IDIAP COMMUNICATION

IDIAP

Martigny - Valais - Suisse

EPFL lab session 1/2 : Introduction to Gaussian Statistics and Statistical Pattern Recognition

Sacha Krstulović

IDIAP–Com 01-06

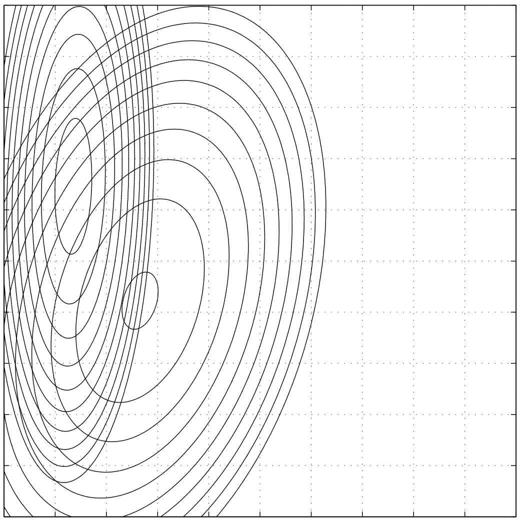
January 2001

Dalle Molle Institute for Perceptual Artificial Intelligence • P.O.Box 592 • Martigny • Valais • Switzerland

phone +41 − 27 − 721 77 11
fax +41 − 27 − 721 77 12
e-mail secretariat@idiap.ch
internet http://www.idiap.ch

Ecole Polytechnique Fédérale de Lausanne

Session 1/2 : Introduction to Gaussian Statistics and Statistical Pattern Recognition



Course :	Speech processing and speech recognition	
Teacher :	Prof. Hervé Bourlard	`bourlard@idiap.ch`
Assistants :	Sacha Krstulović Mathew Magimai-Doss	`sacha@idiap.ch` `mathew@idiap.ch`

Guidelines

The following lab manual is structured as follows :

- each section corresponds to a theme
- each subsection corresponds to a separate experiment.

The subsections begin with useful formulas and definitions that will be put in practice during the experiments. These are followed by the description of the experiment and by an example of how to realize it in MATLAB.

If you follow the examples literally, you will be able to progress into the lab session without worrying about the experimental implementation details. If you have ideas for better MATLAB implementations, you are welcome to put them in practice provided you don't loose too much time : remember that a lab session is no more than 3 hours long.

The subsections also contain questions that you should think about. Corresponding answers are given right after, in case of problem. You can read them right after the question, *but* : the purpose of this lab is to make you

Think !

If you get lost with some of the questions or some of the explanations, DO ASK the assistants or the teacher for help : they are here to make the course understood. There is no such thing as a stupid question, and the only obstacle to knowledge is laziness.

Have a nice lab;

Teacher & Assistants

Before you begin...

If this lab manual has been handed to you as a hardcopy :

1. get the lab package from
   `ftp.idiap.ch/pub/sacha/labs/Session1.tgz`
2. un-archive the package :
   `% gunzip Session1.tgz`
   `% tar xvf Session1.tar`
3. change directory :
   `% cd session1`
4. start MATLAB :
   `% matlab`

Then go on with the experiments...

This document was created by : Sacha Krstulović (sacha@idiap.ch).
This document is currently maintained by : Sacha Krstulović (sacha@idiap.ch). Last modification on November 27, 2001.
This document is part of the package Session4.tgz available by ftp as : ftp.idiap.ch/pub/sacha/labs/Session1.tgz .

Contents

1 Gaussian statistics

1.1 Samples from a Gaussian density

Useful formulas and definitions :

- The *Gaussian probability density function* (Gaussian pdf) for the d-dimensional random variable $x \circlearrowleft \mathcal{N}(\mu, \Sigma)$ (i.e. variable x following the Gaussian, or Normal, probability law) is given by :

$$g_{(\mu,\Sigma)}(x) = \frac{1}{\sqrt{2\pi}^d \sqrt{\det(\Sigma)}} \, e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

  where μ is the mean vector and Σ is the variance-covariance matrix. μ and Σ are the *parameters* of the Gaussian distribution. *Speech features* (also referred as *acoustic vectors*) are examples of d-dimensional variables, and it is usually assumed that they follow a Gaussian distribution.

- If $x \circlearrowleft \mathcal{N}(0, I)$ (x follows a normal law with zero mean and unit variance; I denotes the identity matrix), and if $y = \sqrt{\Sigma}\, x + \mu$, then $y \circlearrowleft \mathcal{N}(\mu, \Sigma)$.

- $\sqrt{\Sigma}$ defines the *standard deviation* of the random variable x (*écart-type* in French). Beware : this square root is meant in the *matrix sense.*

Experiment :

Generate a sample X of N points, i.e. $X = \{x_1, x_2, \cdots, x_N\}$, with $N = 10000$, coming from a 2-dimensional Gaussian process that has mean :

$$\mu = \begin{pmatrix} 730 \\ 1090 \end{pmatrix}$$

and variance :

1. 8000 for both dimensions (*spherical process*) (sample X_1) :

$$\Sigma_1 = \begin{bmatrix} 8000 & 0 \\ 0 & 8000 \end{bmatrix}$$

2. expressed as a *diagonal* covariance matrix (sample X_2) :

$$\Sigma_2 = \begin{bmatrix} 8000 & 0 \\ 0 & 18500 \end{bmatrix}$$

3. expressed as a *full* covariance matrix (sample X_3) :

$$\Sigma_3 = \begin{bmatrix} 8000 & 8400 \\ 8400 & 18500 \end{bmatrix}$$

Use the function `gausview` (`>> help gausview`) to plot the results as clouds of points in the 2-dimensional plane, and to view the corresponding 2-dimensional probability density functions (pdfs) in 2D and 3D.

Example :

```
>> N = 10000;
>> mu = [730 1090]; sigma_1 = [8000 0; 0 8000];
>> X1 = randn(N,2) * sqrtm(sigma_1) + repmat(mu,N,1);
>> gausview(X1,mu,sigma_1,'Sample X1');
```

Repeat the three previous steps for the two other Gaussians. Use the radio buttons to switch the plots on/off. Use the "view" buttons to switch between 2D and 3D. Use the mouse to rotate the plot.

Note : if you don't know what one of the cited MATLAB command does, use
`>> help` *command*
to get some help. If the help doesn't make it clearer, ask the assistants.

Question :

By simple inspection of 2D views of the data and of the corresponding pdf contours, how can you tell which sample corresponds to a spherical process, which sample corresponds to a pdf with a diagonal covariance, and which to a pdf with a full covariance ?

