Also appears in the *Online Symposium for Electronics Engineer 2000*
http://www.techonline.com/osee/

Hidden Markov Models: Fundamentals and Applications
Part 1: Markov Chains and Mixture Models

Valery A. Petrushin
petr@cstar.ac.com
Center for Strategic Technology Research
Accenture
3773 Willow Rd.
Northbrook, Illinois 60062, USA.

Abstract

The objective of this tutorial is to introduce basic concepts of a Hidden Markov Model (HMM) as a fusion of more simple models such as a Markov chain and a Gaussian mixture model. The tutorial is intended for the practicing engineer, biologist, linguist or programmer who would like to learn more about the above mentioned fascinating mathematical models and include them into one's repertoire. This lecture presents Markov Chains and Gaussian mixture models, which constitute the preliminary knowledge for understanding Hidden Markov Models.

Introduction

Why it is so important to learn about these models? First, the models have proved to be indispensable for a wide range of applications in such areas as signal processing, bioinformatics, image processing, linguistics, and others, which deal with sequences or mixtures of components. Second, the key algorithm used for estimating the models – the so-called Expectation Maximization Algorithm -- has much broader application potential and deserves to be known to every practicing engineer or scientist. And last, but not least, the beauty of the models and algorithms makes it worthwhile to devote some time and efforts to learning and enjoying them.

1 Introduction into Markov chains

1.1 Paving the path to the shrine problem

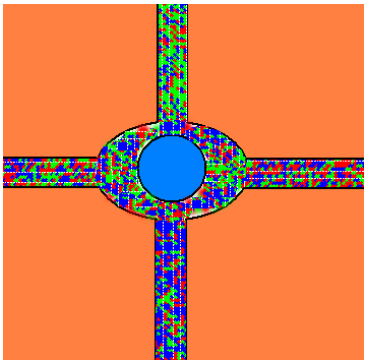

Figure 1.1. Paving problem

Welcome to the planet Median! Median is the capital planet of the Stochastic Empire, which conquered all worlds of the observable part of the Universe and spread its influence to the hidden parts. Ruled by the Glorious and Fair ruler, Emperor Probabil the Great, the Empire reached the most flourishing state of commerce, science and industry ever.

Welcome to Bayes-City, the capital of the Empire! Today, the Emperor's heralds had the honor of announcing a new giant project. The four major Provinces of the Empire have to pave the paths from their residence cities on Median to the St. Likely Hood Shrine, located on the Mean Square in Bayes-City. Each path is 1,000 miles long and has to be paved using large tiles of the following precious stones: ruby (red), emerald (green) and topaz (blue) in accordance to its

Province's random process which states that the color of next tile depends only on the color of the current tile. The square around the St. Likely Hood Shrine will be secretly divided into sectors and paved by the teams from all four Provinces over one night (Figure 1.1). A pilgrim will be allowed to enter the Shrine to enjoy the view of the Bayesian Belief Network, the log that served as a chair to Likely Hood, and the other relics, and also to talk to the high priests of science during the high confidence time intervals if and only if he or she can determine for three sectors of pavement around the Shrine which sector has been paved by the team of which Province.
Imagine you are a pilgrim who journeyed more than 1,000 miles on foot to see the Stochastic Empire's relics. How can you solve the problem?

1.2 Markov chains

We deal with a stochastic or random process which is characterized by the rule that only the current state of the process can influence the choice of the next state. It means the process has no memory of its previous states. Such a process is called a *Markov process* after the name of a prominent Russian mathematician Andrey Markov (1856-1922). If we assume that the process has only a finite or countable set of states, then it is called a *Markov chain*. Markov chains can be considered both in discrete and continuous time, but we shall limit our tutorial to the discrete time finite Markov chains.

