Markov and hidden Markov models

Jiali Lin

Virginia Tech

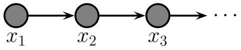
January 15, 2017

Outline

Markov Chains

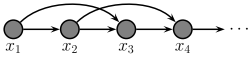
- First order Markov chain

$$p(x_{1:T}) = p(x_1) \prod_{t=2}^{T} p(x_t|x_{t-1})$$



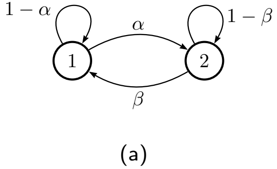
- Second order Markov chain

$$p(x_{1:T}) = p(x_1)p(x_2) \prod_{t=3}^{T} p(x_t|x_{t-1,t-2})$$



State Diagrams

- **State Transition Matrix**: $A_{ij} = p(x_t = j | x_{t-1} = i)$
- State Transition Diagram: One node per discrete state.



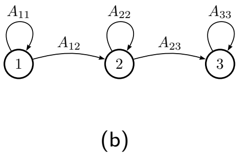


Figure: State transition diagrams for some simple Markov chains. Left: a 2-state chain. Right: a 3-state left-to-right chain.

Hidden Markov Models (HMMs)

$$p(\boldsymbol{x}_{1:T}, \boldsymbol{z}_{1:T}) = p(\boldsymbol{z}_{1:T}) p(\boldsymbol{x}_{1:T} | \boldsymbol{z}_{1:T}) = \left[p(z_1) \prod_{t=2}^{T} p(z_t | z_{t-1}) \right] \left[\prod_{t=1}^{T} p(\boldsymbol{x}_t | z_t) \right]$$

$z_t \rightarrow$ Hidden states taking one of K discrete values.
$x_t \rightarrow$ Observations taking values in space.

- Discrete: M observation symbols

$$p(\boldsymbol{x}_t = l | z_t = k, \boldsymbol{\theta}) = B(k, l)$$

- Continuous Gaussian:

$$p(\boldsymbol{x}_t | z_t = k, \boldsymbol{\theta}) = N(\boldsymbol{x}_t | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Outline

Types of inference problems for temporal models

There are different kinds of inference, which we summarize below.

- **Filtering:** computes the **belief state** $p(z_t|x_{1:t})$ online, or recursively, as the data streams in.
- **Smoothing**: computes $p(z_t|x_{1:T})$ offline, given all the evidence.
- **Fixed lag smoothing** computes $p(z_{t-l}|x_{1:t})$, where $l > 0$ is **lag**.
- **Prediction** computes $p(z_{t+h}|x_{1:t})$, where $h > 0$ is called the **prediction horizon**.
- **MAP estimation** computes arg$\max_{z_{1:T}} p(z_{1:T}|x_{1:T})$, which is a most probable state sequence (also known as **Viterbi decoding**).
- **Posterior samples** $z_{1:T} \sim p(z_{1:T}|x_{1:T})$.
- **Probability of the evidence** computes the probability of the evidence, $p(x_{1:T}) = \sum_{z1:T} p(z_{1:T}, x_{1:T})$.

The main kinds of inference for state-space models

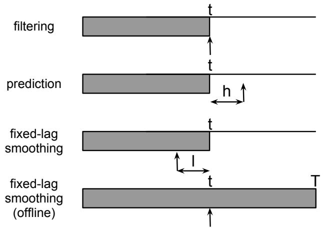


Figure: The main kinds of inference for state-space models. The shaded region is the interval for which we have data. The arrow represents the time step at which we want to perform inference. t is the current time, T is the sequence length, l is the lag and h is the prediction horizon.

Example: The occasionally dishonest casino

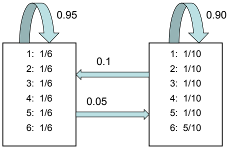
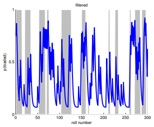


Figure: An HMM for the occasionally dishonest casino. The blue arrows visualize the state transition diagram A. Based on (Durbin et al. 1998, p54).

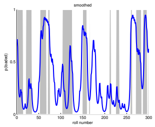
$x_t \in \{1, 2, ..., 6\}$ represents which dice face shows up, and z_t represents the identity of the dice that is being used. Most of the time the casino uses a fair dice, $z = 1$, but occasionally it switches to a loaded dice, $z = 2$, for a short period.