Answer :

For Σ_3, the covariance of both dimensions is not null, so the principal axes of the ellipses are not aligned with the abscissa and ordinate axes.

The cloud of points and the pdf contours corresponding to Σ_2 are elliptic, with their axes parallel to the abscissa and ordinate axes. This is because the first and the second dimension are still independent (their covariance is null again), but this time the variance is different along both dimensions.

They are orthogonal *in the statistical sense, which transposes to a geometric sense (the expectation is a scalar product of random variables; a null scalar product means orthogonality). Intuitively, you can also consider that a null covariance means no sharing of information between the two dimensions : they can evolve independently in a Gaussian way along their respective axes. Besides, the variance is the same in both dimensions, which indicates an equivalent spread of the data along both axes. Hence the circular blob of data points and the name "spherical process".*

$$\mathcal{E}\left[(x_{d1} - \mu_{d1})^T (x_{d2} - \mu_{d2})\right] = 0$$

The cloud of points and the pdf contours corresponding to Σ_1 are circular, because in this case the first and the second dimension of the vectors are independent. As a matter of fact, they have a null covariance :

1.2 Gaussian modeling: mean and variance of a sample

Useful formulas and definitions:

- Mean estimator: $\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i$

- Unbiased covariance estimator: $\hat{\Sigma} = \frac{1}{N-1}\ \sum_{i=1}^{N}(x_i - \mu)^T(x_i - \mu)$

Experiment:

Take the set X_3 of 10000 points generated from $\mathcal{N}(\mu, \Sigma_3)$. Compute an estimate $\hat{\mu}$ of its mean and an estimate $\hat{\Sigma}$ of its variance:

1. with all the available points $\hat{\mu}_{(10000)} =$ $\hat{\Sigma}_{(10000)} =$

2. with only 1000 points $\hat{\mu}_{(1000)} =$ $\hat{\Sigma}_{(1000)} =$

3. with only 100 points $\hat{\mu}_{(100)} =$ $\hat{\Sigma}_{(100)} =$

Compare the estimated value $\hat{\mu}$ with the original value of μ by measuring the Euclidean distance that separates them. Compare the estimated value $\hat{\Sigma}$ with the original value of Σ_3 by measuring the matrix 2-norm of their difference ($\| A - B \|_2$ constitutes a measure of similarity of two matrices A and B; use MATLAB's `norm` command).

Example:

In the case of 1000 points (case 2.):

```
>> X = X3(1:1000,:);
>> N = size(X,1)
>> mu_1000 = sum(X)/N -or- >> mu_1000 = mean(X)
>> sigma_1000 = (X - repmat(mu_1000,N,1))' * (X - repmat(mu_1000,N,1)) / (N-1)
-or- >> sigma_1000 = cov(X)

>> % Comparison of the values:
>> e_mu = sqrt( (mu_1000 - mu) * (mu_1000 - mu)' )
>> % (This is the Euclidean distance between mu_1000 and mu)
>> e_sigma = norm( sigma_1000 - sigma_3 )
>> % (This is the 2-norm of the difference between sigma_1000 and sigma_3)
```

Question:

When comparing the estimated values $\hat{\mu}$ and $\hat{\Sigma}$ with the original values of μ and Σ_3 (using the Euclidean distance and the matrix 2-norm), what can you observe ?

Answer:

The more points, the better the estimates. Furthermore, an accurate mean estimate requires less points than an accurate variance estimate. In general, in any data-based pattern classification technique (as opposed to knowledge-based techniques or expert systems), it is very important to have enough training examples to estimate some accurate models of the data.

1.3 Likelihood of a sample with respect to a Gaussian model

Useful formulas and definitions :

- *Likelihood*: the likelihood of a sample point given a Gaussian model (i.e. given a set of parameters $\Theta = (\mu, \Sigma)$) is the value of the probability density function for that point. In the case of Gaussian models, this amounts to compute the value of the pdf expression given at the beginning of section 1.1.

- *Joint likelihood*: for a set of independent identically distributed (i.i.d.) points, say $X = \{x_1, x_2, \cdots, x_N\}$, the joint (or total) likelihood is the product of the likelihood for each point. For instance, in the Gaussian case:

$$p(X|\Theta) = \prod_{i=1}^{N} p(x_i|\Theta) = \prod_{i=1}^{N} p(x_i|\mu,\Sigma) = \prod_{i=1}^{N} g_{(\mu,\Sigma)}(x_i)$$

Experiment :

Given the 4 Gaussian models:

$$\mathcal{N}_1 : \Theta_1 = \left(\begin{bmatrix} 730 \\ 1090 \end{bmatrix}, \begin{bmatrix} 8000 & 0 \\ 0 & 8000 \end{bmatrix} \right) \qquad \mathcal{N}_2 : \Theta_2 = \left(\begin{bmatrix} 730 \\ 1090 \end{bmatrix}, \begin{bmatrix} 8000 & 0 \\ 0 & 18500 \end{bmatrix} \right)$$

$$\mathcal{N}_3 : \Theta_3 = \left(\begin{bmatrix} 730 \\ 1090 \end{bmatrix}, \begin{bmatrix} 8000 & 8400 \\ 8400 & 18500 \end{bmatrix} \right) \qquad \mathcal{N}_4 : \Theta_4 = \left(\begin{bmatrix} 270 \\ 1690 \end{bmatrix}, \begin{bmatrix} 8000 & 8400 \\ 8400 & 18500 \end{bmatrix} \right)$$

compute the following log-likelihoods for the whole sample X_3 (10000 points):

$$\log p(X_3|\Theta_1),\ \log p(X_3|\Theta_2),\ \log p(X_3|\Theta_3) \text{ and } \log p(X_3|\Theta_4).$$

(First answer the following question and then look at the exemple given on the next page.)

Question :

Why do we want to compute the *log-likelihood* rather than the simple *likelihood* ?

Answer :

Computing the log-likelihood turns the product into a sum:

$$p(X|\Theta) = \prod_{i=1}^{N} p(x_i|\Theta) \quad \Leftrightarrow \quad \log p(X|\Theta) = \sum_{i=1}^{N} \log p(x_i|\Theta)$$

In the Gaussian case, it also avoids the computation of the exponential:

$$\begin{aligned} p(x|\Theta) &= \frac{1}{\sqrt{2\pi}^d \sqrt{\det(\Sigma)}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \\ \log p(x|\Theta) &= -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log(\det(\Sigma)) - \frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu) \end{aligned}$$

Furthermore, since $\log(x)$ *is a monotonically growing function, the log-likelihoods have the same relations of order as the likelihoods:*

$$p(x|\Theta_1) > p(x|\Theta_2) \quad \Leftrightarrow \quad \log p(x|\Theta_1) > \log p(x|\Theta_2)$$

so they can be used directly to classify samples.