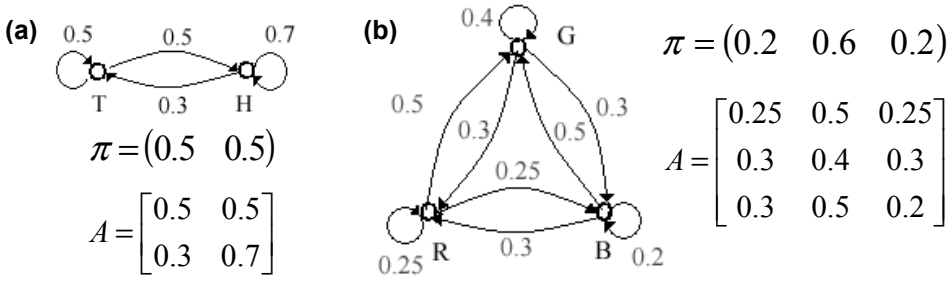

Figure 1.2. Markov chain models for a biased coin (a), and the paving model M_N

Such chains can be described by diagrams (Figure 1.2). The nodes of the diagram represent the *states* (in our case, a state corresponds to a choice of a tile of a particular color) and the edges represent *transitions* between the states. A *transition probability* is assigned to each edge. The probabilities of all edges outgoing from a node must sum to one. Beside that, there is an *initial state probability distribution* to define the first state of the chain.

$$M = \langle \pi, A \rangle \qquad \pi = (\pi_1, \pi_2, ..., \pi_N) \quad A = \{a_{ij}\}_{i,j=\overline{1,N}} \tag{1.1}$$

$$P(q_{k+1} \mid q_1, ..., q_k) = P(q_{k+1} \mid q_k) \tag{1.2}$$

Thus, a Markov chain is uniquely defined by a pair (1.1), where π is the vector of initial probability distribution and A is a stochastic transition matrix. The Markov process characteristic property is represented by (1.2). Figure 1.2 presents Markov chain models for a biased coin and tile generation.

1.3 Training algorithm

We assume that each Province uses its own Markov chain to generate sequences of tiles. We also assume that all tiles have their ordinal numbers carved on them and we have no problem determining the order of tiles in the sequence (in spite of the fact that the tiles cover the two-dimensional surface of the path). The resulting paths could be visually different for pavements of different Provinces. For example, North Province, which is famous for its deposits of emerald, might have larger probabilities for transitions to emerald (green) state and generate greenish paths. But the differences among models could be rather small and the paths produced by the models could look very similar, especially for the paths of short length. That is why we have to find a formal approach for estimating the model's parameters based on

data. Thanks to the Emperor, we have enough data to train our models -- 1,000 miles of data per model! If a tile has the diameter of one yard and the tiles are lined up in rows of 20 tiles per row then we have 35,200 tiles per mile. To estimate parameters of a model with 3 states, we need to calculate 12 values: a 3-element vector of initial state probabilities and a 3 by 3 state transition probability matrix. Let $Q = q_1 q_2 \ldots q_L$ be a training sequence of states.

As estimations for the initial state probabilities, we use frequencies of tiles c_i in the training sequence (1.3). To estimate transition probabilities, we have to count the number of pairs of tiles for each combination of colors c_{ij} (i,j=1,N) and then normalize values of each row (1.4).

$$\pi_i = \frac{c_i}{L} \quad i = \overline{1,N} \tag{1.3}$$

$$a_{ij} = \frac{c_{ij}}{\sum_{j=1}^{N} c_{ij}} \quad i,j = \overline{1,N} \qquad \sum_{i=1}^{N}\sum_{j=1}^{N} c_{ij} = L-1 \tag{1.4}$$

Table 1.1 presents three estimates of parameters for the increasing length of the training sequence.

Table 1.1. Markov chain training results

True	L=1000	L=10000	L=35200
$\pi = (0.2 \quad 0.6 \quad 0.2)$	$\pi = (0.25 \quad 0.45 \quad 0.3)$	$\pi = (0.28 \quad 0.46 \quad 0.26)$	$\pi = (0.29 \quad 0.45 \quad 0.26)$
$A = \begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.3 & 0.4 & 0.3 \\ 0.3 & 0.5 & 0.2 \end{bmatrix}$	$A = \begin{bmatrix} 0.20 & 0.53 & 0.27 \\ 0.26 & 0.36 & 0.38 \\ 0.27 & 0.51 & 0.22 \end{bmatrix}$	$A = \begin{bmatrix} 0.26 & 0.49 & 0.25 \\ 0.3 & 0.4 & 0.3 \\ 0.3 & 0.5 & 0.2 \end{bmatrix}$	$A = \begin{bmatrix} 0.24 & 0.52 & 0.24 \\ 0.3 & 0.4 & 0.3 \\ 0.3 & 0.49 & 0.21 \end{bmatrix}$

Now you know what to do! You should walk along each path, collect data and use (1.3) and (1.4) to create Markov chains for each Province.