- Roll: 664153216162115234653214356634261655234232315142464156.
- Dice: LLLLLLLLLLLLLLFFFFFFLLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFL.

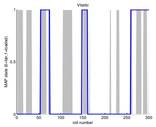
Filtering, smoothing, MAP



(a)



(b)

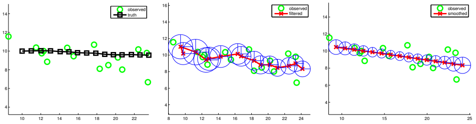


(c)

Figure: Inference in the dishonest casino. Vertical gray bars denote the samples that we generated using a loaded die. (a) Filtered estimate of probability of using a loaded dice. (b) Smoothed estimates. (c) MAP trajectory. Figure generated by `CasinoDemo`.

- If we threshold the estimates at 0.5 and compare to the true sequence, we find that the filtered method makes 71 errors out of 300, the smoothed method makes 49/300, and the MAP path makes 60/300 errors.
- It is not surprising that smoothing makes fewer errors than Viterbi, since the optimal way to minimize bit-error rate is to threshold the posterior marginals.
- Nevertheless, for some applications, we may prefer the Viterbi decoding, to enforce consistency.

Kalman filtering and smoothing



(a) (b) (c)

Figure: Illustration of Kalman filtering and smoothing. (a) Observations (green cirles) are generated by an object moving to the right (true location denoted by black squares). (b) Filtered estimated is shown by dotted red line. Red cross is the posterior mean, blue circles are 95% confidence ellipses derived from the posterior covariance. For clarity, we only plot the ellipses every other time step. (c) Same as (b), but using offline Kalman smoothing. Figure generated by `kalmanTrackingDemo`.

- Suppose $p(z, x)$ is a linear dynamical system subject to Gaussian noise, where $z \in \mathbb{R}^K$ and $x \in \mathbb{R}^D$.
- The analog of forwards-backwards is known as **Kalman filtering/ smoothing**.

The forwards algorithm

We now describe how to recursively compute the filtered marginals, $p(z_t|x_{1:t})$ in an HMM.

- We just apply Bayes rule sequentially to recursively update the belief state:

$$\begin{aligned}\alpha_t(j) = p(z_t = j|\boldsymbol{x}_{1:t}) &= p(z_t = j|\boldsymbol{x}_t, \boldsymbol{x}_{1:t-1}) \\ &\propto\ p(\boldsymbol{x}_t|z_t = j, \cancel{\boldsymbol{x}_{1:t-1}})p(z_t = j|\boldsymbol{x}_{1:t-1})\end{aligned}$$

  where $p(\boldsymbol{x}_t|z_t = j)$ is the likelihood and the prior is the one-step-ahead predictive density

$$p(z_t = j|\boldsymbol{x}_{1:t-1}) = \sum_i p(z_t = j|z_{t-1} = i)\alpha_{t-1}(i)$$

- This is called the **predict-update cycle**.
- In matrix-vector notation, we can write the update in the following simple form:

$$\boldsymbol{\alpha}_t\ \propto\ \boldsymbol{\psi}_t \odot (\boldsymbol{\Psi}^T \boldsymbol{\alpha}_{t-1})$$

- The base case is $\boldsymbol{\alpha}_1\ \propto\ \boldsymbol{\psi}_1 \odot \boldsymbol{\pi}$.

The backwards algorithm

- The key decomposition relies on the fact that we can break the chain into two parts, the past and the future, by conditioning on z_t:

$$\begin{aligned} p(z_t = j | \boldsymbol{x}_{1:T}) &\propto p(z_t = j, \boldsymbol{x}_{t+1:T} | \boldsymbol{x}_{1:t}) \\ &\propto p(z_t = j | x_{1:t}) p(x_{t+1:T} | z_t = j, \cancel{x_{1:t}}) \end{aligned}$$

  We define

$$\begin{aligned} \beta_t(j) &= p(\boldsymbol{x}_{t+1:T} | z_t = j) \\ \gamma_t(j) &= p(z_t = j | x_{1:T}) \end{aligned}$$

  From above equation, we have

$$\gamma_t(j) = \alpha_t(j)\beta_t(j)$$

- We now derive a recursive update for β_t. (It is also possible to directly update γ_t.)

