# MTSC 852 - Pattern Recognition
# Lab Session
# Bayesian Decision Theory

Sokratis Makrogiannis, Ph.D.

October 15, 2015

## Contents

## 1 Bayesian Decision Classifier

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

## 1.2 Two Categories

- Suppose a problem with two categories $\omega_1$ and $\omega_2$.
- 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

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

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

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

## 1.3 Discriminant Functions for the Normal Density

- 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

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

## 1.4 Arbitrary Covariance Matrices

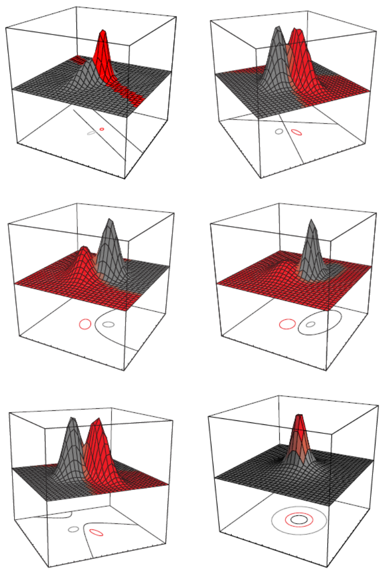
- In this case the covariance matrices are different for each category
- The discriminant function takes the form

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

  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

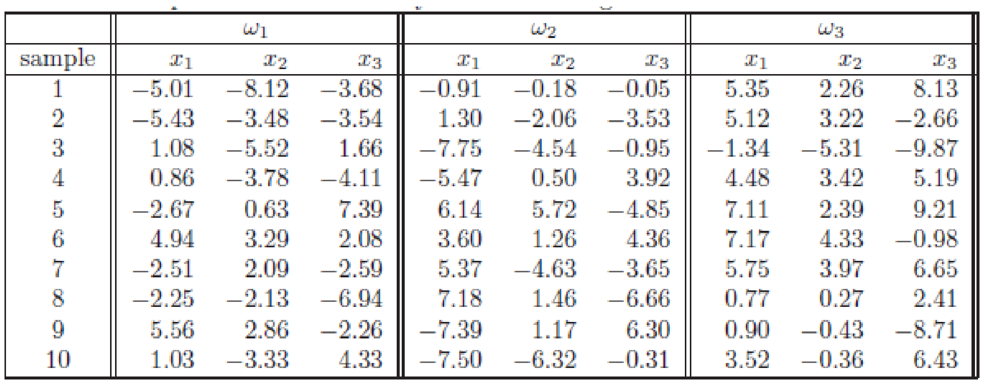


**Exercise 1.** *Consider the problem of classifying 10 samples from dataset in sample.txt and assume $P(\omega_i) = 1/3$ for $i = 1, 2, 3$.*

1. *What is the Mahalanobis distance between each of the following test points and each of the category means:* $(1\,2\,1)^T$ $(5\,3\,2)^T$ $(0\,0\,0)^T$ $(1\,0\,0)^T$*?*

2. *Classify those points.*

3. *Assume instead that* $P(\omega_1) = 0.8$, $P(\omega_2) = P(\omega_3) = 0.1$, *and classify the test points again.*



| | $\omega_1$ | | | $\omega_2$ | | | $\omega_3$ | | |
|---|---|---|---|---|---|---|---|---|---|
| sample | $x_1$ | $x_2$ | $x_3$ | $x_1$ | $x_2$ | $x_3$ | $x_1$ | $x_2$ | $x_3$ |
| 1 | −5.01 | −8.12 | −3.68 | −0.91 | −0.18 | −0.05 | 5.35 | 2.26 | 8.13 |
| 2 | −5.43 | −3.48 | −3.54 | 1.30 | −2.06 | −3.53 | 5.12 | 3.22 | −2.66 |
| 3 | 1.08 | −5.52 | 1.66 | −7.75 | −4.54 | −0.95 | −1.34 | −5.31 | −9.87 |
| 4 | 0.86 | −3.78 | −4.11 | −5.47 | 0.50 | 3.92 | 4.48 | 3.42 | 5.19 |
| 5 | −2.67 | 0.63 | 7.39 | 6.14 | 5.72 | −4.85 | 7.11 | 2.39 | 9.21 |
| 6 | 4.94 | 3.29 | 2.08 | 3.60 | 1.26 | 4.36 | 7.17 | 4.33 | −0.98 |
| 7 | −2.51 | 2.09 | −2.59 | 5.37 | −4.63 | −3.65 | 5.75 | 3.97 | 6.65 |
| 8 | −2.25 | −2.13 | −6.94 | 7.18 | 1.46 | −6.66 | 0.77 | 0.27 | 2.41 |
| 9 | 5.56 | 2.86 | −2.26 | −7.39 | 1.17 | 6.30 | 0.90 | −0.43 | −8.71 |
| 10 | 1.03 | −3.33 | 4.33 | −7.50 | −6.32 | −0.31 | 3.52 | −0.36 | 6.43 |

---

```
% Gaussian noise example.
[n,m] = size(samples);
for i=1:3
    mu{i} = mean(samples(:, (i-1)*3+1:i*3))';
    sigma{i} = zeros(3);
    for j=1:n
        sigma{i} = sigma{i} + ... %The ... continues the line
            (samples(j,(i-1)*3+1:i*3)' - mu{i}) ...
            * (samples(j,(i-1)*3+1:i*3)' - mu{i})';
    end
    sigma{i} = sigma{i}./n;
end
s = [1 2 1; 5 3 2; 0 0 0; 1 0 0]'
for j=1:size(s,2)
    for i=1:3
        d = sqrt((s(:,j)-mu{i})'*inv(sigma{i})*(s(:,j)-mu{i}));
        fprintf('Mahal. dist. for class %d and point %d: %f\n', i,
            j, d);
    end
end
```

```
pw(1,:) =[1/3 0.8];
pw(2,:) =[1/3 0.1];
pw(3,:) =[1/3 0.1];
for p=1:2
    fprintf('\n\n\n\n');
    for j=1:size(s,2)
        class = 0; max_gi = -1000000;
        for i=1:3
            di = (s(:,j)-mu{i})'*inv(sigma{i})*(s(:,j)-mu{i});
            gi = -0.5*di - 1.5*log(2*pi) - 0.5*log(det(sigma{i})) +
                log(pw(i,p));
            if gi > max_gi,
                max_gi = gi;
                class = i;
            end
        end
        fprintf('Point %d classified in category %d\n', j, class);
    end
end
```

---

**Exercise 2.** *Consider the problem of classifying 10 samples from dataset in sample.txt. Assume that the underlying distributions are normal.*

1. *Assume that the prior probabilities for the first two categories are equal ($P(\omega_1) = P(\omega_2) = 1/2$ and $P(\omega_3) = 0$ and design a dichotomizer for those two categories using only the $x_1$ feature value.*

2. *Determine the empirical training error on your samples, that is, the percentage of points misclassified.*

3. *Use the Bhattacharyya bound to bound the error you will get on novel patterns drawn from the distribution.*

4. *Repeat all of the above but now use features $x_1$ and $x_2$.*

5. *Repeat, but use all three feature values.*

6. *Discuss your results. In particular, is it ever possible for a finite set of data that the empirical error might be larger for more data dimensions?*

# MTSC 852 - Pattern Recognition
# Lab Session
# Parametric Estimation

Sokratis Makrogiannis, Ph.D.

