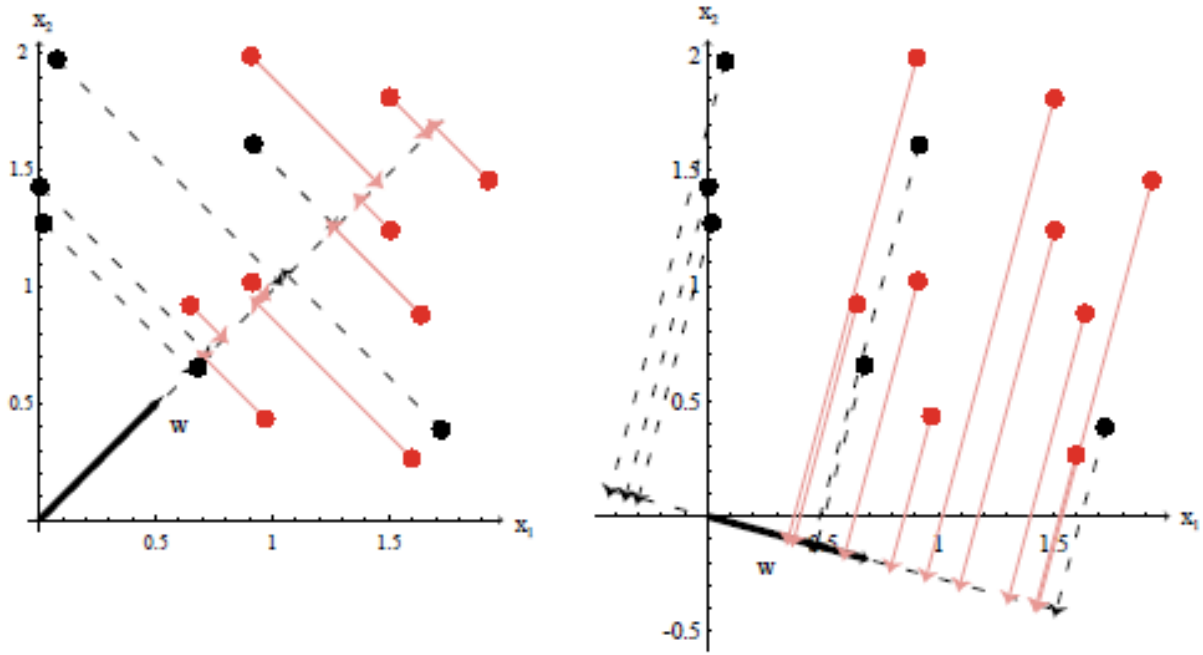


Figure: Projection of data onto different directions defined by $\boldsymbol{w}$. Observe that projection displayed in the right figure produces greater separability than the projection displayed in the left figure

Criterion Function

- Fisher Linear Discriminant seeks maximization of $J(\boldsymbol{w})$ defined as

$$J(\boldsymbol{w}) = \frac{|m_{y1} - m_{y2}|^2}{s_{y1}^2 + s_{y2}^2}$$

where
m_{yi} is the sample mean of ω_i in the projected space:

$$m_{yi} = (1/n_i) \sum_{y \in \mathcal{Y}_i} y = (1/n_i) \sum_{x \in \mathcal{D}_i} \boldsymbol{w}^T \boldsymbol{x} = \boldsymbol{w}^T (1/n_i) \sum_{x \in \mathcal{D}_i} \boldsymbol{x} = \boldsymbol{w}^T m_{xi}$$

s_{yi}^2 is the scatter for projected samples of ω_i:

$$s_{yi}^2 = \sum_{y \in \mathcal{Y}_i} (y - m_{yi})^2$$

Scatter Matrices

Further we define

- Scatter matrices: $S_i = \sum_{x \in \mathscr{D}_i} (\boldsymbol{x} - \boldsymbol{m}_{xi})(\boldsymbol{x} - \boldsymbol{m}_{xi})^T$, $i = 1, 2$
- Within-class scatter matrix: $S_W = S_1 + S_2$
- Because

$$s_{yi}^2 = \sum_{y \in \mathscr{Y}_i} (y - m_{yi})^2 = \sum_{y \in \mathscr{Y}_i} (\boldsymbol{w}^T \boldsymbol{x} - \boldsymbol{w}^T \boldsymbol{m}_{xi})^2$$

$$= \boldsymbol{w}^T \sum_{y \in \mathscr{Y}_i} \left[(\boldsymbol{x} - \boldsymbol{m}_{xi})(\boldsymbol{x} - \boldsymbol{m}_{xi})^T \right] \boldsymbol{w} = \boldsymbol{w}^T S_i \boldsymbol{w},$$

$$s_{y1}^2 + s_{y2}^2 = \boldsymbol{w}^T S_W \boldsymbol{w}$$

Between-class Scatter Matrix

- Consider the numerator of $J(\boldsymbol{w})$:

$$|m_{y1} - m_{y2}|^2 = (m_{y1} - m_{y2})^2 = (\boldsymbol{w}^T \boldsymbol{m}_{x1} - \boldsymbol{w}^T \boldsymbol{m}_{x2})^2$$

$$= \boldsymbol{w}^T (\boldsymbol{m}_{x1} - \boldsymbol{m}_{x2})^2 = \boldsymbol{w}^T (\boldsymbol{m}_{x1} - \boldsymbol{m}_{x2})(\boldsymbol{m}_{x1} - \boldsymbol{m}_{x2})^T \boldsymbol{w}$$

$$= \boldsymbol{w}^T S_B \boldsymbol{w},$$

where S_B is the between-class scatter matrix

$$S_B = (\boldsymbol{m}_{x1} - \boldsymbol{m}_{x2})(\boldsymbol{m}_{x1} - \boldsymbol{m}_{x2})^T$$

Scatter Matrices Notes

Within-class scatter matrix S_W

- Proportional to the sample covariance matrix
- Symmetric and positive-semidefinite
- Nonsingular if $n > d$

Between-class scatter matrix S_B

- Outer product of two vectors
- Symmetric and positive-semidefinite
- Its rank is at most 1

Optimizing the Criterion Function

- We use the scatter matrix definitions it follow that the criterion function is:

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T S_B \boldsymbol{w}}{\boldsymbol{w}^T S_W \boldsymbol{w}}$$

- This is a Rayleigh quotient
- The $\boldsymbol{w}$ that maximizes $J(\boldsymbol{w})$ must satisfy $S_B \boldsymbol{w} = \lambda S_W \boldsymbol{w}$ (generalized eigenvalue problem)
- If S_W is nonsingular, we have the conventional eigenvalue problem

$$S_W^{-1} S_B \boldsymbol{w} = \lambda \boldsymbol{w}$$

Optimizing the Criterion Function

- We do not need to solve

$$S_W^{-1} S_B \boldsymbol{w} = \lambda \boldsymbol{w}$$

- Recall that $S_B \boldsymbol{w}$ is at the direction of $\boldsymbol{m}_1 - \boldsymbol{m}_2$
- Hence the solution is:

$$\boldsymbol{w} = S_W^{-1}(\boldsymbol{m}_{x1} - \boldsymbol{m}_{x2})$$

- After the projection, we make a decision in the unidimensional space

Classification Rule

- Assuming multivariate normal class-conditional densities $p(\boldsymbol{x}|\omega_i)$ with equal covariance matrices Σ, we recall from Ch. 2 that at the decision boundary

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = 0,$$

where

$$\boldsymbol{w} = \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

- When we use the sample means and sample covariance matrix it follows that $\boldsymbol{w}$ is the one that maximizes the Fisher linear discriminant
- In this case, to classify we apply a threshold to Fisher's linear discriminant

Main Concept

- Expectation-Maximization can be used to learn distribution parameters $\boldsymbol{\theta}$ when some features are missing
- EM iteratively estimates the likelihood given the data that we have
- Suppose a training set $\mathscr{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$
- Let $\boldsymbol{x}_k = \{\boldsymbol{x}_{kg}, \boldsymbol{x}_{kb}\}$ be a sample consisting of good and bad features
- Let $\mathscr{D}_g$ be the set of good features and $\mathscr{D}_b$ be the set of bad features
- Likelihood function $l(\cdot)$: $l(\mathscr{D}|\boldsymbol{\theta}) = \ln p(\mathscr{D}_g, \mathscr{D}_b|\theta)$
- We define the function $Q(\boldsymbol{\theta}; \boldsymbol{\theta}^i)$:

$$Q(\boldsymbol{\theta}; \boldsymbol{\theta}^i) = \mathscr{E}_{\mathscr{D}_b}\left[l(\mathscr{D}|\boldsymbol{\theta})|\mathscr{D}_g; \boldsymbol{\theta}^i\right]$$

$$= \mathscr{E}_{\mathscr{D}_b}\left[\ln[p(\mathscr{D}_g, \mathscr{D}_b|\boldsymbol{\theta})]p(\mathscr{D}_b|\mathscr{D}_g, \boldsymbol{\theta}^i)\right]$$

Main Concept

- In $Q(\boldsymbol{\theta};\boldsymbol{\theta}^i)$:

$$Q(\boldsymbol{\theta};\boldsymbol{\theta}^i) = \mathscr{E}_{\mathscr{D}_b}\left[p(\mathscr{D}_b|\mathscr{D}_g,\boldsymbol{\theta}^i)\ln p(\mathscr{D}_g,\mathscr{D}_b|\boldsymbol{\theta})\right]$$

  $\boldsymbol{\theta}^i$: current best estimate for parameter vector $\boldsymbol{\theta}$: candidate for improved estimate
- In EM we calculate the likelihood of data for different $\boldsymbol{\theta}$s and select the candidate that maximizes $Q(\boldsymbol{\theta};\boldsymbol{\theta}^i)$, then set the best candidate to $\boldsymbol{\theta}^{i+1}$
- The procedure is continued till convergence

Algorithm

Algorithm 1 (Expectation-Maximization)

1 **begin** **initialize** $\boldsymbol{\theta}^0, T, i = 0$
2 **do** $i \leftarrow i + 1$
3 **E step** : compute $Q(\boldsymbol{\theta};\ \boldsymbol{\theta}^i)$
5 **M step** : $\boldsymbol{\theta}^{i+1} \leftarrow \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta};\ \boldsymbol{\theta}^i)$
6 **until** $Q(\boldsymbol{\theta}^{i+1};\ \boldsymbol{\theta}^i) - Q(\boldsymbol{\theta}^i;\ \boldsymbol{\theta}^{i-1}) \leq T$
7 **return** $\hat{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^{i+1}$
8 **end**

Figure: Main steps of EM technique

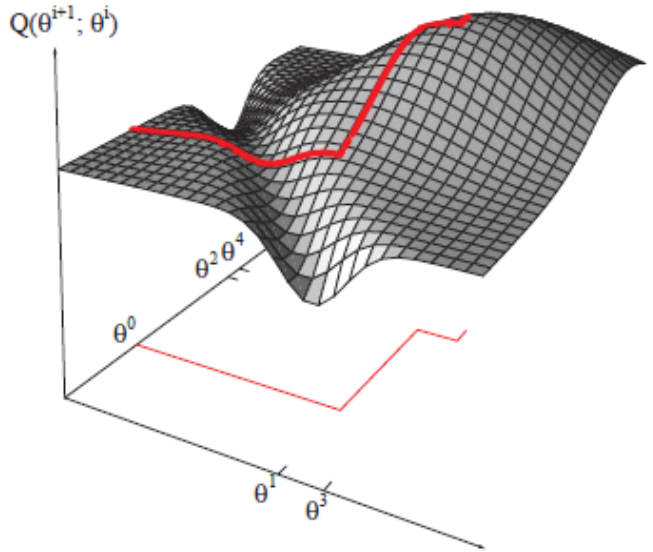


Figure: Starting from an initial estimate $\boldsymbol{\theta}^0$, EM find optimal $\boldsymbol{\theta}^1$ in M step. Then we hold $\boldsymbol{\theta}^1$ constant and find value $\boldsymbol{\theta}^2$ that optimizes $Q(\boldsymbol{\theta};\boldsymbol{\theta}^1)$. The procedure continues till convergence. It is different from gradient search

EM for Gaussian Mixtures

- In several problems we may need to build a richer class of density models than a single Gaussian
- Then we can use a Gaussian mixture model. This model is a linear superposition of Gaussian components: $p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k N(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k)$, where $0 \leq p_k \leq 1$ and $\sum_{k=1}^{K} \pi_k = 1$
- We define a K-dimensional random variable $\boldsymbol{z}$, such that $p(z_k = 1) = \pi_k$
- Another important quantity is $p(z_k = 1|\boldsymbol{x})$, which we call responsibility and denote by $\gamma(z_k)$
- A generalized version of EM is frequently used to estimate parameters of a Gaussian mixture model

EM for Gaussian Mixtures - Algorithm

1. Initialize means μ_k, covariances Σ_k and mixing coefficients π_k, and evaluate the initial value of log-likelihood
2. **E step** Evaluate the responsibilities using current parameter values

$$\gamma(z_{nk}) = \frac{\pi_k N(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j N(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \Sigma_k)}$$

3. **M step** Re-estimate the parameters using current responsibilities

$$\boldsymbol{\mu}_k^{new} = (1/N_k) \sum_{n=1}^{N} \gamma(z_{nk}) \boldsymbol{x}_n$$

$$\Sigma_k^{new} = (1/N_k) \sum_{n=1}^{N} \gamma(z_{nk}) (\boldsymbol{x}_n - \boldsymbol{\mu}_k^{new}) (\boldsymbol{x}_n - \boldsymbol{\mu}_k^{new})^T$$

$$\pi_k^{new} = N_k/N \text{ where, } N_k = \sum_{n=1}^{N} \gamma(z_{nk})$$

4. Evaluate log likelihood

$$\ln p(\boldsymbol{X}|\boldsymbol{\mu}, \Sigma, \boldsymbol{\pi}) = \sum_{n=1}^{N} \left\{ \sum_{k=1}^{K} \pi_k N(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \Sigma_k) \right\}$$

5. Finish if convergence was reached, otherwise go to step 2.

Example: EM for Gaussian Mixtures

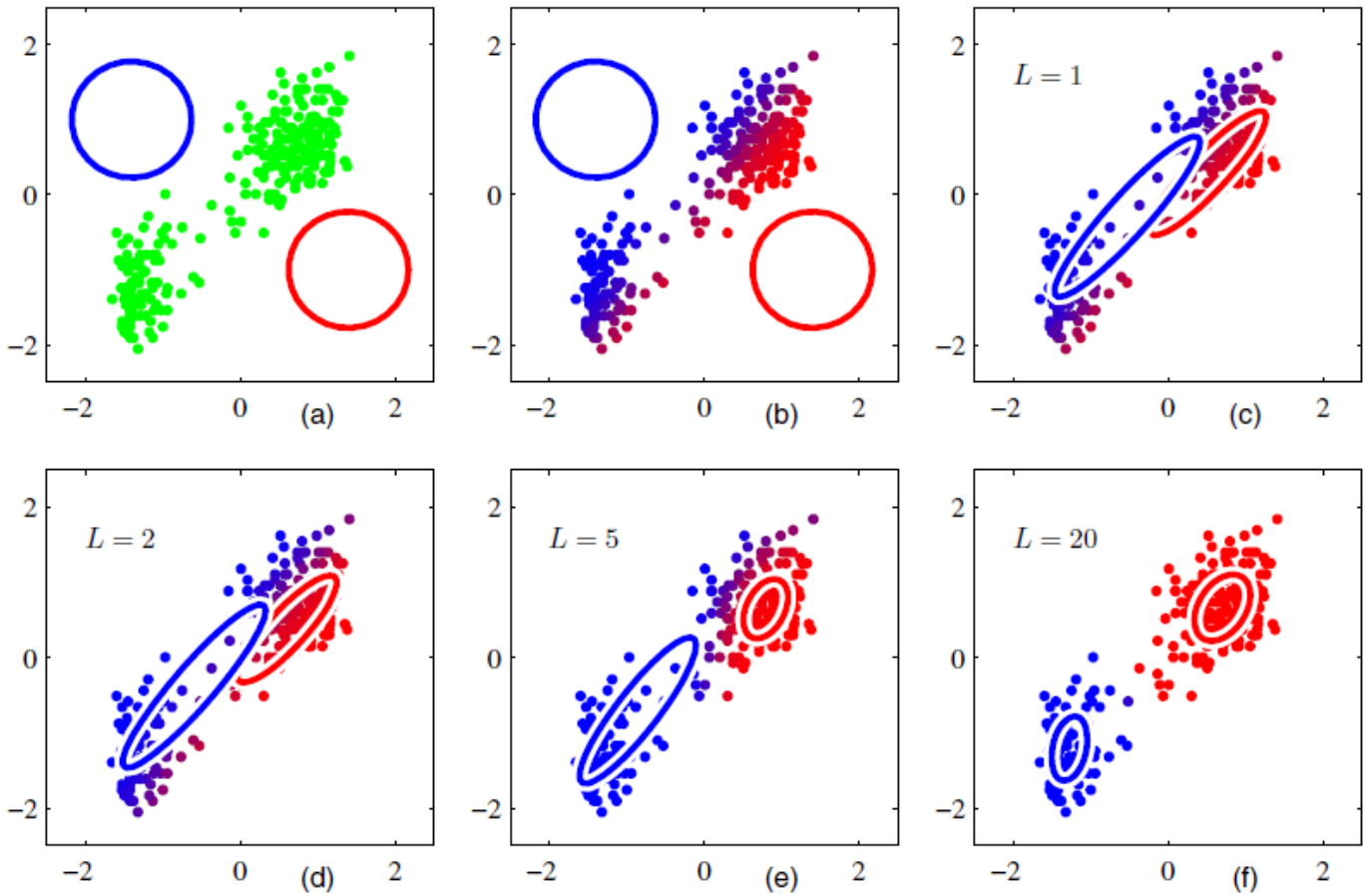


Figure: EM algorithm convergence for a mixture of 3 Gaussian distributions.

