HMM

Sep 2022

Murphy chap17

Introduction

Types of inference problems for temporal models

There are several different kinds of inferential tasks for an HMM (and SSM in general). To illustrate the differences, we will consider an example called the **occasionally dishonest casino**, from (Durbin et al. 1998). In this model, $x_t \in \{1, 2, \ldots, 6\}$ 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. If $z = 1$ the observation distribution is a uniform multinoulli over the symbols $\{1, \ldots, 6\}$. If $z = 2$, the observation distribution is skewed towards face 6 (see Figure 17.9). If we sample from this model, we may observe data such as the following:

Listing 17.1 Example output of `casinoDemo`

```
Rolls:    664153216162115234653214356634261655234232315142464156663246
Die:      LLLLLLLLLLLLLLFFFFFFLLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFLLLLLLLL
```

Introduction

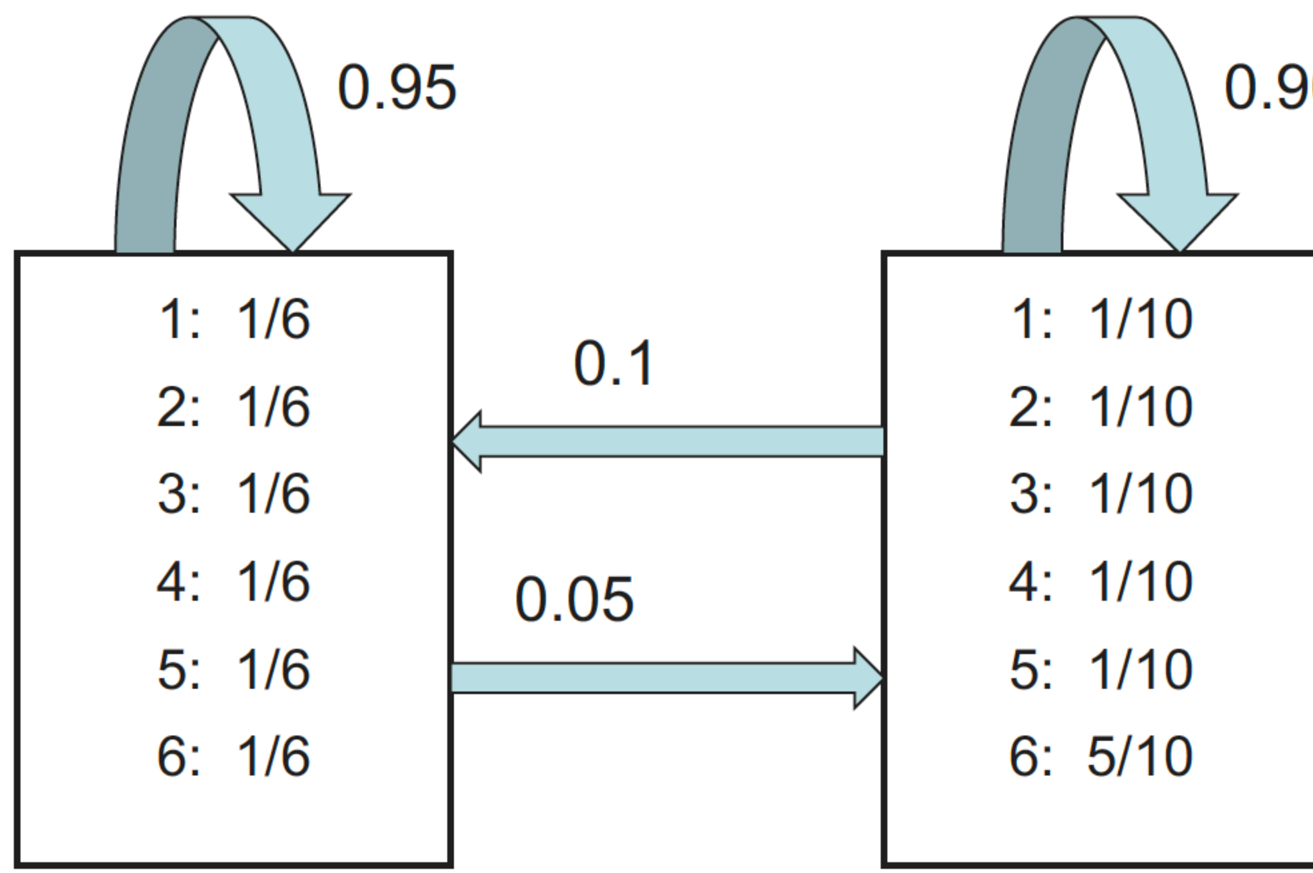
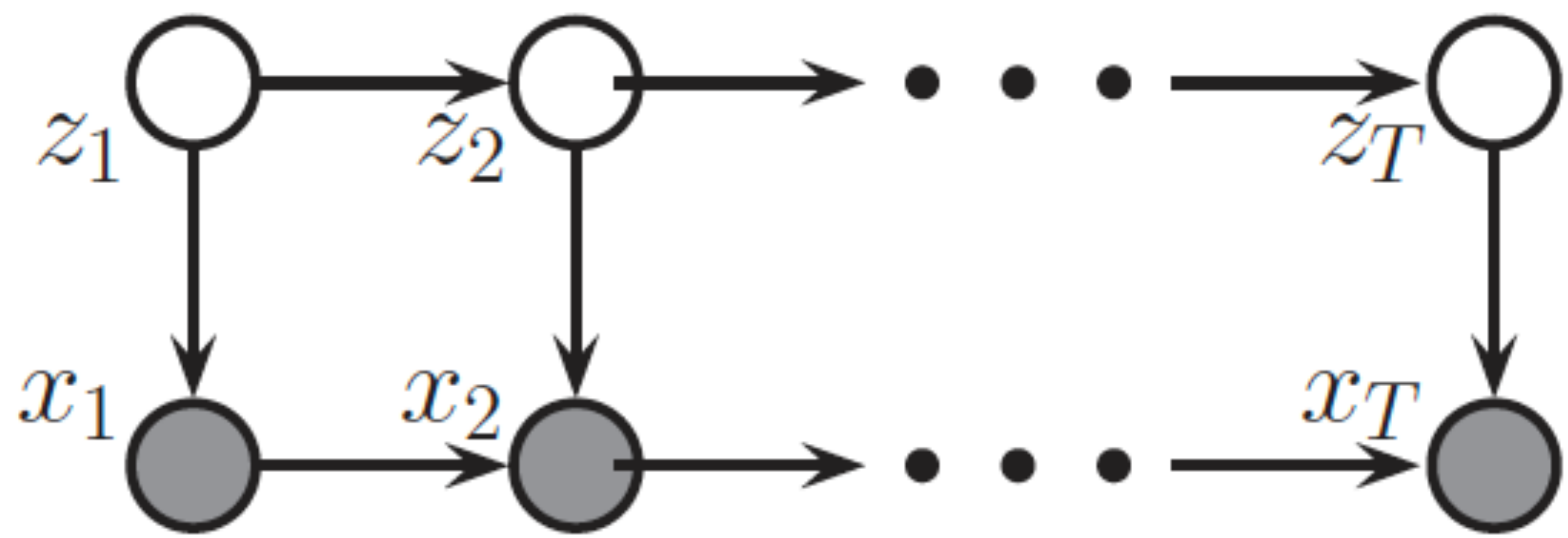