October 15, 2015

## Contents

## 1 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 $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$
- This is equivalent to finding the value $\hat{\theta}$, such that $\hat{\theta} = arg\max p(\mathcal{D}|\theta)$
- Assuming that samples in $\mathcal{D}$ are drawn independently,

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

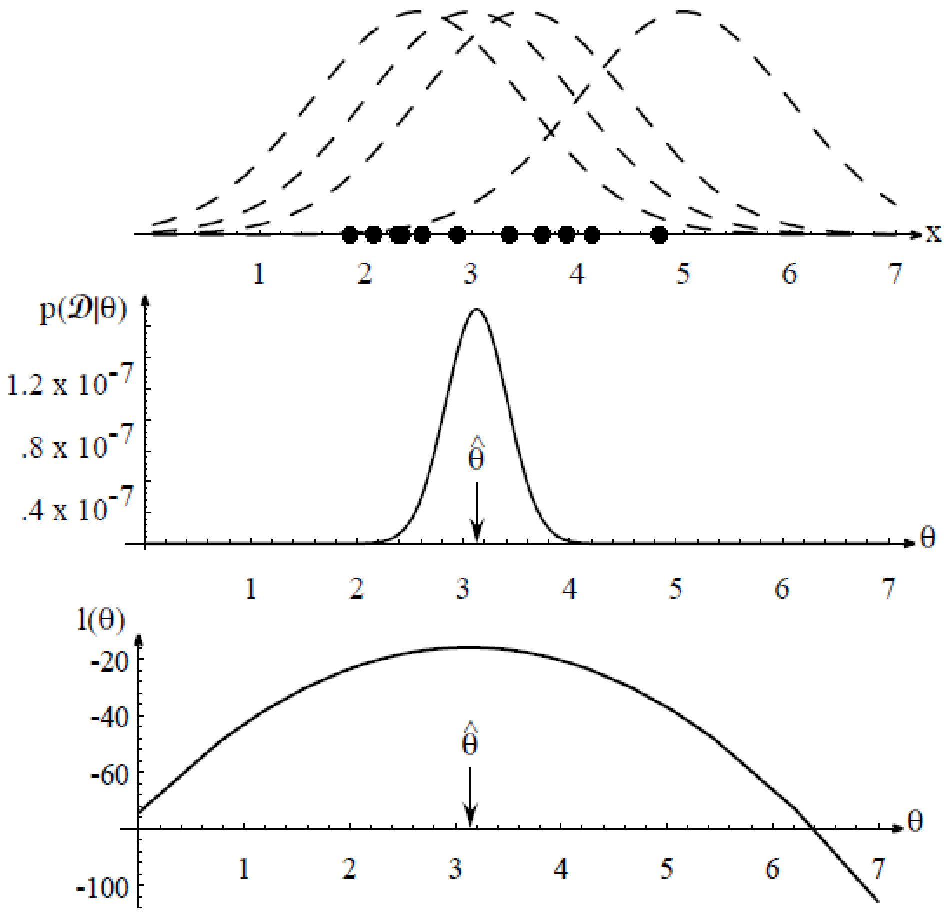


Figure 1: ML technique for parameter estimation.

- If $f(\theta) = p(\mathcal{D}|\theta)$ 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 $\nabla_\theta = [\frac{\partial}{\partial\theta_1} \quad \frac{\partial}{\partial\theta_2} \quad \cdots \quad \frac{\partial}{\partial\theta_p}]^T$

- 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

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

## 2 Bias of Maximum-Likelihood Estimation Technique

- Maximum likelihood estimates for a Gaussian with unknown $\mu$ and unknown $\Sigma$:

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

- 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

**Exercise 1.** *Show that the maximum likelihood (ML) estimation of the mean for a Gaussian is unbiased but the ML estimate of variance is biased (i.e., slightly wrong). Show how to correct this variance estimate so that it is unbiased.*

1. *For this part you will write a program with Matlab to explore the biased and unbiased ML estimations of variance for a Gaussian distribution. Find the data for this problem in the supplementary file ch3_dhs_samples_02.dat. This file contains n=5000 samples from a 1-dimensional Gaussian distribution.*

   (a) *Write a program to calculate the ML estimate of the mean, and report the output.*

   (b) *Write a program to calculate both the biased and unbiased ML estimate of the variance of this distribution. For n=1 to 5000, plot the biased and unbiased estimates of the variance of this Gaussian. This is as if you are being given these samples sequentially, and each time you get a new sample you are asked to re-evaluate your estimate of the variance. Give some interpretation of your plot.*

```
function [Mu, Sigma] = ch3_MLE_Biased(DataMatrix)
% Data: DxN matrix.

[D, N] = size(DataMatrix);

Mu = mean(DataMatrix, 2);

Sigma = (1/N) * ((DataMatrix - repmat(Mu, 1, N)) * (DataMatrix -
    repmat(Mu, 1, N))');

end
```

```
function [Mu, SigmaBiased, SigmaCorrected] =
    ch3_MLE_Unbiased(DataMatrix)
% Data: DxN matrix.

[D, N] = size(DataMatrix);
```

```
[Mu, SigmaBiased] = ch3_MLE_Biased(DataMatrix);

SigmaCorrected = (N/(N-1)) * SigmaBiased;

end
```

---

```
% PR_03_Lab Exercise 1

% Load data.
A = load('ch3_dhs_samples_02.dat');

% Get number of samples and dimensionality.
[N, D] = size(A);

% Initialize variables.
Mu = zeros(N, 1);
SigmaBiased = zeros(N, 1);
SigmaCorrected = zeros(N, 1);

% For each sample:
for i=2:N
    % Compute MLE estimates

    B = A(1:i)';

    [Mu(i), SigmaBiased(i), SigmaCorrected(i)] =
        ch3_MLE_Unbiased(B);

    fprintf('N = %d, \t MLE mean = %f, \t MLE Sigma = %f, MLE
        corrected Sigma = %f\n', ...
        i, Mu(i), SigmaBiased(i), SigmaCorrected(i));
end


% Plot the estimates.
index = 2:N;
figure, plot(index, Mu(index), 'Linewidth', 4); title('MLE Mean');
    xlabel('N'); grid on;
```

```
saveas(gcf, 'MLE_Mean_Lab.png')

figure, plot(index, SigmaBiased(index), 'Linewidth', 4);
    title('MLE Sigma'); xlabel('N'); grid on; hold on;
plot(index, SigmaCorrected(index), 'g--', 'Linewidth', 4);
legend('Biased', 'Corrected');
saveas(gcf, 'MLE_Sigma_Lab.png')
```

## 3 Bayes Classifier and Maximum-Likelihood Estimation

**Exercise 2.** *Generate 10,000 samples from each 2D distribution specified by the following parameters:* $\mu_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ $\Sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ *and* $\mu_2 = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$ $\Sigma_2 = \begin{pmatrix} 4 & 0 \\ 0 & 16 \end{pmatrix}$.