Notes on EM

- EM is very useful when optimization of $Q(\cdot;\cdot)$ is simpler than optimization of ML function $l(\cdot)$
- In EM it is guaranteed that the log likelihood of data will increase monotonically

Nonparametric Techniques

MTSC 852, Fall 2015

Sokratis Makrogiannis, Ph.D.

Department of Mathematical Sciences
Delaware State University

smakrogiannis@desu.edu

October 30, 2015

Outline

Introduction

- In the previous chapter we estimated the class-conditional densities assuming parametric forms
- However unimodal parametric forms or products of functions may not approximate closely the underlying density
- It may be advantageous to use nonparametric techniques that make no assumptions about the forms of the underlying densities
- Here we discuss two nonparametric techniques:
  1. estimating class-conditional densities $p(\boldsymbol{x}|\omega_j)$ from sample patterns, for example using Parzen kernels
  2. estimating directly the posterior probabilities $P(\omega_j|\boldsymbol{x})$. One example is nearest neighbor classification

Density Estimation

- Probability P for vector x in region $\mathscr{R}$:

$$P = \int_{\mathscr{R}} p(\boldsymbol{x}')d\boldsymbol{x}'$$

- Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, be n i.i.d. samples following $p(\boldsymbol{x})$.
- The probability for k of them to be inside $\mathscr{R}$ is:

$$P_k = \left(\begin{array}{c} n \\ k \end{array} \right) P^k (1-P)^{n-k}$$

- The expected value and variance are:

$$\mathscr{E}[k] = nP, \quad Var[k] = nP(1-P)$$

$$\mathscr{E}[k/n] = P, \quad Var[k/n] = \frac{P(1-P)}{n}$$

Density Estimation

- Binomial distribution peaks sharply about the mean. In that case $P \simeq k/n$, especially for very large n.
- Assuming that $\boldsymbol{x}$ does not change much in $\mathscr{R}$, it follows that:

$$P = \int_{\mathscr{R}} p(\boldsymbol{x}')d\boldsymbol{x}' \simeq p(\boldsymbol{x})V,$$

where
V: volume of $\mathscr{R}$
$\boldsymbol{x}$: point in $\mathscr{R}$

- We combine previous results to arrive at:

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

Density Estimation

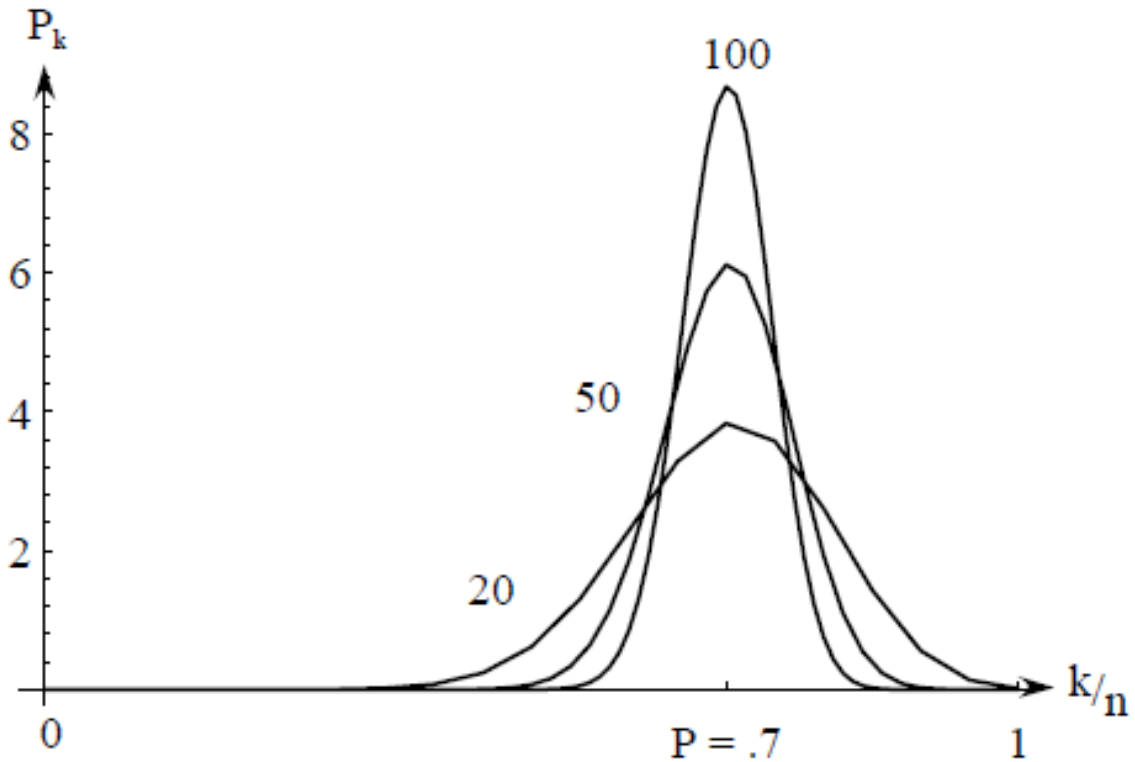


Figure: Binomial distribution convergence. The relative probability P_k peaks more sharply at the true probability as n increases. For $n \to \infty$ the estimate will yield the true probability $P = 0.7$

Density Estimation Considerations

- If we fix V and increase n, then P estimate becomes more accurate, but $p(\boldsymbol{x})$ estimate becomes an average over V:

$$\frac{P}{V} = \frac{\int_{\mathscr{R}} p(\boldsymbol{x}')d\boldsymbol{x}'}{\int_{\mathscr{R}} dx'}$$

- If we decrease V for fixed n to estimate $p(\boldsymbol{x})$, then number of samples $k \to 0$, hence our estimate becomes useless
- So, we cannot reduce V very much in practice

Density Estimation Considerations

- In theory, assume that we have an unlimited number of samples
- We define $\mathscr{R}_k$ regions, where $k = 1, \ldots$ with n increasing with k. Let V_n, k_n, $p_n(\boldsymbol{x})$, be the volume of $\mathscr{R}_n$, the number of samples in $\mathscr{R}_n$, and the density estimate in $\mathscr{R}_n$ respectively. It then follows that:

$$p_n(\boldsymbol{x}) = \frac{k_n/n}{V_n}$$

- To reach an estimate of $p(\boldsymbol{x})$ the following must be satisfied:

$$\lim_{n\to\infty} V_n = 0, \quad \lim_{n\to\infty} k_n = \infty, \quad \lim_{n\to\infty} k_n/n = 0$$

- These conditions ensure that we can estimate $p(\boldsymbol{x})$ accurately, that the frequency ratio will converge to P, and that $p_n(\boldsymbol{x})$ will converge in general

Density Estimation Approaches

We can attempt to satisfy these conditions by the following approaches:

1. Shrink $\mathscr{R}_n$ by defining V_n as a function of n
   - Determine k from the data
   - We must ensure that $p_n(\boldsymbol{x})$ converges to $p(\boldsymbol{x})$
   - Parzen windows use this principle
2. Define k_n as a function of n
   - Then V_n will have to grow till it encloses k_n neighbors of $\boldsymbol{x}$
   - This is the k_n nearest neighbor estimation

Density Estimation Approaches

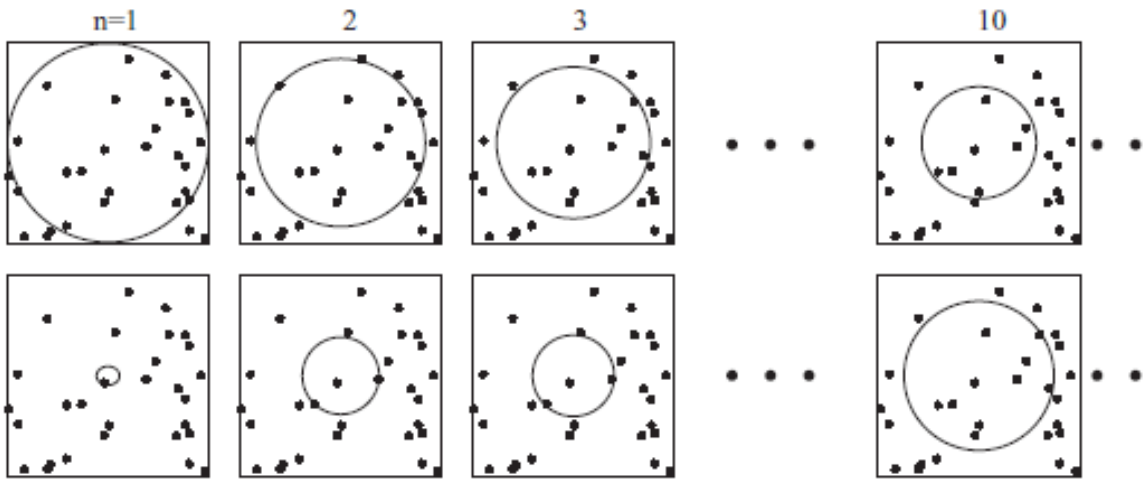


Figure: Two density estimation approaches: reduce V_n to estimate $p_n(\boldsymbol{x})$ more accurately (top), or increase V_n to increase k_n to estimate P_k more accurately (bottom)

Parzen Windows

- The Parzen window method defines a window that may be a function of the number of data points
- More specifically, $\mathscr{R}_n$ is a d-dimensional hypercube
- The volume of the hypecube is:

$$V_n = h_n^d$$

where h_n: edge length of cube

- To yield the number of points in $\mathscr{R}_n$ denoted by k_n we use a window function:

$$\phi(\boldsymbol{u}) = \begin{cases} 1 & |u_j| \leq 1/2 \quad j = 1, \ldots, d \\ 0 & \text{otherwise} \end{cases}$$

Parzen Windows

- We defined the window function as:

$$\phi(\boldsymbol{u}) = \begin{cases} 1 & |u_j| \leq 1/2 \quad j = 1, \ldots, d \\ 0 & \text{otherwise} \end{cases}$$

- Then, the number of points inside the hypercube centered at $\boldsymbol{x}$ is given by:

$$k_n = \sum_{i=1}^{n} \phi\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h_n}\right)$$

- From density estimation we have that:

$$p_n(\boldsymbol{x}) = \frac{k_n/n}{V_n}$$

- By substitution it follows that: $p_n(\boldsymbol{x}) = (1/n)\sum_{i=1}^{n} \frac{1}{V_n}\phi\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right)$

Parzen Windows

- Before, we arrived at:

$$p_n(\boldsymbol{x}) = (1/n) \sum_{i=1}^{n} \frac{1}{V_n} \phi \left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h_n} \right)$$

- This is a general form for estimating density
- This operation can also be considered as interpolation
- We can choose different function types for $\phi(\cdot)$ to handle discontinuities that may be caused by the original window function

Parzen Windows

- Next, we require that $p_n(\boldsymbol{x})$ be a density function
- Hence, we need $\phi(\boldsymbol{x})$ to be a density function with conditions

$$\phi(\boldsymbol{x}) \geq 0$$

and

$$\int \phi(\boldsymbol{u}) d\boldsymbol{u} = 1$$

- If in addition $V_n = h_n^d$, then $p_n(\boldsymbol{x})$ is a density function too

Effect of Window Width h_n

- From previous analysis we observed that the size of $V_n = h_n^d$ can affect density estimation
- Now we focus our interest on h_n
- First, we define $\delta_n(\boldsymbol{x})$ as: $\delta_n(\boldsymbol{x}) = (1/n)\frac{1}{V_n}\phi\left(\frac{\boldsymbol{x}}{h_n}\right)$
- $p_n(\boldsymbol{x})$ becomes: $p_n(\boldsymbol{x}) = (1/n)\sum_{i=1}^{n}\delta_n(\boldsymbol{x} - \boldsymbol{x}_i)$

Effect of Window Width h_n

- If h_n is too large, then
  - $\delta_n(\boldsymbol{x}-\boldsymbol{x}_i)$ becomes a slowly varying function with a low peak
  - $p_n(\boldsymbol{x})$ becomes a smooth and "out-of-focus" estimate of $p(\boldsymbol{x})$
- If h_n is too small, then
  - $\delta_n(\boldsymbol{x}-\boldsymbol{x}_i)$ has a sharper peak and changes faster with distance
  - $p_n(\boldsymbol{x})$ becomes a sum of sharp kernels centered at the training samples affected by noisy samples
- We also note that:
  $\int \delta_n(\boldsymbol{x}-\boldsymbol{x}_i)d\boldsymbol{x} = \int (1/n)\frac{1}{V_n}\phi\left(\frac{\boldsymbol{x}}{h_n}\right)d\boldsymbol{x} = \int \phi(\boldsymbol{u})d\boldsymbol{u} = 1$
- Hence, the distribution is normalized

Effect of Window Width h_n

- If h_n is too large, then the density estimate will be less accurate
- If h_n is too small, then he density estimate will be sensitive to statistical variability
- In real world problems, we must choose the kernel size as a trade-off between the two weaknesses
- In an ideal world with unlimited training samples we could reduce the kernel size and still get an accurate density $p_n(\boldsymbol{x}) = p(\boldsymbol{x})$

Convergence Definition

- We consider $p_n(\boldsymbol{x})$ to be a random variable because it is a function of n random samples $\boldsymbol{x}_i$, $i = 1, \ldots, n$
- We denote the mean and variance of $p_n(\boldsymbol{x})$ by $\bar{p}_n(\boldsymbol{x})$ and $\sigma_n^2(\boldsymbol{x})$
- Then $p_n(\boldsymbol{x})$ will converge to $p(\boldsymbol{x})$, if:

$$\lim_{n \to \infty} \bar{p}_n(\boldsymbol{x}) = p(\boldsymbol{x})$$

and

$$\lim_{n \to \infty} \sigma_n^2(\boldsymbol{x}) = 0$$

Convergence Conditions

- To prove convergence, we place conditions on $p(\boldsymbol{x})$, $\phi(\boldsymbol{u})$, and h_n
- Previous conditions are that $p(\cdot)$ is continuous at $\boldsymbol{x}$, and that $\phi(\cdot)$ is a density function
- We will show that the following conditions have to be met as well:

$$\sup_{\boldsymbol{u}} \phi(\boldsymbol{u}) < \infty, \quad \lim_{\|u\| \to \infty} \phi(\boldsymbol{u}) \prod_{i=1}^{d} u_i = 0$$

$$\lim_{n \to \infty} V_n = 0, \quad \lim_{n \to \infty} nV_n = \infty$$

- These conditions ensure that $\phi(\boldsymbol{u})$ is bounded and that V_n must approach zero but at a slower rate that $1/n$

Convergence of the Mean

- First, we compute $\bar{p}_n(\boldsymbol{x})$:

$$\bar{p}_n(\boldsymbol{x}) = \mathscr{E}[p_n(\boldsymbol{x})] = (1/n)\sum_{i=1}^{n} \mathscr{E}\left[\frac{1}{V_n}\phi\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right)\right]$$

$$= \int \frac{1}{V_n}\phi\left(\frac{\boldsymbol{x}-\boldsymbol{v}}{h_n}\right) p(\boldsymbol{v})d\boldsymbol{v} = \int \delta_n(\boldsymbol{x}-\boldsymbol{v})p(\boldsymbol{v})d\boldsymbol{v}$$

- We observe that $\bar{p}_n(\boldsymbol{x})$ is the result of convolving the window function with the density function, therefore it is a blurred version of the unknown density
- However, when $V_n \to 0$, $\delta_n(\boldsymbol{x}-\boldsymbol{v})$ becomes a delta function at $\boldsymbol{x}$.
- Hence, if $\boldsymbol{p}$ is continuous at $\boldsymbol{x}$, then $\lim_{n\to\infty} \bar{p}_n(\boldsymbol{x}) = p(\boldsymbol{x})$

Convergence of the Variance

- To avoid a noisy estimate, we need to ensure the convergence of variance:

$$\sigma_n^2(\boldsymbol{x}) = \sum_{i=1}^{n} \mathscr{E}\left[\left(\frac{1}{nV_n}\phi\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right) - (1/n)\bar{p}_n(\boldsymbol{x})\right)^2\right]$$

$$= n\mathscr{E}\left[\frac{1}{n^2V_n^2}\phi^2\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right)\right] - (1/n)\bar{p}_n^2(\boldsymbol{x})$$

$$= \frac{1}{nV_n}\int \frac{1}{V_n}\phi^2\left(\frac{\boldsymbol{x}-\boldsymbol{v}}{h_n}\right)p(\boldsymbol{v})d\boldsymbol{v} - (1/n)\bar{p}_n^2(\boldsymbol{x})$$

- If we drop the second term, bound ϕ, and use above expression for $\bar{p}_n(\boldsymbol{x})$ it follows that:

$$\sigma_n^2(\boldsymbol{x}) \leq \frac{\sup\left(\phi(\cdot)\right)\bar{p}_n(\boldsymbol{x})}{nV_n}$$