1.4 Recognition algorithm

Let us assume that you have already built the models for each Province: *MN*, *ME*, *MS*, and *MW*. How can you use them to determine which Province's team paved a sector Q? A natural approach to solve this problem is to calculate the conditional probability of the model *M* for the given sequence of tiles *Q*: *P(M|Q)*.

Using the Bayes' Rule (1.5), we can see that *P(Q)* does not depend on the model. Thus, we should pick up the model, which maximizes the numerator of the formula (1.5).

$$P(M_k \mid Q) = \frac{P(Q \mid M_k) \cdot P(M_k)}{P(Q)} \tag{1.5}$$

where $P(M_k|Q)$ is the posterior probability of model M_k for the given sequence Q; $P(Q|M_k)$ is the likelihood of Q to be generated by M_k ; $P(M_k)$ is the prior probability of M_k ; $P(Q)$ is the probability of Q to be generated by all models.

This criterion is called *maximum aposteriori probability (MAP)* estimation and can be formally expressed as (1.6). However, it is often the case that the prior probabilities are not known or are uniform over all the models. Then we can reduce (1.6) to (1.7). This criterion is called *maximum likelihood (ML)* estimation.

$$M_{k^*} = \arg\max_{M_k} \{P(Q \mid M_k)\} \tag{1.6}$$

$$M_{k^*} = \arg\max_{M_k} \{P(Q \mid M_k) \cdot P(M_k)\} \tag{1.7}$$

How can you calculate the likelihood? Let us consider the sequence "323" and trace on the diagram in Figure 2.1b to see how this sequence could be generated. The first state can be generated with the probability π_3, the transition probability from 3 to 2 is a_{32}, and from 2 to 3 is a_{23}. The probability of the sequence is equal to the product of these three factors (1.8).

$$P(Q \mid M) = \pi_3 a_{32} a_{23} = 0.2 \cdot 0.5 \cdot 0.3 = 0.075 \quad (1.8)$$

Generalizing our derivation for the sequence of arbitrary length we obtain (1.9). Typically the logarithm of likelihood or *log-likelihood* is used for calculations (1.10).

$$P(Q \mid M) = \pi_{q_1} \cdot a_{q_1 q_2} \cdot a_{q_2 q_3} \cdot \ldots \cdot a_{q_{L-1} q_L} = \pi_{q_1} \cdot \prod_{i=2}^{L} a_{q_{i-1} q_i} \quad (1.9)$$

$$\log P(Q \mid M) = \log \pi_{q_1} + \sum_{i=2}^{L} \log a_{q_{i-1} q_i} \quad (1.10)$$

Table 1.2 presents the accuracy of recognition of 1,000 sequences of different length. The test sequences were generated by the "true" models and recognized by the estimated models. For recognition we used the ML criterion, i.e. we calculated the log-likelihood of the given sequence for each of four models and picked up the model with the maximal value. The set of models was trained on "one-mile" data for each model. You can see that the accuracy is poor for the short sequences and gradually improves with growing length of sequences. For reliable recognition we need sequences of more than 100 symbols.

Table 1.2. Recognition accuracy for test sequences of different length

Length	10	20	50	100	150	200	250	300	400	500
Accuracy (%)	55.7	64.7	82	92	96.4	98	99.4	99.7	99.8	100

True models:

North

$$\pi = (0.2 \quad 0.6 \quad 0.2)$$

$$A = \begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.3 & 0.4 & 0.3 \\ 0.3 & 0.5 & 0.2 \end{bmatrix}$$

East

$$\pi = (0.5 \quad 0.25 \quad 0.25)$$

$$A = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.5 & 0.25 & 0.25 \\ 0.4 & 0.3 & 0.3 \end{bmatrix}$$

South

$$\pi = (0.2 \quad 0.2 \quad 0.6)$$

$$A = \begin{bmatrix} 0.2 & 0.3 & 0.5 \\ 0.3 & 0.3 & 0.4 \\ 0.2 & 0.2 & 0.6 \end{bmatrix}$$

West

$$\pi = (0.3 \quad 0.3 \quad 0.4)$$

$$A = \begin{bmatrix} 0.3 & 0.3 & 0.4 \\ 0.3 & 0.4 & 0.3 \\ 0.4 & 0.3 & 0.3 \end{bmatrix}$$