In a classification framework, the computation can be even more simplified: the relations of order will remain valid if we drop the division by 2 and the $d\log(2\pi)$ *term (we have a right to drop these terms because they are* independent of the classes*). If* $\mathcal{N}_1(\Theta_1)$ *and* $\mathcal{N}_2(\Theta_2)$ *have the same variance, we can also drop the* $\log(\det(\Sigma))$ *term (since in this case the variance itself becomes independent of the classes).*

As a summary, log-likelihoods use simpler computation but are readily usable for classification tasks.

Example :

```
>> N = size(X3,1)
>> mu_1 = [730 1090]; sigma_1 = [8000 0; 0 8000];
>> logLike1 = 0;
>> for i = 1:N;
logLike1 = logLike1 + (X3(i,:) - mu_1)*inv(sigma_1)*(X3(i,:) - mu_1)';
end;
>> logLike1 = - 0.5 * ( logLike1 + N*log(det(sigma_1)) + 2*N*log(2*pi) )
```

Note : if you don't understand why the different models generate a different log-likelihood value for the data, use the function `gausview` to compare the relative positions of the models $\mathcal{N}_1$, $\mathcal{N}_2$, $\mathcal{N}_3$ and $\mathcal{N}_4$ with respect to the data set X_3, e.g. :

```
>> mu_1 = [730 1090]; sigma_1 = [8000 0; 0 8000];
>> gausview(X3,mu_1,sigma_1,'Comparison of X3 and N1');
```

Question :

Of $\mathcal{N}_1$, $\mathcal{N}_2$, $\mathcal{N}_3$ and $\mathcal{N}_4$, which model "explains" best the data X_3 ? Which model has the highest number of parameters ? Which model would you choose for a good compromise between the number of parameters and the capacity to represent accurately the data ?

Answer :

The model $\mathcal{N}_3$ produces the highest likelihood for the data set X_3. So we can say that data X_3 is more likely to have been generated by model $\mathcal{N}_3$ than by the other models, or that $\mathcal{N}_3$ explains best the data.

On the other hand, model $\mathcal{N}_3$ has the highest number of parameters (2 terms for the mean, 4 non null terms for the variance). This may seem low in two dimensions, but the number of parameters grows exponentially with the dimension of the data (this phenomenon is called the curse of dimensionality*). Also, the more parameters you have, the more data you need to estimate (or* train*) them.*

In "real world" speech recognition applications, the dimensionality of the speech features is typically of the order of 40 (1 energy coefficient + 12 cepstrum coefficients + their first and second order derivatives = a vector of 39 coefficients). Further processing is applied to orthogonalize *the data (e.g., cepstral coefficients can be interpreted as orthogonalized spectra, and hence admit quasi-diagonal covariance matrices). Therefore, the compromise usually considered is to use models with diagonal covariance matrices, such as the model $\mathcal{N}_2$ in our example.*

2 Statistical pattern recognition

2.1 A-priori class probabilities

Experiment :

Load data from file "vowels.mat". This file contains a database of simulated 2-dimensional speech features in the form of *artificial* pairs of formant values (the first and the second spectral formants, $[F_1, F_2]$). These artificial values represent the features that would be extracted from several occurrences of vowels /a/, /e/, /i/, /o/ and /y/[1]. They are grouped in matrices of size $N \times 2$, where each of the N lines is a training example and 2 is the dimension of the features (in our case, formant frequency pairs).

Supposing that the whole database covers adequately an imaginary language made only of /a/'s, /e/'s, /i/'s, /o/'s and /y/'s, compute the probability $P(q_k)$ of each class q_k, $k \in \{/a/, /e/, /i/, /o/, /y/\}$. What are the most common and the least common phoneme in the language ?

Example :

```
>> clear all; load vowels.mat; whos
>> Na = size(a,1); Ne = size(e,1); Ni = size(i,1); No = size(o,1); Ny = size(y,1);
>> N = Na + Ne + Ni + No + Ny;
>> Pa = Na/N
```

[1] /y/ is the phonetic symbol for "u" like in the French word "*tutu*".

```
>> Pi = Ni/N
```
etc.

Answer :

The probability of using /a/ in this imaginary speech is 0.25. It is 0.3 for /e/, 0.25 for /i/, 0.15 for /o/ and 0.05 for /y/. The most common phoneme is therefore /e/, while the least common is /y/.

2.2 Gaussian modeling of classes

Experiment :

Plot each vowel's data as clouds of points in the 2D plane. Train the Gaussian models corresponding to each class (use directly the `mean` and `cov` commands). Plot their contours (use directly the function `plotgaus(`*mu*`,`*sigma*`,`*color*`)` where *color* `= [R,G,B]`).

Example :

```
>> plotvow; % Plot the clouds of simulated vowel features
```
(Do not close the obtained figure, it will be used later on.)
Then compute and plot the Gaussian models :
```
>> mu_a = mean(a);
>> sigma_a = cov(a);
>> plotgaus(mu_a,sigma_a,[0 1 1]);
>> mu_e = mean(e);
>> sigma_e = cov(e);
>> plotgaus(mu_e,sigma_e,[0 1 1]);
```
etc.

Note your results below :

$\mu_{/a/} =$ $\Sigma_{/a/} =$

$\mu_{/e/} =$ $\Sigma_{/e/} =$

$\mu_{/i/} =$ $\Sigma_{/i/} =$

$\mu_{/o/} =$ $\Sigma_{/o/} =$

$\mu_{/y/} =$ $\Sigma_{/y/} =$

2.3 Bayesian classification

Useful formulas and definitions :

- *Bayes' decision rule*:

$$X \in q_k \text{ if } P(q_k|X,\Theta) \geq P(q_j|X,\Theta),\ \forall j \neq k$$

This formula means : given a set of classes q_k, characterized by a set of known parameters Θ, a set of one or more speech feature vectors X (also called *observations*) belongs to the class which has the highest probability once we actually know (or "see", or "measure") the sample X. $P(q_k|X,\Theta)$ is therefore called the *a posteriori probability*, because it depends on having seen the observations, as opposed to the *a priori* probability $P(q_k|\Theta)$ which does not depend on any observation (but depends of course on knowing how to characterize all the classes q_k, which means knowing the parameter set Θ).

