COMS30035, Machine learning: Sequential Data 3: EM for HMMs

Edwin Simpson
edwin.simpson@bristol.ac.uk

Department of Computer Science, SCEEM
University of Bristol

November 3, 2020

Agenda

- Markov Models
- Hidden Markov Models
- EM for HMMs
- Linear Dynamical Systems
- Bayesian Timeseries Modelling with Gaussian Processes

Hidden Markov Models (HMMs)

- We want to use maximum likelihood to estimate the HMM parameters:
  1. $\boldsymbol{A}$ - transition matrix
  2. π - initial state probabilities
  3. ϕ - parameters of the emission distributions
- We assume the *unsupervised* case where the sequence of states $\boldsymbol{Z}$ is not observed.
- $\ln p(\boldsymbol{X}|\boldsymbol{A}, \pi, \phi) = \ln \sum_{\boldsymbol{Z}} \left\{ p(\boldsymbol{Z}|\boldsymbol{A}, \pi) \prod_{n=1}^{N} p(\boldsymbol{x}_n|\phi, \boldsymbol{z}_n) \right\}$

Likelihood for an HMM

- As with GMMs, there is no closed-form solution to the MLE, so we turn to EM
- Unlike GMM, the likelihood doesn't factorise over the data points:
  1. $\ln p(\boldsymbol{X}|\boldsymbol{A}, \boldsymbol{\pi}, \phi) = \ln \sum_{\boldsymbol{Z}} \left\{ p(\boldsymbol{Z}|\boldsymbol{A}, \boldsymbol{\pi}) \prod_{n=1}^{N} p(\boldsymbol{x}_n|\phi, \boldsymbol{z}_n) \right\}$
  2. The distribution of z_n depends on z_{n-1}, which also depends on z_{n-2}...
  3. Can't just sum over the values of z_n independently for each data point.
  4. So we have to sum over all K^N possible sequences $\boldsymbol{Z}$!

Expectation Maximisation (EM)

- Goal: maximise the expected log likelihood, $Q(\theta^{old}, \theta) = \sum_{\boldsymbol{Z}} p(\boldsymbol{Z}|\boldsymbol{X}, \theta^{old}) \ln p(\boldsymbol{X}, \boldsymbol{Z}|\theta)$.

1. Initialise the parameters with a random guess: $\theta^{old} = \{\boldsymbol{A}, \boldsymbol{\pi}, \boldsymbol{\phi}\}$.
2. **E-step**: use θ^{old} to compute expectations over $\boldsymbol{Z}$ required to compute $Q(\theta^{old}, \theta)$.
3. **M-step**: choose the values of $\theta = \{\boldsymbol{A}, \boldsymbol{\pi}, \boldsymbol{\phi}\}$ that maximise $Q(\theta^{old}, \theta)$.
4. Set $\theta^{old} = \theta$.
5. Repeat steps 2-4 until convergence.

M step

Looking first at the M-step shows us how to optimise the parameters. From this we can see which expectation terms with respect to the latent states we require, which we will compute during the E-step.

- $\pi_k = \gamma(z_{1k}) / \sum_{j=1}^{K} \gamma(z_{1j})$
- $A_j k = \sum_{n=2}^{N} \xi(z_{n-1,j}, z_n k) / \sum_{l=1}^{K} \sum_{n=2}^{N} \xi(z_{n-1,j}, z_{nl})$
- ϕ_k: parameters of posterior emission distributions, with observations weighted by responsibilities, $\gamma(z_{nk})$
  - If we have Gaussian emissions, the equations are the same as for GMM.
  - Discrete observations with value i:

$$\phi_{ki} = p(x_n = i | z_{nk} = 1) = \frac{\sum_{n=1}^{N} \gamma(z_{nk})[x_n = i]}{\sum_{n=1}^{N} \gamma(z_{nk})} \tag{1}$$

E step

- Responsibilities: $\gamma(z_{nk}) = \sum_{\boldsymbol{Z}} p(\boldsymbol{Z}) z_{nk}$
- State pairs: $\xi(z_{n-1,k}, z_{nk}) = \sum_{\boldsymbol{Z}} p(\boldsymbol{Z}) z_{n-1,k} z_{nk}$
- To compute these efficiently, we need the *forward-backward* algorithm.