Estimated "one-mile" models:

North

$$\pi = (0.28 \quad 0.45 \quad 0.27)$$

$$A = \begin{bmatrix} 0.25 & 0.49 & 0.26 \\ 0.3 & 0.4 & 0.3 \\ 0.3 & 0.5 & 0.2 \end{bmatrix}$$

East

$$\pi = (0.43 \quad 0.28 \quad 0.29)$$

$$A = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.5 & 0.25 & 0.25 \\ 0.41 & 0.29 & 0.3 \end{bmatrix}$$

South

$$\pi = (0.23 \quad 0.24 \quad 0.53)$$

$$A = \begin{bmatrix} 0.2 & 0.29 & 0.51 \\ 0.3 & 0.3 & 0.4 \\ 0.2 & 0.2 & 0.6 \end{bmatrix}$$

West

$$\pi = (0.33 \quad 0.33 \quad 0.34)$$

$$A = \begin{bmatrix} 0.3 & 0.29 & 0.41 \\ 0.3 & 0.39 & 0.31 \\ 0.4 & 0.3 & 0.3 \end{bmatrix}$$

1.5 Generalizations

Two important generalizations of the Markov chain model described above are worth to mentioning. They are high-order Markov chains and continuous-time Markov chains.

In the case of a high-order Markov chain of order n, where $n > 1$, we assume that the choice of the next state depends on n previous states, including the current state (1.11).

$$P(q_{k+n} \mid q_1, q_2, \ldots, q_k, \ldots, q_{k+n-1}) = P(q_{k+n} \mid q_k, \ldots, q_{k+n-1}) \quad (1.11)$$

In the case of a continuous-time Markov chain, the process waits for a random time at the current state before it jumps to the next state. The waiting time at each state can have its own probability density function or the common function for all states, for example, an exponential distribution (1.12).

$$f_T(t) = \lambda \cdot e^{-\lambda t} \quad t \geq 0 \quad (1.12)$$

1.6 References and Applications

The theory of Markov chains is well established and has vast literature. Taking all responsibility for my slant choice, I would like to mention only two books. Norris' book [1] is a concise introduction into the theory of Markov chains, and Stewart's book [2] pays attention to numerical aspects of Markov chains and their applications. Both books have some recommendations and references for further reading.

Markov Chains have a wide range of applications. One of the most important and elaborated areas of applications is analysis and design of queues and queuing networks. It covers a range of problems from productivity analysis of a carwash station to the design of optimal computer and telecommunication networks. Stewart's book [2] gives a lot of details on this type of modeling and describes a software package that can be used for queuing networks design and analysis (http://www.csc.ncsu.edu/faculty/WStewart/MARCA/marca.html).

Historically, the first application of Markov Chains was made by Andrey Markov himself in the area of language modeling. Markov was fond of poetry and he applied his model to studies of poetic style. Today you can have fun using the Shannonizer which is a Web-based program (see http://www.nightgarden.com/shannon.htm) that can rewrite your message in the style of Mark Twain or Lewis Carroll.

Another example of Markov chains application in linguistics is stochastic language modeling. It was shown that alphabet level Markov chains of order 5 and 6 can be used for recognition texts in different languages. The word level Markov chains of order 1 and 2 are used for language modeling for speech recognition (http://www.is.cs.cmu.edu/ISL.speech.lm.html).

DNA sequences can be considered as texts in the alphabet of four letters that represent the nucleotides. The difference in stochastic properties of Markov chain models for coding and non-coding regions of DNA can be used for gene finding. GeneMark is the first system that implemented this approach (see http://genemark.biology.gatech.edu/GeneMark/).

The other group of models known as branching processes was designed to cover such application as modeling chain reaction in chemistry and nuclear physics, population genetics (http://pespmc1.vub.ac.be/MATHMPG.html), and even game analysis for such games as baseball (http://www.retrosheet.org/mdp/markov/theory.htm), curling, etc.

And the last application area, which I'd like to mention, is so called Markov chain Monte Carlo algorithms. They use Markov chains to generate random numbers that belong exactly to the desired distribution or, speaking in other words, they create a perfectly random sampling. See the http://dimacs.rutgers.edu/~dbwilson/exact/ for additional information about these algorithms.