- For some classification tasks (e.g. speech recognition), it is more practical to resort to *Bayes' law*, which makes use of *likelihoods*, rather than trying to estimate directly the posterior probability. Bayes' law says:

$$P(q_k|X,\Theta) = \frac{p(X|q_k,\Theta)P(q_k|\Theta)}{p(X|\Theta)}$$

where q_k is a class, X is a sample containing one or more feature vectors and Θ is the parameter set of all the class models.

- The speech features are usually considered equi-probable. Hence, it is considered that $P(q_k|X,\Theta)$ is proportional to $p(X|q_k,\Theta)P(q_k|\Theta)$ for all the classes:

$$\forall k, \;\; P(q_k|X,\Theta) \propto p(X|q_k,\Theta)P(q_k|\Theta)$$

- Once again, it is more convenient to do the computation in the log domain:

$$\log P(q_k|X,\Theta) \simeq \log p(X|q_k,\Theta) + \log P(q_k|\Theta)$$

Question:

1. In our case (Gaussian models for phoneme classes), what is the meaning of the Θ given in the above formulas ?
2. What is the expression of $p(X|q_k,\Theta)$, and of $\log p(X|q_k,\Theta)$?
3. What is the definition of the probability $P(q_k|\Theta)$?

Answer:

1. *In our case, Θ represents the set of all the means μ_k and variances Σ_k, $k \in \{/a/,/e/,/i/,/o/,/y/\}$.*
2. *The expression of $p(X|q_k,\Theta)$ and of $\log p(X|q_k,\Theta)$ correspond to the computation of the Gaussian pdf and its logarithm, already expressed in section 1.3.*
3. *The probability $P(q_k|\Theta)$ is the a-priori class probability for the class q_k (corresponding to the parameters $\Theta_k \in \Theta$). It defines an absolute probability of occurrence for the class q_k. The a-priori class probabilities for our artificial phoneme classes have been computed in the section 2.1.*

Question:

Now, we have modeled each vowel class with a Gaussian pdf (by computing means and variances), we know the probability $P(q_k)$ of each class in the imaginary language, and we assume that the speech *features* (as opposed to speech *classes*) are equi-probable. What is the most probable class q_k for the speech feature points $x = (F_1, F_2)^T$ given in the following table ? (Compute the posterior probabilities according to the example given on the next page.)

x	F_1	F_2	$\log P(q_{/a/}\|x)$	$\log P(q_{/e/}\|x)$	$\log P(q_{/i/}\|x)$	$\log P(q_{/o/}\|x)$	$\log P(q_{/y/}\|x)$	Most prob. class
1.	400	1800						
2.	400	1000						
3.	530	1000						
4.	600	1300						
5.	670	1300						
6.	420	2500						

Example :

Use function gloglike(*point*,*mu*,*sigma*) to compute the likelihoods. Don't forget to add the log of the prior probability ! E.g., for point 1. and class /a/ :

```
>> gloglike([400,1800],mu_a,sigma_a) + log(Pa)
```

Answer :

1. /e/ 2. /y/ 3. /o/ 4. /y/ 5. /a/ 6. /i/

2.4 Discriminant surfaces

Useful formulas and definitions :

- *Discriminant function* : a set of functions $f_k(x)$ allows to classify a sample x into k classes q_k if :

$$x \in q_k \quad \Leftrightarrow \quad f_k(x, \Theta_k) \geq f_l(x, \Theta_l), \; \forall l \neq k$$

In this case, the k functions $f_k(x)$ are called discriminant functions.

Question :

What is the link between discriminant functions and Bayesian classifiers ?

Answer :

$$\begin{aligned} x \in q_k \quad &\Leftrightarrow \quad P(q_k|x) \geq P(q_l|x) \; \forall l \neq k \\ &\Leftrightarrow \quad p(x|q_k)P(q_k) \geq p(x|q_l)P(q_l) \\ &\Leftrightarrow \quad \log p(x|q_k) + \log P(q_k) \geq \log p(x|q_l) + \log P(q_l) \end{aligned}$$

The a-posteriori probability $P(q_k|x)$ that a sample x belongs to class q_k is itself a discriminant function :

Experiment :

The iso-likelihood lines for the Gaussian pdfs $\mathcal{N}\left(\mu_{/i/}, \Sigma_{/i/}\right)$ and $\mathcal{N}\left(\mu_{/e/}, \Sigma_{/e/}\right)$, which we used before to model the class /i/ and the class /e/, are plotted on the next page (figure 1). On a second graph, the iso-likelihood lines for $\mathcal{N}\left(\mu_{/i/}, \Sigma_{/e/}\right)$ and $\mathcal{N}\left(\mu_{/e/}, \Sigma_{/e/}\right)$ (two pdfs with the same covariance matrix $\Sigma_{/e/}$) are represented. On these figures, use a colored pen to join the intersections of the level lines that correspond to equal likelihoods.

Question :

What is the nature of the surface that separates class /i/ from class /e/ when the two models have *different* variances ? Can you explain the origin of this form ?

What is the nature of the surface that separates class /i/ from class /e/ when the two models have the *same* variances ? Why is it different from the previous discriminant surface ?

Answer :

In the case of different variances, the discriminant surface is a parabola. As a matter of fact, in the Gaussian case :

$$\log p(x|\Theta) \simeq -\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu) - \frac{1}{2}\log\left(\det\left(\Sigma\right)\right)$$

which is a quadratic form. Therefore, the equation :

$$\log p(x|\Theta_1) = \log p(x|\Theta_2)$$

which describes the discriminant surface is necessarily a second order equation. Hence, its solution describes a parabolic discriminant surface.

In the case where the covariance matrices are equal, the separation between the class 1 and the class 2 does not depend upon the covariance Σ any more : (Continued on next page...)