The backwards algorithm

- Working right to left, we have

$$\begin{aligned}\beta_{t-1}(i) &= p(\boldsymbol{x}_{t:T}|z_{t-1}=i)\\ &= \sum_j p(z_t=j,\boldsymbol{x}_t,\boldsymbol{x}_{t+1:T}|z_{t-1}=i)\\ &= \sum_j p(\boldsymbol{x}_{t+1:T}|z_t=j,\cancel{\boldsymbol{x}_t},\cancel{z_{t-1}=i})p(z_t=j,\boldsymbol{x}_t|z_{t-1}=i)\\ &= \sum_j p(\boldsymbol{x}_{t+1:T}|z_t=j)p(\boldsymbol{x}_t|z_t=j,\cancel{z_{t-1}=j})p(z_t=j,\boldsymbol{x}_t|z_{t-1}=i)\\ &= \sum_j \beta_t(j)\phi_t(j)\phi(i,j)\end{aligned}$$

- We can write the resulting equation in matrix-vector form as

$$\boldsymbol{\beta}_{t-1} = \boldsymbol{\Psi}(\boldsymbol{\psi}_t \odot \boldsymbol{\beta}_t)$$

- The base case is

$$\beta_T(i) = p(\boldsymbol{x}_{T+1:T}|z_T=i) = p(\emptyset|z_T=i) = 1$$

which is the probability of a non-event.

The Viterbi algorithm

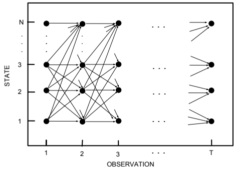


Figure: The trellis of states vs time for a Markov chain. Based on (Rabiner 1989).

- The Viterbi algorithm can be used to compute the most probable sequence of states in a chain-structured graphical model, i.e.,

$$\boldsymbol{z}^* = \operatorname{argmax} p(\boldsymbol{z}_{1:T}|\boldsymbol{x}_{1:T})$$

- This is equivalent to computing a shortest path through the **trellis diagram** where the nodes are possible states at each time step, and the node and edge weights are log probabilities. That is, the weight of a path $z_1, z_2, ..., z_T$ is given by

$$\log \pi_1(z_1) + \log \phi_1(z_1) + \sum_{t=2}^{T} [\log \phi(z_{t-1}, z_t) + \log \phi_t(z_t)]$$

- We already have

$$p(\boldsymbol{x}_{1:T}, \boldsymbol{z}_{1:T}) = p(\boldsymbol{z}_{1:T})p(\boldsymbol{x}_{1:T}|\boldsymbol{z}_{1:T}) = \left[p(z_1)\prod_{t=2}^{T} p(z_t|z_{t-1})\right]\left[\prod_{t=1}^{T} p(\boldsymbol{x}_t|z_t)\right]$$

$$\hat{z} = \operatorname*{argmax}_z p(z|x) = \operatorname*{argmax}_z = p(z,x)$$

- Equivalent to minimizing negative log-probability:

$$\begin{aligned} \phi_1(z_1) &= -\log p(z_1) - \log p(x_1|z_1) \\ \phi_t(z_t) &= -\log p(x_t|z_t) \\ \psi_t(z_{t-1}, z_t) &= -\log p(z_t|z_{t-1}) \end{aligned}$$

$$\hat{z} = \operatorname*{argmin}_z \phi_1(z_1) + \sum_{t=2}^{T}[\psi_t(z_{t-1}, z_t) + \phi_t(z_t)]$$

MAP vs MPM

- The (jointly) most probable sequence of states is not necessarily the same as the sequence of (marginally) most probable states.
- In Viterbi, when we estimate z_t, we "max out" the other variables:

$$z_t^* = \text{arg}\max_{z_t} \max_{\boldsymbol{z}_{1:t-1},\boldsymbol{z}_{t+1:T}} p(\boldsymbol{z}_{1:t-1}, z_t, \boldsymbol{z}_{t+1:T}|\boldsymbol{x}_{1:T})$$

  whereas we when we use forwards-backwards, we sum out the other variables to get a maximum posterior marginal (MPM):

$$\hat{z}_t = \operatorname*{argmax}_{z_t} \max_{\boldsymbol{z}_{1:t-1},\boldsymbol{z}_{t+1:T}} p(\boldsymbol{z}_{1:t-1}, z_t, \boldsymbol{z}_{t+1:T}|\boldsymbol{x}_{1:T})$$

- The sequence of MPMs may not be equal to the joint MAP sequence, e.g., "wreck a nice beach" might be the MPM interpretation of a series of acoustic waveforms, but the MAP sequence is more likely to be "recognize speech".