Forward-backward Algorithm

- A specific example of the *sum-product algorithm*
- We want to compute:
  - Responsibilities: $\gamma(z_{nk}) = p(z_{nk}|\boldsymbol{A}, \boldsymbol{\pi}, \phi) = \sum_{l=1}^{K} \xi(z_{n-1,l}, z_{nk})$
  - State pairs: $\xi(z_{n-1,l}, z_{nk}) = p(z_{n-1,l}, z_{nk}|\boldsymbol{A}, \boldsymbol{\pi}, \phi)$
- Forward pass computes:

$$\mathbb{E}_{z_1,\ldots,n-2}[p(z_{n-1,l} = 1, z_{nk} = 1|\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n-2}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{A}, \boldsymbol{\pi}, \phi)] \quad (2)$$

- Backward pass computes:

$$\mathbb{E}_{z_1,\ldots,n-2,z_{n+1},\ldots,N}[p(z_{n-1,l} = 1, z_{nk} = 1|\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n-2}, \boldsymbol{z}_{n+1}, \ldots, \boldsymbol{z}_N, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \boldsymbol{A}, \boldsymbol{\pi}, \phi)] \quad (3)$$

Forward-backward Algorithm

- Let's take the final output of the backward pass and break it down: $\mathbb{E}_{z_1,\ldots,n-2,z_{n+1},\ldots,N} p(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n | \boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n-2}, \boldsymbol{z}_{n+1}, \ldots, \boldsymbol{z}_N, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \boldsymbol{A}, \boldsymbol{\pi}, \phi)]$
- We can write it as separate terms for time-steps *before n*, the *current* time-step *n*, and time-steps *after n*.

$$\begin{aligned}
\tilde{\xi}(z_{n-1,l}, z_{nk}) =& p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1}, z_{n-1,l} = 1 | \boldsymbol{A}, \boldsymbol{\pi}, \phi) && \text{before} \\
& p(z_{nk} = 1 | z_{n-1,l} = 1, \boldsymbol{A}) p(\boldsymbol{x}_n | z_{nk} = 1, \phi) && \text{current} \\
& p(\boldsymbol{x}_{n+1}, \ldots, x_N | z_{nk} = 1, \boldsymbol{A}, \phi) && \text{after} \\
=& \alpha(z_{n-1,l}) A_{kl} p(\boldsymbol{x}_n | z_{nk} = 1, \phi) \beta(z_{nk}) && (4)
\end{aligned}$$

- $\xi(z_{n-1,l}, z_{nk}) = \tilde{\xi}(z_{n-1,l}, z_{nk}) / \sum_{l=1}^{K} \sum_{k=1}^{K} \tilde{\xi}(z_{n-1,l}, z_{nk})$

Forward Pass

Compute and save the α terms for all states at all time-steps[1].

$$\begin{aligned}\alpha(z_{nk}) &= p(\boldsymbol{x}_1, ..., \boldsymbol{x}_n, z_{nk} = 1 | \boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\phi}) \\ &= p(\boldsymbol{x}_n | z_{nk} = 1, \boldsymbol{\phi}_k) \sum_{l=1}^{K} A_{lk} \alpha(z_{n-1,l}) \end{aligned} \tag{5}$$

1. Initialise $\alpha(z_{1k}) = \pi_k p(\boldsymbol{x}_1 | z_{1k} = 1, \boldsymbol{\phi}_k)$ for all states k.
2. Compute $\alpha(z_{nk})$ for each n from 2 to N.
3. To avoid $\alpha(z_{nk})$ values becoming increasingly small, normalise over $\sum_{k=1}^{K} \alpha(z_{nk})$ at each iteration.

[1] See Bishop (2006) Section 13.2.2 for full derivation of the algorithm.

Backward Pass

Compute and save the β terms for all states and all time-steps[2].

$$\begin{aligned}\beta(z_{nk}) &= p(\boldsymbol{x}_{n+1}, ..., \boldsymbol{x}_N | z_{nk} = 1, \boldsymbol{A}, \boldsymbol{\phi}) \\ &= \sum_{l=1}^{K} A_{kl} p(\boldsymbol{x}_n | z_{n+1,l} = 1, \boldsymbol{\phi}_l) \beta(z_{n+1,l}) \end{aligned} \tag{6}$$

1. Initialise $\beta(z_{Nk}) = 1$ for all states k.
2. Compute $\beta(z_{nk})$ for each n from $N - 1$ to 1.
3. To avoid $\beta(z_{nk})$ values becoming too small after many iterations, normalise over $\sum_{k=1}^{K} \alpha(z_{n+1,k})$ at each iteration..