Figure 17.9 An HMM for the occasionally dishonest casino. The blue arrows visualize the state transition diagram **A**. Based on (Durbin et al. 1998, p54).

Introduction



Introduction

Hidden Markov models

As we mentioned in Section 10.2.2, a **hidden Markov model** or **HMM** consists of a discrete-time, discrete-state Markov chain, with hidden states $z_t \in \{1, \ldots, K\}$, plus an **observation** model

$p(\mathbf{x}_t|z_t)$. The corresponding joint distribution has the form

$$p(\mathbf{z}_{1:T}, \mathbf{x}_{1:T}) = p(\mathbf{z}_{1:T})p(\mathbf{x}_{1:T}|\mathbf{z}_{1:T}) = \left[p(z_1)\prod_{t=2}^{T} p(z_t|z_{t-1})\right]\left[\prod_{t=1}^{T} p(\mathbf{x}_t|z_t)\right] \tag{17.39}$$

The observations in an HMM can be discrete or continuous. If they are discrete, it is common for the observation model to be an observation matrix:

$$p(\mathbf{x}_t = l|z_t = k, \boldsymbol{\theta}) = B(k, l) \tag{17.40}$$

If the observations are continuous, it is common for the observation model to be a conditional Gaussian:

$$p(\mathbf{x}_t|z_t = k, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_t|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{17.41}$$