Details of the algorithm

We implement Viterbi by replacing the sum-operator in forwards-backwards with a max-operator. The former is called the **sum-product**, and the latter the **max-product** algorithm.

- Define the probability of ending up in state j at time t, given that we take the most probable path:

$$\delta_t(j) = \max_{z_1,\dots,z_{t-1}} p(\boldsymbol{z}_{1:t-1}, z_t = j | \boldsymbol{x}_{1:t})$$

- The most probable path to state j at time t must consist of the most probable path to some other state i at time $t-1$, followed by a transition from i to j.

$$\delta_t(j) = \max_i \delta_{t-1}(i)\psi(i,j)\phi_t(j)$$

- We also keep track of the most likely previous state, for each possible state that we end up in:

$$a_t(j) = \text{arg}\max \delta_{t-1}(i)\psi(i,j)\psi_t(j)$$

  $a_t(j)$ tells us the most likely previous state on the most probable path to $z_t = j$. initialize by setting

- We initialize by setting

$$\delta_1(j) = \pi_j \phi_1(j)$$

- and we terminate by computing the most probable final state z_T:

$$z_T = \operatorname*{argmax}_i \delta_T(i)$$

- We can then compute the most probable sequence of states using **traceback**:

$$z_t^* = a_{t+1}(z_{t+1}^*)$$

Outline

Learning for HMMs

We now discuss how to estimate the parameters $\boldsymbol{\theta} = (\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$, where

- $\pi(i) = p(z_1 = i)$ is the initial state distribution.
- $A(i, j) = p(z_t = j | z_{t-1} = i)$ is the transition matrix.
- $\boldsymbol{B}$ are the parameters of the class-conditional densities $p(\boldsymbol{x}_t | z_t = j)$.

Training with fully observed data

- If we observe the hidden state sequences, we can compute the MLEs for $\boldsymbol{A}$ and $\boldsymbol{\pi}$.
- The details on how to estimate $\boldsymbol{B}$ depend on the form of the observation model.
- For example, if each state has a multinoulli distribution with parameters $B_{jl} = p(X_t = l | z_t = j)$

$$\hat{B}_{jl} = \frac{N_{jl}^X}{N_j}, \quad N_{jl}^X = \sum_{i=1}^{N} \sum_{t=1}^{T_i} \mathbb{I}(z_{i,t} = j, x_{i,t} = l)$$

- This result is quite intuitive: we simply add up the number of times we are in state j and we see a symbol l, and divide by the number of times we are in state j.

Training with unobserved data

If the z_t variables are not observed, we apply the EM algorithm to HMMs, also known as the **Baum-Welch** algorithm.

- E step: the expected complete data log likelihood

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \sum_{k=1}^{K} E[N_k^1] \log \pi_k + \sum_{j=1}^{K} \sum_{k=1}^{K} E[N_{jk}] \log A_{jk}$$
$$+ \sum_{i=1}^{T} \sum_{t=1}^{T_i} \sum_{k=1}^{K} p(z_t = k | \boldsymbol{x}_i, \boldsymbol{\theta}^{\text{old}}) \log p(\boldsymbol{x}_{i,t} | \boldsymbol{\phi}_k)$$

where the expected counts are given by

$$E[N_k^1] = \sum_{i=1}^{N} p(z_{i1} = k | \boldsymbol{x}_i, \boldsymbol{\theta}^{\text{old}})$$
$$E[N_{jk}] = \sum_{i=1}^{T} \sum_{t=2}^{T_i} p(z_{i,t-1} = j, z_{i,t} = k | \boldsymbol{x}_i, \boldsymbol{\theta}^{\text{old}})$$
$$E[N_j] = \sum_{i=1}^{T} \sum_{t=1}^{T_i} p(z_{i,t} = j | \boldsymbol{x}_i, \boldsymbol{\theta}^{\text{old}})$$

Training with unobserved data (cont'd)

- M step for $\boldsymbol{A}$ and π is to just normalize the expected counts:

$$\hat{A}_{jk} = \frac{E[N_{jk}]}{\sum_{k'} E[N_{jk'}]}, \ \hat{\pi}_k = \frac{E[N_k^1]}{N}$$

- This result is quite intuitive: we simply add up the expected number of transitions from j to k, and divide by the expected number of times we transition from j to anything else.