GpdsHMM Toolbox

User's guide

Version 0.0

By Sébastien DAVID in collaboration with Miguel A. Ferrer, Carlos M. Travieso and Jesús B. Alonso.
Dpto. De Señales y Comunicaciones, Universidad de Las Palmas de Gran Canaria
Campus de Tafira, 35017 Las Palmas de Gran Canaria
SPAIN
Tel: +34 928 451 269 Fax: +34 928 451 243, e-mail: gpds@gi.ulpgc.es

Illustrations:

Acknowledgement:

A freely distributed version of the gpdsHMM toolbox is available from: http://www.gpds.ulpgc.es/download/index.htm
This toolbox is distributed as binary (dll files) and source code format. For a wide promotion, we ask the users to make a reference to this paper. For any remarks about this toolbox, do not hesitate to contact the authors sending an e-mail to: gpds@gi.ulpgc.es.
The functions for the CHMM use the netlab utility available from: http://www.ncrg.aston.ac.uk/netlab/
This toolbox was developed for the Matlab (© *The MathWorks, Inc.*). We chose this environment since it is widely used for the educational purpose and for the research.

1. Introduction:

A Hidden Markov Model (HMM) is a type of stochastic model appropriate for non stationary stochastic sequences, with statistical properties that undergo distinct random transitions among a set of different stationary processes. In other words, the HMM models a sequence of observations as a piecewise stationary process. Over the past years, Hidden Markov Models have been widely applied in several models like pattern [1, 2], pathologies [3] or speech recognition [4, 5], and DNA sequence analysis [6, 7]. The HMMs are suitable for the classification from one or two dimensional signals and can be used when the information is incomplete or uncertain.

To use a HMM, we need a training phase and a test phase. For the training stage, we usually work with the Baum-Welch algorithm to estimate the parameters (π, A, B) for the HMM [8, 9]. This method is based on the maximum likelihood criterion.

In addition to the Baum-Welch algorithm, it is necessary to estimate the Alfa and Beta matrices thanks to the forward and backward procedures. To compute the most probable state sequence, the Viterbi algorithm is the most suitable.

In order to apply the HMM techniques, the authors have developed a HMM toolbox called gpdsHMM in the Matlab environment. Several toolbox for the HMM already exist [10]. This work was carried out in order to offer a friendlier tool through didactics and graphics examples. This toolbox also contains two new concepts developed recently in the literature: the multi-labeling and the gathering methods (or multi-parameter method) which, when used in suitable conditions, improve significantly the results obtained with the HMM [11].

This HMM toolbox is implemented to enable a quick and easy use of a discrete HMM. We propose in 2 a basic review of the theory. The part three gives a useful description for every function provided within this toolbox. And, in part 4, we tend to propose a continuous HMM. At the end of this file, you can find three useful examples based on the discrete HMM, the continuous HMM and one using the function DHMM2CHMM.

Please read the part 7 of this toolbox to make this toolbox work in the Matlab © environment.

2. HMM theory and example

A hidden Markov Model (HMM) is a type of stochastic model appropriate for no stationary stochastic sequences, with statistical properties that undergo distinct random transitions among a set of different stationary processes. In other words, the HMM models a sequence of observations like a piecewise stationary process. Such models have been used extensively in speech recognition, handwriting recognition, texture classification, blind equalization, etc. In this part, we will focus on the discrete HMM. The HMM theory in the case of the continuous is explained in the part 4.

2.1. Elements of a HMM

A HMM model is basically a stochastic finite state automaton, which generates an observation string, that is, the sequence of observation vectors, $\boldsymbol{O}=\boldsymbol{O}_1, \boldsymbol{O}_2, \ldots, \boldsymbol{O}_T$. Thus, a HMM model consists of a number of N states $S=\{S_i\}$ and the observations string produced as a result of emitting a vector $\boldsymbol{O}_t$ each successive transitions from one state S_i to another state S_j. The state transition probability distribution between state S_i and S_j is $A=\{a_{ij}\}$, and the observation probability distribution of emitting any vector $\boldsymbol{O}_t$ at state S_j is given by $B=\{b_j(\boldsymbol{O}_t)\}$. The probability to be in the state i at the initial instant is $Pi=\{\pi_i\}$.

$$a_{ij}= P(q_{k+1}=S_j \mid q_k=S_i) \quad (1)$$

$$b_j(\boldsymbol{O}_t)= P(\mathbf{O}_t \mid q_t=S_j) \quad (2)$$

$$\pi_i = P(q_0=S_i) \quad (3)$$

Then, given an observation sequence $\boldsymbol{O}$, and a HMM model $\lambda=(A, B, \Pi)$, we can compute $P(\boldsymbol{O} \mid \lambda)$ the probability of the observed sequence thanks to the forward-backward procedure. Concisely, defining the forward variable as the probability of the partial observation sequence $\boldsymbol{O}_1, \boldsymbol{O}_2, \ldots, \boldsymbol{O}_t$ (until time t) and state S_i at time t, given the model λ, as $\alpha_t(i)$. And defining the backward variable as the probability of the partial observation sequence form $t+1$ to the end, given state S_i at time t and the model λ, as $\beta_t(i)$. Both $\alpha_t(i)$ and $\beta_t(i)$ are worked out with the forward-backward procedure.

Forward:

$$\alpha_1(i)= \pi_i \; b_i(\boldsymbol{O}_1) \quad (4)$$

$$\alpha_{t+1}(j)=[\Sigma_i \; \alpha_t(i) \; a_{ij}] \; b_j(\boldsymbol{O}_{t+1}) \quad (5)$$

Backward:

$$\beta_T(i)=a_{iN} \quad (6)$$

$$\beta_{t-1}(i)= [\Sigma_j \; \beta_t(j) \; a_{ji}] \; b_i(\boldsymbol{O}_{t-1}) \quad (7)$$

We can calculate the probability of the observations sequence as:

$$P(\mathbf{O}\mid\lambda)=\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)=\sum_{i=1}^{N}\alpha_T(i) \quad (8)$$

And the probability of being in state *Si* at time *t*, given an observations sequence ***O***, and the model λ, as:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)}$$

The probability of being at state S_i at time *t* and state S_j at time *t+1* is:

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij}\, b_j(\boldsymbol{O}_{t+1})\, \beta_{t+1}(i)}{P(\boldsymbol{O} \mid \lambda)}$$

2.2. Problems of the HMM

In the case in which we wish to use an HMM with a discrete observation density, rather than the continuous observation vectors of parameters $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, …., $\boldsymbol{x}_T$, a vector quantifier (VQ) is required to map each continuous observation vector into a discrete codebook index. Let the code words of the codebook be $\{\boldsymbol{v}_k\}_{k=1,\ldots,M}$ where *M* is the codebook size. Then, the vector $\boldsymbol{O}_t$ of the observation sequence $\boldsymbol{O}_1$, $\boldsymbol{O}_2$, …, $\boldsymbol{O}_T$ is obtained as follows:

$\boldsymbol{O}_t=\boldsymbol{k}$*, with k is the index of the code word* $\boldsymbol{v}_k$ *iff* $d(\boldsymbol{x}_t,\boldsymbol{v}_k)<d(\boldsymbol{x}_t,\boldsymbol{v}_m)$ *for all* $m\neq k$

where $d(\boldsymbol{x}_t,\boldsymbol{v}_k)$ is the distance between $\boldsymbol{x}_t$ and $\boldsymbol{v}_k$. As the number of possible emitted vectors is *M*, the distribution of observation vector in each state $b_j(\boldsymbol{O}_t)$ used in forward backward procedure and Viterbi algorithm is defined by $b_j(\boldsymbol{v}_k)$ being $b_j(\boldsymbol{v}_k)$ the probability of code vector $\boldsymbol{v}_k$ at each state.

To calculate the vector quantifier (VQ also called library of symbols), we have implemented two algorithms. The k-mean and the LBG algorithms create a library of M elements thanks to the training set:

VQ codebook design:

- We want to design the codebook for each parameter in each class.
- We try to find the codebook size and vectors in order to have the overall distortion minimized.

- With fpVQ the training data set and s a vector from fpVQ, R the centroids of the library with r a vector of R (in the function is an element from biblio). And so, d(s, r)= ||s-r||^2

  LBG (Linde, Buzo and Gray) algorithm:
  1. initialization of the library with the centroid r calculated with the vectors fpVQ from this class
  2. Define the vectors r1 = r + ε, r2 = r- ε
  3. The closest vectors from r1 (r2) are s1 (s2)
  4. Search for the centroids from s1 and s2
  5. Make the steps 3-4 several times. UmbralVQ and maxiterVQ are the conditions to stop.

Make 1 to 5 up to obtain the desired numbers or this class

The algorithm k-mean is used to compute the N centroids in each library associated to each parameter in each class. The k-mean problem is to minimize the mean square error (MSE).The idea, in this algorithm, is to build the N centroids adding N new training vectors by step. We update the centroids step by step.

1. The N first training vectors are the centoids(0).
2. Each vector added is assigned to the closest centroid(t) and the centroids(t+1) are recalculated with the new vectors added
3. The algorithm is terminated when centroid(t)= centroid(t+1)
   (in our case it will be terminated when
   Σ ||centroid(t)-centroid(t+1)|| < threshold
   or when the iteration number > maxiter)

2.2.1. Training

But our problem is not to work out the probability of a observation sequence but to model a signature described by a continuous observation vectors of signature parameter $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T$ via an HMM model. To calculate the HMM model of a signature, we associate the observations sequence with the signatures parameters sequence, that is $\boldsymbol{O}_t=\boldsymbol{x}_t$, and to adjust the model parameter $\lambda=(A,B,\Pi)$ to maximize the probability of the observation sequence. The probability maximization is done for the parameters sequences of all the signatures repetitions.

We accomplish the above task thanks to the iterative Baum-Welch methods, which is equivalent to the EM (expectation-modification) procedure.

- ✓ The Baum-Welch method works as follow:

1. Estimate an initial HMM model as $\lambda= (A, B, \Pi)$.
2. Given λ and the observation sequence $\boldsymbol{O}$, we calculate a new model $\overline{\lambda} = (\overline{A}, \overline{B}, \overline{\Pi})$ such that $P(\boldsymbol{O} \mid \overline{\lambda}) > P(\boldsymbol{O} \mid \lambda)$.
3. If the improvement $\dfrac{P(\boldsymbol{O} \mid \overline{\lambda}) - P(\boldsymbol{O} \mid \lambda)}{P(\boldsymbol{O} \mid \overline{\lambda})} < threshold$, then stop.

Put $\overline{\lambda}$ in place of λ and go to step 1.

In this case, discrete HMM, the formulas of Baum-Welch method used in this work to estimate the model $\lambda=(A,B,\Pi)$ (step 2) are the next:

$$\overline{\pi}_i = \gamma_1(i) \qquad \overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T} \gamma_t(i)} \qquad \overline{b}_j(\mathbf{v}_k) = \frac{\sum_{t=1}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}$$

- ✓ Multi-labeling

In the discrete hidden Markov model (DHMM) approach, the conventional VQ technique is applied. For each incoming vector, the quantifier performs a hard decision about which of its code word is the best match, and so the information about how the incoming vector matches other code words is discarded. Because of the signature variability, the vector of parameters can be displaced in such a way that the displacement is a potential source of misrecognition.

Unlike the conventional VQ, multi-labeling makes a soft decision about which codeword is the closest to the input vector, generating an output vector

whose components indicate the relative closeness of each codeword to the input.

So, the multi-labeling codebook used in this work maps the input vector $\boldsymbol{x}_t$ into an observable vector $\boldsymbol{O}_t=\{w(\boldsymbol{x}_t,\boldsymbol{v}_k)\}_{k=1,\ldots,C}$, whose components are calculated with

$$w(\mathbf{x}_t, \mathbf{v}_k) = \frac{1/d(\mathbf{x}_t, \mathbf{v}_k)}{\sum_{m=1}^{C} 1/d(\mathbf{x}_t, \mathbf{v}_m)}$$

These components are positive, their sum is 1. Thus, they provide a heuristic measure describing the likelihood that the input vector $\boldsymbol{x}_t$ would be derived from the class represented by the codeword $\boldsymbol{v}_k$. Under the standard DHMM approach, $w(\boldsymbol{xt},\boldsymbol{v}_k)$ would be taken value 1 for the code word with the best match and value 0 for the rest. In this case, the probability of an observable vector $b_j(\boldsymbol{O}_t)$ is given by

$$b_j(\mathbf{O}_t) = \sum_{k=1}^{C} w(\mathbf{x}_t, \mathbf{v}_k) b_j(\mathbf{v}_k)$$

With respect to Baum-Welch reestimation formulas for the transition probabilities a_{ij} and the initial state probabilities π_i are generalized in the same way. Regarding the reestimation of $b_j(\boldsymbol{v}_k)$, the better recognition scores were obtained just using the next heuristic reestimation formula

$$\overline{b}_j(\mathbf{v}_k) = \frac{\sum_{t=1}^{T} \gamma_t(i)\, w(\mathbf{x}_t, \mathbf{v}_k)}{\sum_{t=1}^{T} \gamma_t(i)}$$

Although above equation does not guarantee the convergence of the training process, in practice its use decrease the required number of iterations. Furthermore, multi-labeling can be simplified using, to work out the observable vector $\boldsymbol{O}_t$, only the L most significant values of $w(\boldsymbol{x}_t,\boldsymbol{v}_k)$ for each $\boldsymbol{x}_t$ (L labels), where L is lower than the codebook size C. The corresponding reductions in computational load make the multi-labeling Hidden Markov Model (MLHMM) approach extremely efficient. The MLHMMM approach is closely related to the Semi Continuous approach.

- ✓ Multi-parameter Vector or Gathering

The gathering is the fact to group together different types of characteristics stored in a vector pattern. Instead of gathering all the parameters in one HMM, the "gathering method" builds one HMM by group of parameters. The input $\boldsymbol{x}_t$ of d-dimension is now described as a group of vector of R characteristics with the sum of the dimension for each characteristic d(r) equal to d.

The parameter p dimensional vector $\boldsymbol{x}_t$ can be a set of R parameters p_r dimensional vector $\boldsymbol{x}_t^r$ linked together, that is $\boldsymbol{x}_t = (\boldsymbol{x}_t^1, \boldsymbol{x}_t^2, ..., \boldsymbol{x}_t^r, ..., \boldsymbol{x}_t^R)$. The above described HMM algorithm will see all the parameters as a vector with just one p-dimensional parameter. To take into account each parameter in a isolated way, we propose to modify the calculation of the observation vector probability $b_j(\boldsymbol{O}_t)$. The other formulas are unchanged.

In the continuous case, where $\boldsymbol{O}_t=\boldsymbol{x}_t$, $b_j(\boldsymbol{O}_t)$ is calculated as

$b_j(\boldsymbol{O}_t) = \prod_{r=1}^{R} b_j(\boldsymbol{O}_t^r)$ being $b_j(\mathbf{O}_t^r) = \sum_{m=1}^{M} c_{jm}^r \aleph(\mathbf{O}_t^r, \boldsymbol{\mu}_{jm}^r, \mathbf{U}_{jm}^r)$

In the discrete case, we calculate a number of *R* vector quantifiers with code vectors $\{\boldsymbol{v}_k^r\}_{k=1,2,...,Cr}^{r=1,2,...,R}$, and quantify each parameter x_t^r with its vector quantifier as follows

$$\boldsymbol{O}_t^r = \boldsymbol{v}_k^r \;\; iff \;\; d(\boldsymbol{x}_t^r, \boldsymbol{v}_k^r) < d(\boldsymbol{x}_t^r, \boldsymbol{v}_m^r) \;\; for \;\; all \;\; m \neq k$$

and

$b_j(\mathbf{O}_t) = \prod_{r=1}^{R} b_j(\mathbf{O}_t^r)$ being $b_j(\boldsymbol{O}_t^r) = b_j(\boldsymbol{v}_k^r)$

$$\sum_{t=1}^{T} \gamma_t(i)$$

and the reestimation formula its $\overline{b}_j(\boldsymbol{v}_k^r) = \frac{s.t.\boldsymbol{O}_t^r = \boldsymbol{v}_k^r}{\sum_{t=1}^{T} \gamma_t(i)}$

Finally, in the multi-labeling case, to take into account each parameter is an isolated way; we mapped each input vector $\boldsymbol{x}_t$ into an observable vector

$\boldsymbol{O}_t = \{w^1(\boldsymbol{x}_t^1, \boldsymbol{v}_k^1), ..., w^r(\boldsymbol{x}_t^r, \boldsymbol{v}_k^r), ..., w^R(\boldsymbol{x}_t^R, \boldsymbol{v}_k^R)\}$ where

$$w^r(\boldsymbol{x}_t^r, \boldsymbol{v}_k^r) = \frac{1/d(\boldsymbol{x}_t^r, \boldsymbol{v}_k^r)}{\sum_{m=1}^{C_r} 1/d(\boldsymbol{x}_t^r, \boldsymbol{v}_m^r)}$$

being the distribution probability worked out as

$$b_j(\boldsymbol{O}_t) = \prod_{r=1}^{R} b_j(\boldsymbol{O}_t^r) = \prod_{r=1}^{R} \left(\sum_{k=1}^{C_r} w(\boldsymbol{x}_t^r, \boldsymbol{v}_k^r) b_j(\boldsymbol{v}_k^r)\right)$$

and the reestimation formula

$$\overline{b}_j(\boldsymbol{v}_k^r) = \frac{\sum_{t=1}^{T} \gamma_t(i)\, w^r(\boldsymbol{x}_t^r, \boldsymbol{v}_k^r)}{\sum_{t=1}^{T} \gamma_t(i)}$$

2.2.2. Classification

We create a HMM for each class and each group of parameters to be classified. Then, as mentioned-above, we train each HMM with its own training set. To train the HMM of the 1st class, we only use the training set of the first class, and so on for the second class up to the end. The Viterbi algorithm can be used to obtain the estimation of the most probable state sequence. Once all the HMMs $\Lambda=(\lambda^1 ... \lambda^W)$ are correctly trained, $P_w=P(O|\lambda^W)$ is calculated for all the λ^W, in order to classify a sequence for the observation ***O***. The unknown observation ***O*** is then classified by the process:

$$w^* = \arg\max_{1 \leq w \leq W} p_w \qquad (12)$$

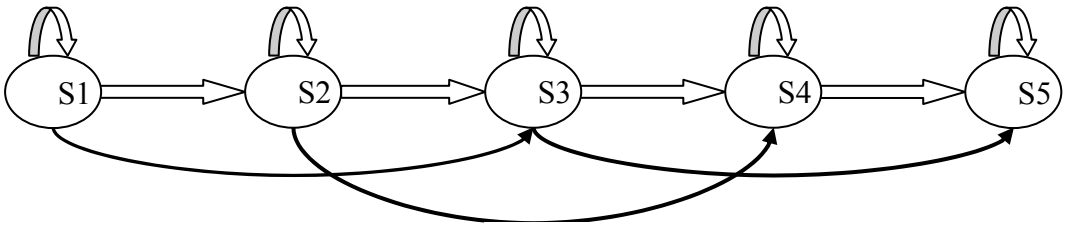
And so, w^* is the optimum class for the observation ***O***.