2 Introduction into Gaussian mixture models

2.1 Revealing the old monk's secret problem

All respectful families on Median have their traditional vast gardens of sculptures. The gardens, which are covered by sand composed of microscopic grains of precious stones, exhibit both the sculptures of concrete things and persons and abstract concepts. The color of the sand is as important as the artistic and scientific value of sculptures. The sand recipes are kept in secret and passed from one generation to another.
A new rich man (let us call him Mr. Newrich), who interpreted the slogan "Keep off Median!" as permission to explore and exploit poor developed suburbs of the Empire, returned to the capital planet and settled himself in a prestigious area. But when he started to design his garden of sculptures, he ran into a problem of sand recipe. After several

unsuccessful attempts to buy a recipe he decided to steal some sand from the workshop of an old monk who served as a sandkeeper for a noble and respectful, but not too rich family, and reveal the secret of the recipe using the stolen sand as data. Mr. Newrich's helpers measured the wavelength of reflected light for 10,000 sand grains. Figure 2.1 presents the histogram for the data. Three concentrations of data near 400 nm, 420 nm and 450 nm allow us to form a hypothesis that sand is a mixture of three components.

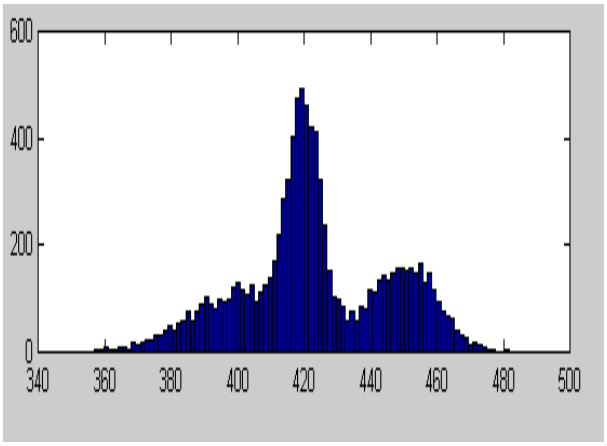

Figure 2.1. Histogram of sand data.

Let us pretend that you are the chief scientist at Allempire Consulting, Inc., and Mr. Newrich pays you a fortune to decode the data. How can you determine the characteristics of each component and proportion of the mixture? It seems that stochastic modeling approach called Gaussian Mixture Models can do the job.

2.2 Gaussian mixture model

Let us assume that components of the mixture have Gaussian distribution with mean μ and variance σ^2 and probability density function (2.1). The choice of the Gaussian distribution is natural and very common when we deal with a natural object or phenomenon, such as sand. But the mixture model approach can be applied to the other distributions from the exponential family.

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \qquad (2.1)$$

We let $o_1, o_2, \ldots, o_L$ denote our observations or data points. Then each data point is assumed to be an independent realization of the random variable O with the three-component mixture probability density function (2.2) and log-likelihood function (2.3). With the maximum likelihood approach to the estimation of our model $M = \langle \nu, N(\mu_1,\sigma_1),\ldots, N(\mu_K,\sigma_K)\rangle$, we need to find such values of ν_i, μ_i and σ_i that maximize the function (2.3). To solve the problem, the Expectation Maximization (EM) Algorithm is used.

$$f(o;M) = \sum_{i=1}^{K} \nu_i f_i(o;\mu_i,\sigma_i) \qquad \sum_{i=1}^{K} \nu_i = 1 \qquad (2.2)$$

$$\log L(M) = \sum_{j=1}^{L} \log\left(\sum_{i=1}^{K} \nu_i f(o_j;\mu_i,\sigma_i)\right) \qquad (2.3)$$

The EM Algorithm is also known as the "Fuzzy k-mean" algorithm. The key idea of the algorithm is to assign to each data point o_i a vector w_i that has as many elements as there are components in the mixture (three in our case). Each element of the vector w_{ij} (we shall call them *weights*) represents the confidence or probability that the i-th data point oi belongs to the

j-th mixture component. All weights for a given data point must sum to one (2.4). For example, a two-component mixture weight vector (1, 0) means that the data point certainly belongs to the first mixture component, but (0.3, 0.7) means than the data point belongs more to the second mixture component than to the first one.

$$w_{ij} = P(N(\mu_j, \sigma_j) \mid o_i), \qquad \sum_{j=1}^{K} w_{ij} = 1 \qquad (2.4)$$

For each iteration, we use the current parameters of the model to calculate weights for all data point (expectation step or E-step), and then we use these updated weights to recalculate parameters of the model (maximization step or M-step).

