Gaussian Statistics and Unsupervised Learning

A Tutorial for the Course *Computational Intelligence*

http://www.igi.tugraz.at/lehre/CI

Barbara Resch (minor changes by Erhard Rank)
Signal Processing and Speech Communication Laboratory
Inffeldgasse 16c/II
phone 873–4436

Abstract

This tutorial presents the properties of the Gaussian probability density function. Subsequently, supervised and unsupervised pattern recognition methods are treated. Supervised classification algorithms are based on a labeled data set. The knowledge about the class membership of this training data set is used for the classification of new samples. Unsupervised learning methods establish clusters from an unlabeled training data set. Clustering algorithms such as the K-means, the EM (expectation-maximization) algorithm, and the Viterbi-EM algorithm are presented.

Usage

To make full use of this tutorial you have to

1. download the file `Gaussian.zip`[1] which contains this tutorial and the accompanying Matlab programs.
2. Unzip `Gaussian.zip` which will generate a subdirectory named `Gaussian/matlab` where you can find all the Matlab programs.
3. Add the path `Gaussian/matlab` to the matlab search path, for example with a command like `addpath('C:\Work\Gaussian\matlab')` if you are using a Windows machine, or by using a command like `addpath('/home/jack/Gaussian/matlab')` if you are on a Unix/Linux machine.

Sources

This tutorial is based on

- EPFL lab notes "Introduction to Gaussian Statistics and Statistical Pattern Recognition" by Hervé Bourlard, Sacha Krstulović, and Mathew Magimai-Doss.

1 Gaussian statistics

1.1 Samples from a Gaussian density

Useful formulas and definitions:

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

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

  where $\boldsymbol{\mu}$ is the mean vector and $\boldsymbol{\Sigma}$ is the covariance matrix. $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the *parameters* of the Gaussian distribution.

[1] http://www.igi.tugraz.at/lehre/CI/tutorials/Gaussian.zip

- The mean vector $\boldsymbol{\mu}$ contains the mean values of each dimension, $\mu_i = E(x_i)$, with $E(x)$ being the *expected value* of x.
- All of the variances c_{ii} and covariances c_{ij} are collected together into the covariance matrix $\boldsymbol{\Sigma}$ of dimension $d \times d$:

$$\boldsymbol{\Sigma} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix}$$

  The covariance c_{ij} of two components x_i and x_j of $\mathbf{x}$ measures their tendency to vary together, i.e., to co-vary,

$$c_{ij} = E\left((x_i - \mu_i)^{\mathsf{T}} (x_j - \mu_j)\right).$$

  If two components x_i and x_j, $i \neq j$, have zero covariance $c_{ij} = 0$ 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). If all components of $\mathbf{x}$ are mutually orthogonal the covariance matrix has a diagonal form.
- $\sqrt{\boldsymbol{\Sigma}}$ defines the *standard deviation* of the random variable $\mathbf{x}$. Beware: this square root is meant in the *matrix sense*.
- If $\mathbf{x} \circlearrowleft \mathcal{N}(\mathbf{0}, \mathbf{I})$ ($\mathbf{x}$ follows a normal law with zero mean and unit variance; $\mathbf{I}$ denotes the identity matrix), and if $\mathbf{y} = \boldsymbol{\mu} + \sqrt{\boldsymbol{\Sigma}}\,\mathbf{x}$, then $\mathbf{y} \circlearrowleft \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

1.1.1 Experiment:

Generate samples X of N points, $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, with $N = 10000$, coming from a 2-dimensional Gaussian process that has mean

$$\boldsymbol{\mu} = \begin{bmatrix} 730 \\ 1090 \end{bmatrix}$$

and variance

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

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 8000 & 0 \\ 0 & 8000 \end{bmatrix}$$

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

$$\boldsymbol{\Sigma}_2 = \begin{bmatrix} 8000 & 0 \\ 0 & 18500 \end{bmatrix}$$

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

$$\boldsymbol{\Sigma}_3 = \begin{bmatrix} 8000 & 8400 \\ 8400 & 18500 \end{bmatrix}$$

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 for the two other variance matrices $\boldsymbol{\Sigma}_2$ and $\boldsymbol{\Sigma}_3$. 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 (must be enabled in `Tools` menu: `Rotate 3D`, or by the ↻ button).

