The EM Algorithm

Roni Rosenfeld

February 1997

Supporting literature:

- A. P. Dempster, N. M. Laird and D. B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm", *Journal of the Royal Statistical Society*, volume 39, number 1, pages 1–38, 1977.
- L. E. Baum, "An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes", *Inequalities, III: Proceedings of a symposium*, 1977.

1 A Simple Example

Given two language models M_A and M_B, consider a hybrid model M_{hybrid} which, every time it is consulted, stochastically chooses which of the two models to use, with probability λ of choosing M_A (and probability $1-\lambda$ of choosing M_B).
We know M_A and M_B fully, but we don't know the value of λ. Given some data $\{w_1, w_2, \ldots, w_N\}$ which is assumed to have been generated by M_{hybrid} , we would like to find a ML estimate for λ, namely that λ which maximizes the log likelihood function:

$$L(w_1, \ldots, w_N; \lambda) = \sum_i \log P_{\text{hybrid}}(w_i) = \sum_i \log(\lambda P_A(w_i) + (1 - \lambda) P_B(w_i)$$

If we were told which of the two models was used in generating each word w_i, we could have estimated λ directly with a simple relative frequency expression:

$$\frac{\text{\#-times-}M_A\text{-was-chosen}}{N}$$

But this is not the case. The stochastic choice between the two models is a *hidden event*. We cannot observe it, but can only collect evidence for it by observing the words that were generated.

Now consider the following algorithm:

INIT: Set an initial guess for λ to some arbitrary non-zero choice (say, 0.5).

ITERATE: at step k:

E-STEP Use the data to compute the expected number of times M_A was used, assuming $\lambda^{[k]}$ – the current estimate for λ:

$$\begin{aligned}
\mathbf{E}_{\lambda^{[k]}}(M = A | w_1, w_2, \ldots, w_N) &= \sum_i \Pr_{\lambda^{[k]}}(M = A | w_i) \\
&= \sum_i \frac{\Pr(M_A, w_i)}{\Pr(w_i)} \\
&= \sum_i \frac{\lambda^{[k]} \cdot P_A(w_i)}{\lambda^{[k]} \cdot P_A(w_i) + (1 - \lambda^{[k]}) \cdot P_B(w_i)}
\end{aligned}$$

M-STEP Derive an improved estimate for λ using the statistics collected in the E-step:

$$\lambda^{[k+1]} = \frac{\mathbf{E}_{\lambda^{[k]}}(M = A | w_1, w_2, \ldots, w_N)}{N}$$

TERMINATE: exit if $L(w_1, \ldots, w_N; \lambda^{k+1}) \approx L(w_1, \ldots, w_N; \lambda^{[k]})$

The above algorithm always converges, and the resulting λ is the ML solution for this data.

2 Sufficient Statistics

Note that, in the example above, in order to estimate λ we don't really need to know the actual sequence of the hidden events. Rather, it would be sufficient to know the total number of times model A was chosen. In statistical terminology, functions of the data which convey everything that is relevent to the estimation of the parameters are called "sufficient statistics". In the example above, the total number of times model A was chosen is a sufficient statistic of the hidden event distribution.

3 The EM algorithm

The above example can be generalized as follows.

Suppose we are given a model family, together with some incomplete data (namely, data containing some hidden events). We would like to use the incomplete data to derive the ML estimate of the various parameters $\{\lambda_j\}$ of the model, namely:

$$\lambda^* = \arg\max_{\lambda}[L(\text{data}; \lambda)]$$

This can be done using the EM algorithm:

DEFINE: the sufficient statistics necessary for estimating the λ's.

INIT: Set an initial guess for the λ's to some arbitrary non-singular choice, call these $\lambda_j^{[0]}$.

ITERATE: at step k:

E-STEP: Use the data to compute the sufficient statistics, assuming the parameter values $\lambda_j^{[k]}$.

M-STEP: Derive $\lambda_j^{[k+1]}$ as a ML estimate based on the sufficient statistics collected in the E-step.

TERMINATE: exit if $L(\text{data}; \{\lambda_j^{[k+1]}\}) \approx L(\text{data}; \{\lambda_j^{[k]}\})$ [1].

Various forms of this algorithm have been used for many years in diverse applications. In 1977, Dempster, Laird & Rubin further formalized the algorithm, gave it its name, and proved that, under very broad conditions, the likelihood function is guaranteed to increase in every iteration. Since L is bounded, the EM algorithm is guaranteed to converge (at least in terms of the likelihood). Convergence may be to a local maximum, although in some cases it can be shown that there is only one maximum.

Let us now consider two more examples:

[1] The λ's themselves are not guaranteed to converge, although in most cases they will.

4 Example 2: optimizing interpolation weights

Suppose we are given a set of language models $P_1, P_2, \ldots, P_M$, and would like to build a combined model by averaging their predictions on a word-by-word basis, namely:

$$P_{\text{combined}}(w_i|h) = \sum_{j=1}^{M} \lambda_j P_j(w_i|h)$$

where the λ's are positive interpolation weights summing to 1.