Let us see how the EM algorithm works in some details. First, we have to provide an initial approximation of model $M^{(0)}$ using one of the following ways:
(1) Partitioning data points arbitrarily among the mixture components and then calculating the parameters of the model.
(2) Setting up the parameters randomly or based on our knowledge about the problem.
After this, we start to iterate E- and M-steps until a stopping criterion gets true.

For the E-step the expected component membership weights are calculated for each data point based on parameters of the current model. This is done using the Bayes' Rule that calculates the weight w_{ij} as the posterior probability of membership of *i*-th data point in *j*-th mixture component (2.5). Compare the formula (2.5) to the Bayes' Rule (1.5).

$$w_{ij} = \nu_j \cdot f(o_i; \mu_j, \sigma_j) \Big/ \sum_{h=1}^{K} \nu_h \cdot f(o_i; \mu_h, \sigma_h) \qquad (2.5)$$

For the M-step, the parameters of the model are recalculated using formulas (2.6), (2.7) and (2.8), based on our refined knowledge about membership of data points.

$$\nu_j^{(t+1)} = \sum_{i=1}^{L} w_{ij} \Big/ L \qquad j = \overline{1, K} \qquad (2.6)$$

$$\mu_j^{(t+1)} = \sum_{i=1}^{L} w_{ij} \cdot o_i \Big/ \sum_{i=1}^{L} w_{ij} \qquad (2.7)$$

$$\sigma_j^{2(t+1)} = \sum_{i=1}^{L} w_{ij} \cdot (o_i - \mu_j)^2 \Big/ \sum_{i=1}^{L} w_{ij} \qquad (2.8)$$

A nice feature of the EM algorithm is that the likelihood function (2.3) can never decrease; hence it eventually converges to a local maximum. In practice, the algorithm stops when the difference between two consecutive values of likelihood function is less than a threshold (2.9).

$$\log L(M^{(t+1)}) - \log L(M^{(t)}) < \varepsilon \qquad (2.9)$$

In order to better understand how the EM algorithm works, let us trace it for the case of two mixture components with a small amount of data points. Suppose we have 10 data points that are arbitrarily partitioned between two components (see Figure 2.3a). The initial model has large variances and means that lie close to each other (see Table 2.1). Using this model, the algorithm calculates a new model with larger distance between means and smaller variances. After 22 iterations the algorithm produces the model that maximizes the likelihood function and optimizes the membership probability distributions for each data point (see Figure 2.3 and Table 2.1). The resulting solution is depicted as two green bell-shaped curves on Figure 2.3d. You can see that we have perfect mixture proportion, and good approximation for the mean of the first component and variance of the second component. In general, the results are amazingly good for a data set of ten points.