1. *Assuming* $P(\omega_1) = P(\omega_2)$
   (a) *Design Bayes classifier for minimum error.*
   (b) *Plot the Bayes decision boundary together with the generated samples.*
   (c) *Classify the samples by the classifier and count the number of misclassified samples.*

2. *Assume that you do not know the true parameters of the Gaussian distributions and that you need to estimate them from the training data using the Maximum Likelihood (ML) approach.*
   (a) *Using the same 10,000 samples from part 1, estimate the parameters of each distribution using ML and classify all samples assuming* $P(\omega_1) = P(\omega_2)$*; then, count the number of misclassified samples and compare your results to those obtained in part 1.*
   (b) *Repeat experiment 2a using 1/10 of the samples (randomly chosen) to estimate the parameters of each distribution using ML and classify all samples assuming* $P(\omega_1) = P(\omega_2)$*; then, count the number*

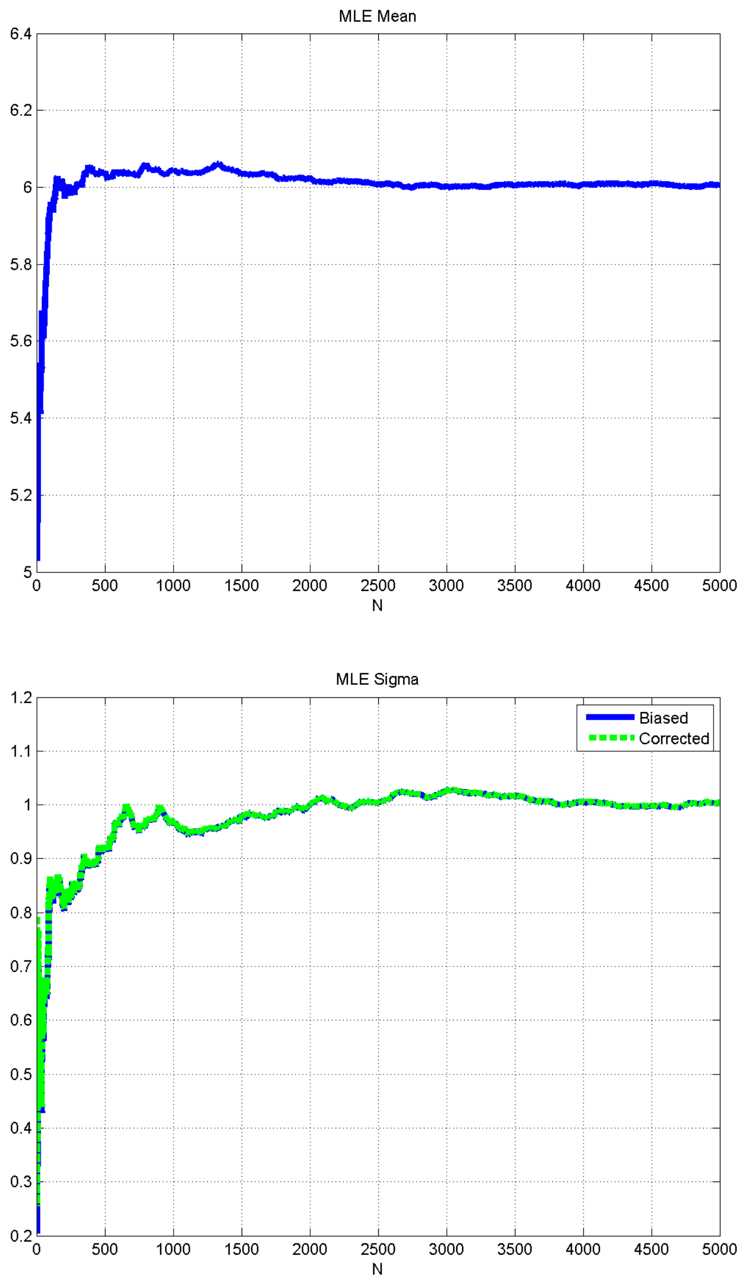


Figure 2: ML Mean estimation (top). ML Sigma estimation bias and correction (bottom).

*of misclassified samples and compare your results to those obtained in experiment 3a.*

```
% PR_03_Lab Exercise 2

% Initialize.
clear all;

% 1: Generate Data
N = 10000;

% Class 1
mu1 = [1 1];
Sigma1 = [1 0; 0 1];
R1 = chol(Sigma1);
X1 = repmat(mu1, N, 1) + randn(N, 2) * R1;
ClassLabels = ones(N, 1);

% Class 2
mu2 = [4 4];
Sigma2 = [4 0; 0 16];
R2 = chol(Sigma2);
X2 = repmat(mu2, N, 1) + randn(N, 2) * R2;

classification_rate = BayesMLClassifier(X1, X2, mu1, mu2, Sigma1,
    Sigma2);
```

```
function classification_rate = BayesMLClassifier(X1, X2, mu1, mu2,
    Sigma1, Sigma2)
% Assumes equal number of samples for each class.

% Prep work.
N = size(X1, 1);
X = [X1; X2];
labels = [zeros(N, 1); ones(N, 1)];

% Design Bayes MAP classifier.
```

```
mahalanobis_distance = @(x, Sigma_inverse, mu) ((x - mu) *
    Sigma_inverse * (x - mu)');

Sigma1_inverse = inv(Sigma1);
for i=1:2*N
    mahalanobis_distance_1(i) = mahalanobis_distance(X(i, :),
        Sigma1_inverse, mu1) ;
    g1(i) = -0.5 * log(det(Sigma1)) - 0.5 *
        mahalanobis_distance_1(i);
end

Sigma2_inverse = inv(Sigma2);
for i=1:2*N
    mahalanobis_distance_2(i) = mahalanobis_distance(X(i, :),
        Sigma2_inverse, mu2) ;
    g2(i) = -0.5 * log(det(Sigma2)) - 0.5 *
        mahalanobis_distance_2(i);
end

dg = g2 - g1;

% Classify samples and count the number of misclassified samples.

decision = dg > 0;

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

% Plot Bayes decision boundary together with generated samples

minX = min(X);
maxX = max(X);

[x, y] = meshgrid(minX(1):maxX(1), minX(2):maxX(2));
Xgrid = [x(:), y(:)];
N2 = size(Xgrid, 1);
for i=1:N2
    mahalanobis_distance_1(i) = mahalanobis_distance(Xgrid(i, :),
        Sigma1_inverse, mu1) ;
    g21(i) = -0.5 * log(det(Sigma1)) - 0.5 *
        mahalanobis_distance_1(i);
end

for i=1:N2
    mahalanobis_distance_2(i) = mahalanobis_distance(Xgrid(i, :),
        Sigma2_inverse, mu2) ;
    g22(i) = -0.5 * log(det(Sigma2)) - 0.5 *
        mahalanobis_distance_2(i);
end

dg2 = g22 - g21;
dg2 = reshape(dg2', size(x));

figure, plot(X1(:,1), X1(:,2), 'bo'); hold on;
plot(X2(:,1), X2(:,2), 'gx');

[c, h] = contour(x, y, dg2); clabel(c, h); colorbar;

saveas(gcf, 'Bayes_Classifier_Lab.png')

end
```