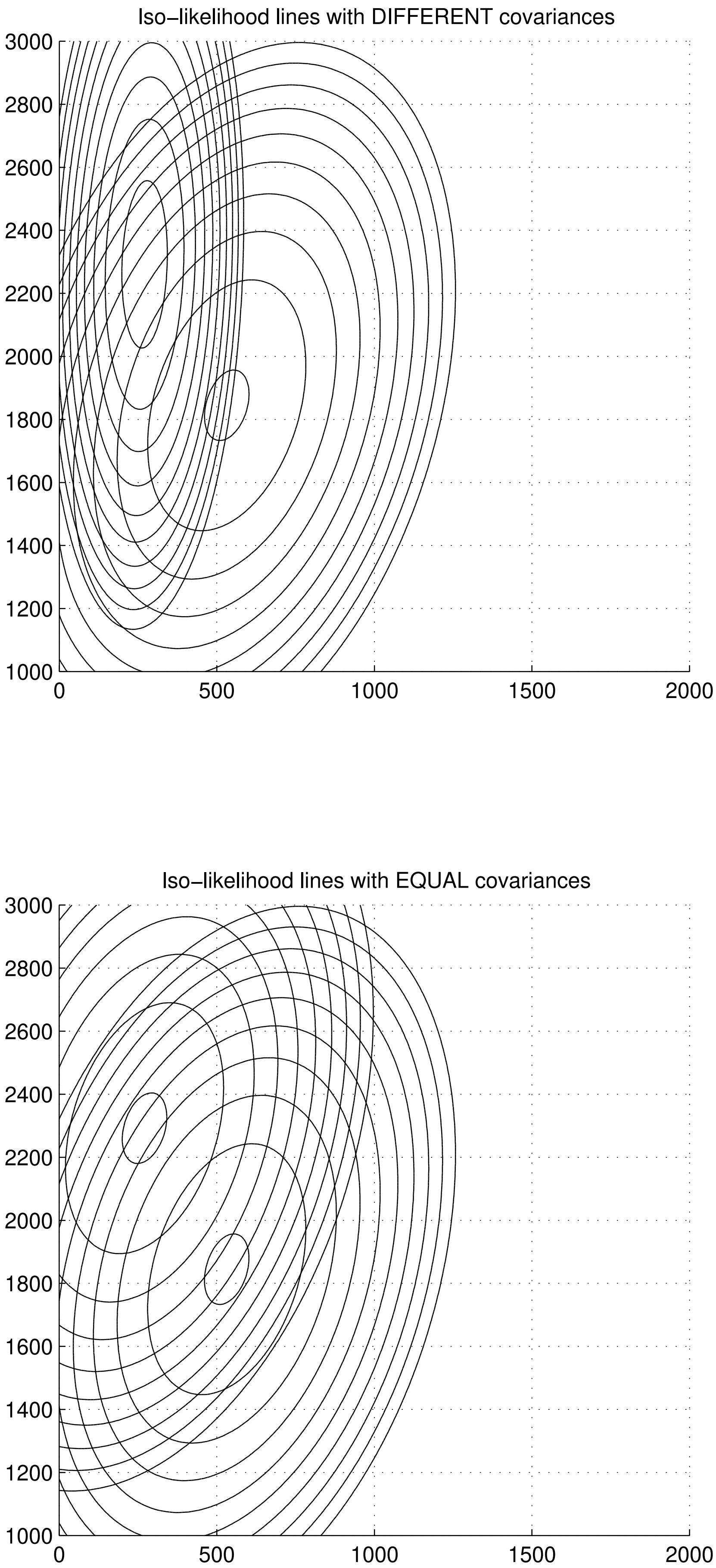


Figure 1: Iso-likelihood lines for the Gaussian pdfs $\mathcal{N}\left(\mu_{/i/}, \Sigma_{/i/}\right)$ and $\mathcal{N}\left(\mu_{/e/}, \Sigma_{/e/}\right)$, then $\mathcal{N}\left(\mu_{/i/}, \Sigma_{/e/}\right)$ and $\mathcal{N}\left(\mu_{/e/}, \Sigma_{/e/}\right)$.

Answer, continued :

$$-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) \;-\frac{1}{2}\log\left(\det\left(\Sigma\right)\right) \;=\; -\frac{1}{2}(x-\mu_2)^T\Sigma^{-1}(x-\mu_2) \;-\frac{1}{2}\log\left(\det\left(\Sigma\right)\right)$$

$$\Leftrightarrow \qquad -\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) = -\frac{1}{2}(x-\mu_2)^T\Sigma^{-1}(x-\mu_2)$$

$$\Leftrightarrow \qquad -\frac{1}{2}x^T\Sigma^{-1}x + \mu_1^T\Sigma^{-1}x - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 = -\frac{1}{2}x^T\Sigma^{-1}x + \mu_2^T\Sigma^{-1}x - \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2$$

Hence, the equation of the surface that separates class q_1 from class q_2 becomes :

$$(\mu_1-\mu_2)^T\Sigma^{-1}x - \frac{1}{2}(\mu_1-\mu_2)^T\Sigma^{-1}(\mu_1-\mu_2) = 0$$

which has a linear form. In the equal covariance case, using a Bayesian classifier with Gaussian densities is therefore equivalent to using discriminant functions of the form :

$$f_k(x) = w_k^T x + w_{k0}$$

where

$$\begin{aligned} w_k^T &= \mu_k^T\Sigma^{-1} \\ w_{k0} &= -\frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k \end{aligned}$$

As a summary, we have shown that Bayesian classifiers with Gaussian models separate the classes with combinations of parabolic surfaces. If the covariance matrices of the models are equal, the parabolic separation surfaces become simple hyper-planes.

3 Unsupervised training

In the previous section, we have computed the models for classes /a/, /e/, /i/, /o/ and /y/ by knowing a-priori which training samples belongs to which class (we were disposing of a *labeling* of the training data). Hence, we have performed a *supervised training* of Gaussian models. Now, suppose that we only have unlabeled training data that we want to separate in several classes (e.g., 5 classes) without knowing a-priori which point belongs to which class. This is called *unsupervised training.* Several algorithms are available for that purpose, among which : the K-means, the Viterbi-EM and the EM (Expectation-Maximization) algorithm.

All these algorithms are characterized by the following components :

- a set of models q_k (not necessarily Gaussian), defined by some parameters Θ (means, variances, priors,...);
- a measure of membership, telling to which extent a data point "belongs" to a model;
- a "recipe" to update the model parameters in function of the membership information.

The measure of membership usually takes the form of a measure of distance or the form of a measure of probability. It replaces the missing labeling information to permit the application of standard parameter estimation techniques. It also defines implicitly a global criterion of "goodness of fit" of the models to the data, e.g. :

- in the case of a distance, the models that are globally closer from the data characterize it better;
- in the case of a probability measure, the models bringing a better likelihood for the data explain it better.

Table 1 summarizes the components of each of the algorithm that will be studied in the following experiments. More detail will be given in the corresponding subsections.