Convergence of the Variance

- Previously we arrived at:

$$\sigma_n^2(\boldsymbol{x}) \leq \frac{\sup(\phi(\cdot))\,\bar{p}_n(\boldsymbol{x})}{nV_n}$$

- We need to start with a large V_n to reach a small variance
- Still, because the numerator is finite w.r.t n, we can still let $V_n \to 0$ as long as $nV_n \to \infty$ to obtain zero variance
- This means that V_n can be $V_1/\sqrt{n}$, for example

Gaussian Kernel Example

- Suppose that the true density $p(\boldsymbol{x})$ is univariate normal, with zero mean, and unit variance
- Suppose we use a Gaussian kernel for density estimation given by:

$$\phi(u) = \frac{1}{\sqrt{2\pi}} e^{-u^2/2}$$

- The density estimate at x is:

$$p_n(x) = (1/n) \sum_{i=1}^{n} \frac{1}{h_n} \phi\left(\frac{x - x_i}{h_n}\right)$$

where $h_n = h_1/\sqrt{n}$

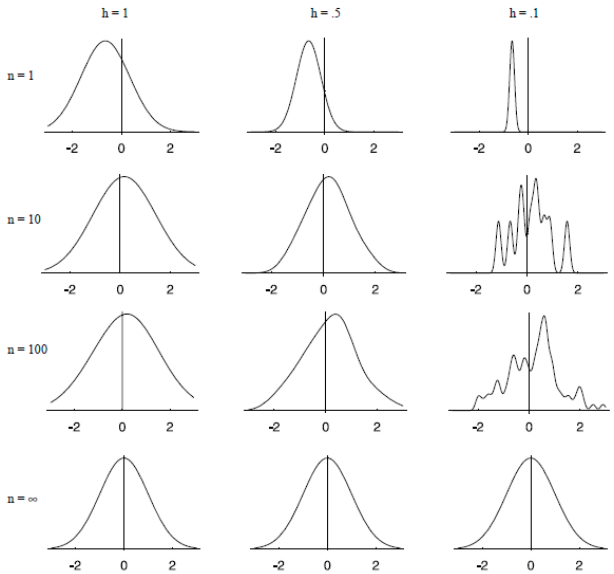


Figure: Parzen kernel density estimation for a univariate normal distribution versus the number of samples and window width. The contribution of each point to the density is more visible for smaller window widths. Larger n improves density estimation

Gaussian Kernel Example

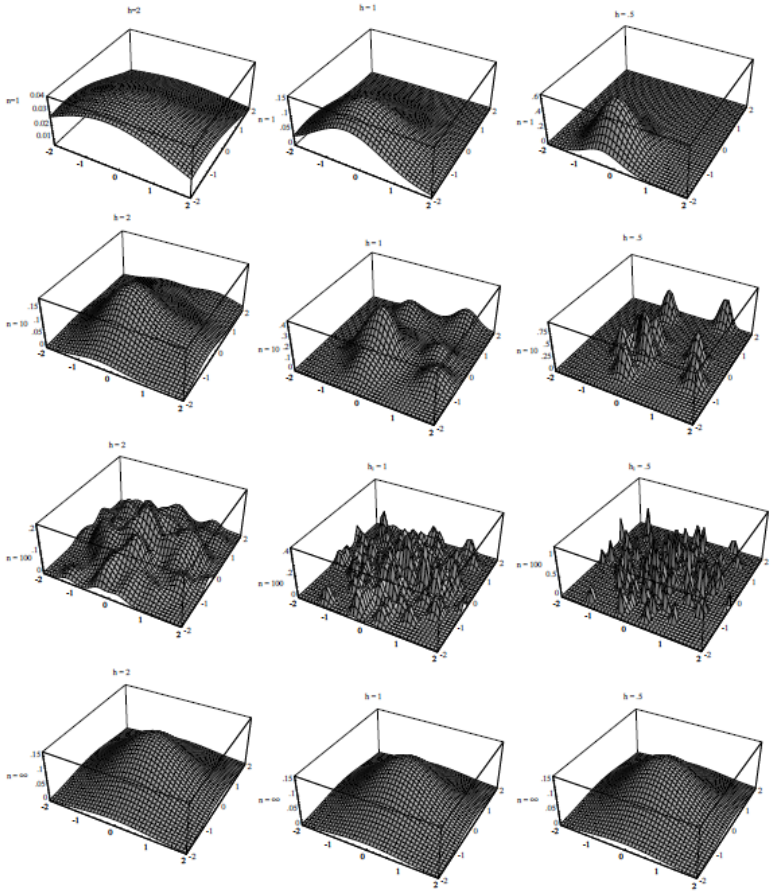


Figure: Parzen kernel density estimation for a bivariate normal distribution versus the number of samples and window width. Smaller window width produces "noisier" estimates for fixed n

Gaussian Kernel Example

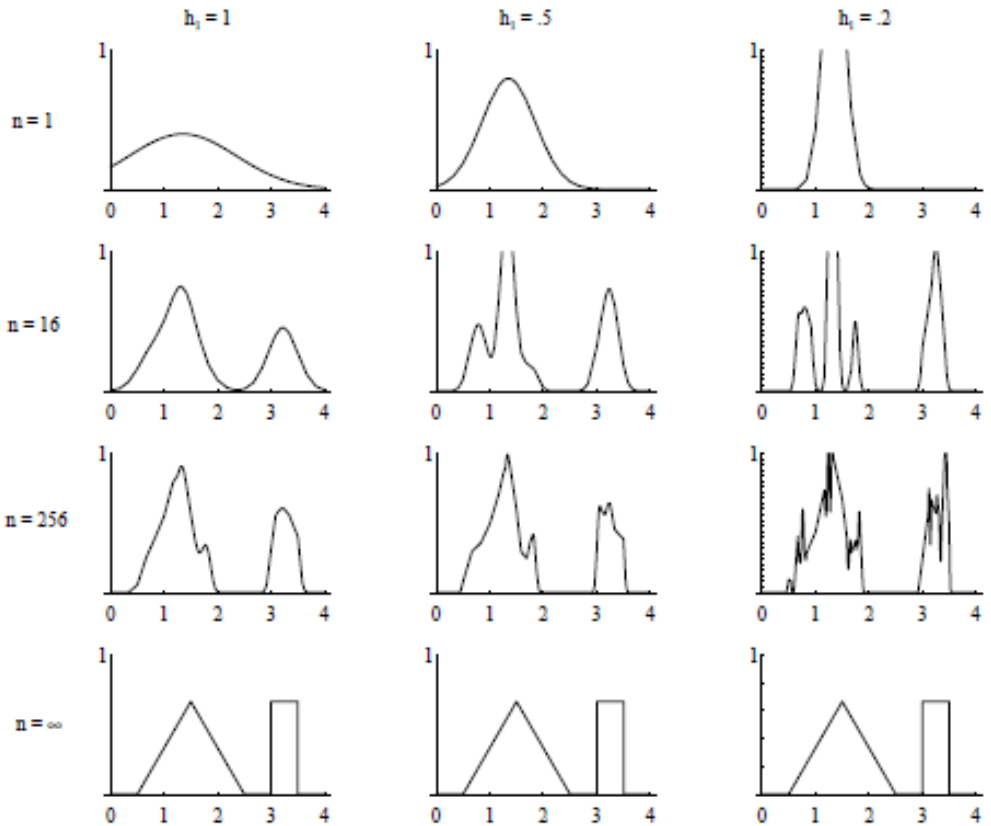


Figure: Parzen kernel density estimation for a mixture of a uniform and a triangular distribution. Observe that more samples improve estimation

Parzen Kernel-based Classification

- In Parzen kernel-based classification we estimate the class-conditional density at each test point and make a decision using Maximum-a-Posteriori rule
- In this classifier we can reduce the training error as much as we wish, but we may cause overfitting
- Gaussian windows are reasonable choices but it may take some experimentation to find the window size

Parzen Kernel-based Classification

- Strengths:
  * no assumption about the density functions – we use the same procedure
- Weaknesses:
  * requirement for a large number of samples
  * computationally demanding because all training samples are used to estimate densities each time
  * when the dimensionality grows the demand for a large number of samples grows exponentially (curse of dimensionality)

Parzen Kernel-based Classification

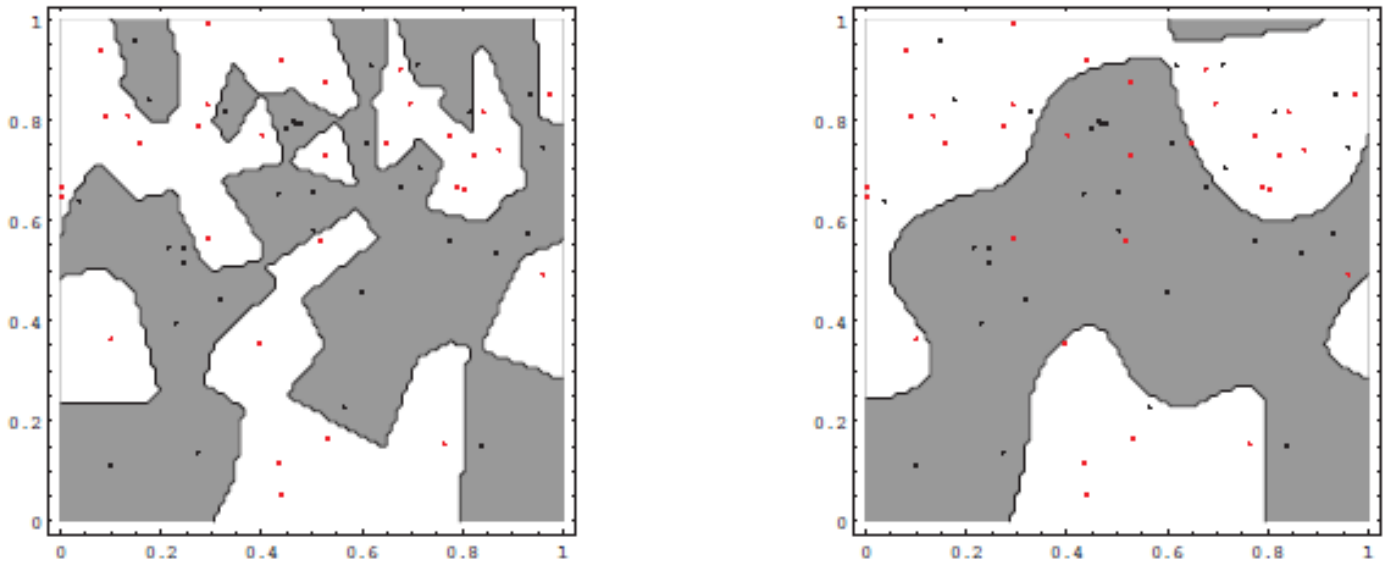


Figure: Decision boundaries versus window width h. Smaller h produces more complicated decision boundaries (left column) than larger h (right column)

Probabilistic Neural Network (PNN) Example

- We will show how Parzen density-based classifier can be implemented as a neural network using PNN
- Suppose we have n patterns, in a d-dimensional space divided into c classes
- PNN will have:
  - an input layer with d input units
  - an intermediate layer with n pattern units. Each input unit is connected to all n pattern units
  - a category layer with c category units. Each pattern unit is connected to only one category unit
- Connections from input to pattern units carry weights $\boldsymbol{w}$, which need to be learned

PNN Topology

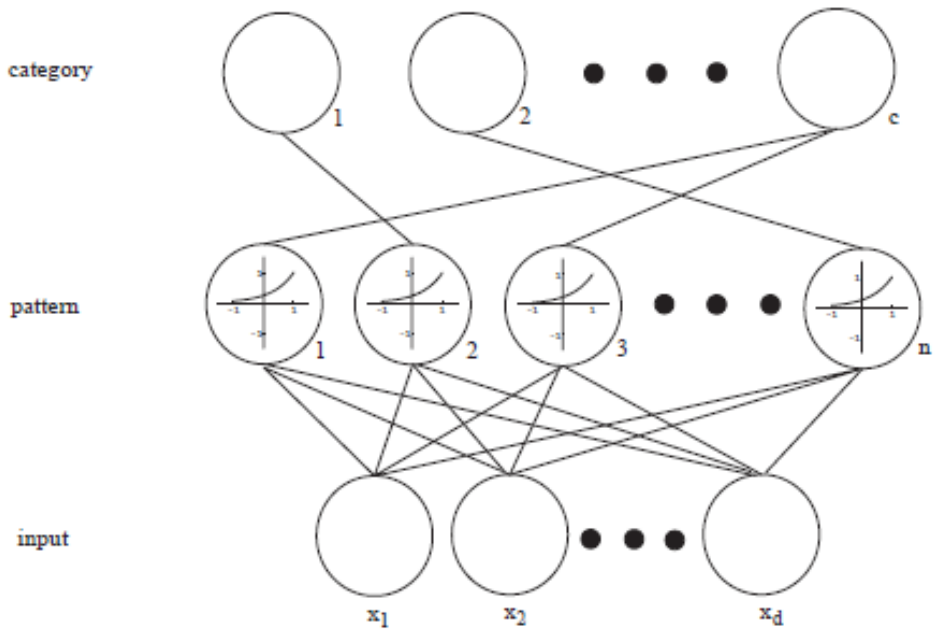


Figure: PNN will have: an input layer with d input units; an intermediate layer with n pattern units. Each input unit is connected to all n pattern units; a category layer with c category units. Each pattern unit is connected to only one category unit

PNN Training

- For each train pattern $\boldsymbol{x}$, we normalize the weight s.t. $\sum_{i=1}^{d} x_i^2 = 1$
- We link the input unit and the corresponding pattern unit with a link that has a weight $\boldsymbol{w}_j = \boldsymbol{x}_j, j = 1, \ldots, n$
- We denote the components of the jth pattern by $x_{jk}, k = 1, \ldots, d$

1 **begin initialize** $j = 0, n = \#\text{patterns}$
2 **do** $j \leftarrow j + 1$
3 **normalize** : $x_{jk} \leftarrow x_{jk} / \left(\sum_{i}^{d} x_{ji}^2 \right)^{1/2}$
4 **train** : $w_{jk} \leftarrow x_{jk}$
5 **if** $\mathbf{x} \in \omega_i$ **then** $a_{ic} \leftarrow 1$
6 **until** $j = n$
7 **end**

Figure: PNN training algorithm

PNN Classification

- We normalize the test pattern $\boldsymbol{x}$
- We compute the net activation by inner product: $net_k = \boldsymbol{w}_k^T \boldsymbol{x}$
- Each output unit sums the contributions from all pattern units using the nonlinear function $e^{\frac{net_k - 1}{\sigma^2}}$, σ: user-defined parameter
- We use the window function to define activation function:

$$\phi\left(\frac{\boldsymbol{x} - \boldsymbol{w}_k}{h_n}\right) \sim e^{-(\boldsymbol{x}-\boldsymbol{w}_k)^T(\boldsymbol{x}-\boldsymbol{w}_k)/2\sigma^2} = e^{\frac{net_k - 1}{\sigma^2}}$$

- Each pattern unit contributes to its associated category unit a signal equal to type of probability of association with a training point
- The sum of estimates gives the discriminant function $g_i(\boldsymbol{x})$. This is a Parzen window estimate for the distribution
- We apply a maximum likelihood rule to assign the pattern to a class

PNN Testing

Algorithm 2 (PNN classification)

1 begin initialize $k = 0, \mathbf{x} =$ test pattern
2 do $k \leftarrow k + 1$
3 $z_k \leftarrow \mathbf{w}_k^t \mathbf{x}$
4 if $a_{kc} = 1$ then $g_c \leftarrow g_c + \exp[(z_k - 1)/\sigma^2]$
5 until $k = n$
6 return $class \leftarrow \arg\max_i g_i(\mathbf{x})$
7 end

Figure: PNN Testing algorithm

Nonparametric Techniques

MTSC 852, Fall 2015

Sokratis Makrogiannis, Ph.D.