[2] See Bishop (2006) Section 13.2.2 for full derivation of the algorithm.

Putting It All Together...

1. Initialise the parameters with a random guess: $\theta^{old} = \{\boldsymbol{A}, \boldsymbol{\pi}, \phi\}$.
2. **E-step** using θ^{old}:
   - 2.1 Run forward pass to compute $\alpha(z_{nk})$
   - 2.2 Run backward pass to compute $\beta(z_{nk})$
   - 2.3 Use $\alpha(z_{n-1,l})$ and $\beta(z_{nk})$ to compute $\xi(z_{n-1,l}, z_{nk})$ and $\gamma(z_{nk})$.
3. **M-step** using $\xi(z_{n-1,l}, z_{nk})$ and $\gamma(z_{nk})$, update $\theta = \{\boldsymbol{\pi}, \boldsymbol{A}, \phi\}$.
4. Set $\theta^{old} = \theta$.
5. Repeat steps 2-4 until convergence.

By summing inside each forward and backward computations, we now have an algorithm that is linear ($\mathcal{O}(N)$) rather than exponential ($\mathcal{O}(K^N)$) in the sequence length 😎.

Predicting the Next Observation

$$p(\boldsymbol{x}_{N+1}|\boldsymbol{x}_1, ..., \boldsymbol{x}_N, \boldsymbol{\theta}) = \sum_{l=1}^{K} \left\{ p(\boldsymbol{x}_{N+1}|z_{N+1,l}, \boldsymbol{\phi}_l) \sum_{k=1}^{K} \gamma(z_{nk}) A_{kl} \right\}$$

Viterbi Algorithm

- Given our estimated model parameters $\theta = \{\pi, \boldsymbol{A}, \phi\}$, how can we predict a sequence of hidden states $\boldsymbol{Z}$?
- Most probable labels (given by the values of $\gamma(z_{nk})$) are not the same as the most probable *sequence*!
- We apply a *max-sum* algorithm called *viterbi* to "decode" the sequence with $\mathcal{O}(N)$ computational cost.

Viterbi Algorithm

- Forward pass: compute the probability of the most likely sequence that leads to each possible state at time n.
- Backward pass: starting with the most likely final state and recursing backwards, choose the previous state $n-1$ that makes the chosen state at n most likely.

Viterbi Algorithm

- Forward pass:
  1. $\omega(\boldsymbol{z}_1) = \ln \pi + \ln p(\boldsymbol{x}|\boldsymbol{z}_1)$
  2. For $n = 2$ to N compute:
     - 2.1 $\omega(\boldsymbol{z}_n) = \max_{\boldsymbol{z}_{n-1}} \{\omega(\boldsymbol{z}_{n-1}) + \ln p(\boldsymbol{z}_n|\boldsymbol{z}_{n-1})\} + \ln p(\boldsymbol{x}_n|\boldsymbol{z}_n)$.
     - 2.2 $\psi(\boldsymbol{z}_n) = \operatorname{argmax}_{\boldsymbol{z}_{n-1}} \{\omega(\boldsymbol{z}_{n-1}) + \ln p(\boldsymbol{z}_n|\boldsymbol{z}_{n-1})\} + \ln p(\boldsymbol{x}_n|\boldsymbol{z}_n)$.
     - 2.3 Passes messages from the start of the sequence to the end.
- Backward pass:
  1. Most likely final state: $\hat{z}_N = \operatorname{argmax}_{k \in \{1,\ldots,K\}} \psi(\boldsymbol{z}_N)_k$.
  2. For $n = N - 1$ to 1: $\hat{z}_n = \psi(\boldsymbol{z}_{n+1})_{\hat{z}_{n+1}}$.
- There are multiple paths leading to each possible state at each step n. We keep only the path with the highest probability, so we don't have to compute the likelihood of every complete path from 1 to N.

Summary

By computing sums and maximums at each timestep we can perform inference over an exponential number of sequences. We use the...

- Forward-backward algorithm, an instance of the more general *sum-product* algorithm to marginalise over sequences of hidden states.
- Viterbi algorithm, an instance of the more general *max-sum* algorithm to find the most likely sequence of hidden states.

Now do the quiz!

Please do the quiz for this lecture on Blackboard.
Next up: linear dynamical systems for modelling continuous states.