Algorithm	**Parameters**	**Membership measure**	**Update method**	**Global criterion**
Kmeans	• mean μ_k	Euclidean distance $d_k(x_n) = \sqrt{(x_n - \mu_k)^T (x_n - \mu_k)}$ (or the square of it)	Find the points closest to $q_k^{(old)}$, then : • $\mu_k^{(new)}$ = mean of the points closest to $q_k^{(old)}$	Least squares
Viterbi-EM	• mean μ_k • variance Σ_k • priors $P(q_k\|\Theta)$	Posterior probability $d_k(x_n) = P(q_k\|x_n, \Theta)$ $\propto \frac{1}{\sqrt{2\pi}^d \sqrt{\det(\Sigma_k)}} e^{-\frac{1}{2}(x_n-\mu_k)^T \Sigma_k^{-1} (x_n-\mu_k)} \cdot P(q_k\|\Theta)$	Do Bayesian classification of each data point, then : • $\mu_k^{(new)}$ = mean of the points belonging to $q_k^{(old)}$ • $\Sigma_k^{(new)}$ = variance of the points belonging to $q_k^{(old)}$ • $P(q_k^{(new)}\|\Theta^{(new)})$ = number of training points belonging to $q_k^{(old)}$ / total number of training points	Maximum likelihood
EM	• mean μ_k • variance Σ_k • priors $P(q_k\|\Theta)$	Posterior probability $d_k(x_n) = P(q_k\|x_n, \Theta)$ $\propto \frac{1}{\sqrt{2\pi}^d \sqrt{\det(\Sigma_k)}} e^{-\frac{1}{2}(x_n-\mu_k)^T \Sigma_k^{-1} (x_n-\mu_k)} \cdot P(q_k\|\Theta)$	Compute $P(q_k^{(old)}\|x_n, \Theta^{(old)})$ (soft classification), then : • $\mu_k^{(new)} = \frac{\sum_{n=1}^N x_n P(q_k^{(old)}\|x_n,\Theta^{(old)})}{\sum_{n=1}^N P(q_k^{(old)}\|x_n,\Theta^{(old)})}$ • $\Sigma_k^{(new)} = \frac{\sum_{n=1}^N P(q_k^{(old)}\|x_n,\Theta^{(old)})(x_n-\mu_k^{(new)})(x_n-\mu_k^{(new)})^T}{\sum_{n=1}^N P(q_k^{(old)}\|x_n,\Theta^{(old)})}$ • $P(q_k^{(new)}\|\Theta^{(new)}) = \frac{1}{N}\sum_{n=1}^N P(q_k^{(old)}\|x_n, \Theta^{(old)})$	Maximum likelihood

Table 1: Characteristics of some usual unsupervised clustering algorithms.

3.1 K-means algorithm

Synopsis of the algorithm :

- Start with K initial prototypes μ_k, $k = 1, \cdots, K$.
- **Do** :
  1. For each data-point x_n, $n = 1, \cdots, N$, compute the squared Euclidean distance from the k^{th} prototype :
  $$\begin{aligned} d_k(x_n) &= \|x_n - \mu_k\|^2 \\ &= (x_n - \mu_k)^T (x_n - \mu_k) \end{aligned}$$
  2. Assign each data-point x_n to its **closest** prototype μ_k, i.e. assign x_n to the class q_k if :
  $$d_k(x_n) \leq d_l(x_n), \;\; \forall l \neq k$$
  Note : using the square of the Euclidean distance for the classification gives the same result as using the true Euclidean distance, since the square root is a monotonically growing function. But the computational load is obviously lighter when the square root is dropped.
  3. Replace each prototype with the mean of the data-points assigned to the corresponding class;
  4. Go to 1.
- **Until** : no further change occurs.

The global criterion defined in the present case is :

$$J = \sum_{k=1}^{K} \sum_{x_n \in q_k} d_k(x_n)$$

and represents the total squared distance between the data and the models they belong to. This criterion is locally minimized by the algorithm.

Experiment :

Use the K-means explorer utility :

```
 KMEANS K-means algorithm exploration tool

   Launch it with KMEANS(DATA,NCLUST) where DATA is the matrix
   of observations (one observation per row) and NCLUST is the
   desired number of clusters.

   The clusters are initialized with a heuristic that spreads
   them randomly around mean(DATA) with standard deviation
   sqrtm(cov(DATA)).

   If you want to set your own initial clusters, use
   KMEANS(DATA,MEANS) where MEANS is a cell array containing
   NCLUST initial mean vectors.

   Example: for two clusters
     means{1} = [1 2]; means{2} = [3 4];
     kmeans(data,means);
```

Launch it with the data sample `allvow`, which was part of file `vowels.mat` and gathers all the simulated vowels data. Do several runs with different cases of initialization of the algorithm:

1. 5 initial clusters determined according to the default heuristic;
2. some initial `MEANS` values equal to some data points;
3. some initial `MEANS` values equal to $\{\mu_{/a/}, \mu_{/e/}, \mu_{/i/}, \mu_{/o/}, \mu_{/y/}\}$.

Iterate the algorithm until its convergence. Observe the evolution of the cluster centers, of the data-points attribution chart and of the total squared Euclidean distance. (It is possible to zoom these plots: left click inside the axes to zoom 2× centered on the point under the mouse; right click to zoom out; click and drag to zoom into an area; double click to reset the figure to the original). Observe the mean values found after the convergence of the algorithm.

Example:

```
>> kmeans(allvow,5);
```
- or -
```
>> means = { mu_a, mu_e, mu_i, mu_o, mu_y };
>> kmeans(allvow,means);
```
Enlarge the window, then push the buttons, zoom etc. After the convergence, use:
```
>> for k=1:5, disp(kmeans_result_means{k}); end
```
to see the resulting means.

Question:

1. Does the final solution depend on the initialization of the algorithm ?
2. Describe the evolution of the total squared Euclidean distance.
3. What is the nature of the discriminant surfaces corresponding to a minimum Euclidean distance classification scheme ?
4. Is the algorithm suitable for fitting Gaussian clusters ?

Answer:

1. *The final solution depends upon the initialization of the algorithm.*
2. *The total squared Euclidean distance decreases in a monotonic way.*
3. *The minimum-distance point attribution scheme is equivalent to linear discriminant surfaces.*
4. *The final solution of the K-means algorithm is unable to lock onto the original Gaussian phoneme classes.*

3.2 Viterbi-EM algorithm for Gaussian clustering

Synopsis of the algorithm:

- Start from K initial Gaussian models $\mathcal{N}(\mu_k, \Sigma_k)$, $k = 1 \cdots K$, characterized by the set of parameters Θ (i.e. the set of all means and variances μ_k and Σ_k, $k = 1 \cdots K$). Set the initial prior probabilities $P(q_k)$ to $1/K$.
- **Do**:
  1. Classify each data-point using Bayes' rule.