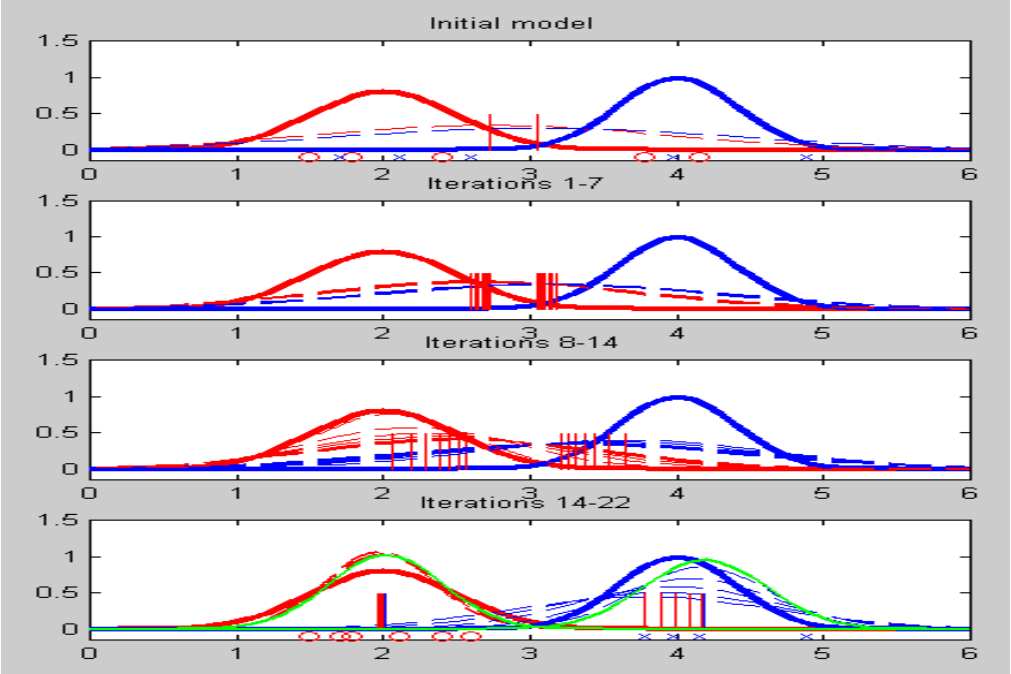

Figure 2.3. EM algorithm convergence.

Table 2.1 EM algorithm convergence (10 points)

t	ν_1	ν_2	μ_1	μ_2	σ_1	σ_2
0	0.5	0.5	2.7190	3.0534	1.1848	1.3272
1	0.5048	0.4952	2.7313	3.0442	1.0835	1.1692
2	0.5045	0.4955	2.7137	3.0619	1.0798	1.1676
			...			
9	0.4972	0.5028	2.5089	3.2593	0.9752	1.1633
10	0.4943	0.5057	2.4532	3.3095	0.9386	1.1556
11	0.4904	0.5096	2.3833	3.3702	0.8874	1.1419
			...			
20	0.5993	0.4007	2.0165	4.1871	0.3901	0.4224
21	0.5998	0.4002	2.0170	4.1892	0.3903	0.4187
22	**0.5999**	**0.4001**	**2.0170**	**4.1893**	**0.3903**	**0.4185**
True	**0.6**	**0.4**	**2.0**	**4.0**	**0.5**	**0.4**

Applying the EM algorithm to Mr. Newrich's data, we obtain the solution presented in Table 2.2. Due to a rather good initial model it took only 17 iterations to converge. The old monk's secret is revealed!

Table 2.2. Monk's secret problem

Proportions	ν_1	ν_2	ν_3
Estimated	0.309	0.392	0.298
True	0.3	0.4	0.3
Means	μ_1	μ_2	μ_3
Estimated	401.146	420.058	450.059
True	400.0	420.0	450.0
S.d.	σ_1	σ_2	σ_3
Estimated	15.646	5.004	9.727
True	15.0	5.0	10.0

The mixture model approach and the EM algorithm have been generalized to accommodate multivariate data and other kind of probability distributions. Another method of generalization is to split multivariate data into several sets or *streams* and create a mixture model for each stream, but have the joint likelihood function.

The special beauty of mixture models is that they work in cases when the components are heavily intersected. For example, for two-dimensional data of 1000 points and three-component mixture with two components having equal means but different variances, we can get a good approximation in 58 iterations (see Figure 2.4). The dark side of the EM algorithm is that it is very slow and is considered to be impractical for large data sets.