**Exercise 3.** *Repeat the previous exercise using the following parametric normal distributions:*

$$\mu_1 = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \Sigma_1 = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \textit{ and } \mu_2 = \begin{pmatrix} 4 \\ 4 \end{pmatrix} \Sigma_2 = \begin{pmatrix} 4 & 2 \\ 2 & 16 \end{pmatrix}.$$

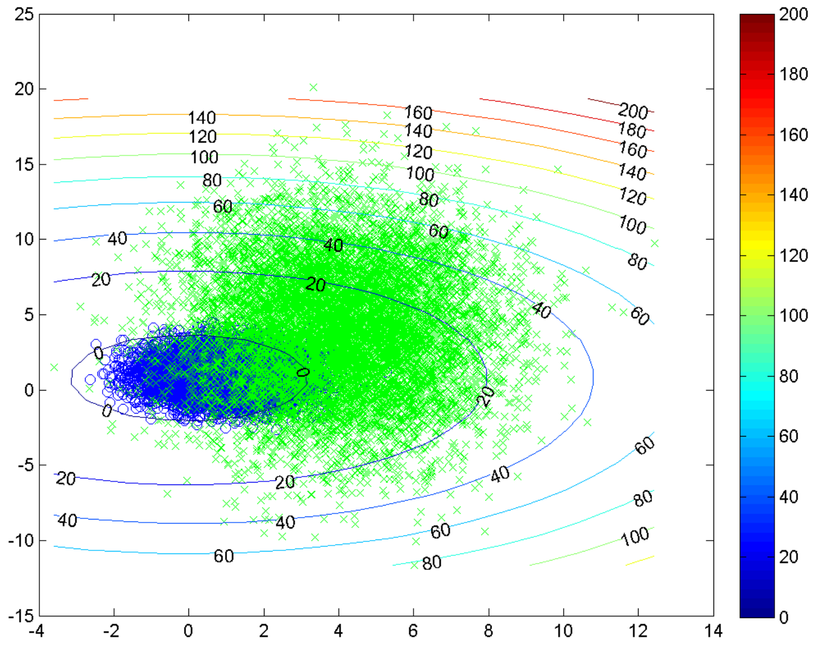


Figure 3: Bayes discriminant function example.

# 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 $\sigma$ 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 $\sigma$ 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 $p(\mu|\mathcal{D})$:

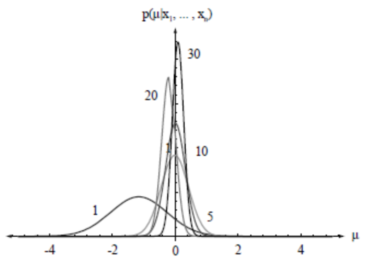
$$p(\mu|\mathcal{D}) = \frac{p(\mathcal{D}|\mu)p(\mu)}{\int p(\mathcal{D}|\mu)p(\mu)d\mu}$$

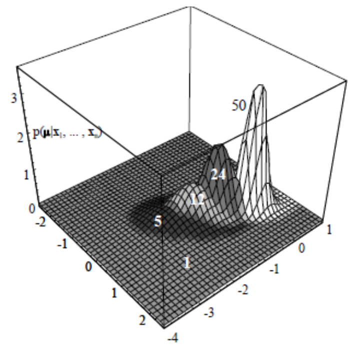
- Let samples $\mathcal{D} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}\}$ be independently drawn. Then: $p(\mathcal{D}|\mu) = \prod_{i=1}^{n} p(x_i|\mu)$
- Then: $p(\mu|\mathcal{D}) = \alpha \cdot \prod_{i=1}^{n} p(x_i|\mu)p(\mu)$
- According to assumptions:

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

- 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,
$$\mu_n = \left(\frac{n\sigma_0^2}{n\sigma_0^2+\sigma^2}\right)\hat{\mu}_n + \frac{\sigma^2}{n\sigma_0^2+\sigma^2}\mu_0, \quad \sigma_n^2 = \frac{\sigma_0^2\sigma^2}{n\sigma_0^2+\sigma^2}$$
$$\hat{\mu}_n = (1/n)\sum_{i=1}^{n} x_i$$
- $\sigma_n$ 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 $\mu_n$. This process is called *Bayesian learning*
- If $\sigma_0 \neq 0$, then $\mu_n$ 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 $\mu_n$ 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,* $\mu_0, \sigma_0, \sigma$ *and training set* $\mathcal{D} = \{x_1, x_2, \ldots, x_n\}$.

2. *Estimate* $\sigma$ *for the* $x_2$ *component of* $\omega_3$ *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.*

| | $\omega_1$ | | | $\omega_2$ | | | $\omega_3$ | | |
|---|---|---|---|---|---|---|---|---|---|
| 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

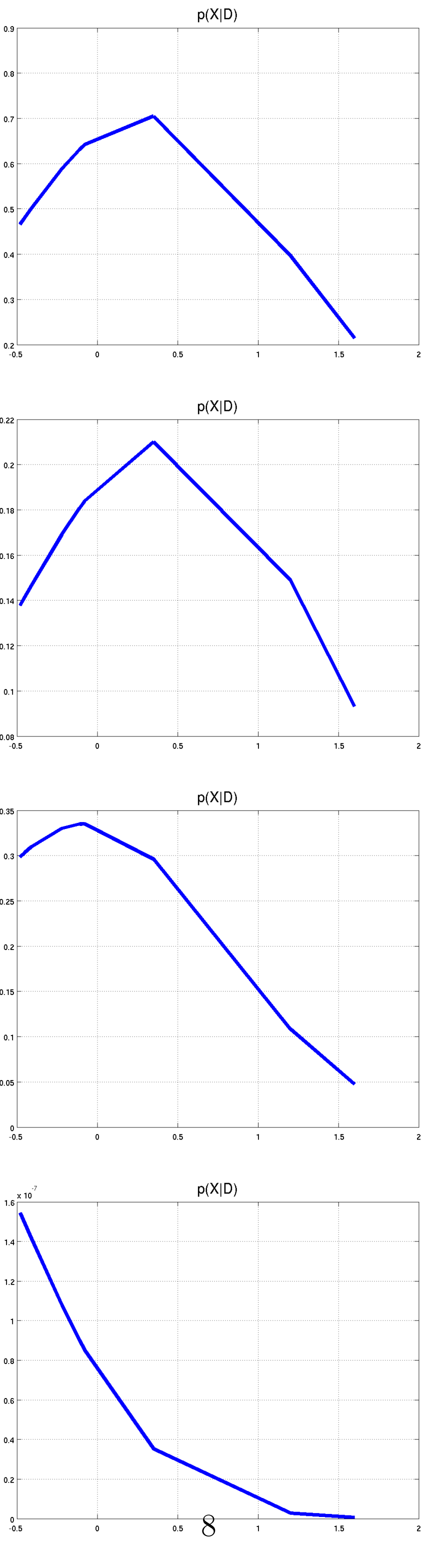


Figure 1: Bayesian parameter estimation example.

