Hidden Markov Models, III. Algorithms

Steven R. Dunbar

March 3, 2017

Outline

Forward Algorithm

The **forward algorithm** or **alpha pass**. For $t = 0, 1, 2, \ldots T-1$ and $i = 0, 1, \ldots, N-1$, define

$$\alpha_t(i) = \mathbb{P}\left[\mathcal{O}_0, \mathcal{O}_1, \mathcal{O}_2, \ldots \mathcal{O}_t, x_t = q_i \,|\, \lambda\right]$$

Then $\alpha_t(i)$ is the probability of the partial observation of the sequence up to time t with the Markov process in state q_i at time t.

Recursive computation

The crucial insight is that the $\alpha(i)$ can be computed recursively as follows

1. Let $\alpha_0(i) = \pi_i b_i(\mathcal{O}_0)$, for $i = 0, 1, \ldots N-1$.
2. For $t = 1, 2, \ldots, T-1$ and $i = 0, 1, \ldots N-1$, compute
$$\alpha_t(i) = \left[\sum_{j=0}^{N-1} \alpha_{t-1}(j) a_{ji}\right] b_i(\mathcal{O}_t)$$
3. Then it follows that
$$\mathbb{P}\left[\mathcal{O} \,|\, \lambda\right] = \sum_{i=0}^{N-1} \alpha_{T-1}(i).$$

The forward algorithm only requires about $N^2 T$ multiplications, a large improvement over the $2TN^T$

Backward Algorithm

The **backward algorithm**, or **beta-pass**. This is analogous to the alpha-pass described in the solution to the first problem of HMMs, except that it starts at the end and works back toward the beginning.

It is independent of the forward algorithm, so it can be done in parallel.

Recursive computation

For $t = 0, 1, 2, \ldots T-1$ and $i = 0, 1, \ldots, N-1$, define

$$\beta_t(i) = \mathbb{P}\left[\mathcal{O}_{t+1}, \mathcal{O}_{t+2}, \ldots \mathcal{O}_{T-1} \,|\, x_t = q_i, \lambda\right].$$

The crucial insight again is that the $\beta_t(i)$ can be computed recursively as follows

1. Let $\beta_{T-1}(i) = 1$, for $i = 0, 1, \ldots N-1$.
2. For $t = T-2, T-3, \ldots, 0$ and $i = 0, 1, \ldots N-1$, compute

$$\beta_t(i) = \sum_{j=0}^{N-1} a_{ij} b_j(\mathcal{O}_{t+1}) \beta_{t+1}(j)$$

The Viterbi (Forward-Backward) Algorithm

For $t = 0, 1, 2, \ldots T-1$ and $i = 0, 1, \ldots N-1$ define the *posteriors*

$$\gamma_t(i) = \mathbb{P}\left[x_t = q_i \,|\, \mathcal{O}, \lambda\right].$$

Since $\alpha_t(i)$ measures the probability up to time t and $\beta_t(i)$ measures the probability after time t

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\mathbb{P}\left[\mathcal{O} \,|\, \lambda\right]}.$$

Recall $\mathbb{P}\left[\mathcal{O} \,|\, \lambda\right] = \sum_{i=0}^{N-1} \alpha_{T-1}(i)$.

The most likely state at time t is the state q_i for which $\gamma_i(t)$ is a maximum.

Problem 3: Training and Parameter Fitting

Given an observation sequence $\mathcal{O}$ and the dimensions N and M, find the model $\lambda = (A, B, \pi)$ that maximizes the probability of $\mathcal{O}$. This can interpreted as training a model to best fit the observed data. We can also view this as search in the parameter space represented by A, B and π.

The solution of Problem 3 attempts to optimize the model parameters so as best to describe how the observed sequence comes about. The observed sequence used to solve Problem 3 is called a *training sequence* since it is used to train the model. This training problem is the crucial one for most applications of hidden Markov models since it creates best models for real phenomena.

Baum-Welch Algorithm

For $t = 0, 1, \ldots, T-2$ and $i, j \in \{0, 1, \ldots, N-1\}$, define

$$\xi_t(i,j) = \mathbb{P}\left[x_t = q_i, x_{t+1} = q_j \,|\, \mathcal{O}, \lambda\right]$$

so $\xi_t(i,j)$ is the probability of being in state q_i at time t and transitioning to state q_j at time $t+1$. They can be written in terms of α, β, A and B as

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(\mathcal{O}_{t+1}) \beta_{t+1}(j)}{\mathbb{P}\left[\mathcal{O} \,|\, \lambda\right]}$$

For $t = 0, 1, \ldots, T-2$, $\gamma_t(i)$ and $\xi_t(i,j)$ are related by

$$\gamma_t(i) = \sum_{j=0}^{N-1} \xi_t(i,j)$$

Interpretation

$$\sum_{t=0}^{T-2} \gamma_t(i) = \mathbb{E}\left[\text{number of visits to } q_i \text{ in } [0, T-2]\right]$$

$$= \mathbb{E}\left[\text{number of transitions from } q_i\right]$$

$$\sum_{t=0}^{T-2} \xi_t(i,j) = \mathbb{E}\left[\text{number of transitions from } q_i \text{ to } q_j\right]$$

Estimation

$$\bar{\pi} = \gamma_0(i) = \text{expected freq. in } q_i \text{ at } t = 0$$

$$\bar{a}_{ij} = \frac{\text{exptd. trans. from } q_i \text{ to } q_j}{\text{exptd. trans. from } q_i} = \frac{\sum_{t=0}^{T-2} \xi(i,j)}{\sum_{t=0}^{T-2} \gamma_t(i)}$$

$$\bar{b}_i(k) = \frac{\text{exptd. times in } q_i \text{ emitting } k}{\text{exptd. times in } q_i} = \frac{\sum_{\substack{t \in \{0,\ldots,T-1\} \\ \mathcal{O}_t = k}} \gamma_t(i)}{\sum_{t=0}^{T-1} \gamma_t(i)}$$

Training and Estimating Parameters

Re-estimation is an iterative process. First we initialize $\lambda = (A, B, \pi)$ with a best guess, or if no reasonable guess is available, we choose random values such that $\pi_i \approx 1/N$ and $a_{ij} \approx 1/N$ and $b_j(k) \approx 1/M$. It is critical that A, B and π be randomized since exactly uniform values results in a local maximum from which the model cannot climb. As always, A, B, π must be row stochastic.