The initialization and stop criteria must be chosen adequately with the HMM. It directly interacts on the relevancy of the HMM [13]. Equi-probable and equal occupancy methods for the initial models are provided as well as iteration and rate of the error for the stop criterion.

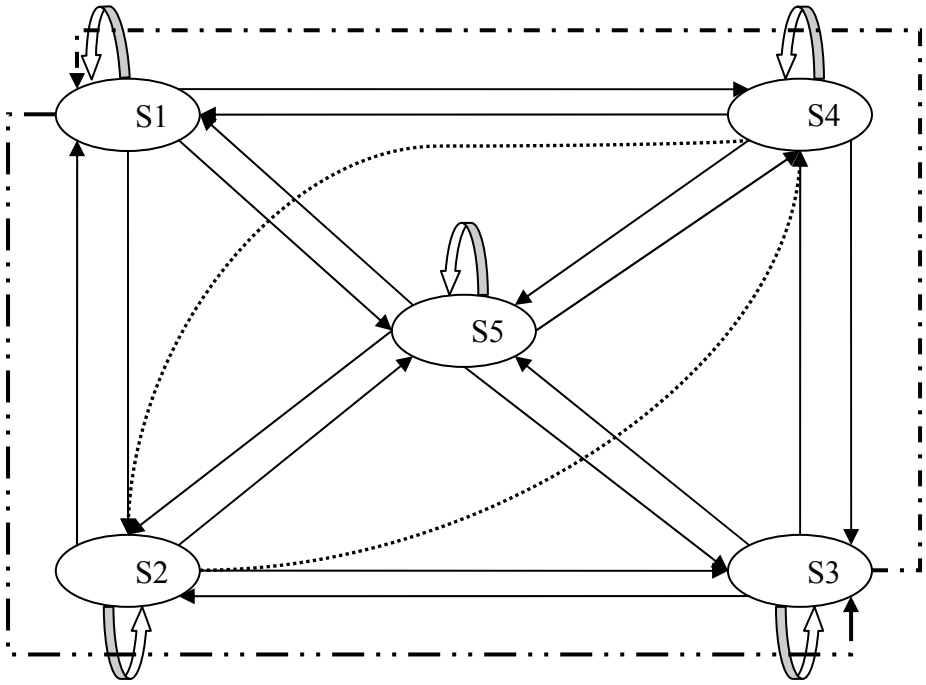
To evaluate the efficiency of the HMM, a function calculates, thanks to the results of the HMM, a matrix of confusion for each set of parameters. The matrix of confusion is a matrix built during the test phase. It shows how and where the HMM fails. Thus, the recognition rate and the matrix of confusion give a good idea about the pertinence for the given set of parameters within the recognition task.

2.3. Types of HMM

There are different types of HMM. In our case, we will present the ergodic and the Bakis (also called left-right HMM) models.



(a)



(b)

Illustration 2:1 illustration of 2 distinct types of HMM. (a) a 5 state left-right HMM (b) a 5 state ergodic model.

The ergodic or fully connected HMM is a HMM with all states linked together (every state can be reached from every others states).

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} with \quad a_{ij} \geq 0$$

The left-right (also called Bakis) is a HMM with the matrix transition defined as:

$a_{ij} = 0 \quad if \quad j < i$ and $a_{ij} = 0 \quad if \quad j > i + \Delta$

For the particular case of a 5 state Bakis with Δ=2, we have the following transition matrix:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 \\ 0 & a_{22} & a_{23} & a_{24} & 0 \\ 0 & 0 & a_{33} & a_{34} & a_{35} \\ 0 & 0 & 0 & a_{44} & a_{45} \\ 0 & 0 & 0 & 0 & a_{55} \end{bmatrix}$$

There are many other HMM types, but with those two classical HMM, the toolbox provides the types of HMM most used. The left-right model is widely used in the problem of voice recognition for example and the ergodic can be used for nearly all problems as it is the more generic type of HMM.

3. Description of the DHMM part of the toolbox

In this part, we describe the different functions developed in the toolbox in order to manage a discrete Hidden Markov Model. Firstly, the main functionalities are summarized for each function and then, we explain its use and the parameters to take into account.

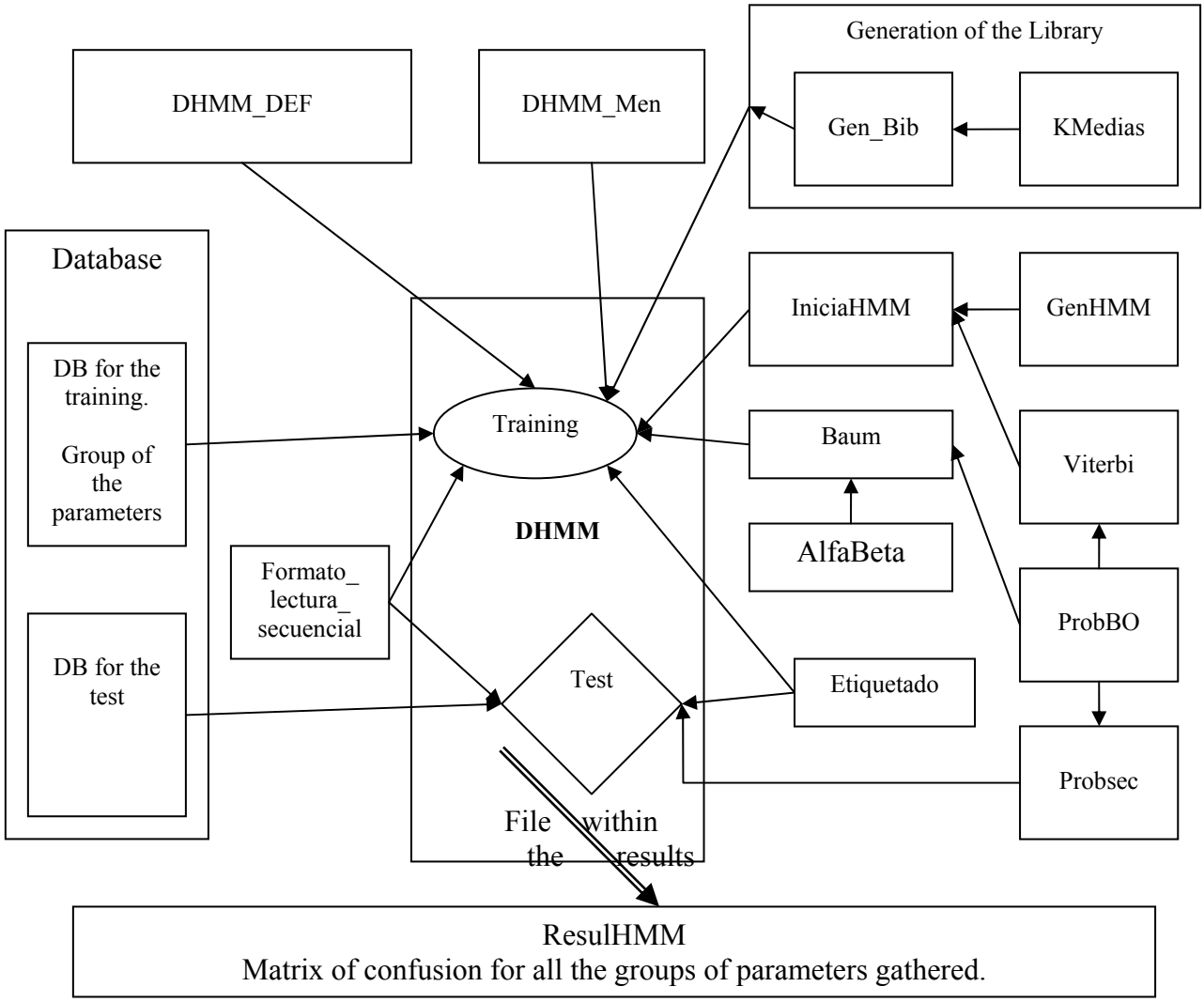


Illustration 3:1 Block diagram of the toolbox for a DHMM.

Some of the functions described in this toolbox are only proposed as examples. In particular, the functions DHMM and DHMM_DEF must be adapted for your own HMM. See the examples provided to have further information about it. In the second part of this point, each function is described with the formats of the inputs and outputs parameters.

Alfa: this function calculates the Alfa from the HMM defined with the matrix A, B and Pi. The Alfa is scaled to avoid the precision problem.
AlfaBeta: this function calculates the Alfa and Beta from the HMM defined with the matrix A, B and Pi. The Alfa is scaled to avoid the problem of the precision.

Baum: this function computes the Baum Welch algorithm in order to estimate the HMM parameters.
DHMM_DEF: this script defines the parameters of the discrete HMM. The type of the HMM and the method to quantify the library are chosen.
DHMM: this is a script to train and test the HMM.
DHMM_MEN: this script is a small program that prints some messages to describe the computation of the HMM.
Etiquetado: this function implements the multi-labeling for the discrete HMM. To do that, we use a clustering algorithm like the k-mean to quantify the library VQ. Given a parameter, the multi-labeling allows to give it various labels. The possible labels are directly linked to the number of symbol by state.
formato_lectura_secuencial: this function splits the database to smaller files (located in the c:\temphmm directory) in order to decrease the HMM computational time and the memory requirement.
genHmm: this function generates a HMM with N states and M symbols by state.
gen_bib: this function computes a library for all the different models of parameters based on the training vectors. According to the set up made in the DHMM_DEF, the library is generated with a LBG algorithm or with a k-means algorithm.
iniciaHMM: this function calculates the HMM optimal initial model for the Baum-Welch training.
kMedias: this function computes a k-mean algorithm for the library VQ.
prodBO: this function calculates the product of the probability to monitor a sequence ***O*** with the probability of the backward B distribution.
ProbSec: this function estimates the log of the probability to monitor a sequence ***O*** given a HMM. We use the log to avoid numerical problem, (formula 4).
Resulhmm: this function calculates the confusion matrix group by group according to the gathering made in the DHMM_DEF.
ROC: this function analyses the results from the HMM. It particularly deals with the problem of false acceptance rate (FAR called here FMR false match rate) and false rejection rate (FRR called here FNMR false non match rate).
Viterbi: this function calculates the sequence of the most probable states given the HMM and the sequence monitored ***O***. We use the algorithm of Viterbi with the log for numerical precision problems.

Each function is now described by alphabetic order. This description is focused on the inputs and outputs parameters. Please refer to the block diagram, to see how the functions interact together,

- ✓ Alfa

This function calculates the alpha from the HMM defined with the matrix A, B and Pi. The alfa is scaled to avoid the precision problem.

```
function [alfa,c]=Alfa(A,B,Pi,O)
```

Entry: A(N, N) is the transition probability matrix from a state I to a state k.

B{Np}(N,M) is the distribution probability symbol matrix for each parameter.
B{ip}(i,k) is the probability to obtain the k^{th} symbol when we are at state I for the set of parameter ip for this HMM (the HMM of a class and a group).

Pi(N,1) is the distribution probability for the initial state i.
Pi(i) is the probability to start at state i.

O is a matrix with the sequence monitored.

Result: alfa is the forward probability matrix.

alfa(i,t) when alfa is not scaled is the probability to observe the sequence O(1,:),….,O(t,:) for state i at the instant t.
We have in particular P(O/(A, B, Pi))= sum(alfa(:,T)).

When alfa is scaled, we have sum(alfa(:,t))=1.

c(T,1) is the vector where we store the scale value for the instant t.
We have c(T,1)= ones(T,1) if the alfa is not scaled.

- ✓ AlfaBeta

This function calculates the alpha and beta from the HMM defined with the matrix A, B and Pi. The alfa and Beta are scaled to avoid the problem of the precision.

```
function [alfa,beta,c]=alfabeta(A,B,Pi,O)
```

Entry: A(N, N) is the transition probability matrix from a state I to a state k.

B{ip}(i,k) is the probability to obtain the k^{th} symbol when we are at state I for the set of parameter ip for this HMM (the HMM of a class and a group).

Pi(N,1) is the distribution probability for the initial state i.
Pi(i) is the probability to start in the state i.

O is a matrix with the sequence monitored.

Result: alfa is the forward probability matrix.
alfa(i,t) when alfa is not scaled is the probability to observe the sequence O(1,:),….,O(t,:) for the state i at the instant t.
We have in particular P(O/(A, B, Pi))= sum(alfa(:,T)).
When alfa is scaled, we have sum(alfa(:,t))=1.

beta is the backward probability matrix.
beta(i,t) when beta is not scaled is the probability to observe the sequence O(t+1,:),….,O(T,:) for the state i at the instant t.

c(T,1) is the vector where we store the scale value for the instant t.
We have c(T,1)= ones(T,1) if the alfa and beta are not scaled.

We have the following relations on the alfa and beta:

Relations between alfa scaled and not scaled

```
h=cumprod(c);
alfa_scaled(:,t)= h(t)*alfa_no_scaled;
```

Relations between beta scaled and not scaled

```
g=cumprod(c(T:-1:1);
beta_scaled(:,t)=g(t)*beta_no_scaled(:,t);
```

If alfa and beta are not scaled, the probability to observe O is (this product is independent of t):

```
alfa( :,t)'*beta( :,t) = sum (alfa(T,:))
```

If alfa and beta are scaled (it depends on t):

```
alfa(t,:)'*beta(t,:) = c(t)*sum (alfa(T,:))
```

- ✓ Baum

This function computes the Baum Welch algorithm in order to estimate the HMM parameters.

function [A1,B1,Pi1,logPOs]=baum(A,B,Pi,lrep)

A, B and Pi are the matrices of the HMM defined with the function genHMM for example.

Entry: A(N, N) is the transition probability matrix from a state I to a state k.

B{ip}(i,k) is the probability to obtain the k^{th} symbol when we are at state I for the set of parameter ip for this HMM (the HMM of a class and a group).

Pi(N,1) is the distribution probability for the initial state i.
Pi(i) is the probability to start in the state i.

Lrep is a matrix with the size of the sequence of labels monitored for this HMM (we have the sequence of labels for all the parameters of the HMM's group and class). It contains all the repetitions of the training set for this HMM (the HMM of a class and a group).

Result: A1(N, N) is the transition probability matrix updated with the Baum Welch algorithm.
B1(N, N) is the distribution probability symbol matrix updated with the Baum Welch algorithm.
Pi(N,1) is the distribution probability for the initial state updated.
logPOs(nr,1): probability that the HMM with the parameters (A,B,Pi) generates each realisation of the labels entries of the subroutine.

This function calls the alfabeta function. The sequences of labels are loaded from the files: c:\temphmm\vpl.tmp during the algorithm.

- ✓ DHMM_DEF

This function is described only as an example to set up the HMM.

function dhmm_def(fhmm)

This function defines the parameters of the discrete HMM. We can split the parameters in 4 groups:

Entry: fhmm is the HMM´s file name.

- ✓ vDB defines the database:
  vDB=[' nc ng agrup Np'];
  where nc is the number of class, ng is the number of group,
  agrup defines how to gather the parameters together. For example, if we have two parameters for the first group. If we fix agrup{1}=[1 3]; we create only one library for this group of parameter. If we fix agrup{1}=[1 2 3]; we create two libraries (one for the first parameter and one for the second parameter of the group). And so on, with three parameters, for agrup{1}=[1 3 4]; we create a library for the first and second parameters and one for the third parameters.

  NP is a vector with the number of each group of parameter.