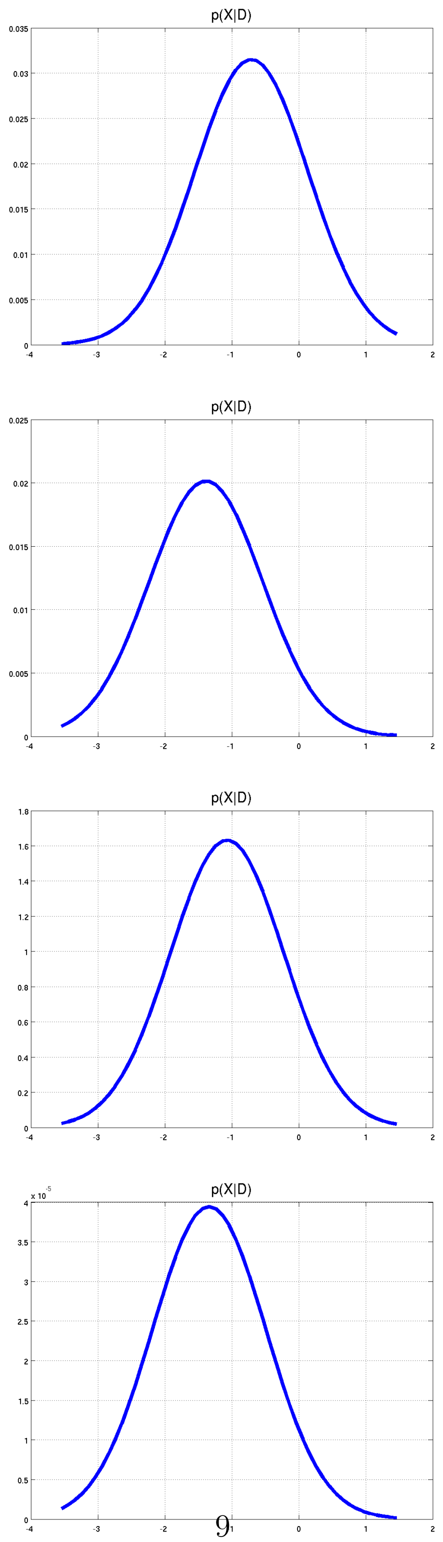


Figure 2: Bayesian parameter estimation example over a densely sampled space.

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

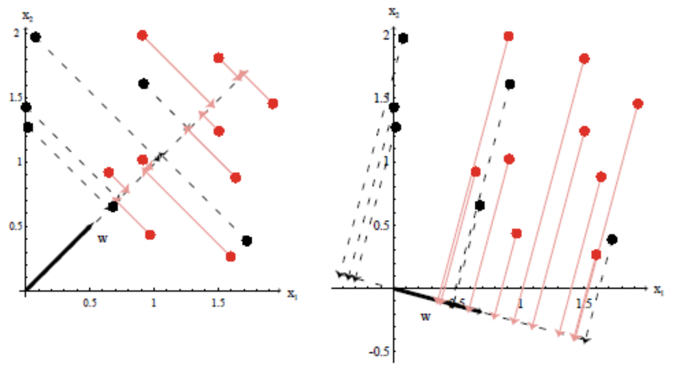


Figure 3: Projection of data onto different directions defined by $\boldsymbol{w}$. Observe that projection displayed in the right figure produces greater separability than the projection displayed in the left figure

## 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 $\Sigma$, 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 $\omega_2$ and $\omega_3$ 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.
```

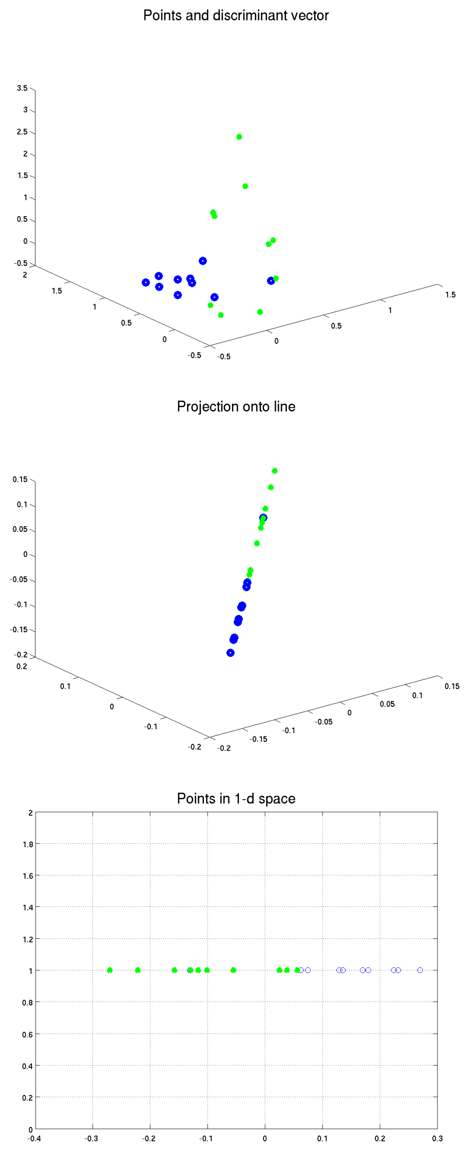


Figure 4: Fisher linear discriminant example.

# MTSC 852 - Pattern Recognition
# Lab Session
# Nonparametric Estimation

Sokratis Makrogiannis, Ph.D.

November 6, 2015

## Contents

# 1 Nearest Neighbor Classifier

## 1.1 $k_n$ Nearest Neighbor Estimation

- In this method, to estimate $p(\boldsymbol{x})$ from $n$ training samples we grow the region $\mathcal{R}_n$ 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

## 1.2 The Nearest Neighbor Rule

- Let $\mathcal{D}^n$ be a set of training points, or prototypes, $\mathcal{D}^n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and let $\boldsymbol{x}$ be a test point that is closest to the training point $\boldsymbol{x}' \in \mathcal{D}^n$

- 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

- 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 $\omega_m$ 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

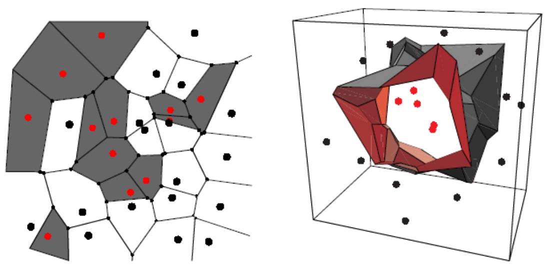


Figure 1: Voronoi Tesselation using NN rule

## 2 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

### 2.1 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: $D(\boldsymbol{a}, \boldsymbol{b}) + D(\boldsymbol{b}, \boldsymbol{c}) \geq D(\boldsymbol{a}, \boldsymbol{c})$

### 2.2 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

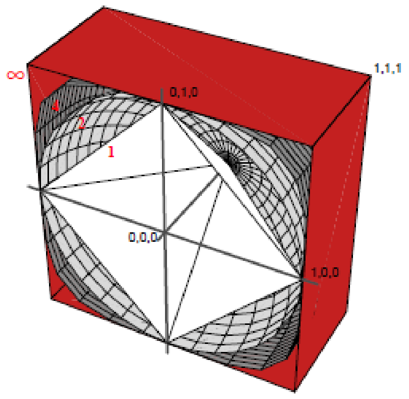


Figure 2: Isosurfaces for $L_1$ (white), $L_2$ (light gray), $L_4$ (dark gray), and $L_\infty$ (pink)

- $L_1$ 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
- $L_\infty$ norm: corresponds to the maximum of the distances between the projections of $\boldsymbol{a}$ and $\boldsymbol{b}$ onto each of the $d$ coordinate axes

**Exercise 1.** *Consider nearest-neighbor classifiers employing different values of $k$ in the $L_k$ norm or Minkowski metric.*

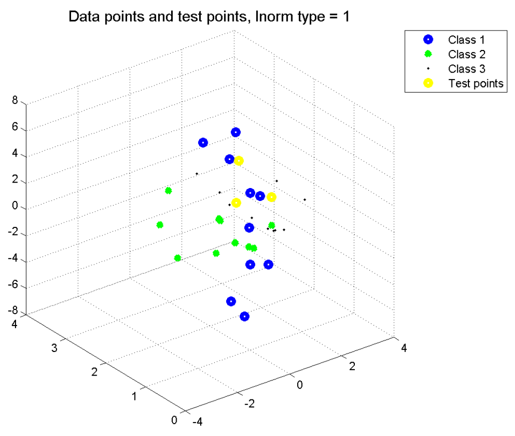
1. *Write a program to implement a nearest-neighbor classifier for $c$ categories, using the Minkowski metric or $L_k$ norm, where $k$ can be selected at classification time.*
2. *Use the three dimensional data in the table above to classify the following points using the $L_k$ norm for $k = 1, 2, 4$ and $\infty$: $(2.21, 1.9, 0.43)^T$, $(0.15, 1.17, 6.19)^T$ and $(0.01, 1.34, 2.60)^T$.*

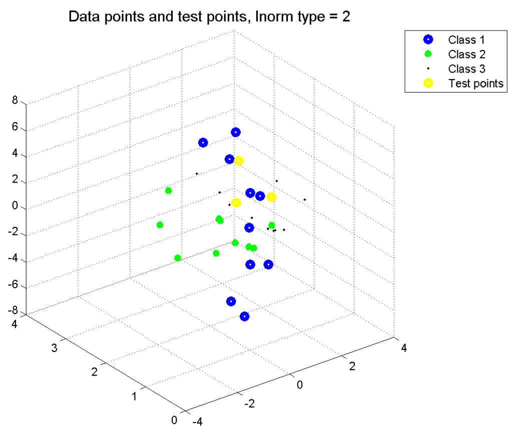
---