Department of Mathematical Sciences
Delaware State University

smakrogiannis@desu.edu

October 29, 2015

Outline

Parzen Windows

- The Parzen window method defines a window that may be a function of the number of data points
- More specifically, $\mathscr{R}_n$ is a d-dimensional hypercube
- The volume of the hypecube is:

$$V_n = h_n^d$$

where h_n: edge length of cube

- To yield the number of points in $\mathscr{R}_n$ denoted by k_n we use a window function:

$$\phi(\boldsymbol{u}) = \begin{cases} 1 & |u_j| \leq 1/2 \quad j = 1, \ldots, d \\ 0 & \text{otherwise} \end{cases}$$

Parzen Windows

- We defined the window function as:

$$\phi(\boldsymbol{u}) = \begin{cases} 1 & |u_j| \leq 1/2 \quad j = 1, \ldots, d \\ 0 & \text{otherwise} \end{cases}$$

- Then, the number of points inside the hypercube centered at $\boldsymbol{x}$ is given by:

$$k_n = \sum_{i=1}^{n} \phi\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h_n}\right)$$

- From density estimation we have that:

$$p_n(\boldsymbol{x}) = \frac{k_n/n}{V_n}$$

- By substitution it follows that: $p_n(\boldsymbol{x}) = (1/n)\sum_{i=1}^{n} \frac{1}{V_n}\phi\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right)$

Parzen Windows

- Before, we arrived at:

$$p_n(\boldsymbol{x}) = (1/n) \sum_{i=1}^{n} \frac{1}{V_n} \phi \left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h_n} \right)$$

- This is a general form for estimating density
- This operation can also be considered as interpolation
- We can choose different function types for $\phi(\cdot)$ to handle discontinuities that may be caused by the original window function

Parzen Windows

- Next, we require that $p_n(\boldsymbol{x})$ be a density function
- Hence, we need $\phi(\boldsymbol{x})$ to be a density function with conditions

$$\phi(\boldsymbol{x}) \geq 0$$

and

$$\int \phi(\boldsymbol{u}) d\boldsymbol{u} = 1$$

- If in addition $V_n = h_n^d$, then $p_n(\boldsymbol{x})$ is a density function too

Effect of Window Width h_n

- From previous analysis we observed that the size of $V_n = h_n^d$ can affect density estimation
- Now we focus our interest on h_n
- First, we define $\delta_n(\boldsymbol{x})$ as: $\delta_n(\boldsymbol{x}) = (1/n)\frac{1}{V_n}\phi\left(\frac{\boldsymbol{x}}{h_n}\right)$
- $p_n(\boldsymbol{x})$ becomes: $p_n(\boldsymbol{x}) = (1/n)\sum_{i=1}^{n}\delta_n(\boldsymbol{x} - \boldsymbol{x}_i)$

Effect of Window Width h_n

- If h_n is too large, then
  - $\delta_n(\boldsymbol{x}-\boldsymbol{x}_i)$ becomes a slowly varying function with a low peak
  - $p_n(\boldsymbol{x})$ becomes a smooth and "out-of-focus" estimate of $p(\boldsymbol{x})$
- If h_n is too small, then
  - $\delta_n(\boldsymbol{x}-\boldsymbol{x}_i)$ has a sharper peak and changes faster with distance
  - $p_n(\boldsymbol{x})$ becomes a sum of sharp kernels centered at the training samples affected by noisy samples
- We also note that:
  $\int \delta_n(\boldsymbol{x}-\boldsymbol{x}_i)d\boldsymbol{x} = \int (1/n)\frac{1}{V_n}\phi\left(\frac{\boldsymbol{x}}{h_n}\right)d\boldsymbol{x} = \int \phi(\boldsymbol{u})d\boldsymbol{u} = 1$
- Hence, the distribution is normalized

Effect of Window Width h_n

- If h_n is too large, then the density estimate will be less accurate
- If h_n is too small, then he density estimate will be sensitive to statistical variability
- In real world problems, we must choose the kernel size as a trade-off between the two weaknesses
- In an ideal world with unlimited training samples we could reduce the kernel size and still get an accurate density $p_n(\boldsymbol{x}) = p(\boldsymbol{x})$

Convergence Definition

- We consider $p_n(\boldsymbol{x})$ to be a random variable because it is a function of n random samples $\boldsymbol{x}_i$, $i = 1, \ldots, n$
- We denote the mean and variance of $p_n(\boldsymbol{x})$ by $\bar{p}_n(\boldsymbol{x})$ and $\sigma_n^2(\boldsymbol{x})$
- Then $p_n(\boldsymbol{x})$ will converge to $p(\boldsymbol{x})$, if:

$$\lim_{n \to \infty} \bar{p}_n(\boldsymbol{x}) = p(\boldsymbol{x})$$

and

$$\lim_{n \to \infty} \sigma_n^2(\boldsymbol{x}) = 0$$

Convergence Conditions

- To prove convergence, we place conditions on $p(\boldsymbol{x})$, $\phi(\boldsymbol{u})$, and h_n
- Previous conditions are that $p(\cdot)$ is continuous at $\boldsymbol{x}$, and that $\phi(\cdot)$ is a density function
- We will show that the following conditions have to be met as well:

$$\sup_{\boldsymbol{u}} \phi(\boldsymbol{u}) < \infty, \quad \lim_{\|u\| \to \infty} \phi(\boldsymbol{u}) \prod_{i=1}^{d} u_i = 0$$

$$\lim_{n \to \infty} V_n = 0, \quad \lim_{n \to \infty} nV_n = \infty$$

- These conditions ensure that $\phi(\boldsymbol{u})$ is bounded and that V_n must approach zero but at a slower rate that $1/n$

Convergence of the Mean

- First, we compute $\bar{p}_n(\boldsymbol{x})$:

$$\bar{p}_n(\boldsymbol{x}) = \mathscr{E}[p_n(\boldsymbol{x})] = (1/n)\sum_{i=1}^{n} \mathscr{E}\left[\frac{1}{V_n}\phi\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right)\right]$$

$$= \int \frac{1}{V_n}\phi\left(\frac{\boldsymbol{x}-\boldsymbol{v}}{h_n}\right) p(\boldsymbol{v})d\boldsymbol{v} = \int \delta_n(\boldsymbol{x}-\boldsymbol{v})p(\boldsymbol{v})d\boldsymbol{v}$$

- We observe that $\bar{p}_n(\boldsymbol{x})$ is the result of convolving the window function with the density function, therefore it is a blurred version of the unknown density
- However, when $V_n \to 0$, $\delta_n(\boldsymbol{x}-\boldsymbol{v})$ becomes a delta function at $\boldsymbol{x}$.
- Hence, if $\boldsymbol{p}$ is continuous at $\boldsymbol{x}$, then $\lim_{n\to\infty} \bar{p}_n(\boldsymbol{x}) = p(\boldsymbol{x})$

Convergence of the Variance

- To avoid a noisy estimate, we need to ensure the convergence of variance:

$$\sigma_n^2(\boldsymbol{x}) = \sum_{i=1}^{n} \mathscr{E}\left[\left(\frac{1}{nV_n}\phi\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right) - (1/n)\bar{p}_n(\boldsymbol{x})\right)^2\right]$$

$$= n\mathscr{E}\left[\frac{1}{n^2V_n^2}\phi^2\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right)\right] - (1/n)\bar{p}_n^2(\boldsymbol{x})$$

$$= \frac{1}{nV_n}\int \frac{1}{V_n}\phi^2\left(\frac{\boldsymbol{x}-\boldsymbol{v}}{h_n}\right)p(\boldsymbol{v})d\boldsymbol{v} - (1/n)\bar{p}_n^2(\boldsymbol{x})$$

- If we drop the second term, bound ϕ, and use above expression for $\bar{p}_n(\boldsymbol{x})$ it follows that:

$$\sigma_n^2(\boldsymbol{x}) \leq \frac{\sup\left(\phi(\cdot)\right)\bar{p}_n(\boldsymbol{x})}{nV_n}$$

Convergence of the Variance

- Previously we arrived at:

$$\sigma_n^2(\boldsymbol{x}) \leq \frac{\sup(\phi(\cdot))\,\bar{p}_n(\boldsymbol{x})}{nV_n}$$

- We need to start with a large V_n to reach a small variance
- Still, because the numerator is finite w.r.t n, we can still let $V_n \to 0$ as long as $nV_n \to \infty$ to obtain zero variance
- This means that V_n can be $V_1/\sqrt{n}$, for example

Gaussian Kernel Example

- Suppose that the true density $p(\boldsymbol{x})$ is univariate normal, with zero mean, and unit variance
- Suppose we use a Gaussian kernel for density estimation given by:

$$\phi(u) = \frac{1}{\sqrt{2\pi}} e^{-u^2/2}$$

- The density estimate at x is:

$$p_n(x) = (1/n) \sum_{i=1}^{n} \frac{1}{h_n} \phi\left(\frac{x - x_i}{h_n}\right)$$

where $h_n = h_1/\sqrt{n}$

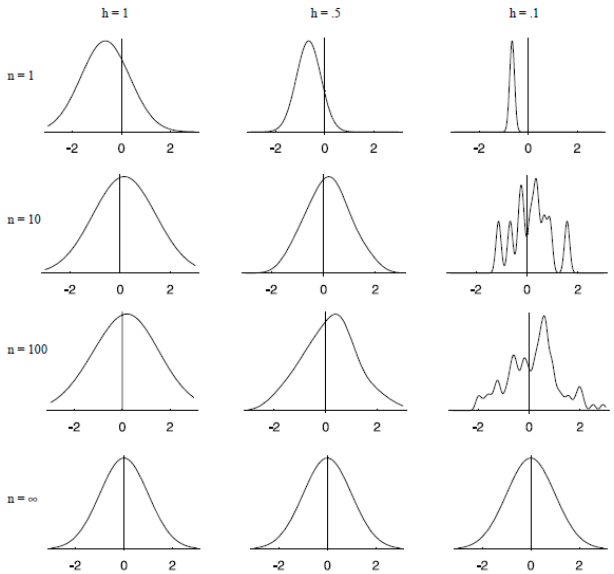


Figure: Parzen kernel density estimation for a univariate normal distribution versus the number of samples and window width. The contribution of each point to the density is more visible for smaller window widths. Larger n improves density estimation

Gaussian Kernel Example

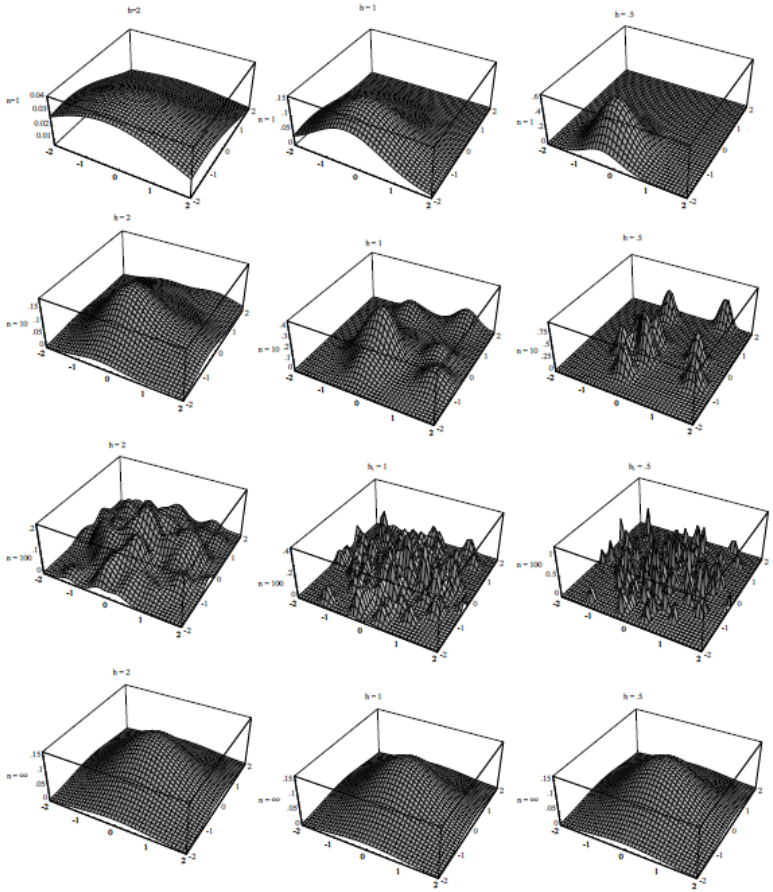


Figure: Parzen kernel density estimation for a bivariate normal distribution versus the number of samples and window width. Smaller window width produces "noisier" estimates for fixed n

Gaussian Kernel Example

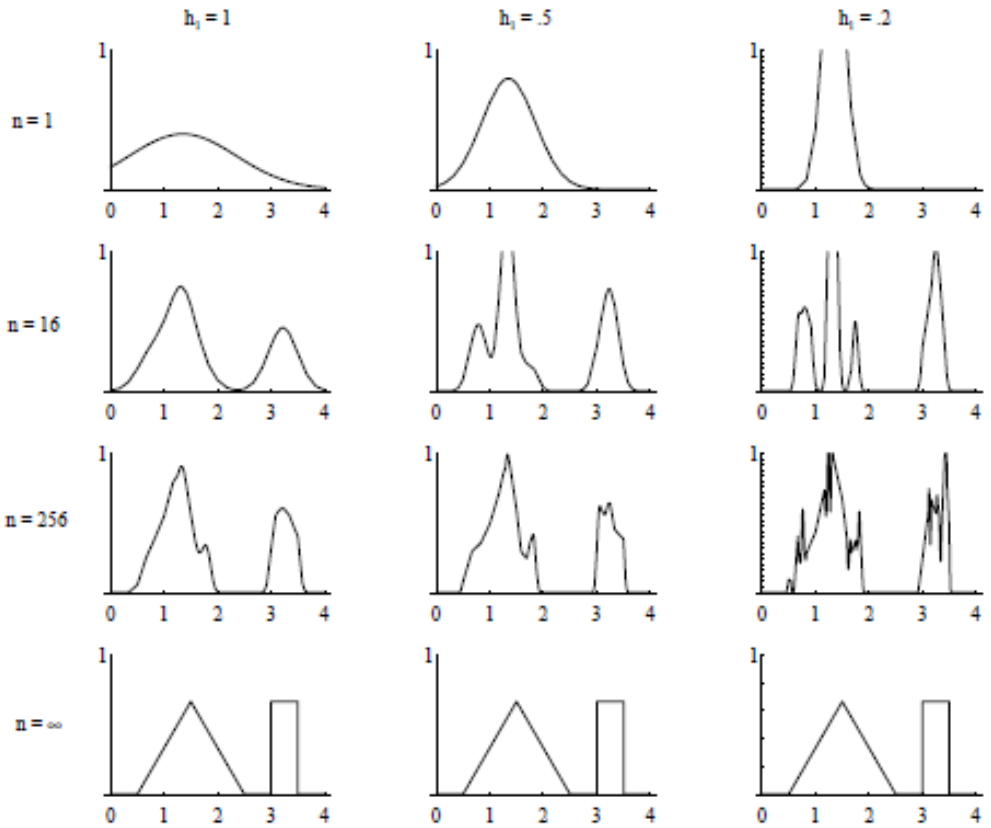


Figure: Parzen kernel density estimation for a mixture of a uniform and a triangular distribution. Observe that more samples improve estimation

Parzen Kernel-based Classification

- In Parzen kernel-based classification we estimate the class-conditional density at each test point and make a decision using Maximum-a-Posteriori rule
- In this classifier we can reduce the training error as much as we wish, but we may cause overfitting
- Gaussian windows are reasonable choices but it may take some experimentation to find the window size

Parzen Kernel-based Classification

- Strengths:
  * no assumption about the density functions – we use the same procedure
- Weaknesses:
  * requirement for a large number of samples
  * computationally demanding because all training samples are used to estimate densities each time
  * when the dimensionality grows the demand for a large number of samples grows exponentially (curse of dimensionality)

Parzen Kernel-based Classification

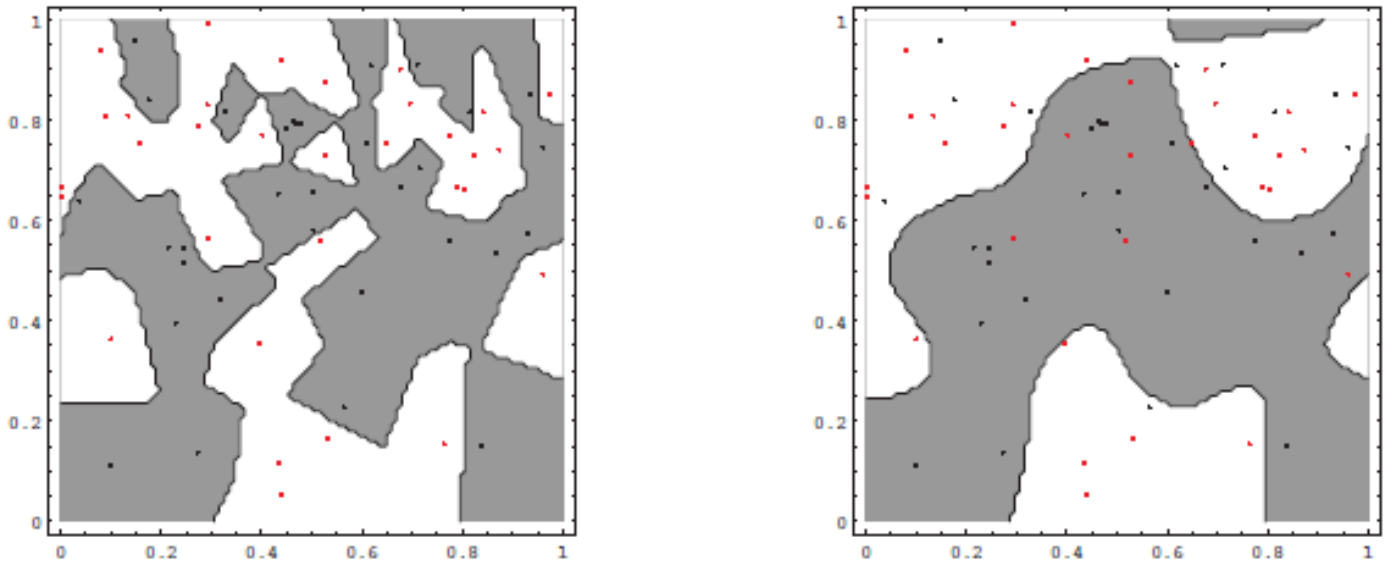


Figure: Decision boundaries versus window width h. Smaller h produces more complicated decision boundaries (left column) than larger h (right column)

Probabilistic Neural Network (PNN) Example

- We will show how Parzen density-based classifier can be implemented as a neural network using PNN
- Suppose we have n patterns, in a d-dimensional space divided into c classes
- PNN will have:
  - an input layer with d input units
  - an intermediate layer with n pattern units. Each input unit is connected to all n pattern units
  - a category layer with c category units. Each pattern unit is connected to only one category unit
- Connections from input to pattern units carry weights $\boldsymbol{w}$, which need to be learned