We have the following relations:
[nc,ng]=size(vl);
agrup{ig}=[1 .... size(vl{1,ig}{1},2)+1];
Np(ig)=length(agrup{ig})-1;

- ✓ vHMM defines the HMM :

vHMM=[' BAKIS salto maxiter umbral maxitermi TOPN Ne Ns A B Pi'];

  if Bakis = 1 implements a Bakis HMM (also called left-right), else an ergodic HMM is defined
  Salto is the maximum path authorized in the Bakis HMM from a state to another.
  Maxiter is the maximum iteration number in the Bakis HMM (condition to stop the algorithm)
  Umbral is the threshold condition to stop the HMM (the error is calculated with the maximum likelihood criterion)
  TOPN is the number of labels to take into account for the multi-labeling during the training phase. For example, TOPN{ig}=1.*ones(Np(ig),1); create one label for all the parameters of all the groups. If we want to put two labels TOPN{ig}=2.*ones(Np(ig),1), we have to change topntest when we change the number of labels too. We can also set a number of labels different for each set of parameters and group. For instance: TOPN{4}=4.*ones(Np(4),1); and TOPN{ig}=1.*ones(Np(ig),1); for ig>1. By experience, the results are quite good fixing only one label by parameter.

  Ne(nc,ng): is the number of states of the HMM for each class and group. We could fix a different number of states for each group ig and each class ic. For example, Ne(1,1) =10 and Ne(2,2)=20…In this

case we have 10 states for the HMM of the first class and first group and 20 for the HMM of the second class and second group.
Ns{ng}(Np): It is the number of symbols for each parameter of each group. We could fix a number of symbols different for each group ig and parameter of the group ip.
A, B and Pi are the matrices of the HMM (it will be calculated thanks to the HMM) but must de defined here.
cell array A=cell(nc,ng);
cell array B=cell(nc,ng);
cell array Pi=cell(nc,ng);

Salhmm is a matrix used to store the probabilities values
We have the following relations:
TOPN=cell(ng,1);

- ✓ vQ is the matrix parameter for the library :
  vVQ=[' LBG dpztoLBG maxiterVQ umbralVQ men biblio'];
  where LBG is the choice of the algorithm to compile the library
  LBG = 1 the algorithm is LBG, in the other case we choose kMedias
  dptzoLBG percentage maximum for the distance of the code vector in the algorithm LBG (see gen_bib)
  maxiterVQ is the maximum number of iteration to compute the library (condition to stop the algorithm)
  umbralVQ is the threshold condition to stop the library computation (condition to stop the algorithm)
  men=1 or 0 if men =1 the library generates message else no.
  biblio is a matrix parameter for the library computation (we define it here but calculate in the gen_bib function)

- ✓ vTEST is the matrix parameter for the test :
  vTEST=[' TOPNtest salhmm'];
  where TOPNtest is the number of labels to take into account for the multi-labeling during the test phase (this works as the TOPN of the training).
  Salhmm is a matrix used to store the probabilities values

We have the following relations:
TOPNtest=cell(ng,1);
salhmm=cell(nc,ng);

- ✓ DHMM

This program calls the different functions to design the HMMs (one for each class and for each group) designed in the function DHMM_DEF. It is here as example to make the HMM work.

function dhmm(fhmm,fptrain,fptest,fhmmout)

Entry: fhmm is the name of the HMMs set up in the function DHMM_DEF

fptrain is the name of the file containing the sequences of parameters to train each HMM with its own group of training set. In each repetition, we find a sequence of parameters for a class and a group.

fptest is the name of the file containing the sequences of parameters to test each HMM with its own group of test set. In each repetition, we find a sequence of parameters for a class and a group.

fsalhmm: Name of the file containing the outputs of each classifier for each sample of the database of the test in a cell array: salhmm{class, group}(repetition).

Variables defined in the fhmm file with the function DHMM_DEF

nc: number of classes.
ng: number of groups.
agrup{ng}: the way to gather the parameters together.
Np(ng): number of parameters by group.
Ne(nc,ng): Number of states of the HMM for each class and group. . We could fix a number of states different for each group ng and each class. For example, Ne(1,1) =10 and Ne(2,2)=20…In this case we have 10 states for the HMM of the first class and first group and 20 for the HMM of the second class and second group.
Ns{ng}(Np): The number of symbols for each parameter of each group. We could fix a number of symbol different for each group ng and parameter of the group Np.
TOPN{ng}(Np): Number of labels for the multi-labeling for each parameter of each group.
Maxitermi is the maximum iteration for the initial model
umbral: Umbral is the threshold condition to stop the HMM (the error is calculated with the maximum likelihood criterion)
maxiter: Maxiter is the maximum iteration number in the Bakis HMM (condition to stop the algorithm)
salto: Salto is the maximum path authorized in the Bakis HMM from a state to another.
BAKIS: if Bakis = 1 implements a Bakis HMM (also called left-right), else an ergodic HMM is defined

The variables of the HMM:

cell array A=cell(nc,ng);

cell array B=cell(nc,ng);
cell array Pi=cell(nc,ng);

The libraries are calculated thanks to the set up made in the DHMM_DEF. We calculate the library for each group and class. The number of centroids is equal to the number of symbols. We store the library in the cell array biblio:
biblio{group}

maxiterVQ is the maximum number of iteration to compute the library (condition to stop the algorithm)
umbralVQ is the threshold condition to stop the library computation (condition to stop the algorithm)
men=1 or 0 if men =1 the library generates message else no.
LBG = 1 the algorithm is LBG, in the other case we choose kMedias
dptzoLBG percentage maximum for the distance of the code vector in the algorithm LBG (see gen_bib)

TOPNtest is the number of labels to take into account for the multi-labeling during the test phase
Salhmm is a matrix used to store the probabilities values

NOTA: use the scripts dhmm_def, and dhmm_men.
NOTA: use the functions: etiquetado, iniciaHMM, genhmm, alfabeta, alfa, probsec, viterbi, baum, gen_bib and kmedias
NOTA: if fptrain=" " we only realize the test
NOTA: if fptest=" " we only make the training

- ✓ DHMM_MEN

This script is a small program that prints some messages to describe the computation of the HMM.

- ✓ Etiquetado

This function implements, for the discrete HMM, the multi-labeling. To do that, we use a clustering algorithm like the k-mean to quantify the library VQ (for further information on the k-mean algorithm see kmedia function). Given a parameter, the multi-labeling gives various labels (this defined for each parameter of each group in the TOPN). The number of labels is defined in the TOPN for each parameter of each group. It can not be bigger than the number of symbols by state.

This function, in particular, labels all the data set given a class and a group.
function pl=etiquetado(vl,agrup,Ns,biblio,TOPN)

Entry: v1 is the database.
agrup defines how to gather the parameters together
Ns number of symbols by state
Biblio is the matrix of the library given this class and this group.
TOPN number of labels to take into account for the multi-labeling during the training phase

Result: p1 is a matrix where we store the labels for all the vectors.

- ✓ formato_lectura_secuencial

This function splits the database to smaller files in order to decrease the HMM computational time.
function [lrep,lgrupo]=formato_lectura_secuencial(fptrain,fplecsec,agrup,fpVQ)

Entry: fptrain is a file containing the full database.
fplecsec: the database split and stored at the direction c:\temphmm\fplecsec_cclase_ggrupo.mat
agrup: if we have agrup, we must prepare the training data set for VQ with a clustering algorithm (this is in particular the case for the multi-labeling for DHMM).
fpVQ: is a file containing the training database for VQ.

Result: lgrupo: is a vector containing the number vectors for every group.
Lrep: is a matrix with the length for each class, each group and repetition. lrep{ic,ig}(ir) contains the length of the ic^{th} class, ig^{th} group and ir^{th} vector (also called repetition).

- ✓ genHmm

This function generates a HMM with N states and M symbols by state.
function [A,B,Pi]=genhmm(N,M,Np,BAKIS,salto);

Entry: N is the state number
M is the symbols number by state
Np is the number of parameters for each symbol
if Bakis = 1 implements a Bakis HMM (also called left-right),else an ergodic HMM is defined
Salto is the maximum path authorized in the Bakis HMM from a state to another.

Result: A, B and Pi are the matrices of the HMM defined within this function.

A(N, N) is the transition probability matrix with the following conditions:

$\Sigma j\ Aij = 1$ (sum(A(i,:)=1))
$Aij \geq 0$ (A(i,j) ≥ 0)
For a Bakis HMM, we have aij=0 if j<i (A(i,j)=0 for j<i)

B(N,M) is the distribution probability symbol matrix
B(i,k) is the probability to obtain the k^{th} symbol when we are at state i.

$Bik \geq 0$ (B(i,k) ≥ 0)
$\Sigma j\ Bij = 1$ (sum(B(i,:)=1))

Pi(N,1) is the distribution probability for the initial state i.
Pi(i) is the probability to start in the state i.

$\Sigma\ Pi(i) = 1$
For a Bakis HMM $Pi(i) = \delta(i)$
With $\delta(i)$ is the Kronecker function equal to 1 for i=0 and null for i<>0

We define here the matrix and update it within the training process of the HMM like the Baum function.

- ✓ gen_bib

function
biblio=gen_bib(fpVQ,Nv,agrup,Ns,LBG,dpztoLBG,maxiterVQ,umbralVQ,
men,biblio)

This function computes a library for all the different models' parameters (a library for) based on the training vectors.
According the set up made in the DHMM_DEF, the library is generated with the LBG algorithm or with the k-means algorithm (see kmedias function for further information about the k-means).

Entry: fpVQ is the matrix with the vectors to train the library (see DHMM and formato_lectura_secuencial functions)
Nv number of vectors for the training
agrup defines how to join the parameters together
LBG is the choice of the algorithm to compile the library
LBG = 1 the algorithm is LBM, in the other case we choose kMedias
dptzoLBG percentage maximum for the distance of the code vector in the algorithm LBG
maxiterVQ is the maximum number of iteration to compute the library (condition to stop the algorithm)
umbralVQ is the threshold condition to stop the library computation (condition to stop the algorithm)
men=1 or 0 if men =1 the library generates message else no.
biblio is a matrix parameter for the library computation (we define it in DHMM_DEF but calculate it here)

Result: biblio is now the library.

This function uses the kmedias function.

Algorithm (LBG) to design the library.

VQ codebook design:
We want to design the codebook for each parameter in each class.
We try to find the codebook size and vectors in order to have the overall distortion minimized.
With fpVQ the training data set and s a vector from fpVQ
R the centroids of the library with r a vector of R (in the function is an element from biblio).
d(s, r)= ||s-r||^2

LBG (Linde, Buzo and Gray) algorithm:

6. initialization of the library with the centroid r calculated with the vectors fpVQ from this class
7. Define the vectors r1 = r + ε, r2 = r- ε
8. The closest vectors from r1 (r2) are s1 (s2)
9. Search for the centroids from s1 and s2
10. Make the steps 3-4 several times. UmbralVQ and maxiterVQ are the conditions to stop.
11. Make 1 to 5 up to obtain the desired numbers or this class

- ✓ iniciaHMM

This function calculates the HMM optimal initial model for the Baum-Welch training.
function [Aa,Ba,Pia]=iniciahmm(Ne,Ns,Np,BAKIS,salto,lrep,maxitermi)

Entry: if Bakis = 1 implements a Bakis HMM (also called left-right), else an ergodic HMM is defined
Salto is the maximum path authorized in the Bakis HMM from a state to another.
Maxitermi is the maximum iteration to calculate the initial HMM.
Ne state number
Ns symbol number for every state
Np number parameters by symbol
Lrep is a Matrix with the size from the training set data for every classes groups and parameters (see also the function formato_lectura_secuencial)

Result: Aa Ba and Pia are the matrices for the parameters of the HMM.

In this function, we will generate a number of symbols in every state bigger than twice Ns. Thus, we make sure to initialize correctly the HMM.

We use the viterbi and the genhmm functions.

- ✓ kMedias

This function computes a k-mean algorithm for the library VQ.
function [biblio]=kmedias(fpVQ,Lt,biblio,maxiter,umbral,men);

Entry: fpVQ: vectors for the training if the library
Lt is the vectors numbers for the training
Biblio: library initial
Maxiter: number maximum for the iteration
Umbral is the threshold condition to stop the library computation
Men =1 this function generates messages else no.

Result: Biblio: library after the training

The algorithm k-mean is used to compute the N centroids in each library associated to each parameter in each class. The k-mean problem is to minimize the mean square error (MSE).
The idea, in this algorithm, is to build the N centroids adding N new training vectors by step. We update the centroids step by step.

1. The N first training vectors are the centoids(0).
2. Each vector added is assigned to the closest centroid(t) and the centroids(t+1) are recalculated with the new vectors added.
3. The algorithm is terminated when centroid (t)= centroid(t+1)
(in our case it will be terminated when
Σ ||centroid(t)-centroid(t+1)|| < threshold
or when the iteration number > maxiter)

- ✓ ProbSec

This function estimates the log of the probability to monitor a sequence O given an HMM. We use the log to avoid numerical problem.

```
function [logPO,logalfaT]=probsec(A,B,Pi,O)
```

Entry: A(N, N) is the transition probability matrix from a state I to a state k.

B{Np}(N,M) is the distribution probability symbol matrix for each parameter.
B(i,k) is the probability to obtain the k^{th} symbol when we are at state i.

Pi(N,1) is the distribution probability for the initial state i.
Pi(i) is the probability to start in the state i.

O is a matrix with the sequence to estimate.

Result: logPO is the log of the probability to monitor the sequence O.
logalfaT: is the log (alfa(:, T)) where alfa is the forward probability matrix and T is the running time for the sequence O.

We use in this function the prodBO function.

- ✓ prodBO

This function calculates the product of the probability to monitor a sequence O with the probability of the backward B distribution.

function prob=prodBO(B,O,nc)

Entry: B{Np}(N,M) is the distribution probability symbol matrix for each parameter.
B(i,k) is the probability to obtain the k^{th} symbol when we are at state i.

O is a matrix with the sequence.
O{Np}{number of vectors, number of the symbols}.

nc is the sequence from O to estimate.

Result: prob is the result of the product. That means prob is the probability for every state to generate the extracted component nc from the sequence O.

- ✓ Resulhmm

This function calculates the confusion matrix group by group.

```
function [Mcmed,Mcvot,recmed]=resulhmm(fhmm)
```

Entry: fhmm is the name of the HMM. We will use it to store the results from the HMM.

Result: Mcmed is the matrix of confusion calculated with a mean criterion.
Mcvot is not yet implemented.
recmed is the average of the recognition.

Example: Mc = resulhmm(fhmm);
Mc{agroup}{ng}: Mc is the matrix with groups gathered group by group and the Ng^{th} combination.
Mc{2}{4} is the 4^{th} combination for the groups joined 2 by 2.

- ✓ ROC

This functions analyses the results from the HMM. In particular it deals with the problem of false acceptance rate (FAR called here FMR false match rate) and false rejection rate (FRR called here FNMR false non match rate).

- ✓ Viterbi

This function calculates the sequence of the most probable states given the HMM and the sequence monitored O. We use the algorithm of Viterbi with the log for the numerical precision problems.

function qP=viterbi(A,B,Pi,O)

Entry: A(N, N) is the transition probability matrix from a state I to a state k.

B{Np}(N,M) is the distribution probability symbol matrix for each parameter.
B(i,k) is the probability to obtain the k^{th} symbol when we are at state i.

Pi(N,1) is the distribution probability for the initial state i.
Pi(i) is the probability to start in the state i.

O is a matrix with the sequence to estimate.

Result: qP(T,1) is the sequence of the more probable states.

4. The CHMM

The elements of the continuous HMM (CHMM) are similar to the DHMM case except that in the discrete case, the symbols observed are quantified in a library or VQ codebook. However for the CHMM, the distribution of the symbols emitted is continuous. Moreover, the multi-labeling do not exist for a CHMM since there are no labels. Thus, as for the DHMM, we define the elements of the CHMM.

4.1. Elements of a CHMM

Thus, a CHMM model consists of a number of N states $S=\{S_i\}$ and the observation string produced as a result of emitting a vector $\boldsymbol{O}_t$ each successive transitions from one state S_i to another state S_j. The state transition probability distribution between state S_i to S_j is $A=\{a_{ij}\}$, and the observation probability distribution of emitting any vector $\boldsymbol{O}_t$ at state S_j is given by $B=\{b_j(\boldsymbol{O}_t)\}$. The probability to be in the state i at the initial instant is $Pi=\{\pi_I\}$.

$$a_{ij}= P(q_{k+1}=S_j \mid q_k=S_i) \quad (1)$$

$$b_j(\boldsymbol{O}_t)= P(\mathbf{O}_t \mid q_t=S_j) \quad (2)$$

$$\pi_i = P(q_0=S_i) \quad (3)$$

In order to use the HMM in continuous, we will make some restrictions on the model form of the probability function (PDF). In our case, we will consider that the general observation can be represented by a finite mixture of Gaussians and with a multi parameter approach:

$b_j(\mathbf{O}_t)=\prod_{r=1}^{R} b_j(\mathbf{O}_t^r)$ being $b_j(\mathbf{O}_t^r)=\sum_{m=1}^{M} c_{jm}^r \aleph(\mathbf{O}_t^r,\boldsymbol{\mu}_{jm}^r,\mathbf{U}_{jm}^r)$ (in this case we have M Gaussians and as for the DHMM the part of the parameter r is pointed with the exponent r).
If we do not use the multi parameters, the relation is:

$$b_j(\mathbf{O}_t)=\sum_{m=1}^{M} c_{jm} \aleph(\mathbf{O}_t,\boldsymbol{\mu}_{jm},\mathbf{U}_{jm})$$

Then, given an observation sequence $\boldsymbol{O}$, and a CHMM model $\lambda=(A, B, \Pi)$, we can compute $P(\boldsymbol{O} \mid \lambda)$ the probability of the observed sequence by means of the forward-backward procedure. Both $\alpha_t(i)$ and $\beta_t(i)$ are worked out by means of the forward-backward procedure.