Questions:

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 (as the sample X_1), which sample corresponds to a process with a diagonal covariance matrix (as X_2), and which to a process with a full covariance matrix (as X_3)?

Find the right statements:

- ☐ In process 1 the first and the second component of the vectors $\mathbf{x}_i$ are independent.
- ☐ In process 2 the first and the second component of the vectors $\mathbf{x}_i$ are independent.
- ☐ In process 3 the first and the second component of the vectors $\mathbf{x}_i$ are independent.
- ☐ If the first and second component of the vectors $\mathbf{x}_i$ are independent, the cloud of points and the pdf contour has the shape of a circle.
- ☐ If the first and second component of the vectors $\mathbf{x}_i$ are independent, the cloud of points and pdf contour has to be elliptic with the principle axes of the ellipse aligned with the abscissa and ordinate axes.
- ☐ For the covariance matrix $\mathbf{\Sigma}$ the elements have to satisfy $c_{ij} = c_{ji}$.
- ☐ The covariance matrix has to be positive definite ($\mathbf{x}^\mathsf{T}\mathbf{\Sigma}\,\mathbf{x} \geq 0$). (If yes, what happens if not? Try it out in MATLAB).

1.2 Gaussian modeling: Mean and variance of a sample

We will now estimate the parameters $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ of the Gaussian models from the data samples.

Useful formulas and definitions:

- Mean estimator: $\hat{\boldsymbol{\mu}} = \dfrac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$
- Unbiased covariance estimator: $\hat{\mathbf{\Sigma}} = \dfrac{1}{N-1}\sum_{i=1}^{N}(\mathbf{x}_i - \boldsymbol{\mu})^\mathsf{T}(\mathbf{x}_i - \boldsymbol{\mu})$

1.2.1 Experiment:

Take the sample X_3 of 10000 points generated from $\mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma}_3)$. Compute an estimate $\hat{\boldsymbol{\mu}}$ of its mean and an estimate $\hat{\mathbf{\Sigma}}$ of its variance:

1. with all the available points $\quad \hat{\boldsymbol{\mu}}_{(10000)} = \quad \hat{\mathbf{\Sigma}}_{(10000)} =$
2. with only 1000 points $\quad \hat{\boldsymbol{\mu}}_{(1000)} = \quad \hat{\mathbf{\Sigma}}_{(1000)} =$
3. with only 100 points $\quad \hat{\boldsymbol{\mu}}_{(100)} = \quad \hat{\mathbf{\Sigma}}_{(100)} =$

Compare the estimated mean vector $\hat{\boldsymbol{\mu}}$ to the original mean vector $\boldsymbol{\mu}$ by measuring the Euclidean distance that separates them. Compare the estimated covariance matrix $\hat{\mathbf{\Sigma}}$ to the original covariance matrix $\mathbf{\Sigma}_3$ by measuring the matrix 2-norm of their difference (the norm $\|\mathbf{A} - \mathbf{B}\|_2$ constitutes a measure of similarity of two matrices $\mathbf{A}$ and $\mathbf{B}$; use MATLAB's `norm` command).