     This step is equivalent to having a set Q of boolean hidden variables that give a labeling of the data by taking the value 1 (belongs) or 0 (does not belong) for each class q_k and each point x_n. The value of Q that maximizes $p(X, Q|\Theta)$ precisely tells which is the most probable model for each point of the whole set X of training data.

     Hence, each data point is assigned to its most probable cluster $q_k^{(old)}$.

2. Update the parameters:
   - update the means:
   $$\mu_k^{(new)} = \text{mean of the points belonging to } q_k^{(old)}$$
   - update the variances:
   $$\Sigma_k^{(new)} = \text{variance of the points belonging to } q_k^{(old)}$$
   - update the priors:
   $$P(q_k^{(new)}|\Theta^{(new)}) = \frac{\text{number of training points belonging to } q_k^{(old)}}{\text{total number of training points}}$$
3. Go to 1.

- **Until**: no further change occurs.

The global criterion defined in the present case is:

$$\begin{aligned} \mathcal{L}(\Theta) &= \sum_X P(X|\Theta) = \sum_Q \sum_X p(X,Q|\Theta) \\ &= \sum_{k=1}^{K} \sum_{x_n \in q_k} \log p(x_n|\Theta_k) \end{aligned}$$

and represents the joint likelihood of the data with respect to the models they belong to. This criterion is locally maximized by the algorithm.

Experiment:

Use the Viterbi-EM explorer utility:

```
VITERB Viterbi version of the EM algorithm

  Launch it with VITERB(DATA,NCLUST) where DATA is the matrix
  of observations (one observation per row) and NCLUST is the
  desired number of clusters.

  The clusters are initialized with a heuristic that spreads
  them randomly around mean(DATA) with standard deviation
  sqrtm(cov(DATA)). Their initial covariance is set to cov(DATA).

  If you want to set your own initial clusters, use
  VITERB(DATA,MEANS,VARS) where MEANS and VARS are cell arrays
  containing respectively NCLUST initial mean vectors and NCLUST
  initial covariance matrices. In this case, the initial a-priori
  probabilities are set equal to 1/NCLUST.

  To set your own initial priors, use VITERB(DATA,MEANS,VARS,PRIORS)
  where PRIORS is a vector containing NCLUST a priori probabilities.

  Example: for two clusters
    means{1} = [1 2]; means{2} = [3 4];
    vars{1} = [2 0;0 2]; vars{2} = [1 0;0 1];
    viterb(data,means,vars);
```

Launch it with the dataset `allvow`. Do several runs with different cases of initialization of the algorithm:

1. 5 initial clusters determined according to the default heuristic;

2. some initial `MEANS` values equal to some data points, and some random `VARS` values (try for instance `cov(allvow)` for all the classes);

3. the initial `MEANS`, `VARS` and `PRIORS` values found by the K-means algorithm.

4. some initial `MEANS` values equal to $\{\mu_{/a/}, \mu_{/e/}, \mu_{/i/}, \mu_{/o/}, \mu_{/y/}\}$, `VARS` values equal to $\{\Sigma_{/a/}, \Sigma_{/e/}, \Sigma_{/i/}, \Sigma_{/o/}, \Sigma_{/y/}\}$, and `PRIORS` values equal to $[P_{/a/}, P_{/e/}, P_{/i/}, P_{/o/}, P_{/y/}]$;

5. some initial `MEANS` and `VARS` values chosen by yourself.

Iterate the algorithm until it converges. Observe the evolution of the clusters, of the data points attribution chart and of the total likelihood curve. Observe the mean, variance and priors values found after the convergence of the algorithm. Compare them with the values computed in section 2.2 (with supervised training).

Example :

```
>> viterb(allvow,5);
```
- or -
```
>> means = { mu_a, mu_e, mu_i, mu_o, mu_y };
>> vars = { sigma_a, sigma_e, sigma_i, sigma_o, sigma_y };
>> viterb(allvow,means,vars);
```
Enlarge the window, then push the buttons, zoom etc. After convergence, use :
```
>> for k=1:5, disp(viterb_result_means{k}); end
>> for k=1:5, disp(viterb_result_vars{k}); end
>> for k=1:5, disp(viterb_result_priors(k)); end
```
to see the resulting means, variances and priors.

Question :

1. Does the final solution depend on the initialization of the algorithm ?

2. Describe the evolution of the total likelihood. Is it monotonic ?

3. In terms of optimization of the likelihood, what does the final solution correspond to ?

4. What is the nature of the discriminant surfaces corresponding to the Gaussian classification ?

5. Is the algorithm suitable for fitting Gaussian clusters ?

Answer :

1. *The final solution strongly depends upon the initialization of the algorithm.*

2. *The total likelihood increases monotonically, which means that the models „explain" the training dataset better and better.*

3. *The final solution corresponds approximately to a local maximum of likelihood in the sense of Bayesian classification with Gaussian models.*

4. *As seen in section 2.4, the discriminant surfaces corresponding to Gaussian clusters with unequal variances have the form of (hyper-)parabolas.*

5. *The final solution of the Viterbi-EM algorithm is able to lock approximately onto the simulated Gaussian phoneme classes if the number and the placement of the initial clusters are correctly guessed* (which is not an easy task).

3.3 EM algorithm for Gaussian clustering

Synopsis of the algorithm:

- Start from K initial Gaussian models $\mathcal{N}(\mu_k, \Sigma_k),\ k = 1 \cdots K$, with equal priors set to $P(q_k) = 1/K$.
- **Do**:
  1. **Estimation step**: compute the probability $P(q_k^{(old)}|x_n, \Theta^{(old)})$ for each data point x_n to belong to the class $q_k^{(old)}$:
  $$P(q_k^{(old)}|x_n, \Theta^{(old)}) = \frac{P(q_k^{(old)}|\Theta^{(old)}) \cdot p(x_n|q_k^{(old)}, \Theta^{(old)})}{p(x_n|\Theta^{(old)})}$$
  $$= \frac{P(q_k^{(old)}|\Theta^{(old)}) \cdot p(x_n|\mu_k^{(old)}, \Sigma_k^{(old)})}{\sum_j P(q_j^{(old)}|\Theta^{(old)}) \cdot p(x_n|\mu_j^{(old)}, \Sigma_j^{(old)})}$$
  This step is equivalent to having a set Q of continuous hidden variables, taking values in the interval $[0, 1]$, that give a labeling of the data by telling to which extent a point x_n belongs to the class q_k. This represents a soft classification, since a point can belong, e.g., by 60% to class 1 and by 40% to class 2 (think of Schrödinger's cat which is 60% alive and 40% dead as long as nobody opens the box or performs Bayesian classification).
  2. **Maximization step**:
     – update the means:
     $$\mu_k^{(new)} = \frac{\sum_{n=1}^N x_n P(q_k^{(old)}|x_n, \Theta^{(old)})}{\sum_{n=1}^N P(q_k^{(old)}|x_n, \Theta^{(old)})}$$
     – update the variances:
     $$\Sigma_k^{(new)} = \frac{\sum_{n=1}^N P(q_k^{(old)}|x_n, \Theta^{(old)})(x_n - \mu_k^{(new)})(x_n - \mu_k^{(new)})^T}{\sum_{n=1}^N P(q_k^{(old)}|x_n, \Theta^{(old)})}$$
     – update the priors:
     $$P(q_k^{(new)}|\Theta^{(new)}) = \frac{1}{N}\sum_{n=1}^N P(q_k^{(old)}|x_n, \Theta^{(old)})$$
  In the present case, all the data points participate to the update of all the models, but their participation is weighted by the value of $P(q_k^{(old)}|x_n, \Theta^{(old)})$.
  3. Go to 1.
- **Until**: the total likelihood increase for the training data falls under some desired threshold.

The global criterion defined in the present case is:

$$\begin{aligned}
\mathcal{L}(\Theta) = \log p(X|\Theta) &= \log \sum_Q p(X, Q|\Theta) \\
&= \log \sum_Q P(Q|X, \Theta) p(X|\Theta) \quad \text{(Bayes)} \\
&= \log \sum_{k=1}^K P(q_k|X, \Theta) p(X|\Theta)
\end{aligned}$$

Applying Jensen's inequality $\left(\log \sum_j \lambda_j y_j \geq \sum_j \lambda_j \log y_j \text{ if } \sum_j \lambda_j = 1\right)$, we obtain:

$$\begin{aligned}
\mathcal{L}(\Theta) &\approx \sum_{k=1}^K P(q_k|X, \Theta) \log p(X|\Theta) \\
&= \sum_{k=1}^K \sum_{n=1}^N P(q_k|x_n, \Theta) \log p(x_n|\Theta)
\end{aligned}$$

Hence, the final J represents a lower boundary for the joint likelihood of all the data with respect to all the models. This criterion is locally maximized by the algorithm.

Experiment :

Use the EM explorer utility :

```
EMALGO EM algorithm explorer

  Launch it with EMALGO(DATA,NCLUST) where DATA is the matrix
  of observations (one observation per row) and NCLUST is the
  desired number of clusters.

  The clusters are initialized with a heuristic that spreads
  them randomly around mean(DATA) with standard deviation
  sqrtm(cov(DATA)*10). Their initial covariance is set to cov(DATA).

  If you want to set your own initial clusters, use
  EMALGO(DATA,MEANS,VARS) where MEANS and VARS are cell arrays
  containing respectively NCLUST initial mean vectors and NCLUST
  initial covariance matrices. In this case, the initial a-priori
  probabilities are set equal to 1/NCLUST.

  To set your own initial priors, use VITERB(DATA,MEANS,VARS,PRIORS)
  where PRIORS is a vector containing NCLUST a priori probabilities.

  Example: for two clusters
    means{1} = [1 2]; means{2} = [3 4];
    vars{1} = [2 0;0 2]; vars{2} = [1 0;0 1];
    emalgo(data,means,vars);
```

Launch it with again the same dataset `allvow`. Do several runs with different cases of initialization of the algorithm :

1. 5 clusters determined according to the default heuristic;
2. some initial `MEANS` values equal to some data points, and some random `VARS` values (e.g. `cov(allvow)` for all the classes);
3. the initial `MEANS` and `VARS` values found by the K-means algorithm.
4. some initial `MEANS` values equal to $\{\mu_{/a/}, \mu_{/e/}, \mu_{/i/}, \mu_{/o/}, \mu_{/y/}\}$, `VARS` values equal to $\{\Sigma_{/a/}, \Sigma_{/e/}, \Sigma_{/i/}, \Sigma_{/o/}, \Sigma_{/y/}\}$, and `PRIORS` values equal to $[P_{/a/}, P_{/e/}, P_{/i/}, P_{/o/}, P_{/y/}]$;
5. some initial `MEANS` and `VARS` values chosen by yourself.

(If you have time, also increase the number of clusters and play again with the algorithm.)

Iterate the algorithm until the total likelihood reaches an asymptotic convergence. Observe the evolution of the clusters and of the total likelihood curve. (In the EM case, the data points attribution chart is not given because each data point participates to the update of each cluster.) Observe the mean, variance and prior values found after the convergence of the algorithm. Compare them with the values found in section 2.2.

Example :

```
>> emalgo(allvow,5);
```
- or -
```
>> means = { mu_a, mu_e, mu_i, mu_o, mu_y };
>> vars = { sigma_a, sigma_e, sigma_i, sigma_o, sigma_y };
>> emalgo(allvow,means,vars);
```
Enlarge the window, then push the buttons, zoom etc. After convergence, use :
```
>> for k=1:5, disp(emalgo_result_means{k}); end
>> for k=1:5, disp(emalgo_result_vars{k}); end
>> for k=1:5, disp(emalgo_result_priors(k)); end
```
to see the resulting means, variances and priors.

Question :

1. Does the final solution depend on the initialization of the algorithm ?
2. Describe the evolution of the total likelihood. Is it monotonic ?
3. In terms of optimization of the likelihood, what does the final solution correspond to ?
4. Is the algorithm suitable for fitting Gaussian clusters ?

Answer :

1. *Here again, the final solution depends upon the initialization of the algorithm.*
2. *Again, the total likelihood always increases monotonically.*
3. *The final solution corresponds to a local maximum of likelihood in the sense of Bayesian classification with Gaussian models.*
4. *The final solution of the EM algorithm is able to lock perfectly onto the Gaussian phoneme clusters, but only if the number and the placement of the initial clusters are correctly guessed (which, once again, is* not an easy task*).*

   In practice, the initial clusters are usually set to the K-means solution (which, as you have seen, may not be an optimal guess). Alternately, they can be set to some "well chosen" data points.

After the lab...

This lab manual can be kept as additional course material. If you want to browse the experiments again, you can use the script :

```
>> lab1demo
```

which will automatically redo all the computation and plots for you (except those of section 3).