Forward:

$$\alpha_1(i)= \pi_I \; b_i(\boldsymbol{O}_1) \quad (4)$$

$$\alpha_{t+1}(j)=[\Sigma_i \; \alpha_t(i) \; a_{ij}] \; b_j(\boldsymbol{O}_{t+1}) \quad (5)$$

Backward:

$$\beta_T(i)=a_{iN} \quad (6)$$

$$\beta_{t-1}(i) = [\Sigma_j \beta_t(j) * a_{ji}]\, b_i(\mathbf{O}_{t-1}) \qquad (7)$$

We can work out the probability of the observation sequence as:

$$P(\mathbf{O}|\lambda) = \sum_{i=1}^{N} \alpha_t(i)\beta_t(i) = \sum_{i=1}^{N} \alpha_T(i) \qquad (8)$$

And the probability of being in state S_i at time t, given the observation sequence $\boldsymbol{O}$, and the model λ, as:

$$\gamma_t(i,k) = \left[\frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}\right]\left[\frac{c_{ik}\aleph(\mathbf{O}_t, \boldsymbol{\mu}_{jm}, \mathbf{U}_{jm})}{\sum_{m=1}^{M} c_{jm}^{r}\aleph(\mathbf{O}_t, \boldsymbol{\mu}_{jm}, \mathbf{U}_{jm})}\right]$$

The probability of being at state S_i at time t and state S_j at time $t+1$ is:

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij}\, b_j(\mathbf{O}_{t+1})\, \beta_{t+1}(i)}{P(\mathbf{O}|\lambda)}$$

4.2. Problems of the CHMM

4.2.1. Training

As we did for the DHMM, we use the Baum-Welch algorithm to recalculate the HMM model of a signature, and we adjust the model parameter $\lambda=(A,B,\Pi)$ to maximize the probability of the observation sequence. The probability maximization is done for the parameter sequences of all the signatures' repetitions.

We accomplish the above task thanks to the iterative Baum-Welch method, which is equivalent to the EM (expectation-modification) procedure.

The Baum-Welch method works as follows:

1. Estimate an initial HMM model as $\lambda = (A, B, \Pi)$.
2. Given λ and the observation sequence $\boldsymbol{O}$, we calculate a new model $\overline{\lambda} = (\overline{A}, \overline{B}, \overline{\Pi})$ such that $P(\boldsymbol{O}|\overline{\lambda}) > P(\boldsymbol{O}|\lambda)$.
3. If the improvement $\dfrac{P(\boldsymbol{O}|\overline{\lambda}) - P(\boldsymbol{O}|\lambda)}{P(\boldsymbol{O}|\overline{\lambda})} < threshold$, then stop.

Put $\overline{\lambda}$ instead of λ and go to step 1.

Thus, as for the DHMM, we reestimate the transition coefficients "A" and the initials states probabilities Pi with the same formulas. Moreover, for the CHMM, the Baum-Welch algorithm needs to estimate the means and variances for the mixture Gaussians and for the coefficients of the mixture of Gaussians [12].

$$\overline{c}_{ik} = \frac{\sum_{t=1}^{T-1} \gamma_t(j,k)}{\sum_{t=1}^{T}\sum_{k=1}^{M} \gamma_t(j,k)} \qquad (11a)$$

$$\overline{\mu}_{ik} = \frac{\sum_{t=1}^{T-1} \gamma_t(j,k) \bullet O_t}{\sum_{t=1}^{T} \gamma_t(j,k)} \qquad (11b)$$

$$\overline{U}_{ik}=\frac{\sum_{t=1}^{T-1}\gamma_t(j,k)\bullet(O_t-\mu_{jk})(O_t-\mu_{jk})'}{\sum_{t=1}^{T}\gamma_t(j,k)} \qquad (11c)$$

4.2.2. Classification

As for the DHMM, we create a CHMM for each class to classify. Then, as mentioned-above, we train each CHMM with its own training set. To train the HMM of the 1st class, we only use the training set of the first class, and so on for the second class to the end. The Viterbi algorithm can be used to obtain the estimation of the most probable state sequence. Once all the HMMs $\Lambda=(\lambda^1 ...\lambda^W)$ are correctly trained, to classify a sequence for the observation ***O***, $P_w=P(O|\lambda^W)$ is calculated for all the λ^W. The unknown observation ***O*** is then classified by the process:

$$w^*=\arg\max_{1\le w\le W} p_w \qquad (12)$$

Thus, w^* is the optimum class for the observation ***O***.

The initialization and stop criteria must be chosen adequately for the HMM. It directly interacts on the relevancy of the HMM [13]. Equi-probable and equal occupancy methods for the initial models are provided as well as iteration and rate of the error for the stop criterion.

As for the DHMM, we build the matrices of confusion for each group of vector (see the examples for further information). The matrix of confusion is a matrix built during the test phase. It shows how and where the HMM fails. Thus, the recognition rate and the matrix of confusion give a good idea about the pertinence for the given set of parameters within the recognition task.

4.3. Description of the CHMM part of the toolbox

In this part, we describe the different functions developed in the gpdsHMM toolbox in order to manage a continuous Hidden Markov Model. Firstly, the main functionalities, for each function, are summarized and then, we give for each function the way to use it and the parameters to take into account. For the CHMM, we use some of the functions mentioned above for the DHMM.

To use the continuous HMM, you need the netlab package. Please refer to the on-line help from netlab for further details.
http://www.ncrg.aston.ac.uk/netlab/

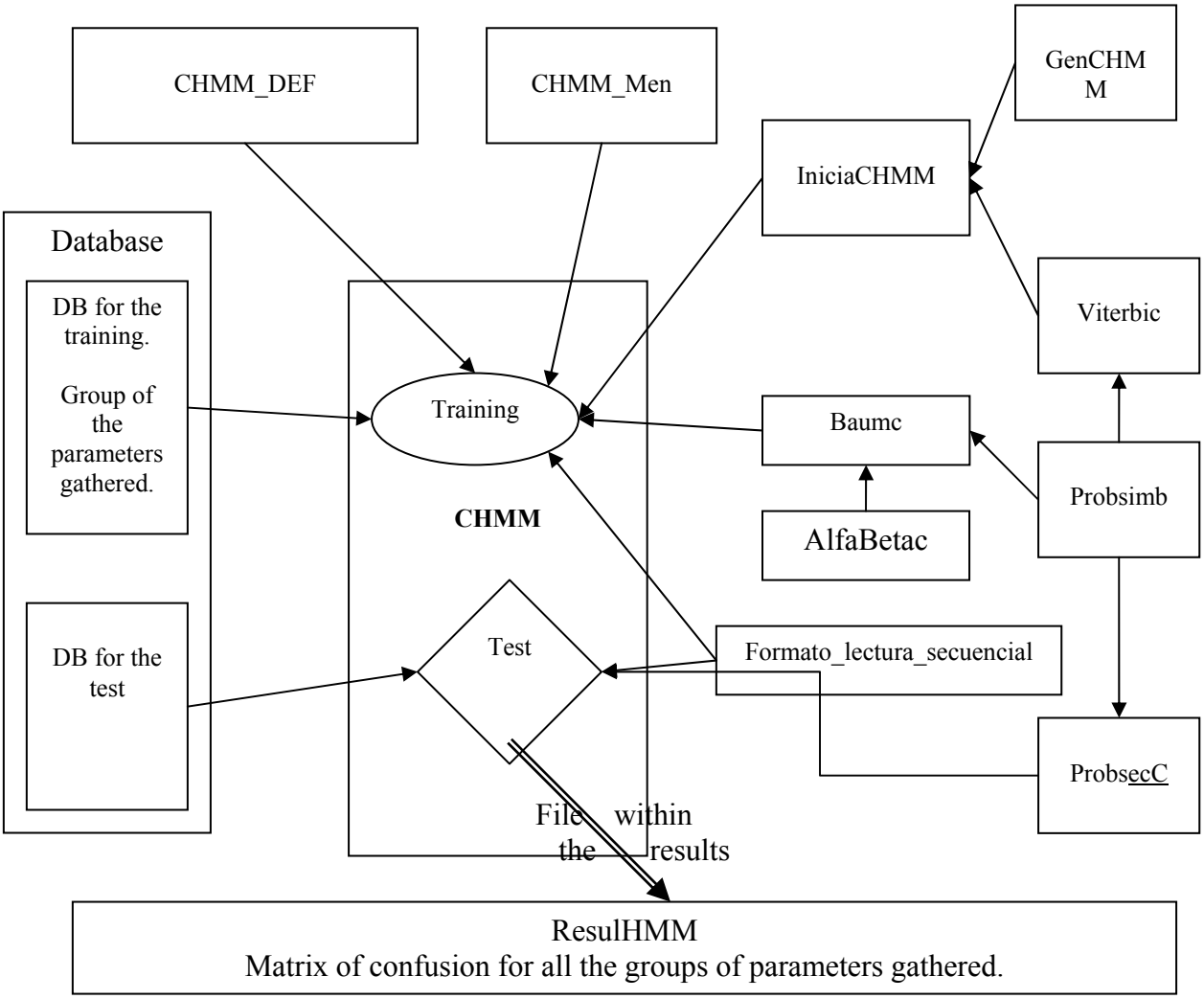


Illustration 4:1 Block-diagram for a CHMM

In the functions described in this toolbox, some are proposed as examples. In particular, the functions CHMM and CHMM_DEF must be adapted for yours own HMM. See the examples provided to have further information about it. In the second part of this point, each function is described with the formats of the inputs and outputs parameters.

AlfaBetac: this function is equivalent to the AlfaBeta function used for the DHMM.
Baumc: it is equal to the function Baum but used for a CHMM instead of a DHMM.
CHMM_DEF: this function defines the parameters for the CHMM and in particular the mixture of Gaussians used in this case.
CHMM: this is a script to train and test the HMM.
CHMM_MEN: it is equal than the DHMM_MEN.
DHMM2CHMM: this function generates a CHMM from a DHMM.
GencHMM: it is equivalent to the genHMM but used for a CHMM.
IniciacHMM: it is equivalent to the iniciaHMM but used for a CHMM.
Probsimb: this function calculates the probability for a vector ***O*** to be generated from any state S_i.
Viterbic: function to calculate the most probable state sequence given a CHMM and a sequence monitored ***O***.

- ✓ AlfaBetac

This function calculates the alpha and beta from the HMM defined with the matrix A, B and Pi. The alfa and beta are scaled to avoid the precision problem.

```
function [alfa,beta,c]=alfabetac(A,B,Med,Var,Pi,vectores,agrup)
```

Entry: A(N, N) is the transition probability matrix from a state I to a state k.

B{Np}{N}(Ngauss{ip},1) is the weight for the Gaussians of the HMM.

Med{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)): Matrix with the values of the medians for the Gaussians of the HMM.

Var{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)): Matrix with the values of the variances for the Gaussians of the HMM.

Pi(N,1) is the distribution probability for the initial state i.
Pi(i) is the probability to start in the state i.

Vectores: is a matrix with the sequence monitored used to estimate all the parameters.

Result: alfa is the forward probability matrix.
alfa(i,t) when alfa is not scaled is the probability to observe the sequence O(1,:),….,O(t,:) for the state i at the instant t for all the parameters.
We have in particular P(O/(A, B, Pi))= sum(alfa(:,T)).
When alfa is scaled, we have sum(alfa(:,t))=1.

beta is the backward probability matrix.
beta(i,t) when beta is not scaled is the probability to observe the sequence O(t+1,:),….,O(T,:) for the state i at the instant t for all the parameters.

c(T,1) is the vector where we store the scale value for the instant t.
We have c(T,1)= ones(T,1) if the alfa and beta are not scaled.

We have the following relations on the alfa and beta:

Relations between alfa scaled and not scaled

```
h=cumprod(c);
alfa_scaled(:,t)= h(t)*alfa_no_scaled;
```

Relations between beta scaled and not scaled

```
g=cumprod(c(T:-1:1);
beta_scaled(:,t)=g(t)*beta_no_scaled(:,t);
```

If alfa and beta are not scaled, the probability to observe O is (this product is independent of t):

```
alfa( :,t)'*beta( :,t) = sum (alfa(T,:))
```

If alfa and beta are scaled (it depends on t):

```
alfa(t,:)'*beta(t,:) = c(t)*sum (alfa(T,:))
```

- ✓ Baumc

This function computes the Baum Welch algorithm in order to estimate the HMM parameters.

function [A1,B1,Med1,Var1,Pi1,logPOs]=baumc(A,B,Med,Var,Pi,nftrain,lrep,agrup)

A, B and Pi are the matrices of the HMM defined with the function genHMM for example.

Entry: A(N, N) is the transition probability matrix from a state I to a state k.

B{Np}{N}(Ngauss{ip},1) is the weight for the Gaussians of the HMM.

Med{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)): Matrix with the values of the medians for the Gaussians of the HMM.

Var{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)): Matrix with the values of the variances for the Gaussians of the HMM.

Pi(i) is the probability to start in the state i.

Lrep is a matrix with the vectors number and symbols number.

A1(N, N) is the transition probability matrix updated with the Baum Welch algorithm.
B1{Np}{N}(Ngauss{ip},1) is the distribution probability symbol matrix updated with the Baum Welch algorithm.
Med{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)): Matrix with the values of the medians for the Gaussians of the HMM updated.
Var{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)): Matrix with the values of the variances for the Gaussians of the HMM updated.
Pi1(N,1) is the distribution probability for the initial state updated.
logPOs(nr,1): probability that the HMM with the parameters (A,B,Pi) generates each realisation of the labels entries of the subroutine.

This function calls the probsimb function.

- ✓ CHMM

This program calls the different functions to design the HMMs (one for each class and for each group) designed in the function CHMM_DEF. . It is here as example to make the HMM work.

function chmm(fhmm,fptrain,fptest,fhmmout)

Entry: fhmm is the name of the HMMs seted up in the function CHMM_DEF

fptrain is the name of the file containing the sequences of parameters to train each HMM with its own group of training set. In each repetition, we find the a sequence of parameters for a class and a group.

fptest is the name of the file containing the sequences of parameters to test each HMM with its own group of test set. In each repetition, we find a sequence of parameters for a class and a group.

fsalhmm: Name of the file containing the outputs of each classifier for each sample of the database of the test in a cell array salhmm{class, group}(repetition).

Variables defined in the fhmm file with the function CHMM_DEF:

nc: number of classes.
ng: number of groups.
agrup{ng}: the way to gather the parameters together.
Np(ng): number of parameters by group.
Ne(nc,ng): Number of states of the HMM for each class and group. We could fix a number of states different for each group ng and each class. For example, Ne(1,1) =10 and Ne(2,2)=20…In this case we have 10 states for the HMM of the first class and first group and 20 for the HMM of the second class and the second group.
Ngauss is the number of Gaussians in the mixture of Gaussians used to represent the distribution of the observation by state. We can change the number of Gaussians for each HMM (for each parameter of each group). Indeed, we have for example, Ngauss{ig}=6.*ones(Np(ig),1); and thus, we could fix Ngauss{1}=6 [1 2] and thus we have 6 Gaussians for the first parameter of the first group and 12 for the second parameter of the first group. We could change those values for the second group.

Maxitermi is the maximum iteration for the initial model
umbral: Umbral is the threshold condition to stop the HMM (the error is calculated with the maximum likelihood criterion)
maxiter: Maxiter is the maximum iteration number in the Bakis HMM (condition to stop the algorithm)
salto: Salto is the maximum path authorized in the Bakis HMM from a state to another.
BAKIS: if Bakis = 1 implements a Bakis HMM (also called left-right), else an ergodic HMM is defined

The variables of the HMM:

cell array A=cell(nc,ng);
cell array B=cell(nc,ng);
cell array Pi=cell(nc,ng);
cell array Med=cell(nc,ng);
cell array Var=cell(nc,ng);

Salhmm is a matrix used to store the probabilities values

NOTA: use the scripts chmm_def, and chmm_men.
NOTA: use the functions: iniciacHMM, genchmm, alfabetac, probsecC, viterbic, baumc
NOTA: if fptrain=" " we only realize the test
NOTA: if fptest=" " we only make the training

- ✓ CHMM_DEF

This function is described only as example to set up the HMM.

function chmm_def(fhmm)

This function defines the parameters of the discrete HMM. We can split the parameters in 4 groups:

Entry: fhmm is the HMM´s file name.

- ✓ vDB defines the database:
  vDB=[' nc ng agrup Np'];
  where nc is the number of class, ng is the number of group,
  agrup defines how to gather the parameters together. For example, if we have two parameters for the first group. If we fix agrup{1}=[1 3]; we create only one library for this group of parameter. If we fix agrup{1}=[1 2 3]; we create two libraries (one for the first parameter and one for the second parameter of the group). And so on, with three parameters, for agrup{1}=[1 3 4]; we create a library for the first and second parameters and one for the fourth parameter.
  NP is a vector with the number parameter of each group.