1.2.2 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 means and covariances:
>> 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)
```

1.2.3 Question:

When comparing the estimated values $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$ to the original values of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}_3$ (using the Euclidean distance and the matrix 2-norm), what can you observe?

Find the right statements:

- ☐ An accurate mean estimate requires more points than an accurate variance estimate.
- ☐ It is very important to have enough training examples to estimate the parameters of the data generation process accurately.

1.3 Likelihood of a sample with respect to a Gaussian model

In the following we compute the likelihood of a sample point $\mathbf{x}$, and the joint likelihood of a series of samples X for a given model $\boldsymbol{\Theta}$ with one Gaussian. The likelihood will be used in the formula for classification later on (sec. 2.3).

Useful formulas and definitions:

- *Likelihood*: the likelihood of a sample point $\mathbf{x}_i$ given a data generation model (i.e., given a set of parameters $\boldsymbol{\Theta}$ for the model pdf) is the value of the pdf $p(\mathbf{x}_i|\boldsymbol{\Theta})$ for that point. In the case of Gaussian models $\boldsymbol{\Theta} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$, this amounts to the evaluation of equation 1.
- *Joint likelihood*: for a set of independent identically distributed (i.i.d.) samples, say $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, the joint (or total) likelihood is the product of the likelihoods for each point. For instance, in the Gaussian case:

$$p(X|\boldsymbol{\Theta}) = \prod_{i=1}^{N} p(\mathbf{x}_i|\boldsymbol{\Theta}) = \prod_{i=1}^{N} p(\mathbf{x}_i|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{i=1}^{N} g_{(\boldsymbol{\mu},\boldsymbol{\Sigma})}(\mathbf{x}_i) \tag{2}$$

1.3.1 Question:

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

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

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

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

$$\begin{aligned} p(\mathbf{x}|\boldsymbol{\Theta}) &= \frac{1}{\sqrt{2\pi}^d \sqrt{\det(\boldsymbol{\Sigma})}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \\ \log p(\mathbf{x}|\boldsymbol{\Theta}) &= \frac{1}{2}\left[-d\log(2\pi) - \log(\det(\boldsymbol{\Sigma})) - (\mathbf{x}-\boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right] \end{aligned} \tag{3}$$

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

$$p(x|\boldsymbol{\Theta}_1) > p(x|\boldsymbol{\Theta}_2) \quad \Leftrightarrow \quad \log p(x|\boldsymbol{\Theta}_1) > \log p(x|\boldsymbol{\Theta}_2),$$

so they can be used directly for classification.

Find the right statements:

We can further simplify the computation of the log-likelihood in eq. 3 for classification by

- ☐ dropping the division by two: $\frac{1}{2}[\ldots]$,
- ☐ dropping term $d\log(2\pi)$,
- ☐ dropping term $\log(\det(\boldsymbol{\Sigma}))$,
- ☐ dropping term $(\mathbf{x}-\boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})$,
- ☐ calculating the term $\log(\det(\boldsymbol{\Sigma}))$ in advance.

We can drop term(s) because:

- ☐ The term(s) are independent of $\boldsymbol{\mu}$.
- ☐ The terms are negligible small.
- ☐ The term(s) are independent of the classes.

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

1.3.2 Experiment:

Given the following 4 Gaussian models $\boldsymbol{\Theta}_i = (\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$

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

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

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

$$\log p(X_3|\boldsymbol{\Theta}_1),\ \log p(X_3|\boldsymbol{\Theta}_2),\ \log p(X_3|\boldsymbol{\Theta}_3),\ \text{and}\ \log p(X_3|\boldsymbol{\Theta}_4).$$

1.3.3 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: 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 (with non-zero values)? Which model would you choose for a good compromise between the number of parameters and the capacity to accurately represent the data?

2 Statistical pattern recognition

2.1 A-priori class probabilities

2.1.1 Experiment:

Load data from file "vowels.mat". This file contains a database of 2-dimensional samples of speech features in the form of formant frequencies (the first and the second spectral formants, $[F_1, F_2]$). The formant frequency samples represent features that would be extracted from the speech signal for several occurrences of the vowels /a/, /e/, /i/, /o/, and /y/[2]. They are grouped in matrices of size $N \times 2$, where each of the N lines contains the two formant frequencies for one occurrence of a vowel.

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, $k \in \{/a/, /e/, /i/, /o/, /y/\}$. Which is the most common and which the least common phoneme in our imaginary language?

2.1.2 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
>> Pi = Ni/N
```

etc.

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]`).

2.2.1 Example:

```
>> plotvow; % Plot the clouds of simulated vowel features
```

(Do not close the figure obtained, 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.

[2] /y/ is the phonetic symbol for "ü"

2.3 Bayesian classification

We will now find how to classify a feature vector $\mathbf{x}_i$ from a data sample (or several feature vectors X) as belonging to a certain class q_k.