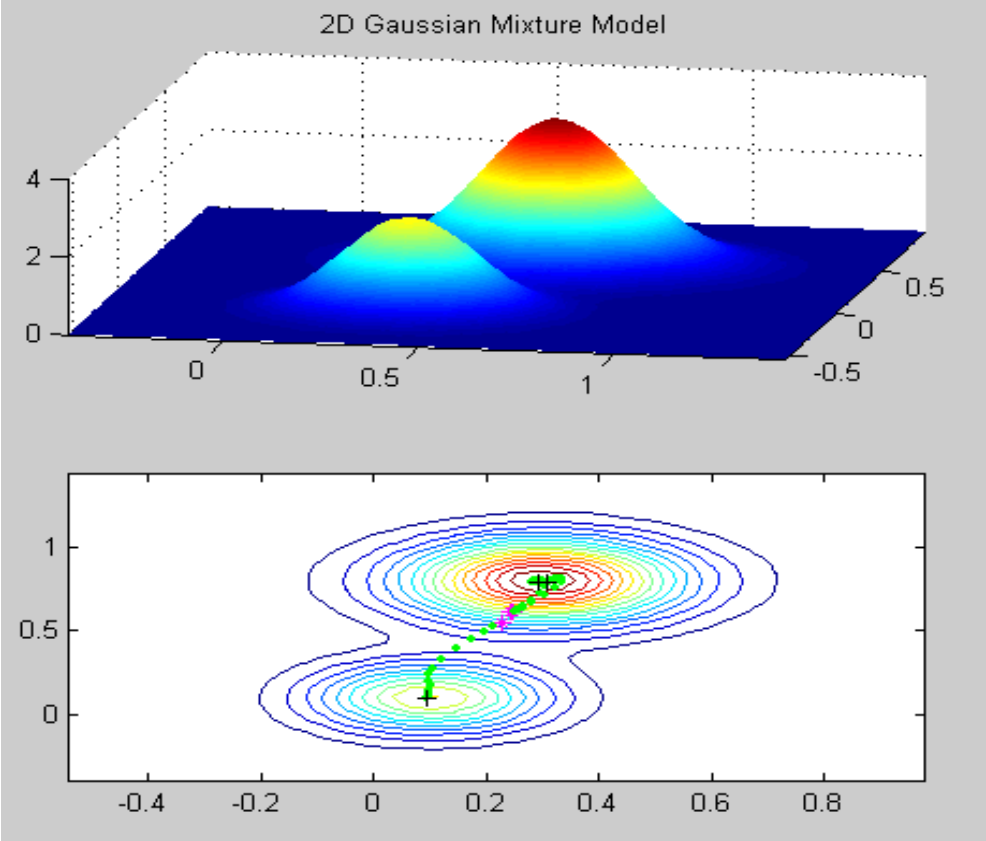

Figure 2.4. Three-component mixture with two components having equal means.

The EM algorithm was invented in late 1970s, but the 1990s marked a new wave of research revived by growing interest to data mining. Recent research is going in two directions:
(1) Generalization of the EM algorithm and extension to other probability distributions, and
(2) Improving the speed of EM algorithm using hierarchical data representations.

2.3 References and Applications

There is a sufficient body of literature devoted to the mixture models. Most of it is intended for researches with a strong mathematical background. I refer to two books [3-4] written by Australian researchers. There are also several commercial and free software packages devoted to mixture models, most of which are available on the Web.
Commercial software:

- MPLUS http://www.statmodel.com/index.html
- MIX http://icarus.math.mcmaster.ca/peter/mix/mix.html

Freeware:

- EMMIX http://www.maths.uq.edu.au/~gjm/emmix/emmix.html
- MCLUST http://www.stat.washington.edu/fraley/mclust/
- MIXCLUS2 http://biometrics.ag.uq.edu.au/software.htm
- NORMIX http://alumni.caltech.edu/~wolfe/
- CAMAN http://ftp.ukbf.fu-berlin.de/sozmed/caman.html

Mixture Models have been in use for more than 30 years. It is a very powerful approach to model-based clustering and classification. It has been applied to numerous problems. Some of the more interesting applications are described below.
Mixture models are widely used in image processing for image representation and segmentation, object recognition, and content-based image retrieval :

- http://www.cs.toronto.edu/vis/
- http://www.maths.uwa.edu.au/~rkealley/diss/diss.html

Mixture models also were used for text classification and clustering:

- http://www.cs.cmu.edu/~mccallum/

Gaussian mixture models proved to be very successful for solving the speaker recognition and speaker verification problems:

- http://www.ll.mit.edu/IST/

There is growing interest in applying this technique to exploratory analysis of data or data mining, and medical diagnosis:

- http://www.cs.helsinki.fi/research/cosco/
- http://www.cnl.salk.edu/cgi-bin/pub-search/

You can learn more about these applications by following the URLs presented above.

References

[1] J.R. Norris (1997) *Markov Chains*. Cambridge University Press.
[2] W.J. Stewart (1994) *Introduction to the Numerical Solutions of Markov Chains*. Princeton University Press.
[3] G.J. McLachlan and K.E. Basford (1988) *Mixture Models: Inference and Application to Clustering*. New York: Marcel Dekker.
[4] G.J. McLachlan and T. Krishnan (1997) *The EM Algorithm and Extensions*. New York: Wiley.