We have the following relations:

```
[nc,ng]=size(vl);
agrup{ig}=[1 .... size(vl{1,ig}{1},2)+1];
Np(ig)=length(agrup{ig})-1;
```

- ✓ vHMM defines the HMM :

vHMM =[' BAKIS salto maxiter umbral maxitermi Ngauss Ne A B Med Var Pi'];

if Bakis = 1 implements a Bakis HMM (also called left-right), else an ergodic HMM is defined
Salto is the maximum path authorized in the Bakis HMM from a state to another.
Maxiter is the maximum iteration number in the Bakis HMM (condition to stop the algorithm)
Umbral is the threshold condition to stop the HMM (the error is calculated with the maximum likelihood criterion)
Maxitermi is the iteration number to find the initial model.
Ne(nc,ng): Number of states of the HMM for each class and group. We could fix a number of states different for each group ng and each class. For example, Ne(1,1) =10 and Ne(2,2)=20…In this case we have 10 states for the HMM of the first class and first group and 20 for the HMM of the second class and second group.
Ngauss is the number of Gaussians in the mixture of Gaussians used to represent the distribution of the observation by state. For instance, Ngauss{ig}=6.*ones(Np(ig),1); Or, we could change it and fix a number different for each parameter of each group For instance: Ngauss{ig}=[1 ; 3];
A, B and Pi are the matrices of the HMM (it will be calculated thanks to the CHMM) but must de defined here.
Med and Var are the matrices of the mean and variances of the Gaussians.

cell array A=cell(nc,ng);
cell array B=cell(nc,ng);
cell array Pi=cell(nc,ng);
cell array Med=cell(nc,ng);
cell array Var=cell(nc,ng);

In the CHMM, we have in particular the cell B, Med and Var with the relation:

```
B{ic,ig}{ip}{ie}=zeros(Ngauss{ig}(ip),1);
Med{ic,ig}{ip}{ie}=zeros(Ngauss{ig}(ip),agrup{ig}(ip+1)-agrup{ig}(ip));
Var{ic,ig}{ip}{ie}=zeros(Ngauss{ig}(ip),agrup{ig}(ip+1)-agrup{ig}(ip));
```

And B{ic,ig}{ip}{ie}is the coefficient of the mixture of Gaussian for the HMM of the class ic, group ig. Calculated for the parameter ip of the group ig and in the state ie. We have the same relation for the variance of the Gaussians of the mixture of Gaussian defined for the HMM of the class ic, group ig, for the parameter ip of the group ig and the state ie (defined in Var). And we have the same relation for the mean of the Gaussians of the mixture of Gaussian (defined in Med).

- ✓ vTEST is the matrix parameter for the test :
  vTEST=[' salhmm'];
  Salhmm is a matrix used to store the probabilities values

We have the following relations:
salhmm=cell(nc,ng);

- ✓ CHMM_MEN

This script is a small program that prints some messages to describe the computation of the HMM.

- ✓ dhmm2chmm

This function is used to migrate from a Discrete HMM to a Continuous HMM. The idea is to use the DHMM trained to calculate the initials parameters of the CHMM. This function uses the Netlab utility.

function dhmm2chmm(filedhmm,filechmm)

filedhmm is the file with the parameters of the discrete HMM.
filechmm is a file containing the parameters of the continuous HMM wanted.

This function eliminates the variable Ns (number of symbols) and initializes the variables Med Var maxitermi and Ngauss (see the CHMM_DEF function for further information).
We generate 1000 samples to simulate each Gaussian with the variance and mean calculated.
We then save the continuous HMM variables in the file “filechmm”.

- ✓ GENCHMM

This function generates a CHMM with N states and Ngauss Gaussians.
function [A,B,Med,Var,Pi]=genchmm(N,vcombl,vmean,vstd,agrup,Ngauss,Np,BAKIS,salto);

Entry: N is the state number
vcombl is a parameter of the mixture of Gaussians. See the function gmminit of the netlab software.
vmean is the vector of the initial means
vstd is the vector of the initial diversion
agrup(Np+1,1) defines how to gather the parameters together
Ngauss(Np,1) number of Gaussians
Np number of parameters
if Bakis = 1 implements a Bakis HMM (also called left-right),else an ergodic HMM is defined
Salto is the maximum path authorized in the Bakis HMM from a state to another.

Result: A, B, Med, Var and Pi are the matrices of the HMM defined within this function.

A(N, N) is the transition probability matrix with the following conditions:
$\Sigma_j A_{ij} = 1$ (sum(A(i,:)=1))
$A_{ij} \geq 0$ (A(i,j) ≥ 0)
For a Bakis HMM, we have aij=0 if j<i (A(i,j)=0 for j<i)

B{Np}{N}(Ngauss{ip},1) is a structure with the weight of each Gaussian for each state and to obtain each parameter.

Med: Med{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)) is the mean for each Gaussian in each state and for each gathering of parameters

Var: Var{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)) is the variance for each Gaussian in each state and for each gathering of parameters

Pi(N,1) is the distribution probability for the initial state i.
Pi(i) is the probability to start in the state i.
Σ Pi(i) =1
For a Bakis HMM Pi(i)= δ(i)
With δ(i) is the Kronecker function equal to 1 for i=0 and null for i<>0

We define here the matrix and update it within the training process of the HMM like the Baum function. This function uses see the netlab software utility. Please refer to this utility for further information.

- ✓ iniciacHMM

This function calculates the CHMM optimal initial model for the Baum-Welch training.

function [Aa,Ba,Meda,Vara,Pia]=
iniciachmm(Ne,Np,BAKIS,salto,nftrain,lrep,agrup,Ngauss,maxitermi)

Entry: Ne state number
Np number parameters by symbol
if Bakis = 1 implements a Bakis HMM (also called left-right), else an ergodic HMM is defined
Salto is the maximum path authorized in the Bakis HMM from a state to another.
Nftrain: is the name of the file with the training data set
Lrep is a Matrix with the size from the training set data for every classes, groups and parameters (see also the function formato_lectura_secuencial)
agrup defines how to gather the parameters together (see the CHMM_DEF).
Ngauss is the number of Gaussians used in the mixture of Gaussians.
Maxitermi is the maximum iteration to calculate the initial HMM.

Result: Aa, Ba, Meda, Vara and Pia are the matrices for the parameters of the HMM. Meda and Vara are the mean and the variance for each Gaussian.

In this function, we will generate a number of symbols in every state bigger than twice Ns. Thus, we make sure to initialize correctly the HMM.

We use the viterbic and the genchmm functions. And the netlab utility (gmm gmminit and gmmem).

- ✓ ProbsecC

This function estimates the log of the probability to monitor a sequence O given a HMM. We use the log to avoid numerical problem.

```
function [logPO,logalfaT]=probsecc(A,B,Med,Var,Pi,O,agrup)
```

Entry: A(N, N) is the transition probability matrix from a state I to a state k.

B{Np}{N}(Ngauss{ip},1) is a structure with the weight of each Gaussian for each state and to obtain each parameter.

Med: Med{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)) is the mean for each Gaussian in each state and for each gathering of parameters

Var: Var{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)) is the variance for each Gaussian in each state and for each gathering of parameters

Pi(N,1) is the distribution probability for the initial state i.
Pi(i) is the probability to start in the state i.

O is a matrix with the sequence to estimate.

agrup(Np+1,1) defines how to gather the parameters together

Result: logPO is the log of the probability to monitor the sequence O.
logalfaT: is the log alfa(:, T) where alfa is the forward probability matrix and T is the running time for the sequence O.

We use in this function the alfabetac and probsimb function.

- ✓ Probsimb

This function caluclates the probability to have the vector of the observation O generated by each state of the HMM. We suppose dfp Gaussian.

```
function [PS,Psk]=probsimb(B,Med,Var,O,agrup)
```

Entry: B{Np}{N}(Ngauss{ip},1) is a structure with the weight of each Gaussian for each state and to obtain each parameter.

Med: Med{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)) is the mean for each Gaussian in each state and for each gathering of parameters

Var: Var{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)) is the variance for each Gaussian in each state and for each gathering of parameters

O is a matrix with the sequence to estimate.

agrup(Np+1,1) defines how to gather the parameters together

Results: PS(N,1): Probability of the vector of the observations O in the state ie.

Psk{Np}{N}(Ngauss{ip},1): Normalizad probability of the vector of the observations O in the state ieusing the symbol k.

Normalizing the data, we have: sum(Psk(i,M*d+1:M*d+M))=1 for d=0,1,...,Dim-1

- ✓ verfdp

This function plots the pdf and the distribution for each state. We stand that the fdp is Gaussian.

```
function [Ptotal,Ototal]=verfdp1(B,Med,Var,agrup)
```

Entry: B{Np}{N}(Ngauss{ip},1) is a structure with the weight of each Gaussian for each state and to obtain each parameter.

Med: Med{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)) is the mean for each Gaussian in each state and for each gathering of parameters

Var: Var{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)) is the variance for each Gaussian in each state and for each gathering of parameters

agrup(Np+1,1) defines how to gather the parameters together

- ✓ Viterbic

This function calculates the sequence of the most probable states given the HMM and the sequence monitored O. We use the algorithm of Viterbi with the log for the numerical precision problems.

```
function qP=viterbic(A,B,Med,Var,Pi,O,agrup)
```

Entry: A(N, N) is the transition probability matrix from a state I to a state k.

B{Np}{N}(Ngauss{ip},1) is a structure with the weight of each Gaussian for each state and to obtain each parameter.

Med: Med{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)) is the mean for each Gaussian in each state and for each gathering of parameters

Var: Var{Np}{N}(Ngauss(ip),agrup(ip+1)-agrup(ip)) is the variance for each Gaussian in each state and for each gathering of parameters

Pi(N,1) is the distribution probability for the initial state i.
Pi(i) is the probability to start in the state i.

O is a matrix with the sequence to estimate.

agrup(Np+1,1) defines how to gather the parameters together

Result: qP(T,1) is the sequence of the most probable states.

5. Examples

5.1. Polygons: the DHMM example

The polygons is an example to use the HMM toolbox. In this part, we can see how the DHMM is initialized and used. First, we generate four different classes of polygons in the function “prueba”. We extract the parameters (i.e. in this case the polar coordinate and the diff of the polar coordinate). Then, we split the database in the function Train and call DHMM_DEF to initialize the HMM with the set up wanted. Please see the block diagram to understand how this example works:

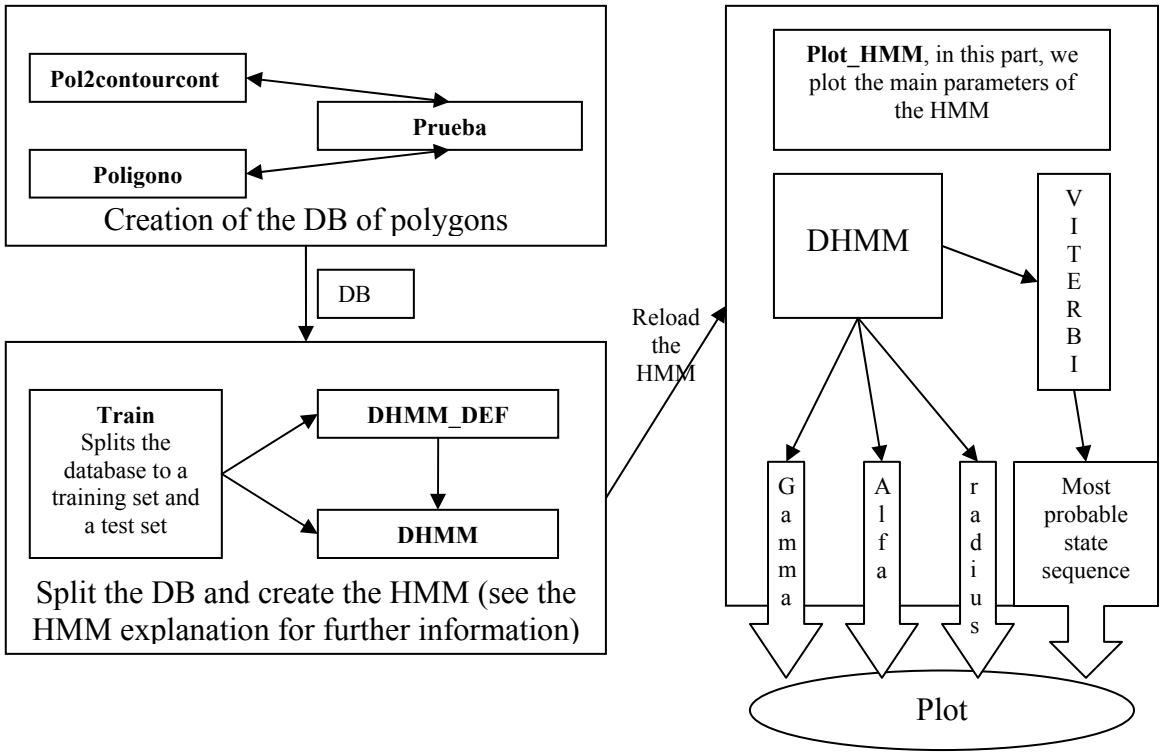


Illustration 5:1 block diagram of the polygons: the DHMM example.

Moreover, this example plots the matrices Gamma, Alfa, radius and the most probable state sequence calculated for each set of the training and for all the different HMMs (one for each class and each group of parameters). Thanks to this example, we can see how the DHMM is defined for a concrete application, what is the format used to store the input parameters for the training and for the test.

- ✓ Prueba

This function first creates 4 classes of different polygons. Then, thanks to the function cart2pol, it parameterizes the contour of the polygons in polar coordinate. The angle and the radius can be chosen as parameters inputs of the DHMM. In our case, we added the diff of the radius and thus made two groups of parameters. The DHMM inputs are stored in a cell. Then, this cell will be saved.

The database is created and each parameter is stored in a structure array cell. We store the parameters as follow:
vlcp{number of the class, number of the group}{number of the repetition}

Once we have calculated the radius and angle for the polygon, we can plot it thanks to the function "Pol2contourcont". "Pol2contourcont" creates an image of the polygons. Putting a 1 in the "if" of the code source of the function, we can show the image created for each class and repetition (see illustration 5:2 and 5:3). We create 200 samples, 100 for the training and 100 for the test. If you want to change it, you can change the number of repetition "nr".

The cell "vlcp" is then saved to be reloaded in the train function.
The function train is called at the end of this function.

- ✓ Pol2contourcont

This function creates the polygons as an image. If you want to plot it, put a 1 in the "if" of the function "prueba" (see the code source of the function prueba). Thus, you obtain the images (see illustration 5:2 and 5:3).

- ✓ Train

Once, we have calculated the inputs, we define the DHMM thanks to the functions Train and DHMM_DEF. The train function first loads the parameters and then splits the data base into the training set and the test set, and calls the DHMM_DEF function and then the DHMM function. The set for the trainings and for the test are taken at random for each class. The training set is saved in a cell format vtrain and the test in vtest. In this function, we also define the percentage for the training and for the test (percentage for the training is ptrain=50, and for the test, 100-ptrain). You can change it. Moreover, we print in the file prueba.txt the messages printed on the screen during the function train.

At the end, we call the function resulhmm to create the matrices of confusion for each group.

- ✓ Plot_HMM

This function plots different parameters characteristic of the DHMM. As you can see in the illustrations 5:4 to 5:7, this script plots the alfa, gamma radius and the most probable state sequence for each DHMM, repetition, class and groups of parameters.

✓ DHMM_DEF

This function defines the DHMM for our example. In the general case, please refer to the 2nd part of this document. It is important to define the right number of classes and groups otherwise, the function fails and an error message is printed.

First, we define vDB the VARIABLE of the database:

```
vDB=[' nc ng agrup Np'];
```

Number of classes nc and number of groups ng.

```
nc=4;
```
here we have 4 classes

```
ng=2;
```
and two groups

How to gather the parameters? We have two groups of parameters, the angle and the radius, and, the diff of the radius and the angle.

```
agrup=cell(ng,1);
Np=zeros(ng,1);
agrup{1}=[1 2 3];
```
We gather (angle and the radius) the first parameters together but create a library for the radius and a library for the angle. If we fix agrup{1}=[1 3]; we create only one library for this group of parameter.

```
agrup{2}=[1 2 3];
```
We gather (angle and diff(radius)) the other parameters together.

```
Ne=10.*ones(nc,ng);
```
10 states are been chosen, if we want to change it: Ne=20.*ones(nc,ng)

```
TOPN{ig}=1.*ones(Np(ig),1);
```
if we want to put two labels TOPN{ig}=2.*ones(Np(ig),1), we have to change topntest when we change the number of labels too. We can also set a number of labels different for each set of parameter and group. For instance: TOPN{4}=4.*ones(Np(4),1); and TOPN{ig}=1.*ones(Np(ig),1); for ig>1.By experience, the results are quite good fixing only one label by parameter.