PNN Topology

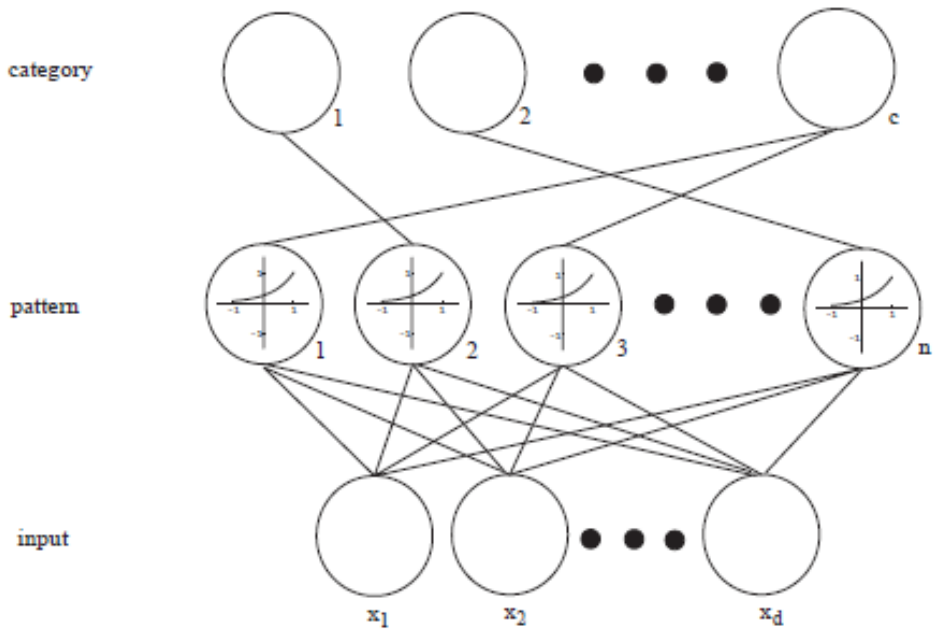


Figure: PNN will have: an input layer with d input units; an intermediate layer with n pattern units. Each input unit is connected to all n pattern units; a category layer with c category units. Each pattern unit is connected to only one category unit

PNN Training

- For each train pattern $\boldsymbol{x}$, we normalize the weight s.t. $\sum_{i=1}^{d} x_i^2 = 1$
- We link the input unit and the corresponding pattern unit with a link that has a weight $\boldsymbol{w}_j = \boldsymbol{x}_j, j = 1, \ldots, n$
- We denote the components of the jth pattern by $x_{jk}, k = 1, \ldots, d$

1 **begin initialize** $j = 0, n = \#\text{patterns}$
2 **do** $j \leftarrow j + 1$
3 $\text{normalize} : x_{jk} \leftarrow x_{jk} / \left(\sum_{i}^{d} x_{ji}^2 \right)^{1/2}$
4 $\text{train} : w_{jk} \leftarrow x_{jk}$
5 **if** $\mathbf{x} \in \omega_i$ **then** $a_{ic} \leftarrow 1$
6 **until** $j = n$
7 **end**

Figure: PNN training algorithm

PNN Classification

- We normalize the test pattern $\boldsymbol{x}$
- We compute the net activation by inner product: $net_k = \boldsymbol{w}_k^T \boldsymbol{x}$
- Each output unit sums the contributions from all pattern units using the nonlinear function $e^{\frac{net_k - 1}{\sigma^2}}$, σ: user-defined parameter
- We use the window function to define activation function:

$$\phi\left(\frac{\boldsymbol{x} - \boldsymbol{w}_k}{h_n}\right) \sim e^{-(\boldsymbol{x}-\boldsymbol{w}_k)^T(\boldsymbol{x}-\boldsymbol{w}_k)/2\sigma^2} = e^{\frac{net_k - 1}{\sigma^2}}$$

- Each pattern unit contributes to its associated category unit a signal equal to type of probability of association with a training point
- The sum of estimates gives the discriminant function $g_i(\boldsymbol{x})$. This is a Parzen window estimate for the distribution
- We apply a maximum likelihood rule to assign the pattern to a class

PNN Testing

Algorithm 2 (PNN classification)

1 begin initialize $k = 0, \mathbf{x} =$ test pattern
2 do $k \leftarrow k + 1$
3 $z_k \leftarrow \mathbf{w}_k^t \mathbf{x}$
4 if $a_{kc} = 1$ then $g_c \leftarrow g_c + \exp[(z_k - 1)/\sigma^2]$
5 until $k = n$
6 return $class \leftarrow \arg\max_i g_i(\mathbf{x})$
7 end

Figure: PNN Testing algorithm

Nonparametric Techniques

MTSC 852, Fall 2015

Sokratis Makrogiannis, Ph.D.

Department of Mathematical Sciences
Delaware State University

smakrogiannis@desu.edu

November 6, 2015

Outline

k_n Nearest Neighbor Estimation

- In this method, to estimate $p(\boldsymbol{x})$ from n training samples we grow the region $\mathscr{R}_n$ with volume V_n around $\boldsymbol{x}$ such that it encloses k_n samples
- The samples enclosed by V_n are the k_n nearest-neighbors of $\boldsymbol{x}$
- We estimate density by

$$p_n(\boldsymbol{x}) = \frac{k_n/n}{V_n}$$

- We can show that the conditions $\lim_{n\to\infty} k_n = \infty$ and $\lim_{n\to\infty} k_n/n = 0$ are necessary and sufficient for $p_n(\boldsymbol{x})$ to converge to $p(\boldsymbol{x})$ at points where $p(\boldsymbol{x})$ is continuous
- Assume that $k_n = \sqrt{n}$. Then for a very large n we have that $V_n \simeq V = 1/(\sqrt{n}p(\boldsymbol{x}))$, following the form $V_1/\sqrt{n}$ that we discussed before
- While $p_n(\boldsymbol{x})$ is continuous, the gradient is not. Still, the points of discontinuity are more frequently not close to the training points

The k-Nearest Neighbor (k-NN) Rule Error Rate

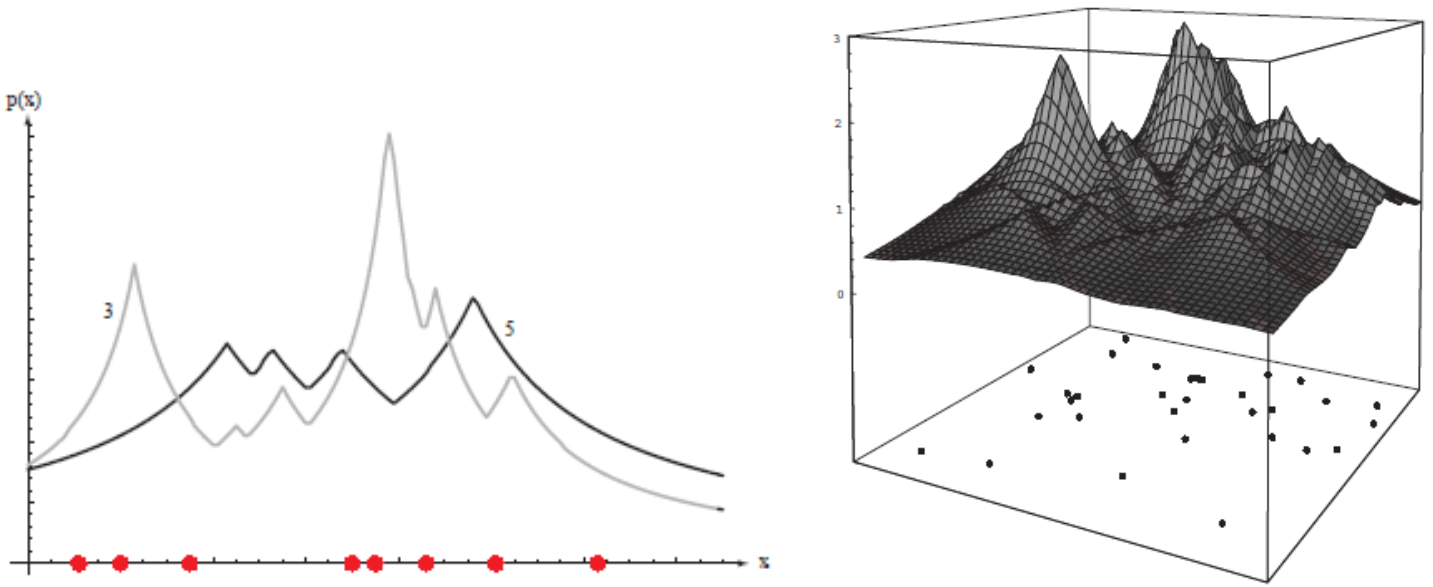


Figure: k_n density estimation for uni- and multi-variate cases

k_n Nearest Neighbor (NN) and Parzen-window Estimation

- We will compare the estimates by Parzen and k_n NN
- For $n = 1$ and $k_n = \sqrt{n} = 1$ the k_n NN estimate becomes

$$p_n(x) = \frac{1}{2|x - x_1|}$$

- This is a poor estimate with integral diverging to infinity
- For larger n, the estimate becomes more accurate but the integral remains infinite
- In Parzen window approach we can change k_n according to $k_n = k_1\sqrt{n}$ by varying k_1

k_n Nearest Neighbor (NN) and Parzen-window Estimation

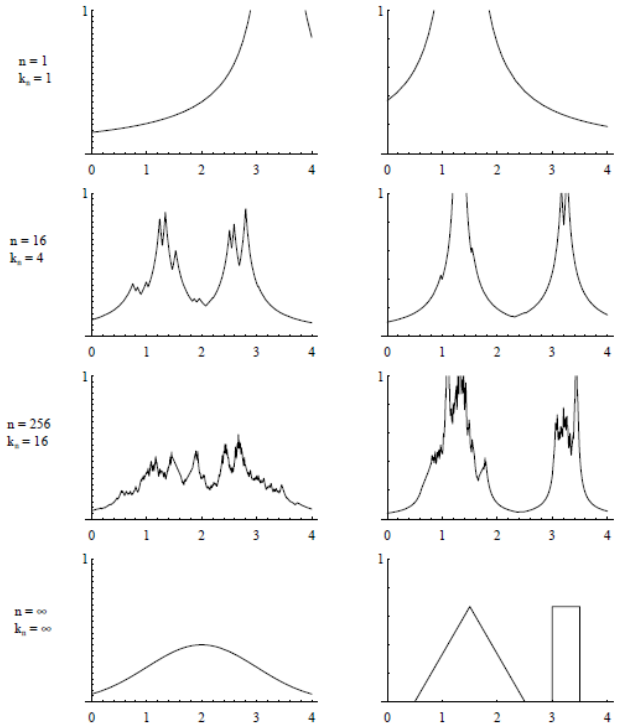


Figure: k_n density estimation for variable n, k_n. Larger k_n improves density estimates

A Posteriori Probability Estimation

- Suppose that we have n training points. n_i of them that belong to category ω_i and that we want to estimate the posterior probability $P(\omega_i|\boldsymbol{x})$ at a point $\boldsymbol{x}$
- We can use the Bayesian decision rule for this purpose. We place a volume V around $\boldsymbol{x}$. Let k be the total number of samples inside volume V, and k_i be the number of samples inside V that belong to ω_i
- Then the likelihood is: $p(\boldsymbol{x}|\omega_i) = \frac{k_i/n_i}{V}$
- Unconditional density: $p(\boldsymbol{x}) = \frac{k/n}{V}$
- Priors: $p(\omega_i) = n_i/n$
- By use of Bayes rule:

$$P_n(\omega_i|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\omega_i)p(\omega_i)}{p(\boldsymbol{x})} = \frac{\frac{k_i/n_i}{V}\frac{n_i}{n}}{\frac{k/n}{V}} = \frac{k_i}{k}$$

The Nearest Neighbor Rule

- Let $\mathscr{D}^n$ be a set of training points, or prototypes, $\mathscr{D}^n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and let $\boldsymbol{x}$ be a test point that is closest to the training point $\boldsymbol{x}' \in \mathscr{D}^n$
- The nearest neighbor rule will classify $\boldsymbol{x}$ to the class of $\boldsymbol{x}'$
- This is a suboptimal procedure; it yields an error rate that is greater than the Bayes rate

The Nearest Neighbor Rule

- We consider the prototype labels to be random variables with probabilities equal to posteriors $P(\omega_i|\boldsymbol{x}')$
- Assuming that $\boldsymbol{x}$ and $\boldsymbol{x}'$ are sufficiently close, it follows that $P(\omega_i|\boldsymbol{x}) \simeq P(\omega_i|\boldsymbol{x}')$
- Then the category ω_m of test point $\boldsymbol{x}$ is found by:

$$\omega_m = arg \max_i P(\omega_i|\boldsymbol{x})$$

- This rule will partition the feature space into regions defined by a neighbor similarity measure
- This is called Voronoi tesselation

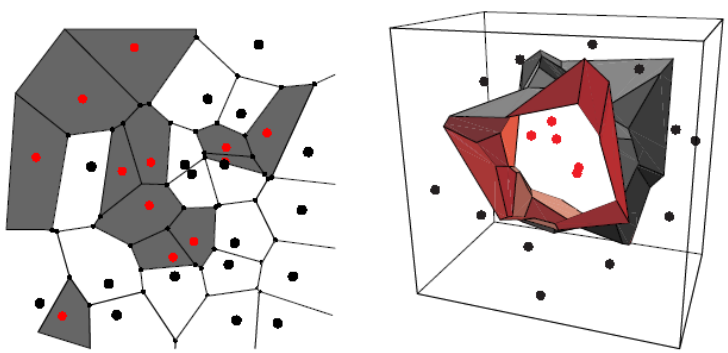


Figure: Voronoi Tesselation using NN rule

The k-Nearest Neighbor (k-NN) Rule

- This rule classifies a point $\boldsymbol{x}$ by examining the labels of the k nearest neighbors and applying majority voting
- We consider the 2-class problem with k being odd
- Labels ω_i in each of the k neighbors are random variables with probabilities $P(\omega_i|\boldsymbol{x})$, $i = 1, 2$
- k-NN rule selects class ω_m, if a majority of k nearest neighbors are labeled ω_m. This event has a probability of

$$\sum_{i=(k+1)/2}^{k} \binom{k}{i} P(\omega_m|\boldsymbol{x})^i [1 - P(\omega_m|\boldsymbol{x})]^{k-i}$$

when k increases, the probability for selecting ω_m increases

The k-Nearest Neighbor (k-NN) Rule

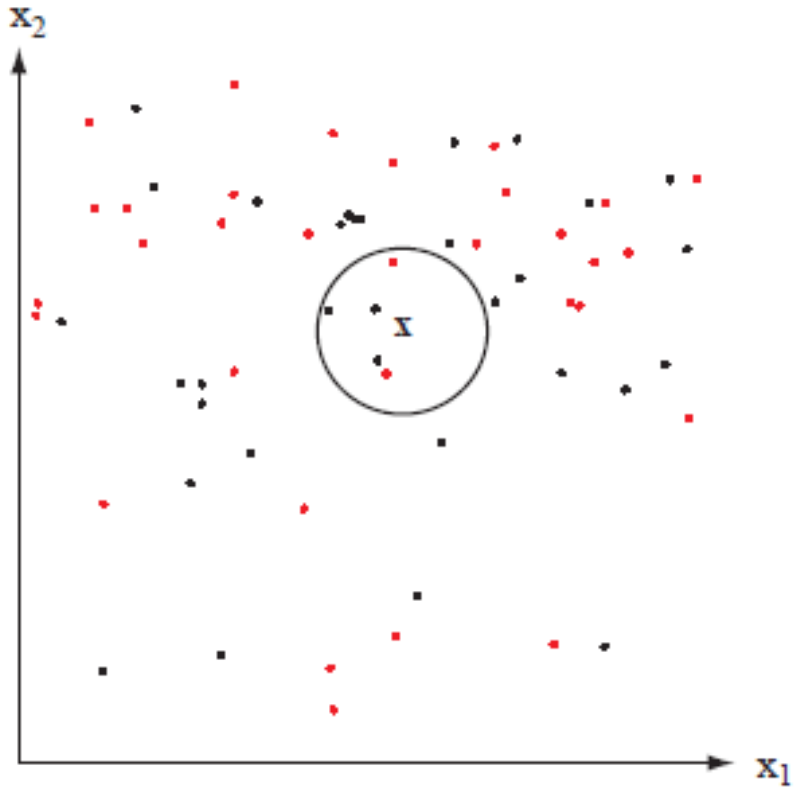


Figure: k NN rule example for $k = 5$

The k-Nearest Neighbor (k-NN) Rule Error Rate

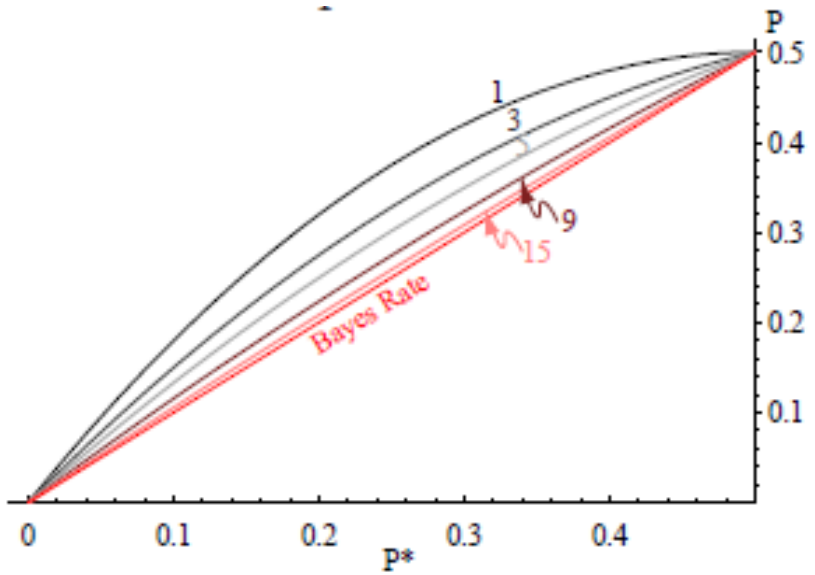


Figure: k NN rule error rate for variable k

Considerations for the k-Nearest Neighbor (k-NN) Rule

- This rule can be considered as an attempt to estimate posterior probabilities $P(\omega_i|\boldsymbol{x})$ from samples
- A large value of k reduces the error rate
- However, we still need to keep the neighbors $\boldsymbol{x}'$ around $\boldsymbol{x}$ to be small enough so that $P(\omega_i|\boldsymbol{x}) \simeq P(\omega_i|\boldsymbol{x}')$
- In practice, this means that k should be a small fraction of n