Iterative Process

1. Initialize $\lambda = (A, B, \pi)$ with a best guess.
2. Compute $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i)$, $\xi_t(i, j)$.
3. Re-estimate the model$\lambda = (A, B, \pi)$.
4. If $\mathbb{P}\left[\mathcal{O} \mid \lambda\right]$ increases by at least some predetermined threshold or the predetermined maximum number of iterations has not been exceeded, go to step 2.
5. Else stop and output $\lambda = (A, B, \pi)$.

Overall Path Problem

Find the most likely overall path given the observations (not the states which are individually most likely).

Dynamic Programming algorithm is the forward algorithm with "sum" replaced by "max"

Dynamic Programming

1. Let $\delta_0(i) = \pi_i b_i(\mathcal{O}_0)$, for $i = 0, 1, \ldots N-1$.
2. For $t = 1, 2, \ldots, T-1$ and $i = 0, 1, \ldots N-1$, compute

$$\delta_t(i) = \max_{j \in \{0,\ldots,N-1\}} [\delta_{t-1}(j) a_{ji} b_i(\mathcal{O}_t)]$$

3. Best overall path is

$$\max_{j \in \{0,\ldots,N-1\}} \delta_{T-1}(j).$$

Note about the path

The Dynamic Programming Algorithm only gives the optimal probability, not the path.

Must keep track of preceding states at each stage, trace back from the highest-scoring final state.

Underflow problems

All computations involve multiple products of probabilities, hence tend to 0 quickly at T increases. Therefore, underflow is a serious problem.

Solution is to scale all computations.

Scaled Forward Algorithm

1. 1. Let $\tilde{\alpha}_0(i) = \alpha_0(i)$ for $i = 0, 1, \ldots, N-1$.
   2. Let $c_0 = 1 \Big/ \sum_{j=0}^{N-1} \tilde{\alpha}_0(j)$
   3. Let $\hat{\alpha}(i) = c_0 \tilde{\alpha}_0(i)$ for $i = 0, 1, \ldots, N-1$.
2. For $i = 0, 1, \ldots, N-1$, compute

$$\tilde{\alpha}(i) = \sum_{j=0}^{N-1} \tilde{\alpha}_{t-1}(j) a_{ji} b_i(\mathcal{O}_t)$$

3. Let $c_t = 1 \Big/ \sum_{j=0}^{N-1} \tilde{\alpha}(j)$
4. By induction $\tilde{\alpha}_t(i) = c_0 c_1 \cdots c_t$ Then $\tilde{\alpha}_t(i) = \dfrac{\alpha_t(i)}{\sum_{j=0}^{N-1} \alpha_t(j)}$

Variable Factory with Dynamic Programming 1

$$\delta_0(0) = \pi_0 b_0(\mathcal{O}_0) = (0.8)(0.99) = 0.792$$
$$\delta_0(1) = \pi_1 b_1(\mathcal{O}_0) = (0.2)(0.96) = 0.192$$

Variable Factory with Dynamic Programming 2

$$\delta_1(0) = \max \begin{cases} \delta_0(0)a_{00}b_0(\mathcal{O}_1) & = (0.792)(0.9)(0.01) \\ & = 0.007128 \\ \delta_0(1)a_{10}b_0(\mathcal{O}_1) & = (0.192)(0)(0.01) \\ & = 0.0 \end{cases}$$

$$\delta_1(1) = \max \begin{cases} \delta_0(0)a_{01}b_1(\mathcal{O}_1) & = (0.792)(0.1)(0.04) \\ & = 0.003168 \\ \delta_0(1)a_{11}b_1(\mathcal{O}_1) & = (0.192)(1)(0.04) \\ & = 0.00768 \end{cases}$$

Variable Factory Dynamic Programming 3

$$\delta_2(0) = \max \begin{cases} \delta_1(0)a_{00}b_0(\mathcal{O}_2) & = (0.007128)(0.9)(0.99) \\ & = 0.006351 \\ \delta_1(1)a_{10}b_0(\mathcal{O}_2) & = (0.00768)(0)(0.96) \\ & = 0.0 \end{cases}$$

$$\delta_2(1) = \max \begin{cases} \delta_1(0)a_{01}b_1(\mathcal{O}_2) & = (0.007128)(0.1)(0.99) \\ & = 0.0007057 \\ \delta_1(1)a_{11}b_1(\mathcal{O}_2) & = (0.00768)(1)(0.96) \\ & = 0.007373 \end{cases}$$

Variable Factory Dynamic Programming 4

$\delta_t(i)$	0 (a)	1 (u)	2 (a)
0	0.792	0.007128	0.006351
1	0.192	0.00768	0.007373

Tracing back the source of the red maximum, the maximum overall state sequence was 111, same as exhaustive search.

Remarks on Efficiency

The simple example required only 18 multiplications compared to 40 for the exhaustive listing of all possibilities.

Not surprising, since DP throws out one path of two at every stage.

Similar efficiency for larger problems