We chose a Bakis HMM with a maximum path of 1, maximum iteration of the Baum Welch is 30, threshold to stop the HMM is 0.005, Maxitermi is the iteration number to find the initial model 10.

```
BAKIS=1;
salto=1;
maxiter=30;
umbral=0.005;
maxitermi=10;
```

And LBG is the choice of the algorithm to compile the library.
LBG = 1 the algorithm is LBG, in the other case we choose the k-mean (see “kMedias” function).
dptzoLBG percentage maximum for the distance of the code vector in the algorithm LBG (see gen_bib).
maxiterVQ is the maximum number of iteration to compute the library (condition to stop the algorithm).

umbralVQ is the threshold condition to stop the library computation (condition to stop the algorithm).
men=1 or 0 if men =1 the library generates message else no.
biblio is a matrix parameter for the library computation (we define it here but calculate in the gen_bib function).

Here we have chosen:

```
LBG=1;
dpztoLBG=0.1;
maxiterVQ=40;
umbralVQ=0.001;
men=1;
```

vTEST is the matrix parameter for the test : where TOPNtest is the number of labels to take into account for the multi-labeling during the test phase, Salhmm is a matrix used to store the probabilities values

`TOPNtest{ig}=1.*ones(Np(ig),1);` To take into account two labels: TOPNtest{ig}=2.*ones(Np(ig),1). We can also set a number of labels different for each set of parameters and group. For instance: `TOPNtest{1}=4.*ones(Np(1),1)` and `TOPNtest{ig}=2.*ones(Np(ig),1)` for ig>1.By experience, the results are quite good fixing only one label by parameter.

Analysis of the example

Then, the train function calls the DHMM function from the toolbox and calculate the different models of the HMM (one for each class and group of parameters) according to the set up made in the DHMM_DEF. Thanks to this example, we can see the way to set up a discrete HMM and to use it in order to classify four different classes of polygons with two different groups of parameters. We use a Bakis DHMM and the LBG algorithm associated to the k-mean to calculate the libraries for each set of parameters. We use only one label but changing the set up in the DHMM_DEF, we could put more labels and a number different for each parameter of each group. The results obtained are analyzed thanks to the matrix of confusion.

Thus, the results obtained are of the range of:

```
RATE OF RECOGNITION BY GROUP 1: 95
RATE OF RECOGNITION BY GROUP 2: 99
RATE OF RECOGNITION BY GROUP 1  2: 98
```

And the matrices of confusion:
Matrix_of_confusion_first_group = Mc{1}{1}

```
100    0    0    0
  0   64    0   36
  4    0   96    0
  0    0    0  100
```

Matrix_of_confusion_second_group = Mc{1}{2}

```
100    0    0    0
```

```
  0   99    0    1
  5    0   95    0
  0    0    0  100
```

Matrix_of_confusion_union = Mc{2}{1}

```
100    0    0    0
  0   90    0   10
  2    0   98    0
  0    0    0  100
```

As we can see in the first matrix, the second class is confused with the fourth one (for 36 samples) and the third with the first one (for 4 samples). We can see the confusions for the second group and the union of the outputs in the matrices Mc{1}{2} and Mc{2}{1}.

Thanks to the illustrations 5-2 and 5-3, we can see the 4 classes of polygons and the variation of the radius with the angle for each class. The illustrations 5-3 to 5-7 show how the different discrete HMM interprets the variation of the radius. In particular, we can see in the two graphs of the most probable state sequence, the differences between the HMM of the first class and of the fourth class. This example gives us a good clue to understand the process to classify a sample thanks to the technology of the Hidden Markov Model.

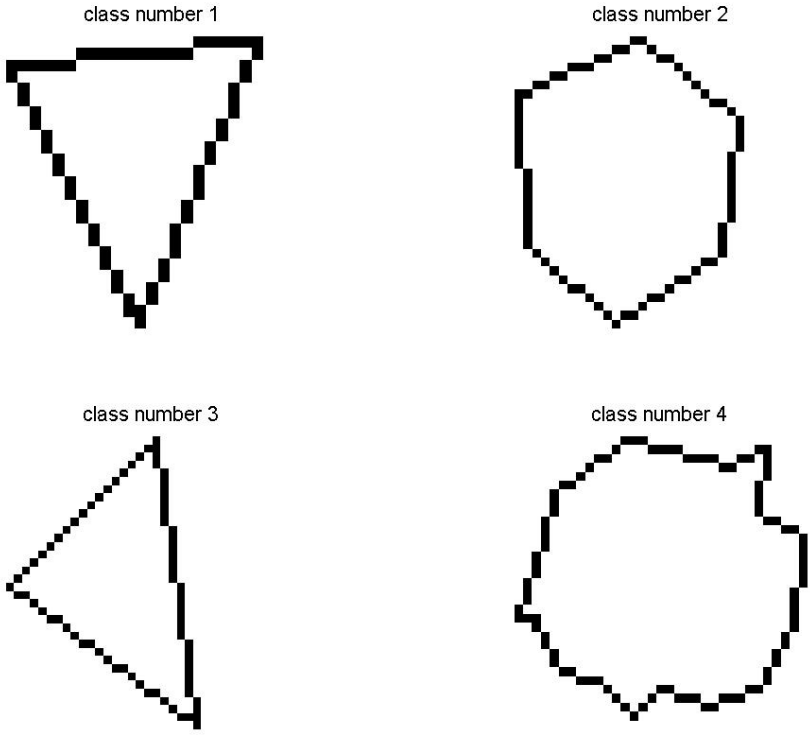


Illustration 5:2 Samples of the 4 classes of the example.

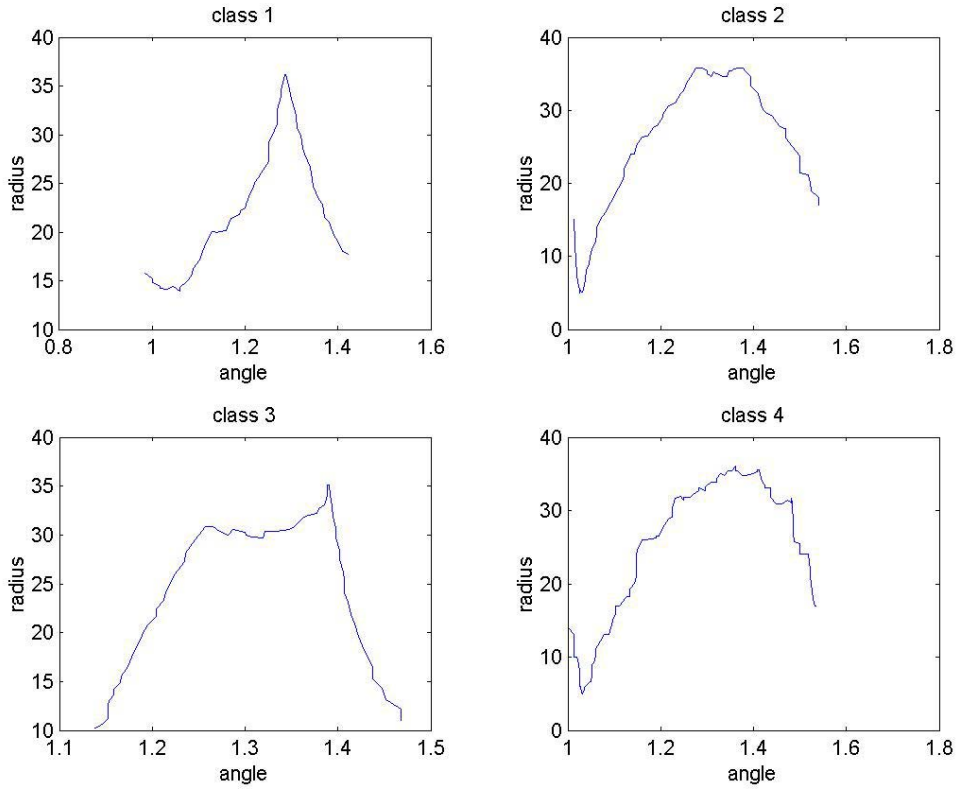


Illustration 5:3 The radius and angle for the 4 polygons of the illustration 5:1

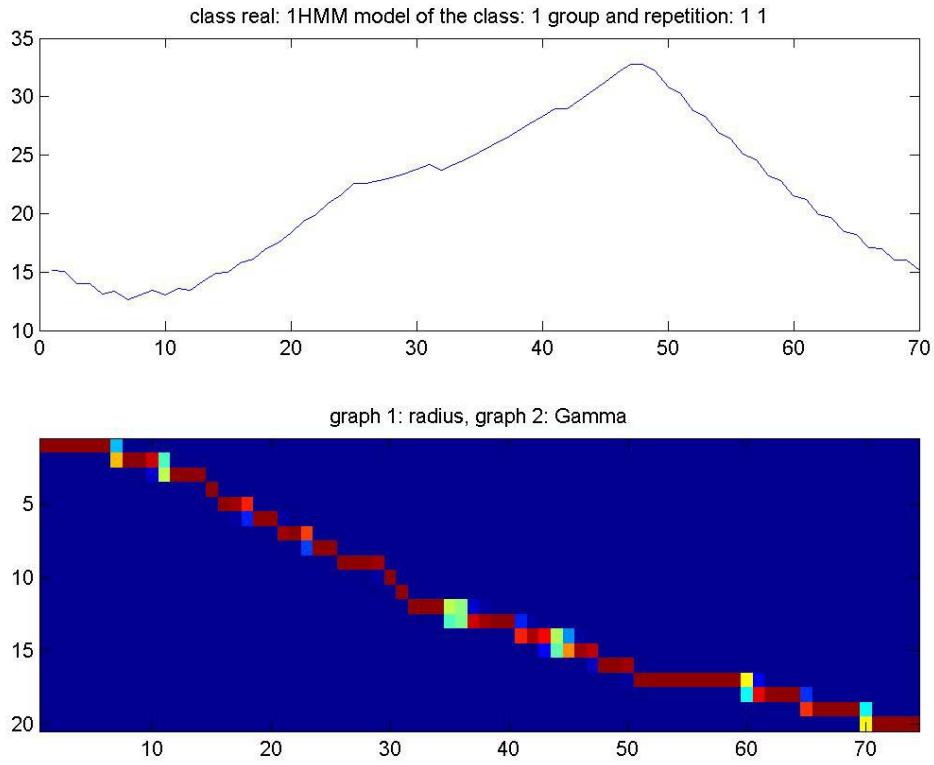


Illustration 5:4 For the first repetition of the first class, the radius is plotted in the graph 1 and gamma is calculated for the group 1 of the first DHMM.

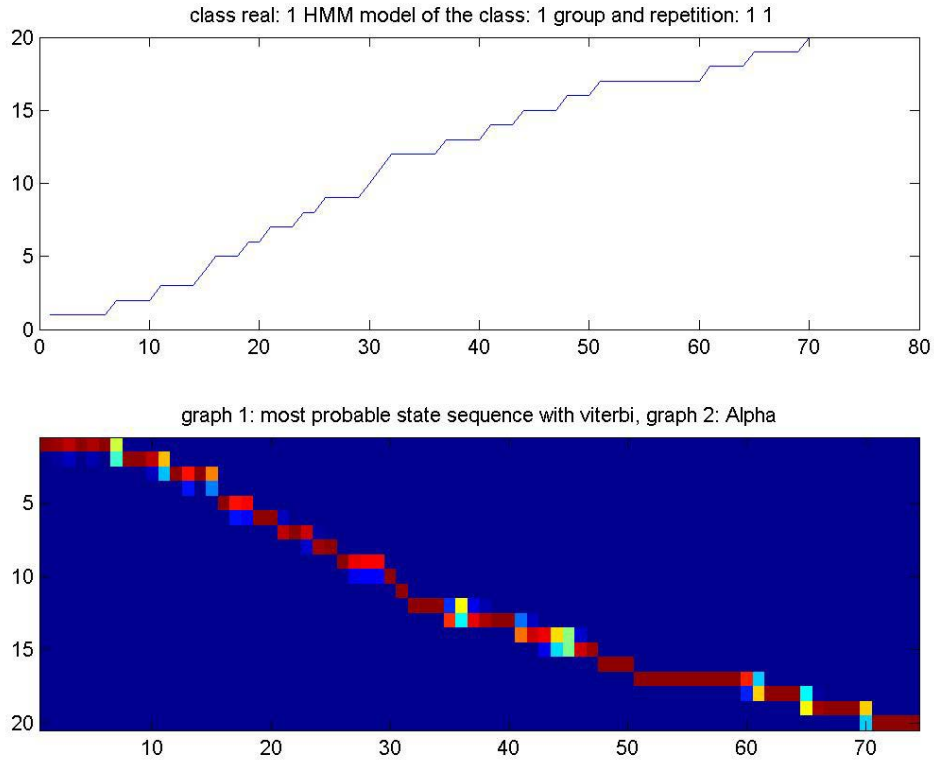


Illustration 5:5 For the first repetition of the first class, the most probable state sequence is plotted in the graph 1 and alpha is calculated for the group 1 of the first DHMM.

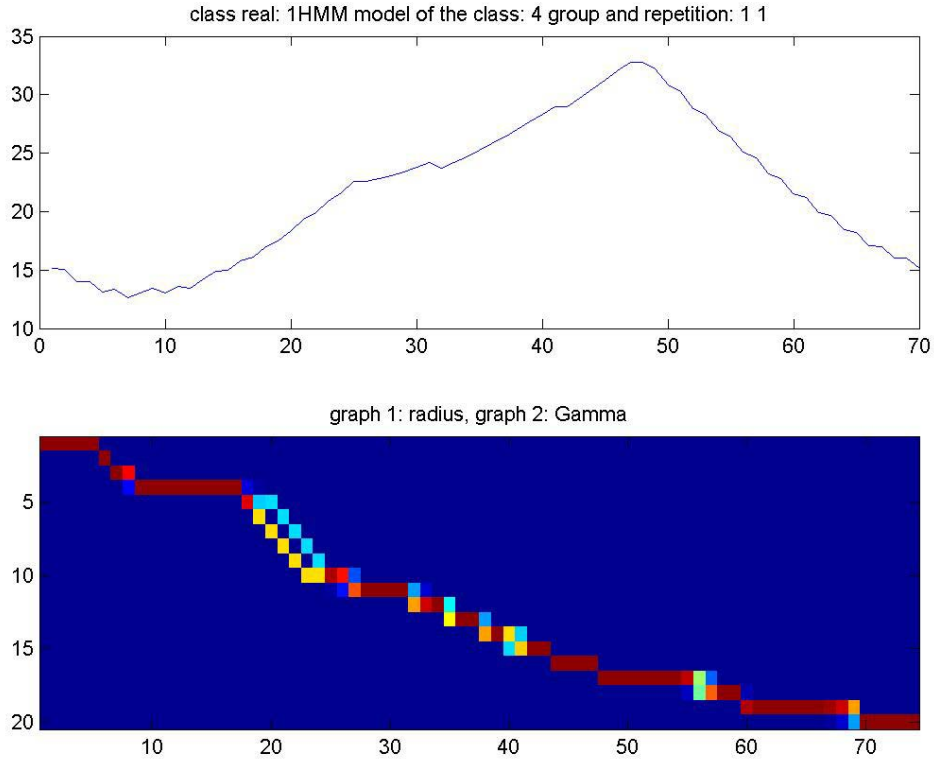


Illustration 5:6 For the first repetition of the first class, the radius is plotted in the graph 1 and gamma is calculated for the group 1 of the fourth DHMM.

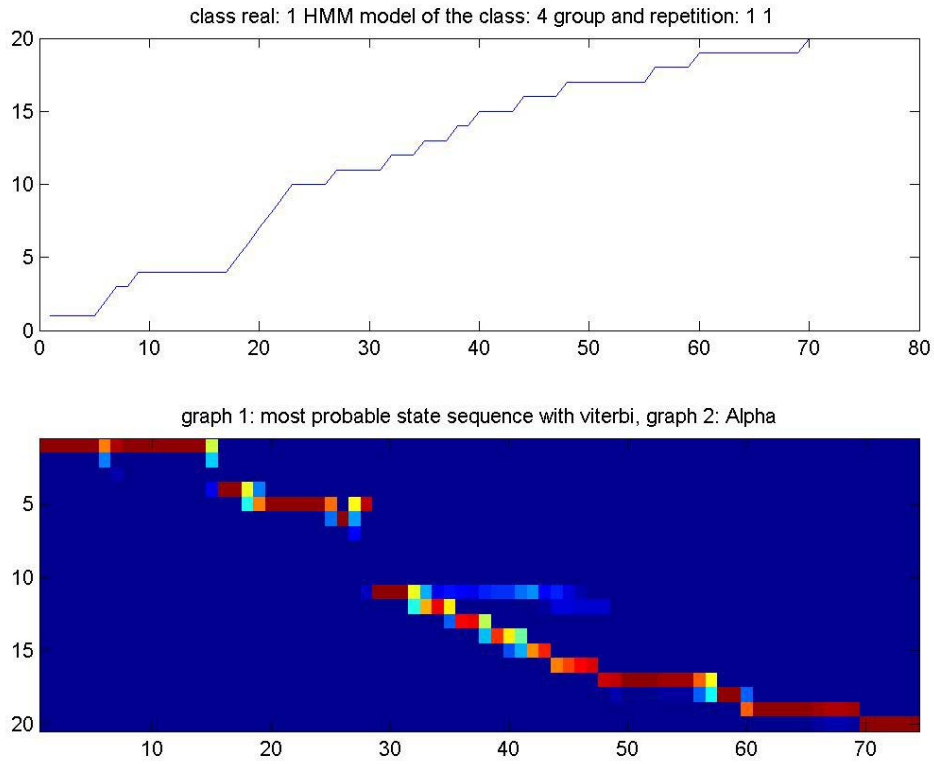


Illustration 5:7 For the first repetition of the first class, the most probable state sequence is plotted in the graph 1 and alpha is calculated for the group 1 of the fourth DHMM.