Useful formulas and definitions:

- *Bayes' decision rule*:

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

  This formula means: given a set of classes q_k, characterized by a set of known parameters in model $\boldsymbol{\Theta}$, 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. $P(q_k|X,\boldsymbol{\Theta})$ is therefore called the *a posteriori probability*, because it depends on having seen the observations, as opposed to the *a priori* probability $P(q_k|\boldsymbol{\Theta})$ 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 $\boldsymbol{\Theta}$).

- For some classification tasks (e.g. speech recognition), it is practical to resort to *Bayes' law*, which makes use of *likelihoods* (see sec. 1.3), rather than trying to directly estimate the posterior probability $P(q_k|X,\boldsymbol{\Theta})$. Bayes' law says:

$$P(q_k|X,\boldsymbol{\Theta}) = \frac{p(X|q_k,\boldsymbol{\Theta})\ P(q_k|\boldsymbol{\Theta})}{p(X|\boldsymbol{\Theta})} \tag{4}$$

  where q_k is a class, X is a sample containing one or more feature vectors and $\boldsymbol{\Theta}$ is the parameter set of all the class models.

- The speech features are usually considered equi-probable: $p(X|\boldsymbol{\Theta}) = \text{const.}$ (uniform prior distribution for X). Hence, $P(q_k|X,\boldsymbol{\Theta})$ is proportional to $p(X|q_k,\boldsymbol{\Theta})P(q_k|\boldsymbol{\Theta})$ for all classes:

$$P(q_k|X,\boldsymbol{\Theta}) \propto p(X|q_k,\boldsymbol{\Theta})\ P(q_k|\boldsymbol{\Theta}), \quad \forall k$$

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

$$\log P(q_k|X,\boldsymbol{\Theta}) \propto \log p(X|q_k,\boldsymbol{\Theta}) + \log P(q_k|\boldsymbol{\Theta}) \tag{5}$$

In our case, $\boldsymbol{\Theta}$ represents the set of *all* the means $\boldsymbol{\mu}_k$ and variances $\boldsymbol{\Sigma}_k$, $k \in \{/\text{a}/, /\text{e}/, /\text{i}/, /\text{o}/, /u/\}$ of our data generation model. $p(X|q_k,\boldsymbol{\Theta})$ and $\log p(X|q_k,\boldsymbol{\Theta})$ are the joint likelihood and joint log-likelihood (eq. 2 in section 1.3) of the sample X with respect to the model $\boldsymbol{\Theta}$ for class q_k (i.e., the model with parameter set $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$).

The probability $P(q_k|\boldsymbol{\Theta})$ is the a-priori class probability for the class q_k. It defines an absolute probability of occurrence for the class q_k. The a-priori class probabilities for our phoneme classes have been computed in section 2.1.

2.3.1 Experiment:

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 (sec. 2.1), which we assume to be the correct a priori probabilities $P(q_k|\boldsymbol{\Theta})$ for each class given our model $\boldsymbol{\Theta}$. Further we assume that the speech *features* $\mathbf{x}_i$ (as opposed to speech *classes* q_k) are equi-probable[3].

What is the most probable class q_k for each of the formant pairs (features) $\mathbf{x}_i = [F_1, F_2]^\mathsf{T}$ given in the table below? Compute the values of the functions $f_k(\mathbf{x}_i)$ for our model $\boldsymbol{\Theta}$ as the right-hand side of eq. 5: $f_k(\mathbf{x}_i) = \log p(\mathbf{x}_i|q_k,\boldsymbol{\Theta}) + \log P(q_k|\boldsymbol{\Theta})$, proportional to the log of the posterior probability of $\mathbf{x}_i$ belonging to class q_k.