```
% Nearest neighbor classifier using different $L_k$ norms.
% Load data.
clear all;
A = load('ch4_dhs_samples.dat');
D{1} = A(:, 1:3);
D{2} = A(:, 4:6);
D{3} = A(:, 7:9);
[d, n] = size(D{1});

% Set test data.
test_points = [ 2.21 1.9 0.43; -0.15 1.17 6.19; 0.01 1.34 2.60];
n_test = size(test_points, 1);

% For lnorm_type=1,2,4,\infty
for lnorm_type=[1,2,4]

    % Plot data points.
    figure, plot3(D{1}(:,1), D{1}(:,2), D{1}(:,3), 'bo',
        'linewidth', 4); hold on;
    plot3(D{2}(:,1), D{2}(:,2), D{2}(:,3), 'gx', 'linewidth', 4);
    plot3(D{3}(:,1), D{2}(:,2), D{3}(:,3), 'k.', 'linewidth', 4);
    % For each test point:
    for i=1:n_test
        % Estimate Parzen kernel class-conditional densities on
            training points for
        % each class and find arg max.
        [predicted_class, min_distance] =
            ch4_nn_classification(test_points(i,:), D, lnorm_type);

        % Display result on screen.
        fprintf('lnorm_type= %.2f \t test point = %.3f,%.3f,%.3f \t
            distance= %.3f \t predicted class: %d \n', lnorm_type,
            test_points(i,:), min_distance, predicted_class);

        % Plot test point.
        plot3(test_points(i,1), test_points(i,2), test_points(i,3),
            'yo', 'linewidth', 4);

    end
    fprintf('\n');
    grid on;
    title(['Data points and test points, lnorm type = ',
        num2str(lnorm_type)], 'fontsize', 14);
    legend('Class 1', 'Class 2', 'Class 3', 'Test points');
    saveas(gcf, ['NN_Lab_', 'L', num2str(lnorm_type), '.png'])

end
```

---

```
function [predicted_class, min_distance] =
    ch4_nn_classification(test_point, training_points_by_class,
    lnorm_type)
% Classification using NN rule.
% syntax: [predicted_class, min_distance] =
% ch4_nn_classification(test_point, training_points_by_class, knn,
    lnorm_type);
% S. Makrogiannis, Delaware State Univ, 11/2015.

% Get the pattern and class info.
c = length(training_points_by_class);
[n, d] = size(training_points_by_class{1});
minkowski_metric = @(x, k) ( sum(abs(x).^k).^(1/k) );
distance_vector = zeros(c,n);