5.2. Polygons: the CHMM example

The polygons is an example to use the HMM toolbox. In this part, we can see how the CHMM is initialized and used. First, we generate four different classes of polygons in the function “prueba”. We extract the parameters (i.e. in this case the polar coordinate and the diff of the polar coordinate). Then, we split the database in the function “trainCHMM” and call CHMM_DEF to initialize the HMM with the parameters wanted. Please see the block diagram to understand how this example works:

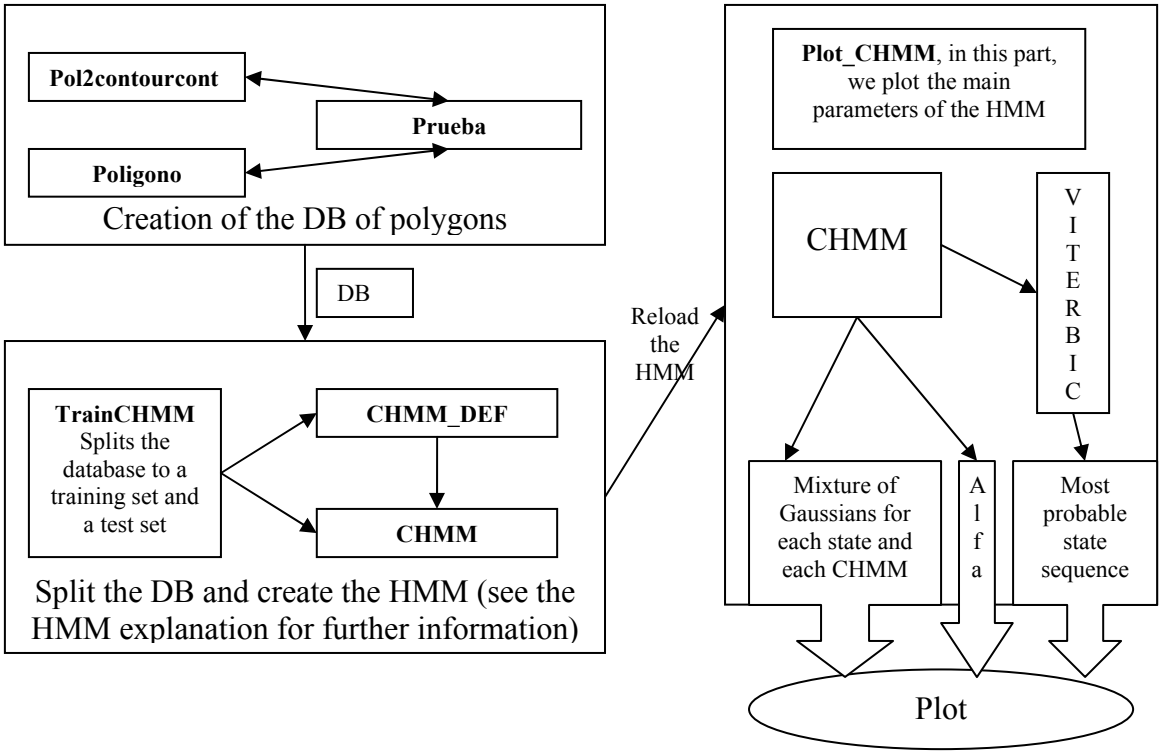


Ilustración 5:8 Block diagram of the polygons: the CHMM example.

Moreover, this example plots the matrices Alfa, the most probable state sequence calculated for each set of the training and for all the different HMMs. The mixture of Gaussians is also drawn for each state of each HMM. Thanks to this example, we can see how the CHMM is defined for a concrete application, what format we use to store the input parameters for the training and for the test. As for the discrete HMM example, the inputs for the HMM are saved in a cell array format.

- ✓ Prueba

First, this function creates 4 classes of different polygons. Then, thanks to the function “cart2pol”, it parameterizes the contour of the polygons in polar coordinate. The angle and the radius can be chosen as parameters inputs of the CHMM. In our case, we added the diff of the radius and thus made two groups of parameters. The CHMM inputs are stored in a cell. Then, this cell will be saved.

The database is created, and each parameter is stored in a structure array cell. We store the parameters as follow:
vlcp{number of the class, number of the group}{number of the repetition}

Once we have calculated the radius and angle for the polygon, we can plot them thanks to the function “Pol2contourcont”. “Pol2contourcont” creates an image of the polygon. Putting a 1 in the “if” of the code source of the function, we can show the image created for each class and repetition (see illustration 5:2 and 5:3). We create 200 samples, 100 for the training and 100 for the test. If you want to change it, you can change the number of repetition “nr”.

The cell “vlcp” is then saved to be reloaded in the “trainCHMM” function. The function “trainCHMM” is called at the end of this function.

- ✓ Pol2contourcont

This function creates the polygons as an image. If you want to plot it, you put a 1 in the “if” of the function “prueba” (see the code source of the function “prueba”). Thus, you obtain the images (see illustration 5:2 and 5:3).

- ✓ trainCHMM

Once, we have calculated the inputs, we define the CHMM thanks to the functions “trainCHMM” and CHMM_DEF. The “trainCHMM” function first loads the parameters and then splits the data base into the training set and the test set, and calls the CHMM_DEF function and then the CHMM function. The set of samples for the trainings and for the tests are taken at random for each class. The training set is saved in a cell format vtrain and the test in vtest. In this function, we also define the percentage for the training and for the test (percentage for the training is ptrain=50, and for the test, 100-ptrain). You can change it. Moreover, we print in the file prueba.txt the messages printed on the screen during the function “trainCHMM”.

At the end, we call the function “resulhmm” to create the matrices of confusion for each group.

- ✓ Plot_CHMM

This function plots different parameters characteristic of the DHMM. As you can see in the illustrations 5:9 to 5:10, this script plots the alfa, the most

probable state sequence for each CHMM, repetition, class and groups of parameters. It also plots the mixture of Gaussians for each state of each HMM.

- ✓ CHMM_DEF

This function defines the CHMM for our example. In the general case, please refer to the 2nd part of this document. It is important to define the right number of classes and groups otherwise, the function fails and an error message is printed.

First, we define vDB the VARIABLE of the database:

```
vDB=[' nc ng agrup Np'];
```

Number of classes nc and number of groups ng.

```
nc=4; here we have 4 classes
ng=2; and two groups
```

How to gather the parameters? We have two groups of parameters, the angle and the radius, and, the diff of the radius and the angle.

```
agrup=cell(ng,1);
Np=zeros(ng,1);
agrup{1}=[1  3]; We gather (angle and the radius) the first parameters
                 together.
agrup{2}=[1  2   3]; We gather (angle and diff(radius)) the other
                 parameters together. In this case, we have different
                 values for the mixture of Gaussians for the angle and
                 the diff(radius).
Ne=12.*ones(nc,ng); 12 states are been chosen, if we want to change it:
                 Ne=20.*ones(nc,ng). We can also fix a different
                 number of states for each HMM. For instance, we
                 could fix different number of states for each class and
                 group's HMM by Ne= [3 4; 5 6; 7 8; 9 10];
Ngauss{ig}=6.*ones(Np(ig),1); Ngauss is the number of Gaussians in the
                 mixture of Gaussians used to represent the
                 distribution of the observation by state. Here, we have
                 put 6 for each parameter of each group. Or, we could
                 change it and fix a number different for each
                 parameter of each group. For instance: Ngauss{1}=[1 ;
                 3];
```

We chose a Bakis HMM with a maximum path of 1, maximum iteration of the Baum Welch is 30, threshold to stop the HMM is 0.005, Maxitermi is the iteration number to find the initial model 10.

```
BAKIS=1;
salto=1;
maxiter=30;
umbral=0.005;
maxitermi=10;
```

vTEST is the matrix parameter for the test : Salhmm is a matrix used to store the probabilities values output for each repetition.

Matrices of the HMM

```
A=cell(nc,ng);
B=cell(nc,ng);
Med=cell(nc,ng);
Var=cell(nc,ng);
Pi=cell(nc,ng);
```

In the CHMM, we have in particular the cell B, Med and Var with the relations:

```
B{ic,ig}{ip}{ie}=zeros(Ngauss{ig}(ip),1);
Med{ic,ig}{ip}{ie}=zeros(Ngauss{ig}(ip),agrup{ig}(ip+1)-agrup{ig}(ip));
Var{ic,ig}{ip}{ie}=zeros(Ngauss{ig}(ip),agrup{ig}(ip+1)-agrup{ig}(ip));
```

And B{ic,ig}{ip}{ie}is the coefficient of the mixture of Gaussian for the HMM of the class ic, group ig. Calculated for the parameter ip of the group ig and in the state ie. We have the same relation for the variance of the Gaussians of the mixture of Gaussian defined for the HMM of the class ic, group ig, for the parameter ip (of this group) and the state ie. And we have the same relation for the mean of the Gaussians of the mixture of Gaussian (defined in Med).

Analysis of the example

Then, the “trainCHMM” function calls the CHMM function from the toolbox and calculate the different models of the HMM (one for each class and group of parameters) according to the set up made in the CHMM_DEF. Thanks to this example, we can see the way to set up a continuous HMM and to use it in order to classify four different classes of polygons with two different groups of parameters. We use a Bakis CHMM. The results obtained are analyzed thanks to the matrices of confusion.

Thus, the results obtained are of the range of:

```
RATE OF RECOGNITION BY GROUP 1: 91.25
RATE OF RECOGNITION BY GROUP 2: 95.5
RATE OF RECOGNITION BY GROUP 1  2: 99
```

And the matrices of confusion:

Matrix_of_confusion_first_group = Mc{1}{1}

```
 96     0     4     0
  0    75    11    14
  0     0   100     0
  0     6     0    94
```

Matrix_of_confusion_second_group = Mc{1}{2}

```
100     0     0     0
 12    84     0     4
  2     0    98     0
  0     0     0   100
```

Matrix_of_confusion_union = Mc{2}{1}

100	0	0	0
1	96	0	3
0	0	100	0
0	0	0	100

As we can see in the first matrix, the second class is confused with the third (for 11 samples) and with the fourth (for 14 samples). The first is confused with the third (4 samples), and the fourth with the second (6 samples). We can see the confusions for the second group and the union of the outputs in the matrices Mc{1}{2} and Mc{2}{1}.

Thanks to the illustrations 5-2 and 5-3, we can see the 4 classes of polygons and the variation of the radius with the angle for each class. The illustrations 5-10 and 5-11 show how the different continuous HMMs interpret the variation of the radius. In particular, we can see in the two graphs of the most probable state sequence the differences between the HMM of the first class and of the fourth class. This example gives us a good clue to understand the process to classify a sample thanks to the technology of the continuous Hidden Markov Model. If we compare it with the discrete HMM, we see that the information is much more diffused in the case of the CHMM. With the illustration 5-9, we can see the mixture of Gaussians every states of each CHMM.

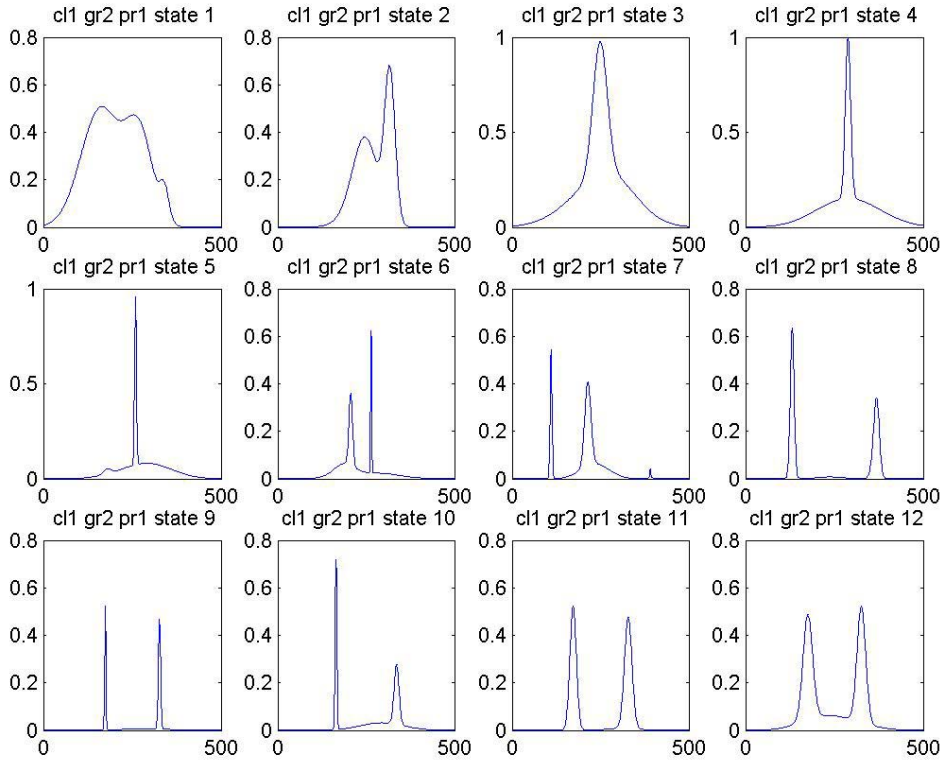


Illustration 5:9 Mixture of Gaussians from the CHMM of the class 1(cl1), group of parameter 2 (gr2) is plotted for the parameter 1 of this group (pr1) and for every states (state 1 to 12) of this HMM.

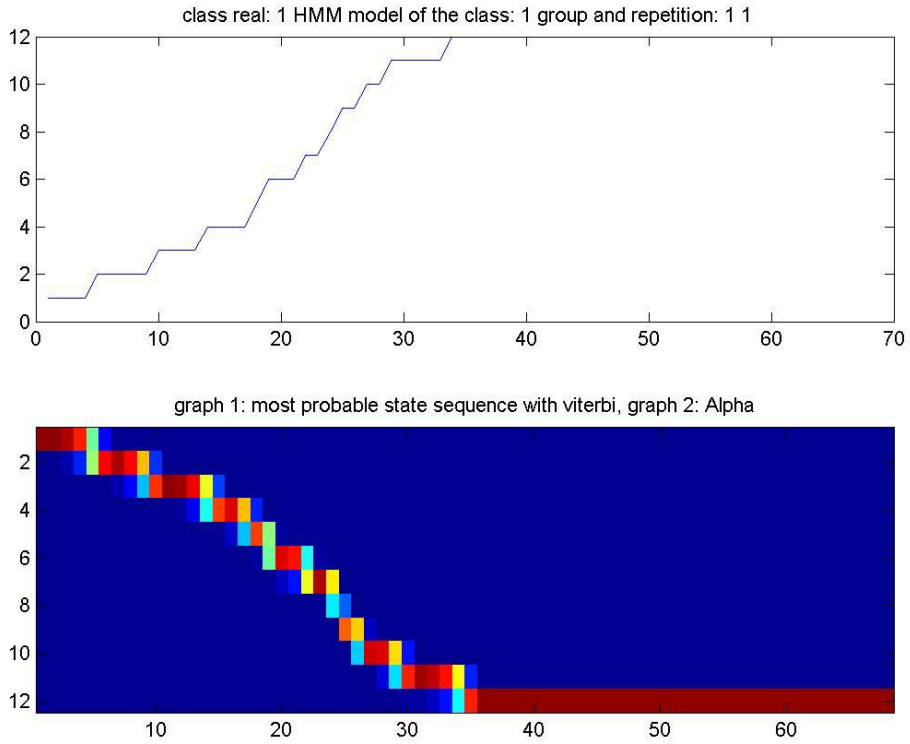


Illustration 5:10 For the first repetition of the first class, the most probable state sequence is plotted in the graph 1 and Alpha is calculated for the group 1 of the first CHMM.

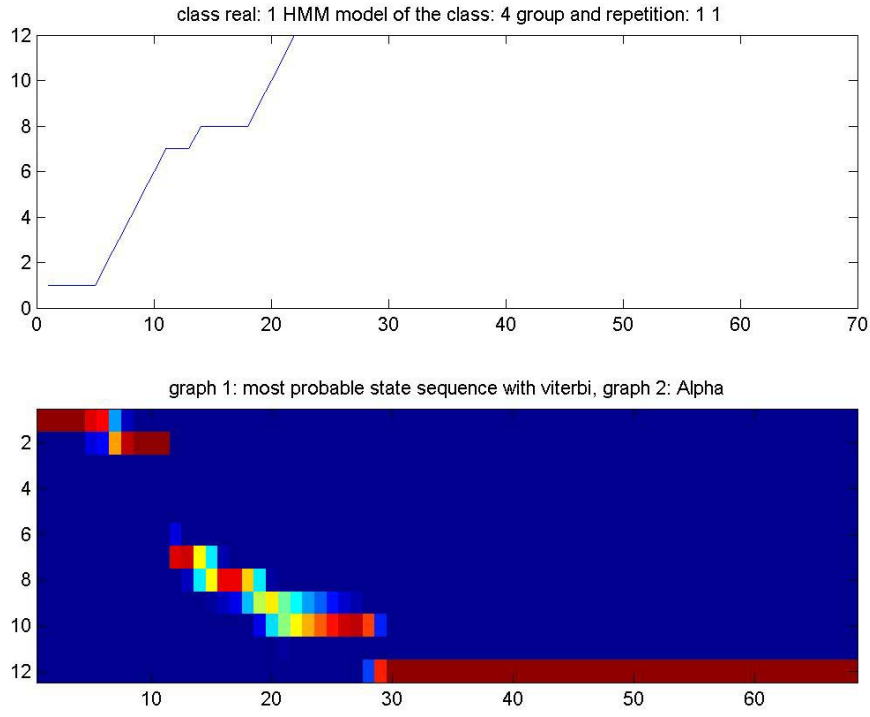


Illustration 5:11 For the first repetition of the first class, the most probable state sequence is plotted in the graph 1 and Alpha is calculated for the group 1 of the fourth CHMM.