Computational Complexity of the k-Nearest Neighbor Rule (k-NN)

- Suppose n samples with d dimensions. We are looking for the nearest neighbor to a point $\boldsymbol{x}$
- The simplest approach for this is:
  1. Calculate Euclidean distances from all training points to $\boldsymbol{x}'$
  2. Find the point with minimum distance
- Computational complexity is $O(dn^2)$

Reducing Computational Complexity of the k-Nearest Neighbor Rule (k-NN)

Basic approaches

1. Partial distance
2. Prestructuring with a search tree
3. Editing prototypes - by eliminating non-informative prototypes during training

Reducing Computational Complexity of the k-Nearest Neighbor Rule (k-NN)

Partial distance

1. We calculate distances using a subset of the full d-dimensional space

$$D_r(\boldsymbol{a}, \boldsymbol{b}) = \left(\sum_{k=1}^{r} (a_k - b_k)^2 \right)^{1/2}, r \leq d$$

2. If the distance exceeds a limit, then we do not compute further

Reducing Computational Complexity of the k-Nearest Neighbor Rule (k-NN)

Prestructuring with a search tree

1. Here we create a search tree with linked prototypes
2. In classification stage, we compute distances from $\boldsymbol{x}$ to linked "entry" prototypes
3. We find the closest prototype and find its linked prototypes
4. We compute distances from $\boldsymbol{x}$ to these prototypes
5. Repeat until we reach no other tree elements

Reducing Computational Complexity of the k-Nearest Neighbor Rule (k-NN)

Editing prototypes

We eliminate non-informative prototypes during training

Algorithm 3 (Nearest-neighbor editing)

1 **begin initialize** $j = 0, \mathcal{D} =$ data set, $n =$ #prototypes
2 construct the full Voronoi diagram of $\mathcal{D}$
3 **do** $j \leftarrow j + 1$; for each prototype $\mathbf{x}'_j$
4 Find the Voronoi neighbors of $\mathbf{x}'_j$
5 **if** any neighbor is not from the same class as $\mathbf{x}'_j$ **then** mark $\mathbf{x}'_j$
6 **until** $j = n$
7 Discard all points that are not marked
8 Construct the Voronoi diagram of the remaining (marked) prototypes
9 **end**

Metrics and Nearest Neighbor Classification

- The central component of a nearest neighbor classifier is the distance function $D(\cdot, \cdot)$ between patterns
- $D(\cdot, \cdot)$ is usually a metric

Properties of Metrics

Let $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ be three vector data points in a vector space $\mathbb{R}^d$ with dimensionality d. Then a metric $D : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ must have the following properties

- Nonnegativity: $D(\boldsymbol{a}, \boldsymbol{b}) \geq 0$
- Reflexivity: $D(\boldsymbol{a}, \boldsymbol{b}) = 0 \Leftrightarrow \boldsymbol{a} = \boldsymbol{b}$
- Symmetry: $D(\boldsymbol{a}, \boldsymbol{b}) = D(\boldsymbol{b}, \boldsymbol{a})$
- Triangle inequality: $D(\boldsymbol{a}, \boldsymbol{b}) + D(\boldsymbol{b}, \boldsymbol{c}) \geq D(\boldsymbol{a}, \boldsymbol{c})$

Metrics

Minkowski Metric

A general class of metrics for d-dimensional patterns is the Minkowski metric given by

$$L_k(\boldsymbol{a}, \boldsymbol{b}) = \left(\sum_{i=1}^{d} |a_i - b_i|^k \right)^{1/k}.$$

This is also called the L_k norm

- L_1 norm: Manhattan or city block distance. This is the shortest path between $\boldsymbol{a}$ and $\boldsymbol{b}$. In this path each segment is parallel to the coordinate axes
- L_2 norm: Euclidean distance
- L_∞ norm: corresponds to the maximum of the distances between the projections of $\boldsymbol{a}$ and $\boldsymbol{b}$ onto each of the d coordinate axes

Minkowski Metrics Example

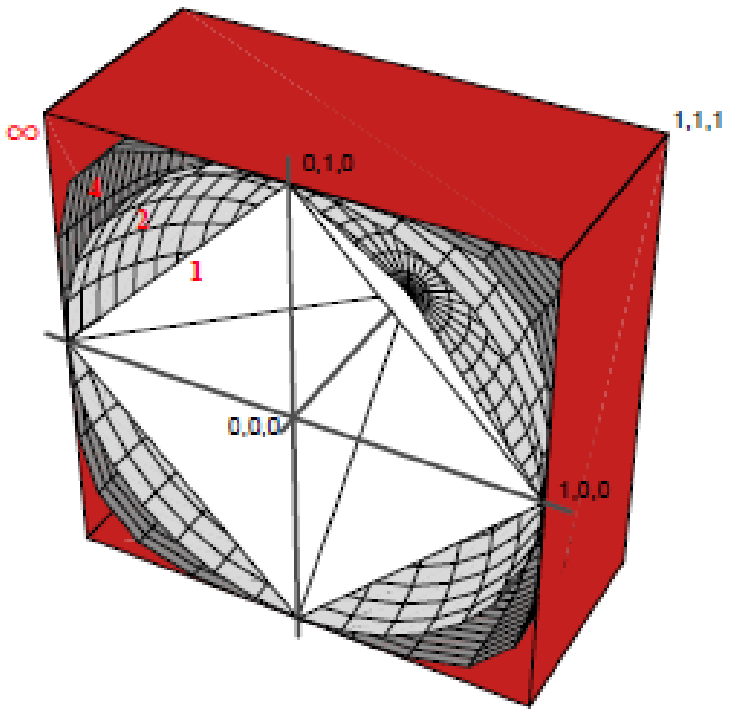


Figure: Isosurfaces for L_1 (white), L_2 (light gray), L_4 (dark gray), and L_∞ (pink)

Metrics

Tanimoto Metric

This is mostly used in taxonomy to find a distance between two sets S_1 and S_2:

$$D_{Tanimoto}(S_1, S_2) = \frac{n_1 + n_2 - 2n_{12}}{n_1 + n_2 - n_{12}}$$

where n_1, n_2 are the cardinalities of sets S_1 and S_2, and n_{12} is the cardinality of $S_1 \cap S_2$

Tangent Distance

- On several occasions the computation of a specific metric in an NN classifier may be sensitive to the problem of invariance
- For example, suppose we need to classify digits using 10×10 pixel grayscale images
- Then a small translation of a digit by a few pixels may change the metric computation significantly
- One can reduce this problem by use of the tangent distance

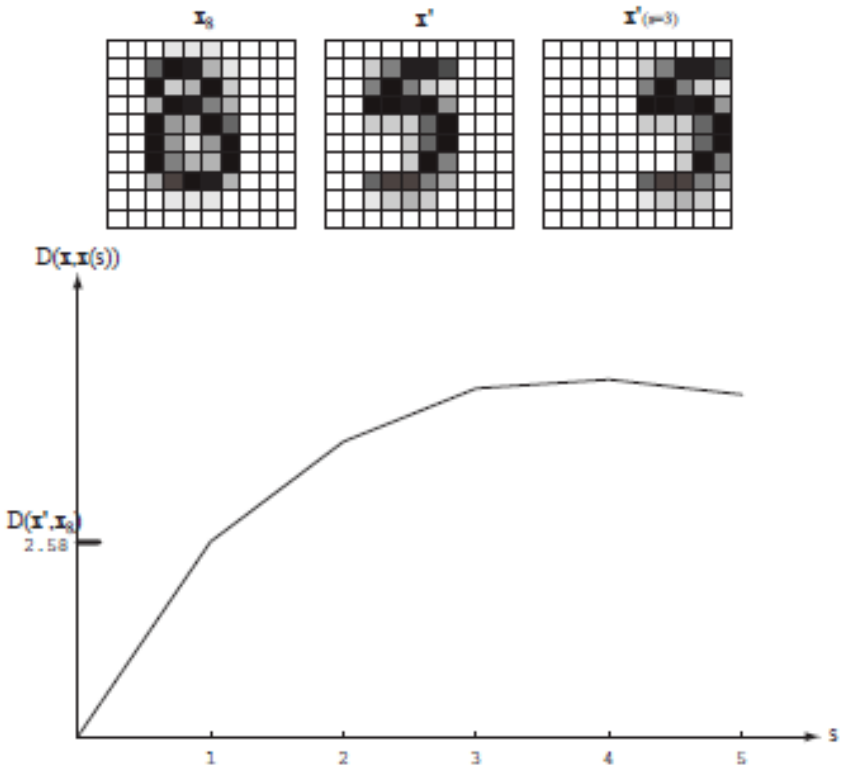


Figure: In this example the digit 8 is closer to the prototype of digit 5 than the shifted digit 5

Tangent Distance

- The general approach for addressing variability is to construct tangent vectors for all transformations
- Let $\boldsymbol{x}'$ be a prototype, α_i be a transformation parameter, and $F_i(\boldsymbol{x}';\alpha_i)$ be the transformed prototype
- Then for each transformation we can construct the tangent vector

$$\boldsymbol{TV}_i = F_i(\boldsymbol{x}';\alpha_i) - \boldsymbol{x}'$$

- This process produces an $r \times d$ matrix $\boldsymbol{T}$ for each vector $\boldsymbol{x}'$ that corresponds to the tangent vectors at $\boldsymbol{x}'$

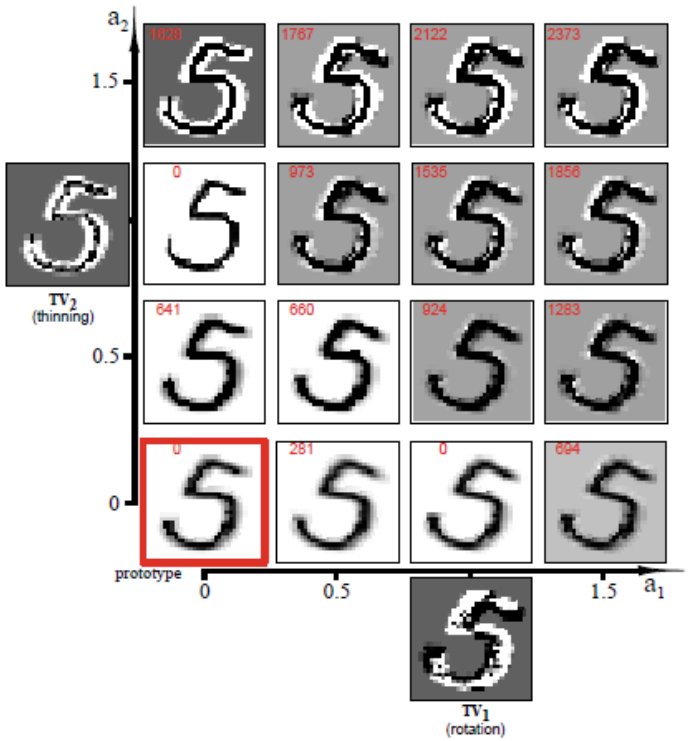


Figure: Generation of tangent vectors using rotation and line thinning

Tangent Distance

- To reduce the computation load we can sample the tangent space to create a linear subspace
- Then we compute the tangent distance between $\boldsymbol{x}'$ and $\boldsymbol{x}$

$$D_{tan}(\boldsymbol{x}', \boldsymbol{x}) = \min_{a} \|(\boldsymbol{x}' + \boldsymbol{T}\boldsymbol{\alpha}) - \boldsymbol{x}\|$$

- So we are looking for the smallest Euclidean distance from $\boldsymbol{x}'$ to $\boldsymbol{x}$
- In classification we need to find optimizing value of α_i
- In that case we find the optimal $\boldsymbol{\alpha}$ using gradient descent or matrix methods

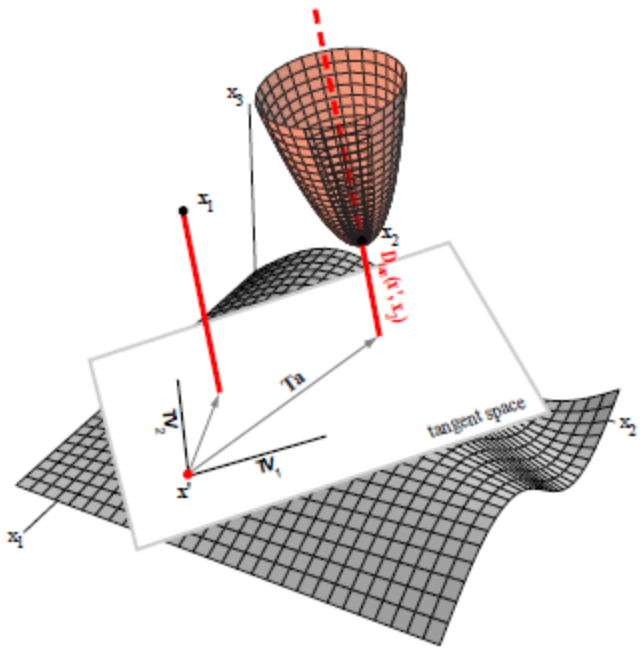


Figure: Euclidean space produced by tangent vectors TV_1 and TV_2

Linear Discriminant Functions

MTSC 852, Fall 2015

Sokratis Makrogiannis, Ph.D.

Department of Mathematical Sciences
Delaware State University

smakrogiannis@desu.edu

November 12, 2015

Outline

Introduction

- Pattern classification approaches can be categorized into *generative* and *discriminant*
- *Generative approaches* first estimate models of class-conditional probability density that generate the training patterns, then define the discriminant function. These methods usually assume some parametric form of density
- *Discriminant approaches* find the discriminant function directly without making any assumptions about the distribution of the training patterns
- This chapter deals with discriminant functions that are either linear in components of pattern vector $\boldsymbol{x}$, or linear in a function of $\boldsymbol{x}$

Introduction

- Linear discriminant functions are analytically tractable and computationally efficient
- They can be used for trial classifiers
- We can find a linear discriminant function by minimizing a criterion function
- The usual criterion function is related to the training error, that is the average loss for classifying the training patterns
- A major consideration is to ensure that the discriminant function will classify at a good rate unknown, new patterns

Linear Discriminant Function

- A linear discriminant function $g(\boldsymbol{x})$ is typically of the form:

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$$

  where $\boldsymbol{x}$ is the pattern vector, $\boldsymbol{w}$ is the weight vector, and w_0 is the bias or threshold weight
- In general, when we have c classes, we need to find c discriminant functions

Two categories

- Suppose a problem with two categories ω_1 and ω_2
- Then we can define a single discriminant function by

$$g(\boldsymbol{x}) = g_1(\boldsymbol{x}) - g_2(\boldsymbol{x})$$

- The decision rule is:

$$\text{Decide } \omega_1 \text{ if } g(\mathbf{x}) > 0; \text{ decide } \omega_2 \text{ if } g(\boldsymbol{x}) < 0$$

- The decision surface is represented by $g(\boldsymbol{x}) = 0$. For a linear discriminant function the decision surface is a hyperplane H
- Let points $\boldsymbol{x}_1, \boldsymbol{x}_2$ on the decision surface. Then:

$$\boldsymbol{w}^T \boldsymbol{x}_1 = \boldsymbol{w}^T \boldsymbol{x}_2 \Leftrightarrow \boldsymbol{w}^T (\boldsymbol{x}_1 - \boldsymbol{x}_2) = 0$$

- We observe that $\boldsymbol{w}$ is normal to the hyperplane

Two categories

- $g(\boldsymbol{x})$ also gives a measure of the distance of the point $\boldsymbol{x}$ to the hyperplane
- We can write $\boldsymbol{x}$ as:

$$\boldsymbol{x} = \boldsymbol{x}_p + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$$

  where $\boldsymbol{x}_p$ is the normal projection of $\boldsymbol{x}$ to H, r is the algebraic distance
- Given that $g(\boldsymbol{x}_p) = 0$, it follows that

$$g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 = \boldsymbol{w}^T(\boldsymbol{x}_p + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}) + w_0 = \boldsymbol{w}^T\boldsymbol{x}_p + r\frac{\boldsymbol{w}^T\boldsymbol{w}}{\|\boldsymbol{w}\|} + w_0$$

$$\Leftrightarrow g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}_p + r\|\boldsymbol{w}\| + w_0 \Leftrightarrow g(\boldsymbol{x}) = r\|\boldsymbol{w}\| \Leftrightarrow r = \frac{g(\boldsymbol{x})}{\|\boldsymbol{w}\|}$$

