h2m : A set of MATLAB functions for the EM estimation of hidden Markov models with Gaussian state-conditional distributions

Olivier Cappé
ENST dpt. Signal / CNRS-URA 820,
46 rue Barrault, 75634 Paris cedex 13, France.
`cappe@sig.enst.fr`

July 16, 1997

Contents

1 About `h2m`

`h2m`is a set of functions that implement the EM algorithm [1], [2] in the case of mixture models or hidden Markov models with (multivariate) Gaussian state-conditional distribution. More specifically, three special cases have been considered

1. Gaussian mixture models.
2. Ergodic (or fully connected) hidden Markov models.
3. Left-right hidden Markov models.

In fact, the case 2 and 3 above do not significantly differ except for the fact that in the case of a left-right HMM, one needs to estimate the parameters from multiple observation sequences. In all three cases, it is possible to use either diagonal or full covariance matrices for the state-conditional distributions.

2 Changes and bug fixes

There is a list of the changes that have been made since the first version of this manual (Mar. 18, 1997) in the file `h2m/doc/CHANGES`. The current version number which should be found in all the mfiles is 1.3.

From a practical point of view, the main changes have been: *(i)* There was an omission in the scaling procedure in the unnumbered version (which could cause overflows when the input data was too small). *(ii)* In versions 1.1 - 1.2, this was corrected but a brand new bug was introduced in the same part of the code (function `hmm_fb`), which prevented a correct re-estimation of the transition matrix.

Minor changes include: the renaming of `multgen` to `randindx`, and some new functions (`gauseval`, `gauslogv` and `hmm_psim` - see list in section 7).

3 Data structures

No specific data structures have been used, so that a HMM consists of:

pi0 Row vector containing the probability distribution for the first (unobserved) state: $\pi_0(i) = P(s_1 = i)$.

A Transition matrix: $a_{ij} = P(s_t + 1 = j | s_t = i)$.

mu Mean vectors (of the state-conditional distributions) stacked as row vectors, such that `mu(i,:)` is the mean (row) vector corresponding to the i-th state of the HMM.

Sigma Covariance matrices. These are stored one above the other in two different way depending on whether full or diagonal covariance matrices are used: for full covariance matrices,

```
Sigma((1+(i-1)*p):(i*p),:)
```

(where p is the dimension of the observation vectors) is the covariance matrix corresponding to the i-th state; for diagonal covariance matrices, `Sigma(i,:)` contains the diagonal of the covariance matrix for the i-th state (ie. the diagonal coefficients stored as row vectors).

For a left-right HMM, `pi0` is assumed to be deterministic (ie. `pi_0 = [1 0 ... 0]`) and `A` can be made sparse in order to save memory space (`A` should be upper triangular for a left-right model).

A Gaussian mixture model, is rather similar except that as the underlying jump process being i.i.d., `pi0` and `A` are replaced by a single row vector containing the mixture weights `w` ($w(i) = P(s_t = i)$).

Most functions (those that have `mu` and `Sigma` among their input arguments) are able to determine the dimensions of the model (size of observation vectors and number of states) and the type of covariance matrices (full or diagonal) from the size of their input arguments. This is achieved by the two functions `hmm_chk` and `mix_chk`.

For more specialized variables such as those that are used during the forward-backward recursions, I have tried to use the notations of L. R. Rabiner in [3] (or [4]) which seem pretty standard:

alpha Forward variables: $\alpha_t(i) = P(X_1, \ldots, X_t, S_t = i)$.

beta Backward variables: $\beta_t(i) = P(X_{t+1}, \ldots, X_T | S_t = i)$.

gamma A posteriori distributions of the states:

$$\gamma_t(i) = P(S_t = i | X_1, \ldots, X_T)$$

I have also tried to use systematically the convention of multivariate data analysis that the matrices should have "more rows than columns", so that the observation vectors are stacked in `X` as row vectors (the number of observed vectors being usually greater than their dimension). The same is true for `alpha`, `beta` and `gamma` which are $T \times N$ matrices (where T is the number of observation vectors and N the number of states).

4 Examples

The file `doc/examples.m` contains some MATLAB code corresponding to low dimensional examples of the three basic HMM types that are handled by `h2m`. These are:

4.1 Ergodic model with full covariance matrices

Let `X` denote a matrix containing `T` observed vectors, the EM estimation of the parameters take the following form:

```
for i = 1:n_iter
  [alpha, beta, logscale, dens] = hmm_fb(X, A, pi0, mu, Sigma);
  logl(i) = log(sum(alpha(T,:))) + logscale;
  [A, pi0] = hmm_tran(alpha, beta, dens, A, pi0);
  [mu, Sigma] = hmm_dens(X, alpha, beta, COV_TYPE);
end;
```

`COV_TYPE` is just a flag that should be set to 0 when using full covariance matrices and to 1 for diagonal covariance matrices.