We would like to choose the best possible λs, namely those weights that will minimize perplexity of new data. Given a representative set of new data $\{w_1, w_2, \ldots, w_N\}$, we need to find the λ's that minimize perplexity over it, or, alternatively, maximize the data's likelihood. Even though the motivation is different, the mathematical form of P_{combined} is the same as that of P_{hybrid} in the previous example. Thus mathematically speaking this is a trivial generalization of the first example to any number of components. The sufficient statistic for ML-estimating λ_j is $\mathbf{E}(\text{Model}_\text{j}|\text{data})$. We initialize the λ's to any non-zero value (e.g. $\lambda_j = \frac{1}{M}$). Then, in iteration k, the E-step computes, for every model j:

$$\begin{aligned} \mathbf{E}_{\lambda^{[k]}}(\text{Model}_\text{j}|\text{w}_1, \text{w}_2, \ldots, \text{w}_\text{N}) &= \sum_i \Pr_{\lambda^{[k]}}(\text{Model}_\text{j}|\text{w}_\text{i}) && (1) \\ &= \sum_i \frac{\lambda_j^{[k]} \cdot P_j(w_i)}{\sum_j \lambda_j^{[k]} \cdot P_j(w_i)} && (2) \\ & && (3) \end{aligned}$$

and the M-step computes, for every model j:

$$\lambda_j^{[k+1]} = \frac{\mathbf{E}_{\lambda^{[k]}}(\mathrm{Model_j}|\mathrm{w_1}, \mathrm{w_2}, \ldots, \mathrm{w_N})}{\mathrm{N}}$$

5 Example 3: Training a Hidden Markov Model

Hidden Markov Models (HMMs) are widely used in speech recognition and many other applications. A (first order) Markov model is a finite state model where the stochastic function governing the transition between states depends only on the last state visited. Namely, let X_t be the random variable denoting the active state at time t. Then:

$$P(X_{t+1}|X_1, X_2, \ldots, X_t) = P(X_{t+1}|X_t)$$

This is known as the (first-order) Markov property.

In a hidden Markov model, the state the system is in is not directly observable. Instead, at every time step t, a symbol y_t is emitted from a finite alphabet[2][3]. Again, the stochastic function governing the selection of the symbol depends only on x_t — the state the system is in at time t.

Let a_{ij} be the probability of transitioning into state j given that the model is in state i ($\sum_j a_{ij} = 1$), and let $b_i(k)$ be the probability of emitting symbol k given that the model is in state i. Together, the a_{ij}'s and $b_i(k)$'s fully specify the model.

Suppose now that we are given a sequence of symbol observations, and are asked to estimate the model's parameters (the a's and b's). This problem is known as "training the HMM".

[2] in continuous HMMs, the value being emitted is from a continuous domain.

[3] In some formulations, the symbol emissions are associated with state *transitions*.

One approach would be maximum likelihood training, namely finding the parameters that maximize the likelihood function:

$$L(data|a_{ij}, b_i(k)) = \sum_{x_0, x_1, \ldots, x_N} \prod_{t=1}^{N} a_{x_{t-1}, x_t} b_{x_{t-1}, x_t}(y_t)$$

where the summation is over all possible state sequences, and, for any given sequence, the product computes the joint likelihood of that sequence with the observed symbols.

The EM algorithm provides a straightforward solution. We start by identifying the sufficient statistics needed for estimating the parameters. As before, we don't really need to know the exact state sequence. We can ML-estimate a_{ij} from the expected number of times state i was visited (C_i), and the expected number of times the transition $i \rightarrow j$ occurred ($C_{i \rightarrow j}$). Similarly, we can estimate $b_i(k)$ from the expected number of times state i was visited (C_i) and the expected fraction of these during which symbol k was emitted ($C_i(k)$).

To compute the latter quantities based on some tentative values of the parameters, we need to define a few intermediate variables. Let λ represent the tentative parameters $\{a_{ij}, b_i(k)\}$, and let:

$$\alpha_t(i) \stackrel{\text{def}}{=} \Pr(y_1, \ldots, y_t \ , \ x_t = i|\lambda)$$

$$\beta_t(i) \stackrel{\text{def}}{=} \Pr(y_{t+1}, \ldots, y_T|x_t = i, \lambda)$$

$$\gamma_t(i) \stackrel{\text{def}}{=} \Pr(x_t = i|y_1, \ldots, y_T, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{\sum_j \alpha_t(j)\beta_t(j)}$$

Given the tentative parameters λ and the data $y_1, \ldots, y_T$, the α's and β's can be computed in polynomial time using a simple dynamic programming approach, and then the γ's can be derived from them by summation.

Now at every iteration, the E-step computes:

$$C_i = \sum_t \gamma_t(i)$$

$$C_{i \to j} = \sum_t \Pr(x_t = i, x_{t+1} = j | y_1, \ldots, y_T, \lambda) = \sum_t \frac{\alpha_t(i) a_{ij} b_j(y_{t+1}) \beta_{t+1}(j)}{\sum_i \sum_j \alpha_t(i) a_{ij} b_j(y_{t+1}) \beta_{t+1}(j)}$$

$$C_i(k) = \sum_{t \,:\, y_t = k} \gamma_t(i)$$

and the M-step re-estimates the parameters as:

$$a_{ij} = \frac{C_{i \to j}}{C_i}$$

$$b_i(k) = \frac{C_i(k)}{C_i}$$