Introduction

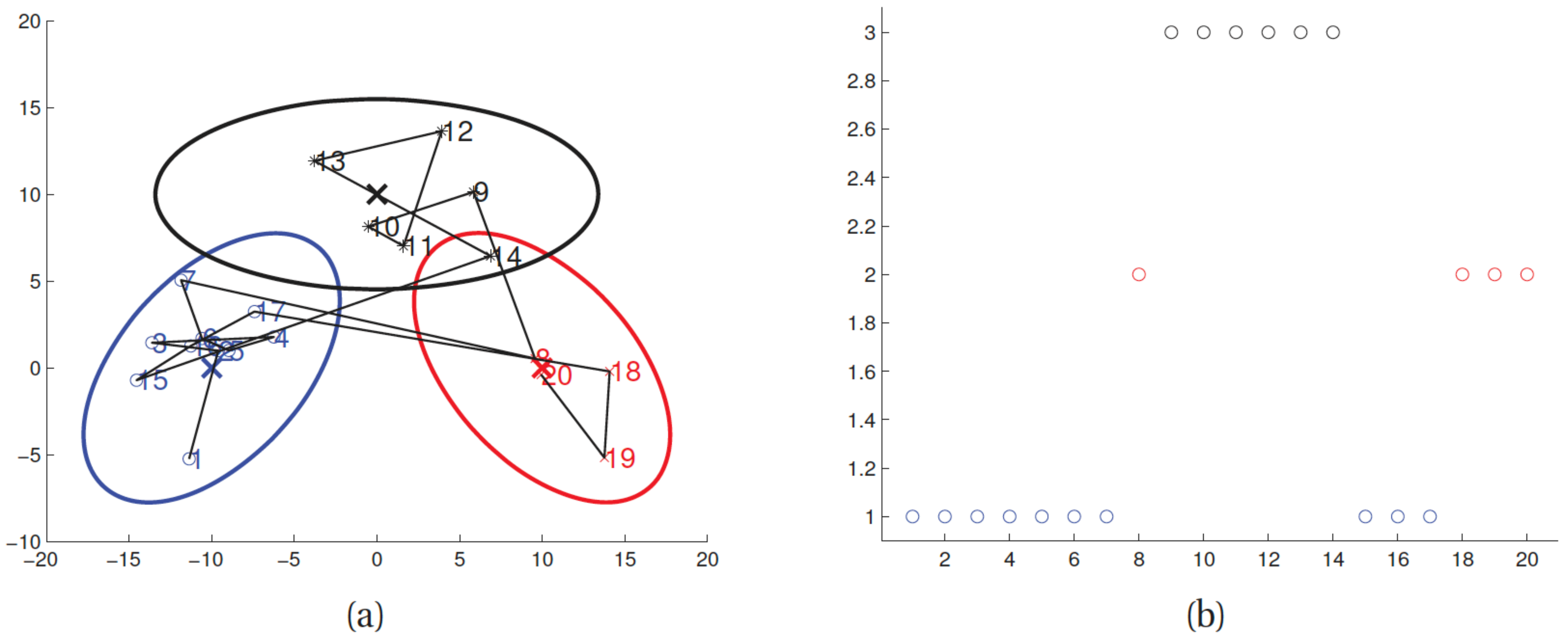


Figure 17.7 (a) Some 2d data sampled from a 3 state HMM. Each state emits from a 2d Gaussian. (b) The hidden state sequence. Based on Figure 13.8 of (Bishop 2006b). Figure generated by `hmmLillypadDemo`.

Introduction

- **Filtering** means to compute the **belief state** $p(z_t|\mathbf{x}_{1:t})$ online, or recursively, as the data streams in. This is called "filtering" because it reduces the noise more than simply estimating the hidden state using just the current estimate, $p(z_t|\mathbf{x}_t)$. We will see below that we can perform filtering by simply applying Bayes rule in a sequential fashion. See Figure 17.10(a) for an example.

- **Smoothing** means to compute $p(z_t|\mathbf{x}_{1:T})$ offline, given all the evidence. See Figure 17.10(b) for an example. By conditioning on past and future data, our uncertainty will be significantly reduced. To understand this intuitively, consider a detective trying to figure out who committed a crime. As he moves through the crime scene, his uncertainty is high until he finds the key clue; then he has an "aha" moment, his uncertainty is reduced, and all the previously confusing observations are, in **hindsight**, easy to explain.

- **Fixed lag smoothing** is an interesting compromise between online and offline estimation; it involves computing $p(z_{t-\ell}|\mathbf{x}_{1:t})$, where $\ell > 0$ is called the lag. This gives better performance than filtering, but incurs a slight delay. By changing the size of the lag, one can trade off accuracy vs delay.

Introduction

- **Prediction** Instead of predicting the past given the future, as in fixed lag smoothing, we might want to predict the future given the past, i.e., to compute $p(z_{t+h}|\mathbf{x}_{1:t})$, where $h > 0$ is called the prediction **horizon**. For example, suppose $h = 2$; then we have

$$p(z_{t+2}|\mathbf{x}_{1:t}) = \sum_{z_{t+1}} \sum_{z_t} p(z_{t+2}|z_{t+1})p(z_{t+1}|z_t)p(z_t|\mathbf{x}_{1:t}) \tag{17.42}$$

It is straightforward to perform this computation: we just power up the transition matrix and apply it to the current belief state. The quantity $p(z_{t+h}|\mathbf{x}_{1:t})$ is a prediction about future hidden states; it can be converted into a prediction about future observations using

$$p(\mathbf{x}_{t+h}|\mathbf{x}_{1:t}) = \sum_{z_{t+h}} p(\mathbf{x}_{t+h}|z_{t+h})p(z_{t+h}|\mathbf{x}_{1:t}) \tag{17.43}$$

This is the posterior predictive density, and can be used for time-series forecasting (see (Fraser 2008) for details). See Figure 17.11 for a sketch of the relationship between filtering, smoothing, and prediction.