[3]Note, that – also for not equi-probable features – finding the most probable class q_k according to eq. 4 does not depend on the denominator $p(X|\boldsymbol{\Theta})$, since $p(X|\boldsymbol{\Theta})$ is independent of q_k!

i	$\mathbf{x}_i = [F_1, F_2]^\mathsf{T}$	$f_{/\mathrm{a}/}(\mathbf{x}_i)$	$f_{/\mathrm{e}/}(\mathbf{x}_i)$	$f_{/\mathrm{i}/}(\mathbf{x}_i)$	$f_{/\mathrm{o}/}(\mathbf{x}_i)$	$f_{/\mathrm{y}/}(\mathbf{x}_i)$	Most prob. class q_k
1	$[400, 1800]^\mathsf{T}$						
2	$[400, 1000]^\mathsf{T}$						
3	$[530, 1000]^\mathsf{T}$						
4	$[600, 1300]^\mathsf{T}$						
5	$[670, 1300]^\mathsf{T}$						
6	$[420, 2500]^\mathsf{T}$						

2.3.2 Example:

Use function `gloglike(point,mu,sigma)` to compute the log-likelihoods $\log p(\mathbf{x}_i|q_k, \boldsymbol{\Theta})$. Don't forget to add the log of the prior probability $P(q_k|\boldsymbol{\Theta})$! E.g., for the feature set x_1 and class /a/ use
`>> gloglike([400,1800],mu_a,sigma_a) + log(Pa)`

2.4 Discriminant surfaces

For the Bayesian classification in the last section we made use of the *discriminant functions* $f_k(\mathbf{x}_i) = \log p(\mathbf{x}_i|q_k, \boldsymbol{\Theta}) + \log P(q_k|\boldsymbol{\Theta})$ to classify data points $\mathbf{x}_i$. This corresponds to establishing *discriminant surfaces* of dimension $d-1$ in the vector space for $\mathbf{x}$ (dimension d) to separate regions for the different classes.

Useful formulas and definitions:

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

$$\mathbf{x} \in q_k \quad \Leftrightarrow \quad f_k(\mathbf{x}, \boldsymbol{\Theta}_k) \geq f_l(\mathbf{x}, \boldsymbol{\Theta}_l), \quad \forall l \neq k$$

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

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

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

As in our case the samples $\mathbf{x}$ are two-dimensional vectors, the discriminant surfaces are one-dimensional, i.e., lines at equal values of the discriminant functions for two distinct classes.

2.4.1 Experiment:

The iso-likelihood lines for the Gaussian pdfs $\mathcal{N}(\boldsymbol{\mu}_{/\mathrm{i}/}, \boldsymbol{\Sigma}_{/\mathrm{i}/})$ and $\mathcal{N}(\boldsymbol{\mu}_{/\mathrm{e}/}, \boldsymbol{\Sigma}_{/\mathrm{e}/})$, which we used before to model the class /i/ and the class /e/, are plotted in figure 1, first graph. On the second graph in figure 1, the iso-likelihood lines for $\mathcal{N}(\boldsymbol{\mu}_{/\mathrm{i}/}, \boldsymbol{\Sigma}_{/\mathrm{e}/})$ and $\mathcal{N}(\boldsymbol{\mu}_{/\mathrm{e}/}, \boldsymbol{\Sigma}_{/\mathrm{e}/})$ (two pdfs with the *same* covariance matrix $\boldsymbol{\Sigma}_{/\mathrm{e}/}$) are represented.

On these figures, use a colored pen to join the intersections of the level lines that correspond to equal likelihoods. Assume that the highest iso-likelihood lines (smallest ellipses) are of the same height. (You can also use `isosurf` in MATLAB to create a color plot.)

2.4.2 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?