Two categories

- A linear discriminant function produces a hyperplane decision surface
- The weight vector $\boldsymbol{w}$ determines the orientation and the bias w_0 determines the location of the hyperplane
- $g(\boldsymbol{x})$ is proportional to the distance of point $\boldsymbol{x}$ from the hyperplane
- $\boldsymbol{w}$ points toward the positive side of the hyperplane where $g(\boldsymbol{x}) > 0$

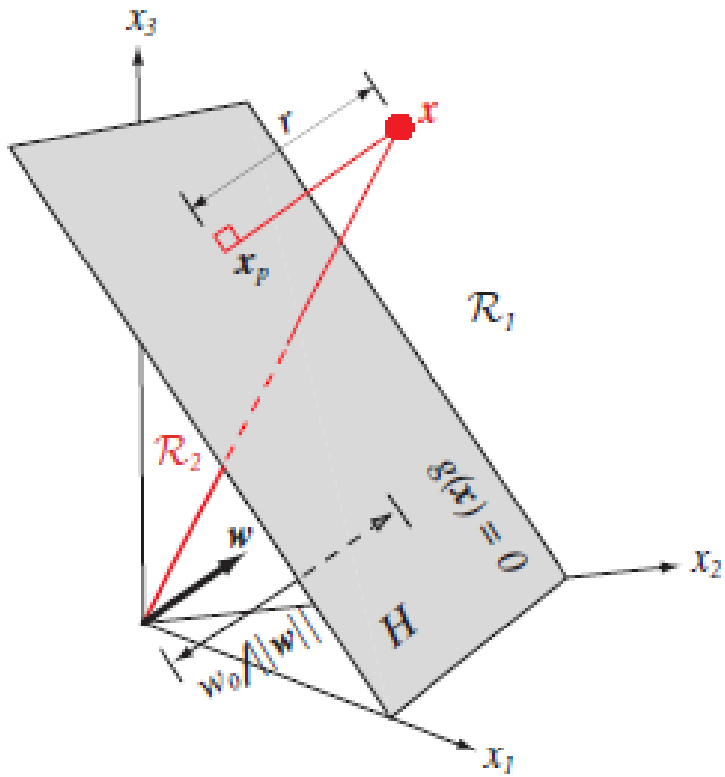


Figure: Linear decision boundary for two categories

Multiple categories

- To address classification into multiple c categories one could i) make c decisions for ω_i versus not ω_i, or ii) define $c(c-1)/2$ discriminants for all possible class pairs
- However these strategies can lead to regions of undefined classification

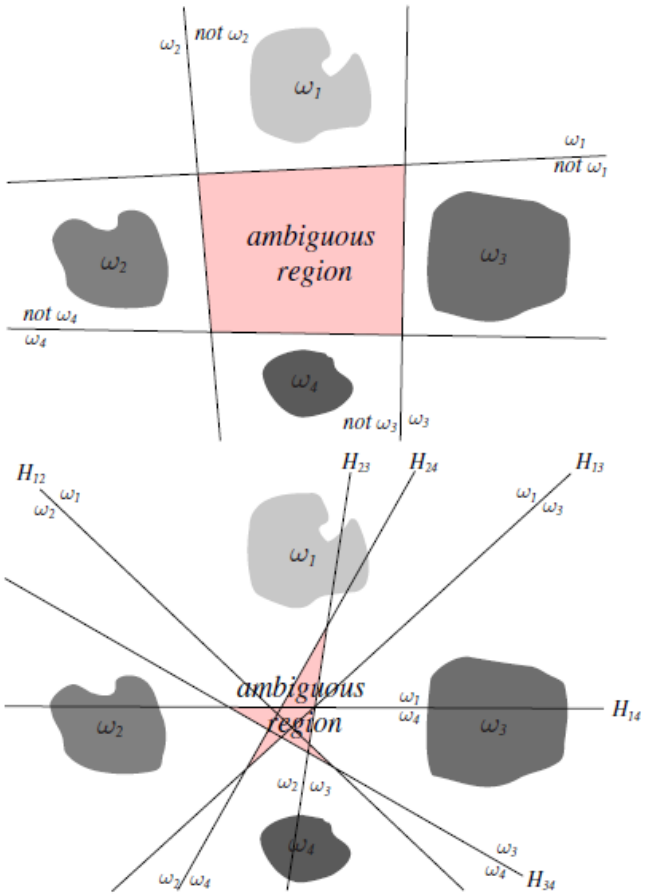


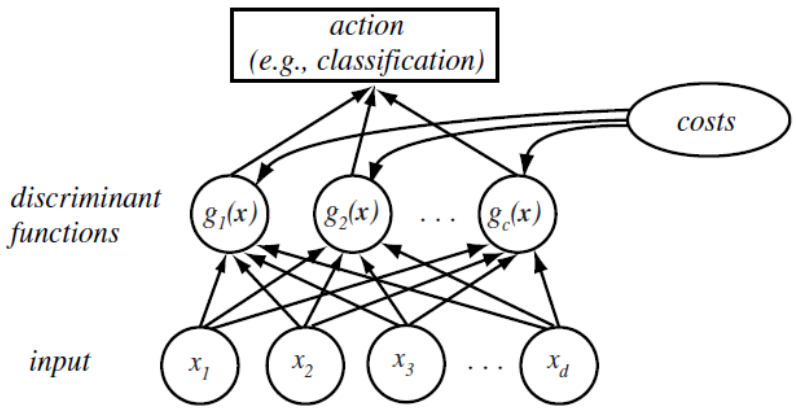
Figure: One-against-all and taking all pairwise discriminant may lead to ambiguous decision regions

Multiple categories

- To avoid these shortcomings we can use a set of discriminant functions $g_i(x), i = 1, ..., c$ corresponding to each category, with a decision rule:

  Assign feature vector $\mathbf{x}$ to class ω_i, if $g_i(\mathbf{x}) > g_j(\mathbf{x}) \quad \forall j \neq i$

- This classifier can be considered as a network or linear machine that computes c discriminant functions and makes a decision using a maximum operator



Multiple categories

- Let $\mathscr{R}_i, \mathscr{R}_j$ be the decision regions corresponding to categories ω_i, ω_j
- If $\mathscr{R}_i, \mathscr{R}_j$ are contiguous, then at the boundary H_{ij} between $\mathscr{R}_i, \mathscr{R}_j$ we have that

$$g_i(\boldsymbol{x}) = g_j(\boldsymbol{x}) \Leftrightarrow (\boldsymbol{w}_i - \boldsymbol{w}_j)^T \boldsymbol{x} + (w_{i0} - w_{j0}) = 0$$

- We observe that $\boldsymbol{w}_i - \boldsymbol{w}_j$ is normal to H_{ij} and similarly to the two-category case the distance from $\boldsymbol{x}$ to H_{ij} is $(g_i(\boldsymbol{x}) - g_j(\boldsymbol{x})/\|\boldsymbol{w}_i - \boldsymbol{w}_j\|$
- In a linear machine the decision regions are convex, and every decision region is singly connected, therefore this classifier is best suited for unimodal class-conditional densities $p(\boldsymbol{x}|\omega_i)$

Multiple categories

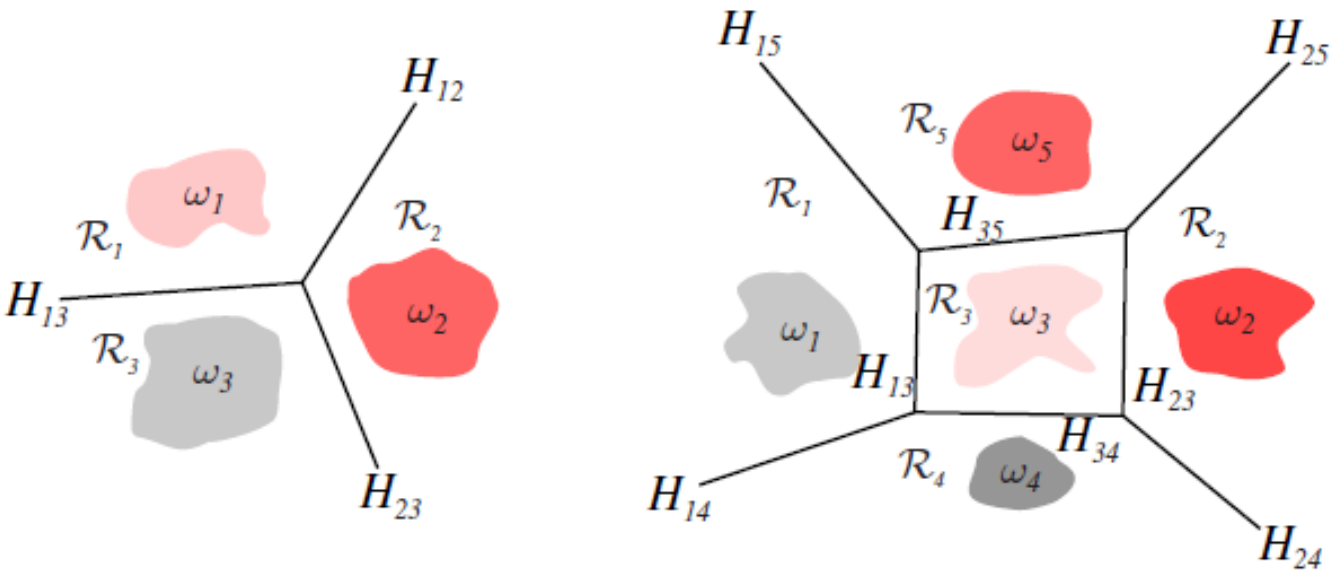


Figure: Decision Boundaries for linear machines

Higher Order Discriminant Functions

- Linear discriminant:

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

  where w_i: components of $\boldsymbol{x}$

- Quadratic discriminant includes second order terms $x_i x_j$:

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

- By adding higher order terms we generate a polynomial discriminant

Generalized Linear Discriminant Functions

- The generalized discriminant function is defined as:

$$g(\boldsymbol{x}) = \sum_{i=1}^{\hat{d}} \alpha_i y_i(\boldsymbol{x}) = \boldsymbol{\alpha}^T \boldsymbol{y}$$

  where $\boldsymbol{\alpha}$: $\hat{d}$-dimensional vector
  $y_i(\boldsymbol{x})$: arbitrary functions of $\boldsymbol{x}$, also called ϕ functions
- $g(\boldsymbol{x})$ may be nonlinear in $\boldsymbol{x}$ but is linear in $\boldsymbol{y}$
- Functions $y_i(\boldsymbol{x})$ map points from a d-dimensional space to a $\hat{d}$-dimensional space
- These mappings may simplify the original problem to that of finding a linear discriminant function

Generalized Linear Discriminant Functions - Example

- Let $g(\boldsymbol{x})$ be:

$$g(\boldsymbol{x}) = \alpha_1 + \alpha_2 x + \alpha_3 x^2$$

- Then $\boldsymbol{y}$ is :

$$\boldsymbol{y} = \begin{pmatrix} 1 \\ x \\ x^2 \end{pmatrix}$$

- The inherent dimensionality of the data is 1
- The decision regions in $\boldsymbol{y}$-space are convex, but they are non-convex in x-space

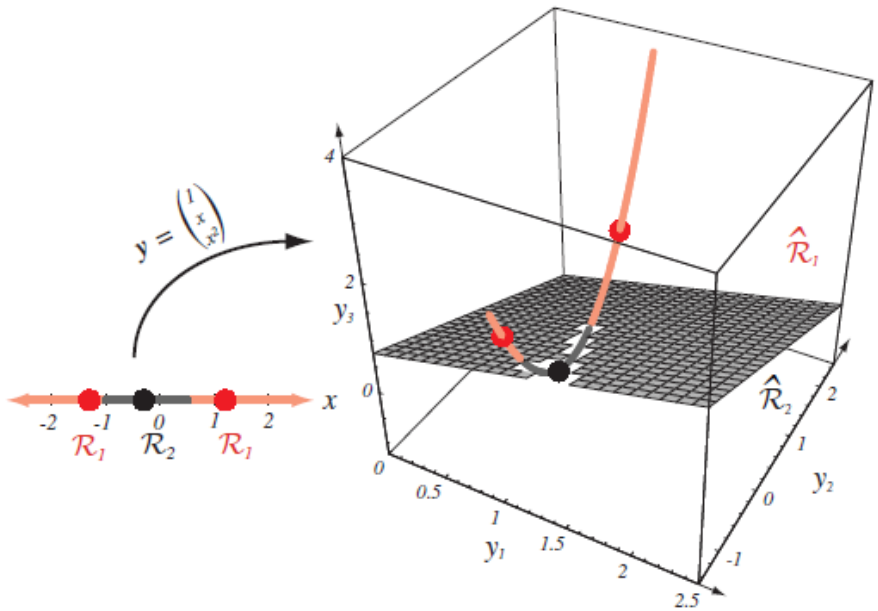


Figure: The ϕ functions transform a line to a parabola and creates a non-simply connected decision region in the x-space

Generalized Linear Discriminant Functions

- The mapping to a higher dimensional space while interesting in concept, it implies a requirement for a lot more training data because of the curse of dimensionality
- We can address this drawback by imposing a constraint of large margins between the training samples. This approach is used in Support Vector Machines (SVM)

Linear Discriminant Functions and Augmented Vectors

- For the linear discriminant we have that:

$$g(\boldsymbol{x}) = w_0 + \sum_{i=1}^{d} w_i x_i = \sum_{i=0}^{d} w_i x_i$$

  where $x_0 = 1$
- Then $g(\boldsymbol{x}) = \boldsymbol{\alpha}^T \boldsymbol{y}$:

$$\boldsymbol{y} = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{pmatrix} = \begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix} \text{ and } \boldsymbol{\alpha} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{pmatrix} = \begin{pmatrix} w_0 \\ \boldsymbol{w} \end{pmatrix}$$

- This is a transformation from a d-dimensional to a $d+1$-dimensional space creates a hyperplane passing from the origin
- We reduce the problem of finding a weight vector $\boldsymbol{w}$ and a threshold weight w_0 to that of finding just a weight vector $\boldsymbol{\alpha}$

Linear Discriminant Functions

MTSC 852, Fall 2015

Sokratis Makrogiannis, Ph.D.

Department of Mathematical Sciences
Delaware State University

smakrogiannis@desu.edu

November 17, 2015

Outline

The Two-category Linearly Separable Behavior

Linearly separable samples

- Suppose a set of samples $\mathscr{D} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n\}$
- We are looking for a weight vector $\boldsymbol{a}$ in a linear discriminant function $g(\boldsymbol{y}) = \boldsymbol{a}^T \boldsymbol{y}$
- If there exists a weight vector that classifies all samples correctly, then the samples are said to be linearly separable
- By the decision rule, a sample $\boldsymbol{y}_i$ is classified correctly to ω_1, if $\boldsymbol{a}^T \boldsymbol{y}_i > 0$, and is classified correctly to ω_2, if $\boldsymbol{a}^T \boldsymbol{y}_i < 0$
- If we negate the samples in ω_2 –a process called normalization– then we are looking for a weight vector $\boldsymbol{a}$, such that $\boldsymbol{a}^T \boldsymbol{y}_i > 0, \forall \boldsymbol{y}_i \in \mathscr{D}$
- Such a vector is called a separating vector, or solution vector

The Two-category Linearly Separable Behavior

- The weight vector can be considered as a point in the weight space
- Each sample places a constraint on the location of the solution vector
- Assuming normalization, the solution vector must be located on the positive side of the separating hyperplane
- The solution vector lies on the intersection of n half-spaces. This region is called the solution region

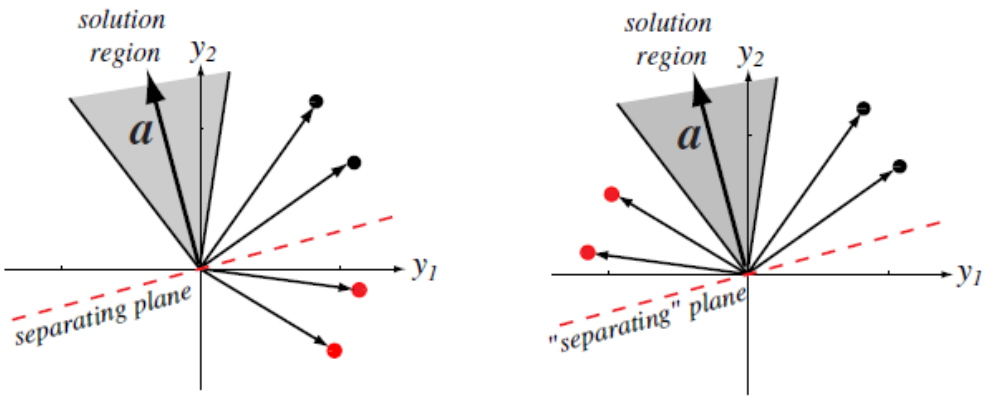


Figure:

The Two-category Linearly Separable Behavior

Margins

- Based on previous discussion, we can tell that the solution vector is not unique
- We can impose additional constraints:
  one is to look for the unit vector that maximizes the minimum distance from the samples to the separating hyperplane
  another is to look for the minimum length weight vector $\boldsymbol{a}$ such that $\boldsymbol{a}^T\boldsymbol{y} \geq b$, where b is a positive constant called margin
- The objective is to find a solution vector that is closer to the middle of the solution region, therefore the classifier will perform well for unlabeled samples

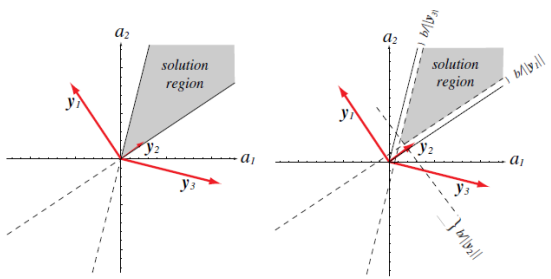


Figure:

Gradient Descent

- To find a solution of $\boldsymbol{a}^T \boldsymbol{y}_i > 0$ we can define a criterion function $J(\boldsymbol{a})$ and seek the minimizing vector $\boldsymbol{a}$
- Function minimization can be achieved by the gradient descent technique
- The gradient descent begins from an initial value of $\boldsymbol{a}$ and moves along the negative of the gradient:

$$\boldsymbol{a}(k+1) = \boldsymbol{a}(k) - n(k)\nabla J(\boldsymbol{a}(k))$$

where n is a positive scale factor or learning rate (step size)

- This method is dependent on $n(k)$: if it's too small, convergence is slow, whereas if it's too large, the algorithm may overshoot or diverge

Algorithm 1 (Basic gradient descent)

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

Setting the Learning Rate $n(k)$

- The second order Taylor expansion of the criterion function $J(\boldsymbol{a})$ at $\boldsymbol{a}$ is:

$$J(\boldsymbol{a}) \simeq J(\boldsymbol{a}(k)) + \boldsymbol{\nabla} J^T (\boldsymbol{a} - \boldsymbol{a}(k)) + (1/2)(\boldsymbol{a} - \boldsymbol{a}(k))^T \boldsymbol{H} (\boldsymbol{a} - \boldsymbol{a}(k))$$

- By using the gradient descent equation we get:

$$J(\boldsymbol{a}(k+1)) \simeq J(\boldsymbol{a}(k)) + n(k)\|\boldsymbol{\nabla} J\|^2 + (1/2) n^2(k) \boldsymbol{\nabla} J^T \boldsymbol{H} \boldsymbol{\nabla} J$$

  where $\boldsymbol{H}$ is the Hessian matrix of second order derivatives $\frac{\partial^2 J}{\partial a_i \partial a_j}$

- We can show that $J(\boldsymbol{a}(k+1))$ is minimized by:

$$n(k) = \frac{\|\boldsymbol{\nabla} J\|^2}{\boldsymbol{\nabla} J^T \boldsymbol{H} \boldsymbol{\nabla} J}$$

- If the criterion function is quadratic in the region of interest, then $\boldsymbol{H}$ is a constant and n is a constant as well

Newton's Algorithm

- In this method we choose $\boldsymbol{a}(k+1)$ to minimize the second order expansion
- This yields the equation:

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

- Newton's algorithm achieves faster convergence
- However it requires that the Hessian is non-singular and implies increased computational cost $O(d^3)$

Algorithm 2 (Newton descent)

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

Figure:

Gradient Descent vs. Newton's Algorithm

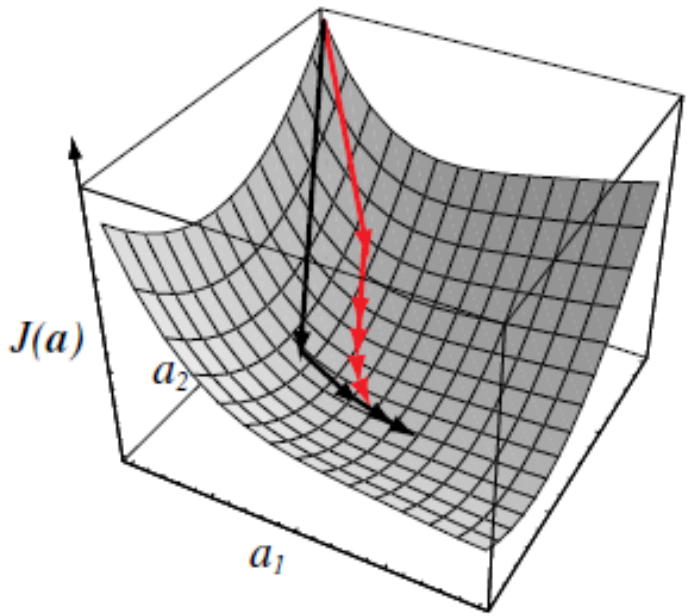


Figure: Newton's algorithm (black line) converges in fewer steps than gradient descent (red line), however it is more computational intensive as it requires inversion of the Hessian

The Perceptron Criterion Function

- The only missing piece is now the criterion function $J(\boldsymbol{a})$
- Since we need to minimize this function, one obvious option is the number of misclassified samples. This function is discontinuous, therefore gradient calculation may be problematic
- We can use the linear discriminant function to define the Perceptron criterion function after normalizing the patterns:

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

  where $\mathscr{Y}(a)$ is the set of samples misclassified by $\boldsymbol{a}$
- This criterion function is always nonnegative because $\boldsymbol{a}^T \boldsymbol{y} \leq 0$ for any misclassified sample

The Perceptron Criterion Function

- We have that:

$$\nabla J_p = \sum_{y \in \mathcal{Y}} (-\boldsymbol{y})$$

- The update rule in gradient descent becomes:

$$\boldsymbol{a}(k+1) = \boldsymbol{a}(k) + n(k) \sum_{y \in \mathcal{Y}_k} \boldsymbol{y}$$

where $\mathcal{Y}_k$ is the set of samples misclassified by $\boldsymbol{a}(k)$

In the Batch Perceptron algorithm the next vector is computed by adding the a multiple of the sum of misclassified samples to the weight vector from previous iteration

Algorithm 3 (Batch Perceptron)

1 begin initialize $\mathrm{a}, \eta(\cdot)$, criterion $\theta, k = 0$
2 do $k \leftarrow k + 1$
3 $\mathrm{a} \leftarrow \mathrm{a} + \eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} \mathbf{y}$
4 until $\eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} \mathbf{y} < \theta$
5 return a
6 end

Perceptron Learning

If classes are linearly separable then the perceptron rule will converge to a valid solution

- A version of the perceptron rule uses variable learning rate
- However, this technique will converge only under specific conditions

On the other hand, if the two classes are not linearly separable, the perceptron rule will not converge

- Because there will always be at least one misclassified sample, the corrections in perceptron rule will continue with no end
- One approach to solve this problem is to use variable rates $n(k)$ approaching zero as $k \to \infty$

Linear Discriminant Functions

MTSC 852, Fall 2015

Sokratis Makrogiannis, Ph.D.

Department of Mathematical Sciences
Delaware State University

smakrogiannis@desu.edu

November 19, 2015

Outline

Minimum Squared Error Techniques

- The previous criterion functions use the misclassified samples
- Here we introduce techniques that use all samples for estimation of the weight vector
- Now we are looking to solve $\boldsymbol{a}^T \boldsymbol{y}_i = b_i$, where b_i is a set of arbitrary constants

Minimum Squared Error (MSE) Techniques

- So we are looking for a solution of a system of linear equations
- Using matrix notation, we are looking to find a weight vector that satisfies

$$\begin{pmatrix} y_{10} & y_{12} & \cdots & y_{1d} \\ y_{20} & y_{22} & \cdots & y_{2d} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ y_{n0} & y_{n2} & \cdots & y_{nd} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ \vdots \\ b_n \end{pmatrix}, \quad \boldsymbol{Ya} = \boldsymbol{b}$$

- If $\boldsymbol{Y}$ were non-singular, we could solve for $\boldsymbol{a} = \boldsymbol{Y}^{-1} b$
- But $\boldsymbol{Y}$ is rectangular with more rows than columns, therefore our system is overdetermined and usually there is no exact solution

Minimum Squared Error (MSE) Techniques

- Still, we can find an approximate solution of $\boldsymbol{Ya} = \boldsymbol{b}$ by minimizing

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

- We can define the equivalent criterion function $J_s(\boldsymbol{a})$:

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

- We can solve this problem using a gradient search technique
- Otherwise, we form the gradient ∇J_s and set it to zero:

$$\nabla J_s = \sum_{i=1}^{n} 2(\boldsymbol{a}^T \boldsymbol{y}_i - \boldsymbol{b}_i)\boldsymbol{y}_i = 2\boldsymbol{Y}^T(\boldsymbol{Ya} - \boldsymbol{b})$$

$$\boldsymbol{Y}^T(\boldsymbol{Ya} - \boldsymbol{b}) = 0 \Leftrightarrow \boldsymbol{Y}^T\boldsymbol{Ya} = \boldsymbol{Y}^T\boldsymbol{b}$$

Minimum Squared Error (MSE) Techniques

- In the equation $\boldsymbol{Y}^T\boldsymbol{Y}\boldsymbol{a} = \boldsymbol{Y}^T\boldsymbol{b}$, we observe that $\boldsymbol{Y}^T\boldsymbol{Y}$ is a $d \times d$ square matrix that is often nonsingular
- Then the solution is:

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

$$\Leftrightarrow \boldsymbol{a} = \boldsymbol{Y}^\dagger\boldsymbol{b}$$

  where the $d \times n$ matrix $\boldsymbol{Y}^\dagger \equiv (\boldsymbol{Y}^T\boldsymbol{Y})^{-1}\boldsymbol{Y}^T$ is called the pseudoinverse of $\boldsymbol{Y}$ with $\boldsymbol{Y}^\dagger\boldsymbol{Y} = \boldsymbol{I}$, but in general $\boldsymbol{Y}\boldsymbol{Y}^\dagger \neq \boldsymbol{I}$
- $\boldsymbol{Y}^\dagger$ is defined as:

$$\boldsymbol{Y}^\dagger \equiv \lim_{\varepsilon \to 0}(\boldsymbol{Y}^T\boldsymbol{Y} + \varepsilon\boldsymbol{I})^{-1}\boldsymbol{Y}^T$$

- We can show that this limit always exists and that $\boldsymbol{a} = \boldsymbol{Y}^\dagger\boldsymbol{b}$ is a solution to the MSE problem
- We note that the MSE solution depends on $\boldsymbol{b}$. If $\boldsymbol{b}$ is fixed, the MSE solution does not necessarily yield the separating vector in a linearly separable case
- But MSE may yield a useful discriminant function in both separable and nonseparable cases

The LMS Procedure

The MSE criterion function $J_s(\boldsymbol{a}) = \|\boldsymbol{Ya} - \boldsymbol{b}\|^2$ can be minimized by gradient descent

- It addresses problems of singularity of $\boldsymbol{Y}^T\boldsymbol{Y}$
- We don't have to deal with large matrices in data of high dimensionality

The LMS Procedure

From previous result, the gradient of the MSE criterion function is:

$$\nabla J_s = 2\boldsymbol{Y}^T(\boldsymbol{Ya} - \boldsymbol{b})$$

Then the update rule becomes

$$\boldsymbol{a}(k+1) = \boldsymbol{a}(k) - n(k)\nabla J(\boldsymbol{a}(k)) = \boldsymbol{a}(k) + n(k)\boldsymbol{Y}^T(\boldsymbol{b} - \boldsymbol{Ya}(k))$$

- We can show that if $n(k) = n(1)/k$, where $n(1)$ is a positive constant, then this rule converges to a solution of $\boldsymbol{Y}^T(\boldsymbol{Ya}(k) - \boldsymbol{b}) = 0$

- Widrow-Hoff or LMS rule

We can consider each sample sequentially to derive:

$$\boldsymbol{a}(k+1) = \boldsymbol{a}(k) + n(k)(b(k) - \boldsymbol{a}^T(k)\boldsymbol{y}^k)\boldsymbol{y}^k$$

Algorithm 10 (LMS)

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

Comparing Perceptron with MSE

- The perceptron rule always finds a solution if classes are linearly separable but does not converge if classes are nonseparable
- The MSE approach will converge to a solution but it may not be the separating hyperplane if classes are linearly separable

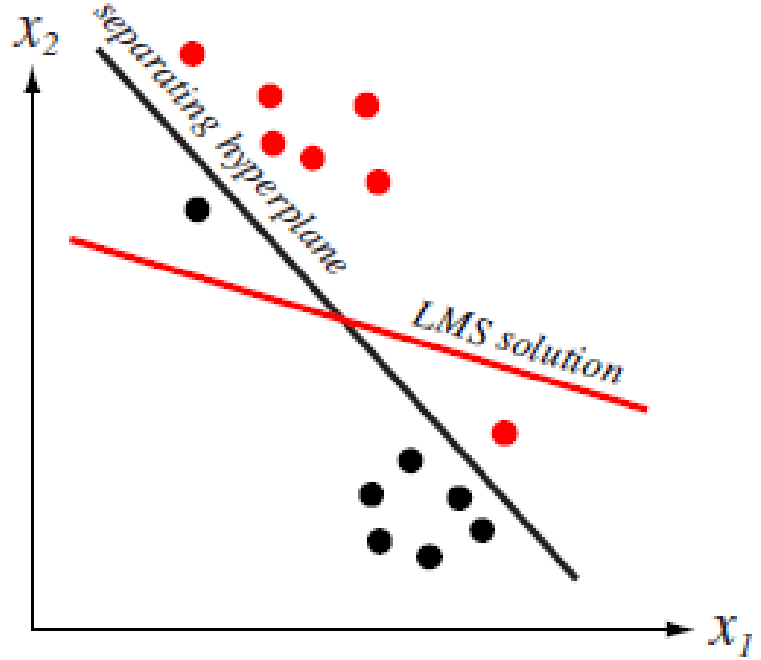


Figure: The LMS algorithm does not necessarily converge to the separating hyperplane even if the classes are linearly separable

Ho-Kashyap Techniques

- We noted that the MSE solution does not necessarily converge to the separating hyperplance for linearly separable classes
- This is mainly due to the selection of a fixed arbitrary margin vector $\boldsymbol{b}$ in MSE
- However by definition of linear separability, if the classes are linearly separable, then $\exists \hat{\boldsymbol{a}}, \hat{\boldsymbol{b}} : \boldsymbol{Y}\hat{\boldsymbol{a}} = \hat{\boldsymbol{b}} > 0$
- One approach for finding the separating hyperplane using MSE would be to find both the separating vector $\boldsymbol{a}$ and margin vector $\boldsymbol{b}$
- This can be achieved in the following steps
  1. Use gradient descent to find $\boldsymbol{b}$
  2. Use MSE approach to find $\boldsymbol{a}$
  3. Iterate until convergence

Ho-Kashyap Descent Procedure

- The gradients of MSE criterion function $J_s(\boldsymbol{a}) = \|\boldsymbol{Ya} - \boldsymbol{b}\|^2$ in $\boldsymbol{a}$ and $\boldsymbol{b}$ are

$$\nabla_{\boldsymbol{a}} J_s = \sum_{i=1}^{n} 2(\boldsymbol{a}^T \boldsymbol{y}_i - \boldsymbol{b}_i)\boldsymbol{y}_i = 2\boldsymbol{Y}^T(\boldsymbol{Ya} - \boldsymbol{b})$$

$$\nabla_{\boldsymbol{b}} J_s = \sum_{i=1}^{n} (-2)(\boldsymbol{a}^T \boldsymbol{y}_i - \boldsymbol{b}_i) = -2(\boldsymbol{Ya} - \boldsymbol{b})$$

- For a specific $\boldsymbol{b}$, the MSE solution for $\boldsymbol{a}$ is $\boldsymbol{a} = \boldsymbol{Y}^{\dagger}\boldsymbol{b}$, where $\boldsymbol{Y}^{\dagger} \equiv (\boldsymbol{Y}^T\boldsymbol{Y})^{-1}\boldsymbol{Y}^T$
- We use gradient descent to find $\boldsymbol{b}$; however we must keep the constraint $\boldsymbol{b} > 0$

Ho-Kashyap Descent Procedure

- We initialize $\boldsymbol{b}$ to a positive value and set to zero all positive components of $\nabla_{\boldsymbol{b}} J_s$ by use of $\nabla_{\boldsymbol{b}} J_s^- = 1/2[\nabla_{\boldsymbol{b}} J_s - |\nabla_{\boldsymbol{b}} J_s|]$ in the update rule:

  $$\boldsymbol{b}(k+1) = \boldsymbol{b}(k) - (n/2)[\nabla_{\boldsymbol{b}} J_s - |\nabla_{\boldsymbol{b}} J_s|]$$

- We compute the weight vector $\boldsymbol{a}(k+1) = \boldsymbol{Y}^{\dagger}\boldsymbol{b}(k+1)$

Algorithm 11 (Ho-Kashyap)

```
1  begin initialize a, b, η(·) < 1, criteria b_min, k_max
2        do k ← k + 1
3           e ← Ya − b
4           e+ ← 1/2(e + Abs[e])
5           b ← a + 2η(k)e+
6           a ← Y†b
7           if Abs[e] ≤ b_min then return a, b and exit
8        until k = k_max
9    Print NO SOLUTION FOUND
10 end
```