Notice that at each step, the log-likelihood is computed from the forward variables using a correction term `logscale` returned by `hmm_fb` (for forward-backward) which contains the sum of the logarithmic scaling factors used during the computation of `alpha` and `beta`. In the present version scaling of the forward variable is performed at each time index $2 \leq t \leq T$ (which means that each row of the `alpha` matrix sums to one, except the first one). This systematic scaling appears to be much safer when using input data with largely varying range. The backward variable is scaled using the same normalization factors as indicated in [3] (using exactly the same normalization factors is important for the re-estimation of the coefficients of the transition matrix). Note that the mere suppression of the scaling procedure would lead to numerical problems in almost every cases of interest (when the length of the observation sequences if greater than 50 for instance) despite the double precision representation used by MATLAB.

If you don't want to see what's going on you can simply use

```
[A, pi0, mu, Sigma, logl] = hmm(X, A, pi0, mu, Sigma, n_iter);
```

which calls the very same piece of code except for the fact that the messages concerning the execution time are suppressed: all the computational functions print those messages by default but this can be suppressed by supplying and optional argument (named `QUIET`) different from zero.

4.2 Left-right HMM

This case is not really different from the last one except that the case of multiple observation sequences has been considered:

```
for i = 1:n_iter
  [A, logl(i), gamma] = hmm_mest(X, st, A, mu, Sigma);
  [mu, Sigma] = mix_par(X, gamma, COV_TYPE);
end;
```

In this case, the matrix X contains all the observation sequences and the vector st yields the index corresponding to the beginning of each sequence so that X(1:st(2)-1,:) contains the vectors that correspond to the first observation sequence, and so on until X(st(length(st)),length(X(1,:),:) which corresponds to the last observation sequence.

The transition parameters are re-estimated inside hmm_mest and the a posteriori distribution of the states are returned in gamma. Once again, if you don't want to see what's happening you could more simply use

```
[A, mu, Sigma, logl] = hmm(X, st, A, mu, Sigma, n_iter);
```

In fact, if you need to estimate the parameter of an ergodic model using multiple (independent) observation sequences, you may use hmm_mest as well, but hmm_mest assumes that pi_0 = [1 0 ... 0] (ie. that the Markov chain starts from the first state).

4.3 Mixture model

The EM estimation in the case of mixture model is achieved through

```
for i = 1:n_iter
  [gamma, logl(i)] = mix_post(X, w, mu, Sigma);
  [mu, Sigma, w] = mix_par(X, gamma, COV_TYPE);
end;
```

or, more simply

```
[w, mu, Sigma, logl] = mix(X, w, mu, Sigma, n_iter);
```

5 Implementation issues

5.1 Initialization

Initialization plays an important part in iterative algorithms such as the EM. Usually the choice of the initialization point will strongly depend on the application considered. I use only two very basic methods of initializing the parameters:

For left-right models hmm_mint initializes all the model parameters using a uniform "hard" segmentation of each data sequence (each sequence is splitted in N consecutive sections, where N is he number of states in the HMM, the vectors thus associated with each state are used to obtain initial parameters of the state-conditional distributions).

For mixture models svq implements a binary splitting vector quantization algorithm. This usually provides efficient initial estimates of the parameters of the Gaussian densities. Note that `svq` uses the unweighted Euclidean distance as performance criterion. If the components of the input vectors are strongly correlated and/or of very different magnitude, it would be preferable to use `svq` on the vectors $\mathbf{\Phi x}_t$ where $\mathbf{\Phi}$ is the Cholevski factor associated with $\mathrm{Cov}^{-1}(\mathbf{x})$ (ie. $\mathbf{\Phi}'\mathbf{\Phi} = \mathrm{Cov}^{-1}(\mathbf{x})$.

5.2 Modifications of the EM recursions

Many people introduce some modifications of the EM algorithm in order to avoid some known pitfalls. The fact that the likelihood becomes infinite for singular covariance matrices is maybe the problem most often encountered in practice. Solutions include thresholding the individual variances coefficients in the diagonal case or adding a constant matrix at each iteration. This is certainly useful, particularly in the case where there is "not enough" training data (compared to the complexity of the model). There is however a risk of modifying the properties of the EM algorithm with such modifications, in particular the likelihood may decrease at some iteration. No such modification has been used here.

5.3 Computation time and memory usage

Each function is implemented in a rather straightforward way and should be easy to read. In some case (such as `hmm_tran` for instance) however, the code may be less easy to read because of aggressive "matlabization" (vectorization) which helps save computing time. Note that one of the most time-consuming operation is the computation of Gaussian density values (for all input vectors and all states of the model). In the case I use more frequently (Gaussian densities with diagonal covariance), I have included a mex-file (`c_dgaus`) which is used in priority if it is found on MATLAB's search path (expect a gain of a factor 5 to 10 on `hmm_fb` and `mix_post`). Some routines could easily be made more efficient (`hmm_vit` for instance) if someone has some time to do so.