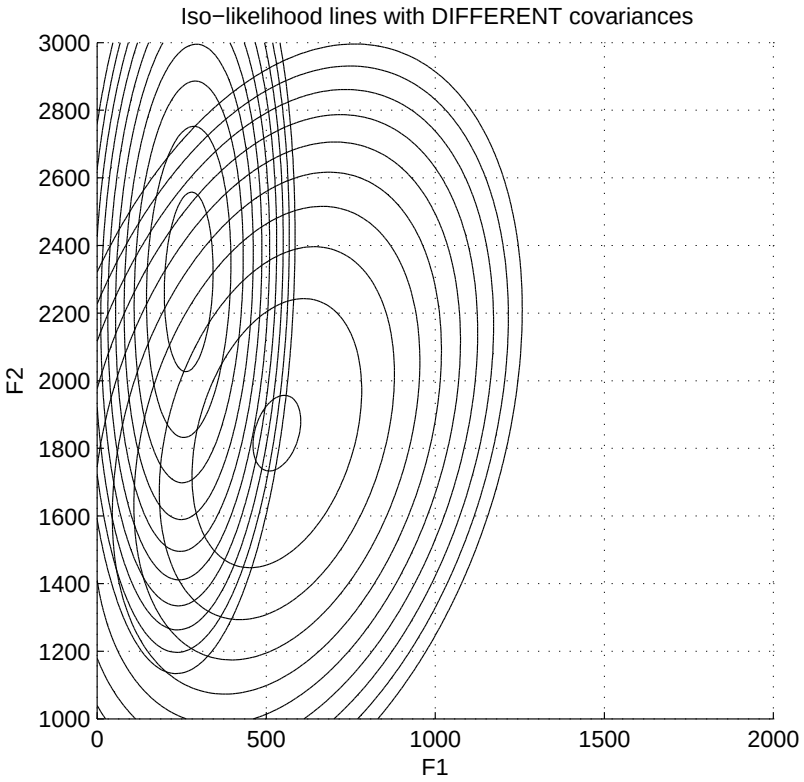

Figure 1: Iso-likelihood lines for the Gaussian pdfs $\mathcal{N}(\boldsymbol{\mu}_{/\mathrm{i}/}, \boldsymbol{\Sigma}_{/\mathrm{i}/})$ and $\mathcal{N}(\boldsymbol{\mu}_{/\mathrm{e}/}, \boldsymbol{\Sigma}_{/\mathrm{e}/})$ (top), and $\mathcal{N}(\boldsymbol{\mu}_{/\mathrm{i}/}, \boldsymbol{\Sigma}_{/\mathrm{e}/})$ and $\mathcal{N}(\boldsymbol{\mu}_{/\mathrm{e}/}, \boldsymbol{\Sigma}_{/\mathrm{e}/})$ (bottom).

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?

Show that in the case of two Gaussian pdfs with *equal covariance matrices*, the separation between class 1 and class 2 does not depend upon the covariance $\mathbf{\Sigma}$ any more.

As a summary, we have seen that Bayesian classifiers with Gaussian data 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 the 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 them: the K-means, the EM (Expectation-Maximization), and the Viterbi-EM algorithm.

All these algorithms are characterized by the following components:

- a set of models for the classes q_k (not necessarily Gaussian), defined by a parameter set $\mathbf{\Theta}$ (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 as a function of the membership information.

The measure of membership usually takes the form of a distance measure 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 measure, the models that are closer to the data characterize it better;
- in the case of a probability measure, the models with a larger likelihood for the data explain it better.

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

3.1 K-means algorithm

Synopsis of the algorithm:

- Start with K initial prototypes $\boldsymbol{\mu}_k$, $k = 1, \ldots, K$.
- **Do**:
  1. For each data-point $\mathbf{x}_n$, $n = 1, \ldots, N$, compute the squared Euclidean distance from the k^{th} prototype:
  $$\begin{aligned} d_k(\mathbf{x}_n) &= \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2 \\ &= (\mathbf{x}_n - \boldsymbol{\mu}_k)^{\mathsf{T}}(\mathbf{x}_n - \boldsymbol{\mu}_k) \end{aligned}$$
  2. Assign each data-point $\mathbf{x}_n$ to its *closest prototype* $\boldsymbol{\mu}_k$, i.e., assign $\mathbf{x}_n$ to the class q_k if
  $$d_k(\mathbf{x}_n) < d_l(\mathbf{x}_n), \qquad \forall l \neq k$$
  Note: using the squared 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 smaller when the square root is dropped.

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

Table 1: Characteristics of some usual unsupervised clustering algorithms.