% Compute distances using L_k norm
for i=1:c
    for j=1:n
        distance_vector(i,j) = minkowski_metric(test_point -
            training_points_by_class{i}(j,:), lnorm_type);
    end
end
% Find nearest neighbor.
[min_distance, min_index] = min(distance_vector(:));
[predicted_class, ~] = ind2sub([c, n], min_index);
end
```





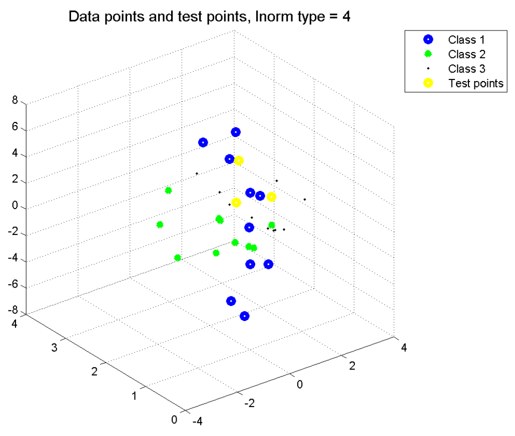


Figure 3: NN classifier.

# 3 Parzen Density Classifier

## 3.1 Parzen windows

- The Parzen window method defines a window that may be a function of the number of data points
- More specifically, $\mathcal{R}_n$ 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 $\mathcal{R}_n$ denoted by $k_n$ we use a window function:

  $$\phi(\boldsymbol{u}) = \begin{cases} 1 & |u_j| \leq 1/2 \quad j = 1, \ldots, d \\ 0 & \text{otherwise} \end{cases}$$
- Based on window function definition, 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: $p_n(\boldsymbol{x}) = (1/n) \sum_{i=1}^{n} \frac{1}{V_n} \phi\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right)$

## 3.2 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:

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

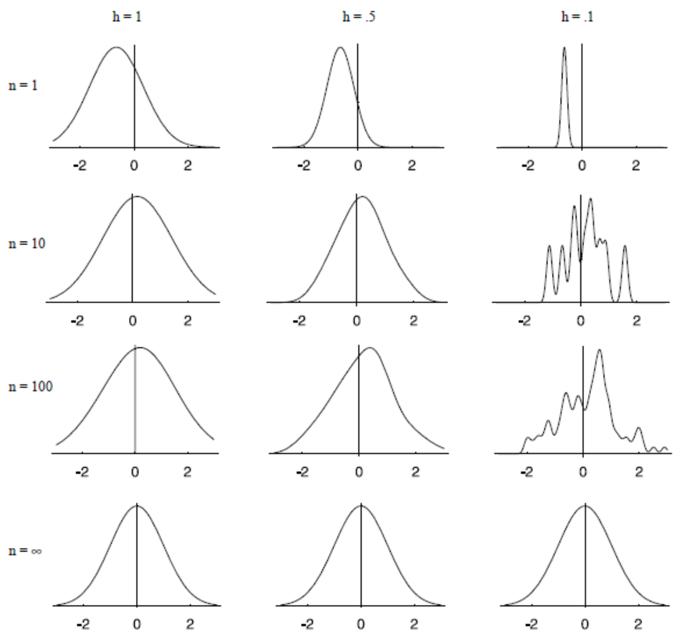


Figure 4: Parzen kernel density estimation for a univariate normal distribution versus the number of samples and window width. The contribution of each point to the density is more visible for smaller window widths. Larger $n$ improves density estimation

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

## 3.3 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

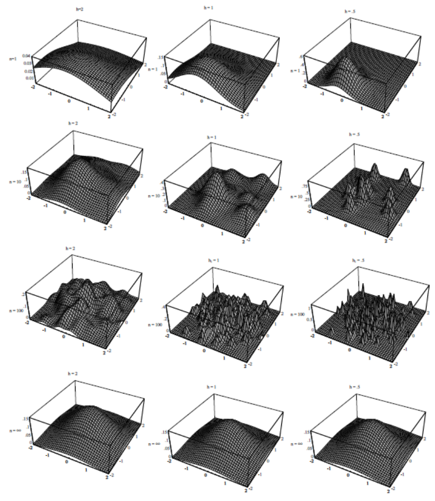


Figure 5: Parzen kernel density estimation for a bivariate normal distribution versus the number of samples and window width. Smaller window width produces "noisier" estimates for fixed $n$

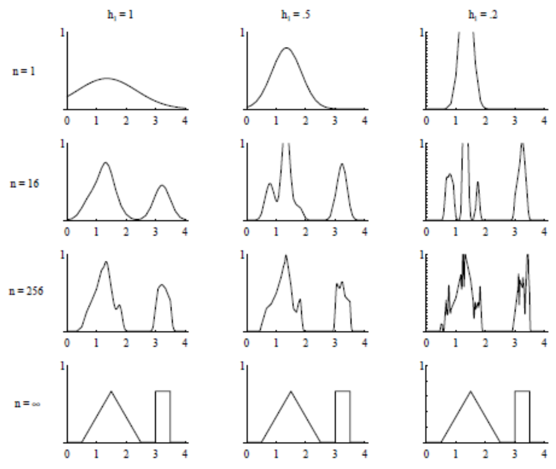


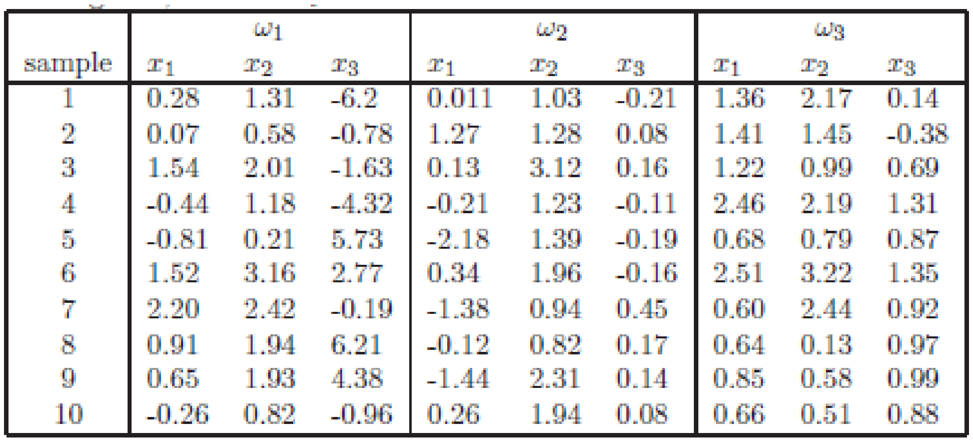
Figure 6: Parzen kernel density estimation for a mixture of a uniform and a triangular distribution. Observe that more samples improve estimation

- Gaussian windows are reasonable choices but it may take some experimentation to find the window size

**Exercise 2.** *Consider Parzen-window estimates and classifiers for points in the table above. Let your window function be a spherical Gaussian, i.e.,*

$$\phi((\boldsymbol{x} - \boldsymbol{x}_i)/h) \simeq \exp[-(\boldsymbol{x} - \boldsymbol{x}_i)^T(\boldsymbol{x} - \boldsymbol{x}_i)/(2h^2)]$$

1. *Write a program to classify an arbitrary test point x based on the Parzen window estimates. Train your classifier using the three-dimensional data from your three categories in the table above. Set h = 1 and classify the following three points:* $(0.50, 1.0, 0.0)^T$, $(0.31, 1.51, 0.50)^T$ *and* $(0.3, 0.44, 0.1)^T$.
2. *Repeat with* $h = 0.1$.



|  | $\omega_1$ |  |  | $\omega_2$ |  |  | $\omega_3$ |  |  |
|---|---|---|---|---|---|---|---|---|---|
| sample | $x_1$ | $x_2$ | $x_3$ | $x_1$ | $x_2$ | $x_3$ | $x_1$ | $x_2$ | $x_3$ |
| 1 | 0.28 | 1.31 | -6.2 | 0.011 | 1.03 | -0.21 | 1.36 | 2.17 | 0.14 |
| 2 | 0.07 | 0.58 | -0.78 | 1.27 | 1.28 | 0.08 | 1.41 | 1.45 | -0.38 |
| 3 | 1.54 | 2.01 | -1.63 | 0.13 | 3.12 | 0.16 | 1.22 | 0.99 | 0.69 |
| 4 | -0.44 | 1.18 | -4.32 | -0.21 | 1.23 | -0.11 | 2.46 | 2.19 | 1.31 |
| 5 | -0.81 | 0.21 | 5.73 | -2.18 | 1.39 | -0.19 | 0.68 | 0.79 | 0.87 |
| 6 | 1.52 | 3.16 | 2.77 | 0.34 | 1.96 | -0.16 | 2.51 | 3.22 | 1.35 |
| 7 | 2.20 | 2.42 | -0.19 | -1.38 | 0.94 | 0.45 | 0.60 | 2.44 | 0.92 |
| 8 | 0.91 | 1.94 | 6.21 | -0.12 | 0.82 | 0.17 | 0.64 | 0.13 | 0.97 |
| 9 | 0.65 | 1.93 | 4.38 | -1.44 | 2.31 | 0.14 | 0.85 | 0.58 | 0.99 |
| 10 | -0.26 | 0.82 | -0.96 | 0.26 | 1.94 | 0.08 | 0.66 | 0.51 | 0.88 |

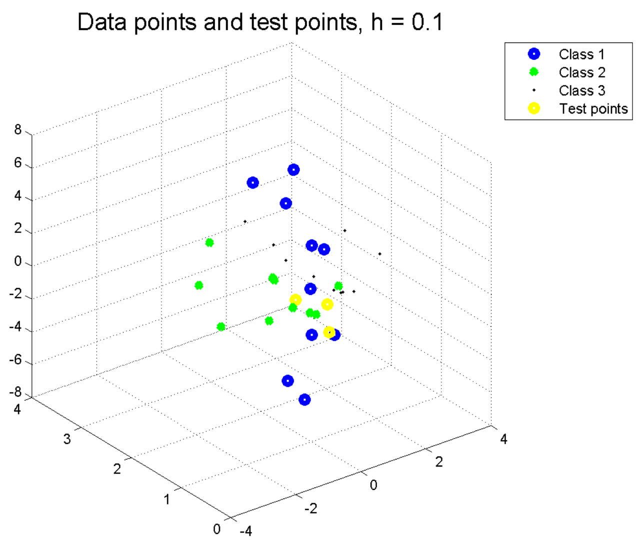
Table 1: Three-dimensional data sampled from three categories.