These routines are not particularly fast, especially if you are using full covariance matrices or if you can't compile the mex-file `c_dgaus`. I apologize if it looks like a plain old commercial, but: execution time get approximately divided by 10 on functions such as `hmm_fb` when using the MATLAB compiler `mcc`. In the four components 2-D mixture model (last example of file `doc/examples.m`) with 50 000 observation vectors, the execution time (on a SUN SPARC workstation) for each EM iteration is: 3 seconds when using diagonal covariance with the mex-file `c_dgaus`; 3 minutes when using full covariance; 10 seconds for full covariance matrices when the files `mix_post` and `mix_par` have been compiled using MATLAB's `mcc`.

Users of MATLAB compiler should compile in priority the file `gauseval` (computation of the Gaussian densities) which represents the main computational load in many of the routines. Compiling the high-level functions like `mix`, `hmm` (and `vq`) is not possible at the moment simply because I used some variable names ending with a trailing underscore (sorry for that!) It wouldn't be very useful anyway since only the compilation of the low-level functions significantly speeds up the computation. Note that functions compiled with `mcc` can't handle sparse matrices, which is a problem for left-right HMMs (for this reason, I don't recommend compiling a function like `hmm_fb`).

Memory space is also a factor to take into account: typically, using more than 50 000 training vectors of dimension 20 with HMMs of size 30 is likely to cause problems on most computers. Usually, the largest matrices are `alpha` and `beta` (forward and backward probabilities), `gamma` (a posteriori distribution for the states) and `dens` (values of Gaussian densities). The solution would consists in reading the training data from disk-files by blocks... but this is another story!

6 Other functions

The functions `hmm_gen` and `mix_gen` generate data vectors according to a given model. This is useful for testing algorithms on "prototype data". `hmm_psim` generates a random sequence of HMM state conditional to an observation sequence. This can be used for doing Monte-Carlo simulations (the way it works is described, for instance, in [5] as "sampling the indicator variables").

`gauseval` and `gauslogv` computes values of the Gaussian probability density (or the logarithm of it for `gauslogv`) for several Gaussian distributions and several observed vectors at the same time. Computing as many values as possible at the same time is much faster than calling the function several times (especially when the number of Gaussian distributions is large).

`gauselps` plots the 2-D projections of the Gaussian ellipsoids corresponding to the Gaussian distribution (this is certainly one of the most useful things in order to see what's going on, at least for low dimensional models).

7 List of functions

c_dgaus Computes a set of multivariate normal density values in the case of diagonal covariance matrices (mexfile)

gauselps Plots 2D projections of gaussian ellipsoids.

gauseval Computes a set of multivariate normal density values.

gauslogv Computes a set of multivariate normal log-density values.

hmm Performs multiple iterations of the EM algorithm.

hmm_chk Checks the parameters of an HMM and returns its dimensions.

hmm_dens Reestimates the gaussian parameters for an HMM.

hmm_fb Implements the forward-backward recursion (with scaling).

hmm_gen Generates a sequence of observation given an HMM.

hmm_mest Reestimates the transition parameters for multiple observation sequences.

hmm_mint Initializes the distribution parameters using multiple observations (left-right model).

hmm_psim Generates a random sequence of conditional HMM states.

hmm_tran Reestimates the transition part of an HMM.

hmm_vit Computes the most likely sequence of states (Viterbi DP algorithm).

lrhmm Performs multiple iterations of the EM algorithm for a left-right model.

mix Performs multiple iterations of the EM algorithm for a mixture model.

mix_chk Checks the parameters of a mixture model and return its dimensions.

mix_gen Generates a sequence of observation for a gaussian mixture model.

mix_par Reestimates mixture parameters.

mix_post Computes a posteriori probabilities for a gaussian mixture model.

randindx Generates random indexes with a specified probability distribution.

svq Vector quantization using successive binary splitting steps.

vq Vector quantization using the K-means (or LBG) algorithm.

8 Downloading `h2m`

`h2m` is available by anonymous FTP as a unix gz-compressed tape archive at address `ftp://sig.enst.fr/pub/cappe/mfiles/h2m.tar.gz`.

References

[1] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statist. Soc. Ser. B*, 39(1):1–38 (with discussion), 1977.

[2] C. F. J. Wu. On the convergence properties of the EM algorithm. *Annals of Statistics*, 11(1):95–103, 1983.

[3] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE*, 77(2):257–285, February 1989.

[4] L. R. Rabiner and B-H. Juang. *Fundamentals of speech recognition.* Prentice-Hall, 1993.

[5] C. K. Carter and R. Kohn. On Gibbs sampling for state space models. *Biometrika*, 81(3):541–553, 1994.