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 for the present case is

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

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 `kmeans` 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. initial `MEANS` values equal to some data points;
3. initial `MEANS` values equal to $\{\boldsymbol{\mu}_{/\mathrm{a}/}, \boldsymbol{\mu}_{/\mathrm{e}/}, \boldsymbol{\mu}_{/\mathrm{i}/}, \boldsymbol{\mu}_{/\mathrm{o}/}, \boldsymbol{\mu}_{/\mathrm{y}/}\}$.

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.

Think about the following 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?

3.2 Viterbi-EM algorithm for Gaussian clustering

Synopsis of the algorithm:

- Start from K initial Gaussian models $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k),\ k = 1 \dots K$, characterized by the set of parameters $\boldsymbol{\Theta}$ (i.e., the set of all means and variances $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$, $k = 1 \dots K$). 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 $\mathbf{x}_n$. The value of Q that maximizes $p(X, Q|\boldsymbol{\Theta})$ precisely tells which is the most probable model for each point of the whole set X of training data.
     Hence, each data point $\mathbf{x}_n$ is assigned to its most probable cluster q_k.
  2. Update the parameters (i is the iteration index):
     - update the means:
     $$\boldsymbol{\mu}_k^{(i+1)} = \text{mean of the points belonging to } q_k^{(i)}$$
     - update the variances:
     $$\boldsymbol{\Sigma}_k^{(i+1)} = \text{variance of the points belonging to } q_k^{(i)}$$
     - update the priors:
     $$P(q_k^{(i+1)}|\boldsymbol{\Theta}^{(i+1)}) = \frac{\text{number of training points belonging to } q_k^{(i)}}{\text{total number of training points}}$$
  3. Go to 1.
- **Until**: no further change occurs.

The global criterion in the present case is

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

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 `viterb` with the dataset `allvow`. Do several runs with different initializations of the algorithm:

1. 5 initial clusters determined according to the default heuristic;
2. 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. initial `MEANS` values equal to $\{\boldsymbol{\mu}_{/\mathrm{a}/}, \boldsymbol{\mu}_{/\mathrm{e}/}, \boldsymbol{\mu}_{/\mathrm{i}/}, \boldsymbol{\mu}_{/\mathrm{o}/}, \boldsymbol{\mu}_{/\mathrm{y}/}\}$, `VARS` values equal to $\{\boldsymbol{\Sigma}_{/\mathrm{a}/}, \boldsymbol{\Sigma}_{/\mathrm{e}/}, \boldsymbol{\Sigma}_{/\mathrm{i}/}, \boldsymbol{\Sigma}_{/\mathrm{o}/}, \boldsymbol{\Sigma}_{/\mathrm{y}/}\}$, and `PRIORS` values equal to $[P_{/\mathrm{a}/}, P_{/\mathrm{e}/}, P_{/\mathrm{i}/}, P_{/\mathrm{o}/}, P_{/\mathrm{y}/}]$;
5. 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 (using 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?

3.3 EM algorithm for Gaussian clustering

Synopsis of the algorithm:

- Start from K initial Gaussian models $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k),\ k = 1 \dots K$, with equal priors set to $P(q_k) = 1/K$.
- **Do**:
  1. **Estimation step**: compute the probability $P(q_k^{(i)}|\mathbf{x}_n, \boldsymbol{\Theta}^{(i)})$ for each data point $\mathbf{x}_n$ to belong to the class $q_k^{(i)}$:

$$\begin{aligned} P(q_k^{(i)}|\mathbf{x}_n, \boldsymbol{\Theta}^{(i)}) &= \frac{P(q_k^{(i)}|\boldsymbol{\Theta}^{(i)}) \cdot p(\mathbf{x}_n|q_k^{(i)}, \boldsymbol{\Theta}^{(i)})}{p(\mathbf{x}_n|\boldsymbol{\Theta}^{(i)})} \\ &= \frac{P(q_k^{(i)}|\boldsymbol{\Theta}^{(i)}) \cdot p(\mathbf{x}_n|\boldsymbol{\mu}_k^{(i)}, \boldsymbol{\Sigma}_k^{(i)})}{\sum_j P(q_j^{(i)}|\boldsymbol{\Theta}^{(i)}) \cdot p(\mathbf{x}_n|\boldsymbol{\mu}_j^{(i)}, \boldsymbol{\Sigma}_j^{(i)})} \end{aligned}$$