```
% Parzen kernel density estimation and classification.

% Load data.
clear all; close all;
A = load('ch4_dhs_samples.dat');
D{1} = A(:, 1:3);
```

```
D{2} = A(:, 4:6);
D{3} = A(:, 7:9);
[d, n] = size(D{1});

% test_points = [ 0.5 1 0; 0.31 1.51 -0.5; -0.3 0.44 -0.1];
test_points = [ 2.2 2.42 -0.19; 0.31 1.51 -0.5; -0.3 0.44 -0.1];
n_test = size(test_points, 1);

% Classification stage.

% For each h value:
for h=[1,2]

    % Plot data points.
    figure, plot3(D{1}(:,1), D{1}(:,2), D{1}(:,3), 'bo',
        'linewidth', 4); hold on;
    plot3(D{2}(:,1), D{2}(:,2), D{2}(:,3), 'gx', 'linewidth', 4);
    plot3(D{3}(:,1), D{2}(:,2), D{3}(:,3), 'k.', 'linewidth', 4);
    % For each test point:
    for i=1:n_test
        % Estimate Parzen kernel class-conditional densities on
            training points for
        % each class and find arg max.
        [predicted_class, max_density] =
            ch4_parzen_classification(test_points(i,:), D, h);

        % Display result on screen.
        fprintf('h= %.2f \t test point = %.3f,%.3f,%.3f \t density=
            %.3f \t predicted class: %d \n', h, test_points(i,:),
            max_density, predicted_class);

        % Plot test point.
        plot3(test_points(i,1), test_points(i,2), test_points(i,3),
            'yo', 'linewidth', 4);

    end
    fprintf('\n');

    grid on;
    title(['Data points and test points, h = ', num2str(h)],
       'fontsize', 18);
    legend('Class 1', 'Class 2', 'Class 3', 'Test points');
    saveas(gcf, ['Parzen_Density_Lab_', 'h_', num2str(h), '.png'])

end
```

```
function parzen_density = ch4_parzen_density(estimation_point,
    training_points, h)
% Nonparametric density estimation using Parzen kernels with
% Gaussian window function.
% syntax: parzen_density = ch4_parzen_density(estimation_point,
% training_points, h);
% S. Makrogiannis, Delaware State Univ, 11/2015.

% Get number of training points and dimensionality d.
[n, d] = size(training_points);
parzen_density = 0;

% Estimate density at estimation_point.
window_function = @(x) ( (1/sqrt(2*pi)) * exp( (-0.5) * x^2 ) );
minkowski_metric = @(x, k) ( sum(abs(x).^k).^(1/k) );
delta_function = @(x, x_tr, k, h, d) ( (1/h^d) * window_function(
    minkowski_metric(x - x_tr, k) / h ) );

% For each training point.
for i=1:n
    % Compute parzen density contributions.
    delta = delta_function(estimation_point, training_points(i, :),
       2, h, d);

    % Sum up to estimate density.
    parzen_density = parzen_density + delta;
end

parzen_density = parzen_density / n;

end
```

```
function [predicted_class, max_density] =
    ch4_parzen_classification(test_point, training_points_by_class,
    h)
% syntax: [predicted_class, estimated_density] =
% ch4_parzen_classification(test_point, training_points_by_class,
    h);
% Classification using Nonparametric density estimation using
    Parzen kernels with
% Gaussian window function.
% S. Makrogiannis, Delaware State Univ, 11/2015.

% Get the pattern and class info.
c = length(training_points_by_class);
[n, d] = size(training_points_by_class{1});

% Estimate Parzen kernel class-conditional densities on training
    points for each class.
for i=1:c
    estimated_density(i) = ch4_parzen_density(test_point,
        training_points_by_class{i}, h);
end

% Normalize to yield pdf.
estimated_density = estimated_density / sum(estimated_density);

% Because we have equal priors we can make a Bayesian decision
    using just the likelihoods.
% Find arg max.
[max_density, predicted_class] = max(estimated_density);

end
```



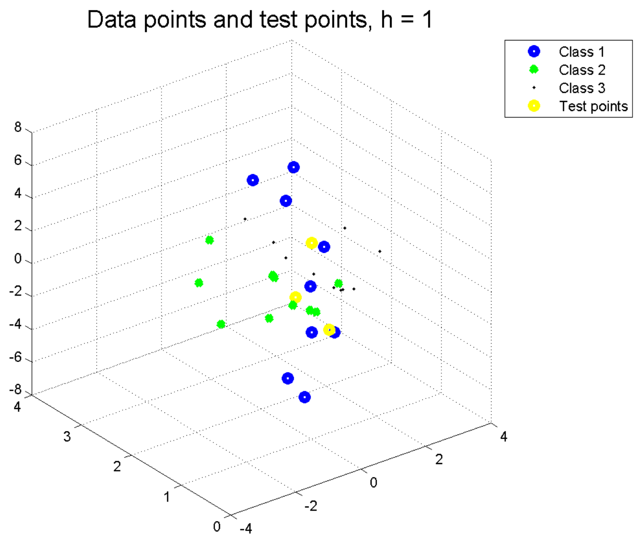


Figure 7: Parzen Density