Introduction

- **MAP estimation** This means computing $\arg\max_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T}|\mathbf{x}_{1:T})$, which is a most probable state sequence. In the context of HMMs, this is known as **Viterbi decoding** (see

- **Posterior samples** If there is more than one plausible interpretation of the data, it can be useful to sample from the posterior, $\mathbf{z}_{1:T} \sim p(\mathbf{z}_{1:T}|\mathbf{x}_{1:T})$. These sample paths contain much more information than the sequence of marginals computed by smoothing.

- **Probability of the evidence** We can compute the **probability of the evidence**, $p(\mathbf{x}_{1:T})$, by summing up over all hidden paths, $p(\mathbf{x}_{1:T}) = \sum_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T}, \mathbf{x}_{1:T})$. This can be used to classify sequences (e.g., if the HMM is used as a class conditional density), for model-based clustering, for anomaly detection, etc.

Assumptions

$$o_t \perp \{o_1, ..., o_{t-1}, o_{t+1}, ..., o_\tau, s_1, ..., s_{t-1}, s_{t+1}, ..., s_\tau\} | s_t.$$

$$s_{t+1} \perp \{o_1, ..., o_t\} | s_t.$$

Forward algorithm

$$
\begin{aligned}
&P(o_1, ..., o_t, s_t = x_j) \\
&= P(o_t | o_1, ..., o_{t-1}, s_t = x_j) P(o_1, ..., o_{t-1}, s_\tau = x_j) \\
&= P(o_t | s_t = x_j) \sum_i P(o_1, ..., o_{t-1}, s_{t-1} = x_i, s_t = x_j) \\
&= P(o_t | s_t = x_j) \sum_i P(s_t = x_j | o_1, ..., o_{t-1}, s_{t-1} = x_i) P(o_1, ..., o_{t-1}, s_{t-1} = x_i) \\
&= P(o_t | s_t = x_j) \sum_i P(s_t = x_j | s_{t-1} = x_i) P(o_1, ..., o_{t-1}, s_{t-1} = x_i).
\end{aligned}
$$

$$p_t(i) = P(o_1, ..., o_t, s_t = x_i), \ \alpha_{ij} = P(s_t = x_j | s_{t-1} = x_i), \ \beta_j(o_t) = P(o_t | s_t = x_j),$$

$$p_t(j) = \left(\sum_i p_{t-1}(i) \alpha_{ij} \right) \beta_j(o_t)$$

$$P(o_1, ..., o_t) = \sum_j p_t(j).$$

Forward algorithm

$$
\begin{aligned}
P(o_1, ..., o_t) &= \sum_s P(o_1, ..., o_t, s_t = s) \\
&= \sum_s P(o_t | o_1, ..., o_{t-1}, s_t) P(s_t | o_1, ..., o_{t-1}) P(o_1, ..., o_{t-1}) \\
&= \sum_s P(o_t | s_t) P(s_t) P(o_1, ..., o_{t-1}) = \left(\sum_s P(o_t, s_t) \right) P(o_1, ..., o_{t-1}),
\end{aligned}
$$

Why?

Backward algorithm

$$
\begin{aligned}
&P(o_{t+1}, ..., o_\tau | s_t = x_i) \\
&= \sum_j P(o_{t+1}, ..., o_\tau, s_{t+1} = x_j | s_t = x_i) \\
&= \sum_j P(o_{t+2}, ..., o_\tau | o_{t+1}, s_{t+1} = x_j, s_t = x_i) P(o_{t+1}, s_{t+1} = x_j | s_t = x_i) \\
&= \sum_j P(o_{t+2}, ..., o_\tau | s_{t+1} = x_j) P(o_{t+1} | s_{t+1} = x_j) P(s_{t+1} = x_j | s_t = x_i).
\end{aligned}
$$

$$q_t(i) = P(o_{t+1}, ..., o_\tau | s_t = x_i),\ \ q_\tau(i) = 1,$$

$$q_t(i) = \sum_j \alpha_{ij} \beta_j(o_{t+1}) q_{t+1}(j).$$

$$P(o_1, ..., o_\tau) = \sum_i q_0(i) P(S_0 = x_i).$$

Viterbi algorithm