     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 $\mathbf{x}_n$ 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:

$$\boldsymbol{\mu}_k^{(i+1)} = \frac{\sum_{n=1}^N \mathbf{x}_n P(q_k^{(i)}|\mathbf{x}_n, \boldsymbol{\Theta}^{(i)})}{\sum_{n=1}^N P(q_k^{(i)}|\mathbf{x}_n, \boldsymbol{\Theta}^{(i)})}$$

     - update the variances:

$$\boldsymbol{\Sigma}_k^{(i+1)} = \frac{\sum_{n=1}^N P(q_k^{(i)}|\mathbf{x}_n, \boldsymbol{\Theta}^{(i)})\ (\mathbf{x}_n - \boldsymbol{\mu}_k^{(i+1)})(\mathbf{x}_n - \boldsymbol{\mu}_k^{(i+1)})^\mathsf{T}}{\sum_{n=1}^N P(q_k^{(i)}|\mathbf{x}_n, \boldsymbol{\Theta}^{(i)})}$$

     - update the priors:

$$P(q_k^{(i+1)}|\boldsymbol{\Theta}^{(i+1)}) = \frac{1}{N} \sum_{n=1}^N P(q_k^{(i)}|\mathbf{x}_n, \boldsymbol{\Theta}^{(i)})$$

     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 $P(q_k^{(i)}|\mathbf{x}_n, \boldsymbol{\Theta}^{(i)})$.
  3. Go to 1.
- **Until**: the total likelihood increase for the training data falls under some desired threshold.

The global criterion in the present case is the joint likelihood of all data with respect to all the models:

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

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:

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

Hence, the criterion $J(\mathbf{\Theta})$ represents a lower boundary for $\mathcal{L}(\mathbf{\Theta})$. 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 `emalgo` 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. 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. initial `MEANS` values equal to $\{\boldsymbol{\mu}_{/\mathrm{a}/}, \boldsymbol{\mu}_{/\mathrm{e}/}, \boldsymbol{\mu}_{/\mathrm{i}/}, \boldsymbol{\mu}_{/\mathrm{o}/}, \boldsymbol{\mu}_{/\mathrm{y}/}\}$, `VARS` values equal to $\{\mathbf{\Sigma}_{/\mathrm{a}/}, \mathbf{\Sigma}_{/\mathrm{e}/}, \mathbf{\Sigma}_{/\mathrm{i}/}, \mathbf{\Sigma}_{/\mathrm{o}/}, \mathbf{\Sigma}_{/\mathrm{y}/}\}$, and `PRIORS` values equal to $[P_{/\mathrm{a}/}, P_{/\mathrm{e}/}, P_{/\mathrm{i}/}, P_{/\mathrm{o}/}, P_{/\mathrm{y}/}]$;
5. 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?

Questions to K-means, Viterbi-EM and EM algorithm:

Find the right statements:

☐ With the K-means algorithm the solution is independent upon the initialization.

☐ With the K-means algorithm the discriminant surfaces are linear.

☐ The K-means algorithm is well suited for fitting Gaussian clusters

☐ In the K-means algorithm the global criterion used to minimize is the maximum likelihood.

☐ In all 3 algorithms the measure used as global criterion decreases in a monotonic way.

☐ In the Viterbi-EM and EM algorithm the solution is highly dependent upon initialization.

☐ The EM algorithm is best suited for fitting Gaussian clusters.

☐ It is an easy task to guess the parameters for initialization.

☐ With the Viterbi-EM algorithm the discriminant surfaces are linear.

☐ With the EM algorithm the discriminant surfaces have the form of (hyper)parabola.

☐ The EM algorithm needs less computational effort then the Viterbi-EM algorithm.

☐ In the EM algorithm and the Viterbi-EM algorithm, the same global criterion is used.

☐ The EM algorithm finds a global optimum.