5.3. Dhmm2chmm example

In this part, we can see how the CHMM is initialized thanks to the function “dhmm2chmm”. First, we generate four different classes of polygons in the function “prueba”. We extract the parameters (i.e. in this case the polar coordinate and the diff of the polar coordinate). Then, we split the database in the function “train” and call DHMM_DEF to initialize the DHMM with the parameters wanted. Then we optimize the DHMM with the function ”DHMM” as in the example “polygons: the DHMM example”. Please see the block diagram to understand how this example works:

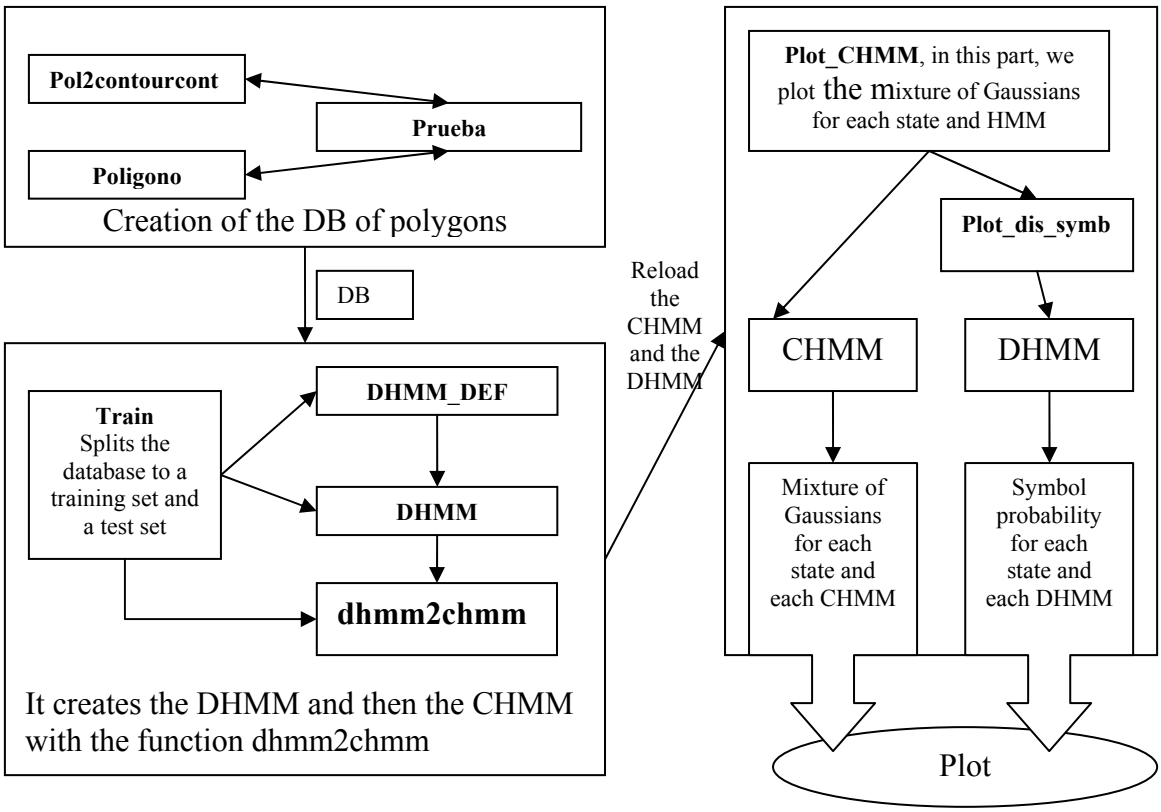


Illustration 5:12 Block diagram of the “dhmm2chmm” example.

Moreover, this example plots different HMMs. The mixture of Gaussians is drawn for each state of each HMM and parameter. We compare the mixture of Gaussians with the probability distribution symbols for the DHMM. Thus, we can see how the CHMM is initialized thanks to the DHMM. Thanks to this example, we can see how the CHMM can be defined for a concrete application, what format we use to store the input parameters for the training and for the test. This example shows in particular the advantages of the “dhmm2chmm” function to initialize the CHMM.

- ✓ Prueba

First, this function creates 4 classes of different polygons. Then, thanks to the function “cart2pol”, it parameterizes the contour of the polygons in polar coordinate. The angle and the radius can be chosen as parameters inputs of the HMM. In our case, we added the diff of the radius and thus made two groups of parameters. The HMM inputs are stored in a cell. Then, this cell will be saved.

The database is created, and each parameter is stored in a structure array cell. We store the parameters as follow:
vlcp{number of the class, number of the group}{number of the repetition}

Once we have calculated the radius and angle for the polygon, we can plot them thanks to the function “Pol2contourcont”. “Pol2contourcont” creates an image of the polygon. Putting a 1 in the “if” of the code source of the function, we can show the image created for each class and repetition (see illustration 5:2 and 5:3). We create 200 samples, 100 for the training and 100 for the test. If you want to change it, you can change the number of repetition “nr”.

The cell “vlcp” is then saved to be reloaded in the “train” function.
The function “train” is called at the end of this function.

- ✓ Pol2contourcont

This function creates the polygons as an image. If you want to plot it, you put a 1 in the “if” of the function “prueba” (see the code source of the function “prueba”). Thus, you obtain the images (see illustration 5:2 and 5:3).

- ✓ train

Once, we have calculated the inputs, we define the DHMM thanks to the functions “train” and DHMM_DEF. The “train” function first loads the parameters and then splits the data base into the training set and the test set, and calls the DHMM_DEF function and then the DHMM function. The set of samples for the trainings and for the tests are taken at random for each class. The training set is saved in a cell format vtrain and the test in vtest. In this function, we also define the percentage for the training and for the test (percentage for the training is ptrain=50, and for the test, 100-ptrain). You can change it. Moreover, we print in the file prueba.txt the messages printed on the screen during the function “train”. Once we have calculated the DHMM, we use the function “dhmm2chmm” to generate the CHMM initialized with the parameters of the DHMM.

At the end, we call the function “resulhmm” to create the matrices of confusion for each group.

- ✓ Plot_CHMM

As you can see in the illustrations 5:13 to 5:14, this script plots mixture of Gaussians for each CHMM, repetition, class and groups of parameters. It also plots the distribution probability symbols for each DHMM. This function calls the function “plot_dis_symb” in order to plot the distribution probability symbols for each DHMM.

- ✓ plot_dis_symb

This function only plots the probability distribution symbols for each parameter and states given a DHMM (a group, a class).

Analysis of the example

The set up of the CHMM is made thanks to the functions “DHMM_DEF” (where we define the DHMM) and then thanks to the function “dhmm2chmm”. Please refer to the source codes of those two functions to see the set up made of the CHMM. In this example, we can see thanks to the illustrations 5:13 and 5:14 how the mixture of Gaussians is set up with the probability distribution of the symbols.

The results obtained with the DHMM are:
RATE OF RECOGNITION BY GROUP 1: 98.25
RATE OF RECOGNITION BY GROUP 2: 84.5
RATE OF RECOGNITION BY GROUP 1 2: 97.5

The results obtained with the CHMM (without the training) are:
RATE OF RECOGNITION BY GROUP 1: 90.5
RATE OF RECOGNITION BY GROUP 2: 72.75
RATE OF RECOGNITION BY GROUP 1 2: 86

We obtain those results thanks to the function CHMM without the training. To make that, we call the function:
chmm('hmmpoligonos2.mat','','vtest');

As for the others examples, we can plot the matrices of confusion. Mc is the matrix for the DHMM. Mc2 is the matrix for the CHMM.

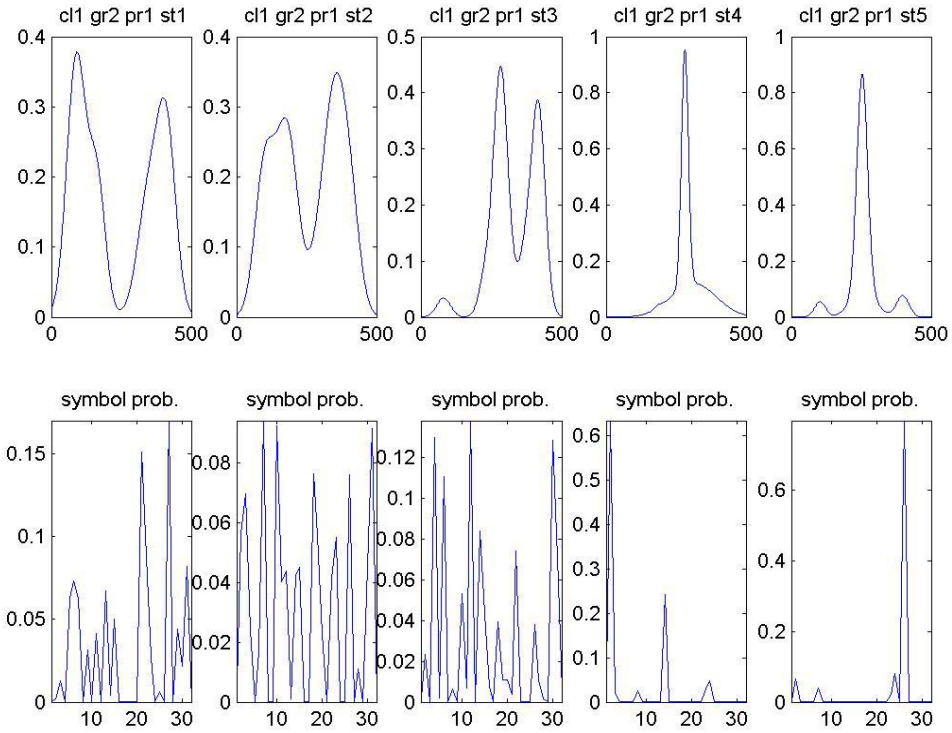


Illustration 5:13 Above is the mixture of Gaussians for the CHMM obtained thanks to the "dhmm2chmm" function for the class 1, group 2, parameter 1 and 5 first states. Below is the probability of symbols for the DHMM for the class 1, group 2, parameter 1 and first five states.

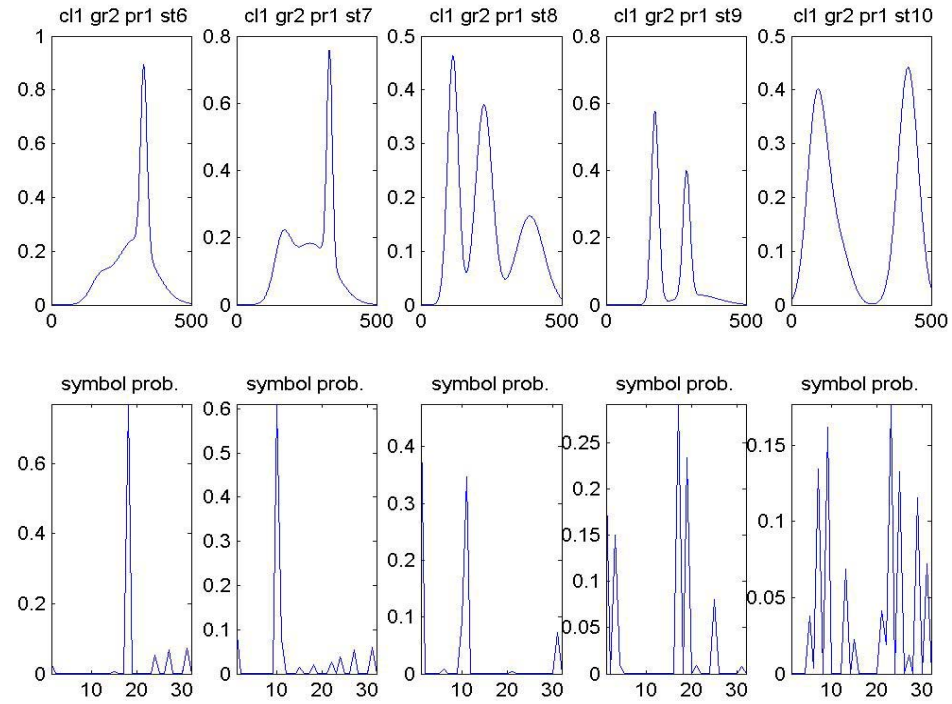


Illustration 5:14 Above is the mixture of Gaussians for the CHMM obtained thanks to the "dhmm2chmm" function for the class 1, group 2, parameter 1 and 5 last states. Below is the probability of symbols for the DHMM for the class 1, group 2, parameter 1 and last five states.

6. References

[1] J. A. Sánchez, C. M. Travieso, I. G. Alonso, M. A. Ferrer, *Handwritten recognizer by its envelope and strokes layout using HMM's*, 35rd *Annual 2001 IEEE Internacional Carnahan Conference on Security Technology, (IEEE ICCST'01)*, London, UK, 2001, 267-271.

[2] M. A. Ferrer, J. L. Camino, C. M. Travieso, C. Morales, *Signature Classification by Hidden Markov Model*, *33rd Anual 1999 IEEE Internacional Carnahan Conference on Security Technology, (IEEE ICCST'99), Comisaría General de Policía Científica, Ministerio del Interior, IEEE Spain Section, COIT, SSR-UPM, Seguritas Seguridad España S.A*, Madrid, Spain, Oct. 1999, 481-484.

[3] J. B. Alonso, C.Carmona, J. de León y M. A. Ferrer, Combining Neural Networks and Hidden Markov Models for Automatic Detection of Pathologies, *16_th Biennial International Eurasip Conference Biosignal 2002*, Brno, Check Republic, June 2002.

[4] Renals, S., Morgan, N., Bourlard, H., Cohen, M. & Franco, H. (1994), Connectionist probability estimators in HMM speech recognition, *IEEE Transactions on Speech and Audio Processing 2(1)*, 1994, 161-174.

[5] L.R. Bahl, P.F. Brown, P.V. de Souza, and R.L. Mercer, Maximum mutual information estimation of HMM parameters for speech recognition,. *In Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing,* , Tokyo, Japan, December 1986, 49-52

[6] Yin, M.M., Wang, J.T.L., Application of hidden Markov models to gene prediction in DNA, *Information Intelligence and Systems, 1999.*] *Proceedings. International Conference on*, 1999, 40 – 47.

[7] Cohen, A., Hidden Markov models in biomedical signal processing, *Engineering in Medicine and Biology Society, 1998. Proceedings of the 20th Annual International Conf. of the IEEE, Vol. 3*, 1998, 1145 – 1150.

[8] L. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical* Statistics*, 41(1),* 1970, 164-171.

[9] L. Baum, An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, *3*, 1972, 1-8.

[10] Al-Ani, T.; Hamam, Y., An integrated environment for hidden Markov models, a Scilab toolbox, *Computer-Aided Control System Design, 1996., Proceedings of the 1996 IEEE International Symposium on*, Sept. 1996, 446 – 451.

[11] J.Hernando, C.Nadeu, José B. Mariño, Speech recognition in a noisy environment based on LP of the one-sided autocorrelation sequence and robust similarity measuring techniques, *Speech communications*, *vol. 21*, 1997, 17-31.

[12]L. R. Rabiner. Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition *Readings in Speech Recognition, chapter A*, 1989, 267-295.

[13] M.A. Ferrer, I. Alonso, C. Travieso, Influence of initialization and Stop Criteria on HMM based recognizers, Electronics letters of IEE, *Vol. 36*, June 2000, 1165-1166.

[14] Notes of the lectures of M.A. Ferrer : http://www.gpds.ulpgc.es/ (see Docencia).

[15] A HMM toolbox for Matlab, Kevin Murphy,

http://www.ai.mit.edu/~murphyk/Software/HMM/hmm.html

[16] The HTK toolbox from the Cambridge University: http://htk.eng.cam.ac.uk/

[17] Netlab softaware: http://www.ncrg.aston.ac.uk/netlab

7. Installation

- The first step is to download the HMM toolbox from:
  http://www.gpds.ulpgc.es/download/index.htm

- Then, extract the HMM toolbox in its own PC. The directories should be created as:

..\hmm_toolbox\toolbox
..\hmm_toolbox\examples
..\hmm_toolbox\examples\dhmm
..\hmm_toolbox\examples\chmm
..\hmm_toolbox\examples\dhmm2chmm

- Then download the netlab utility from
  http://www.ncrg.aston.ac.uk/netlab/
  and install the files in ..\hmm_toolbox\toolbox\netlab.

- Create the directory:
  c:\temphmm

- Add the paths corresponding to the mentioned above directories.