Dynamic and temporal models

ISTA 410 / INFO 510: Bayesian Modeling and Inference

U. of Arizona School of Information

November 18, 2020

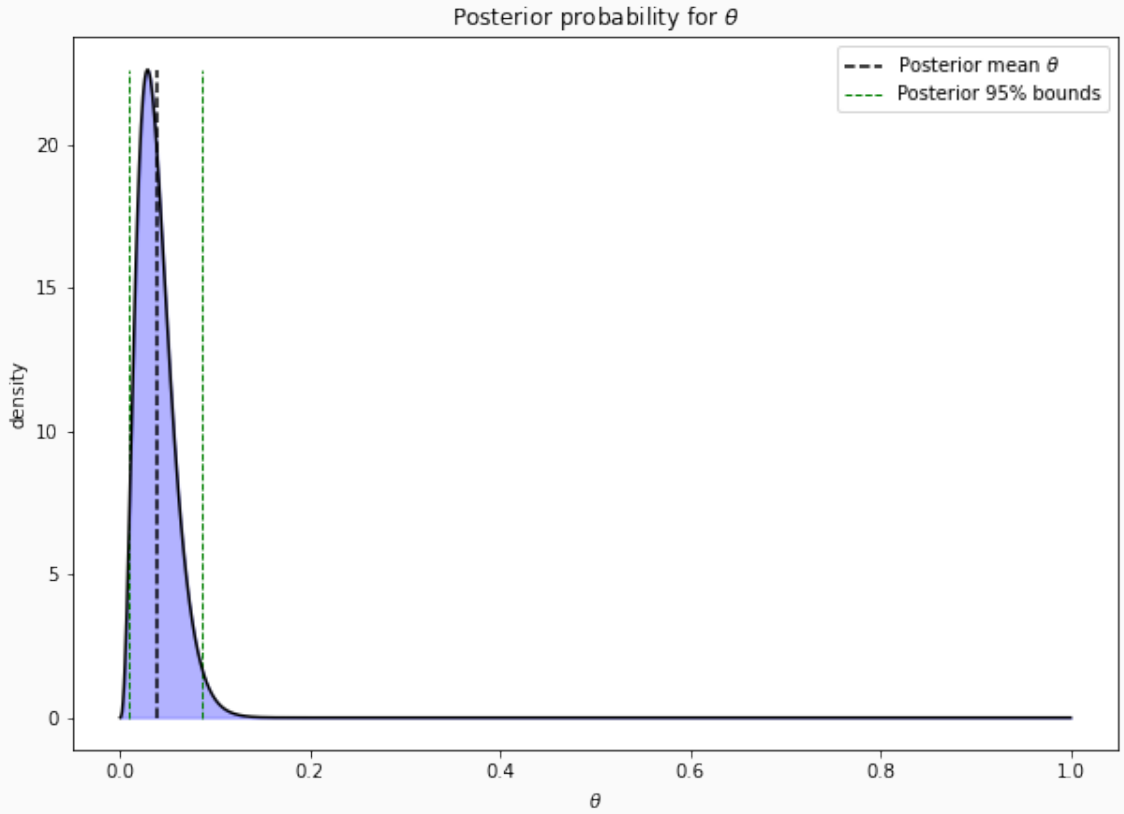
Beta-binomial model in the news

Before we go on: recent news!

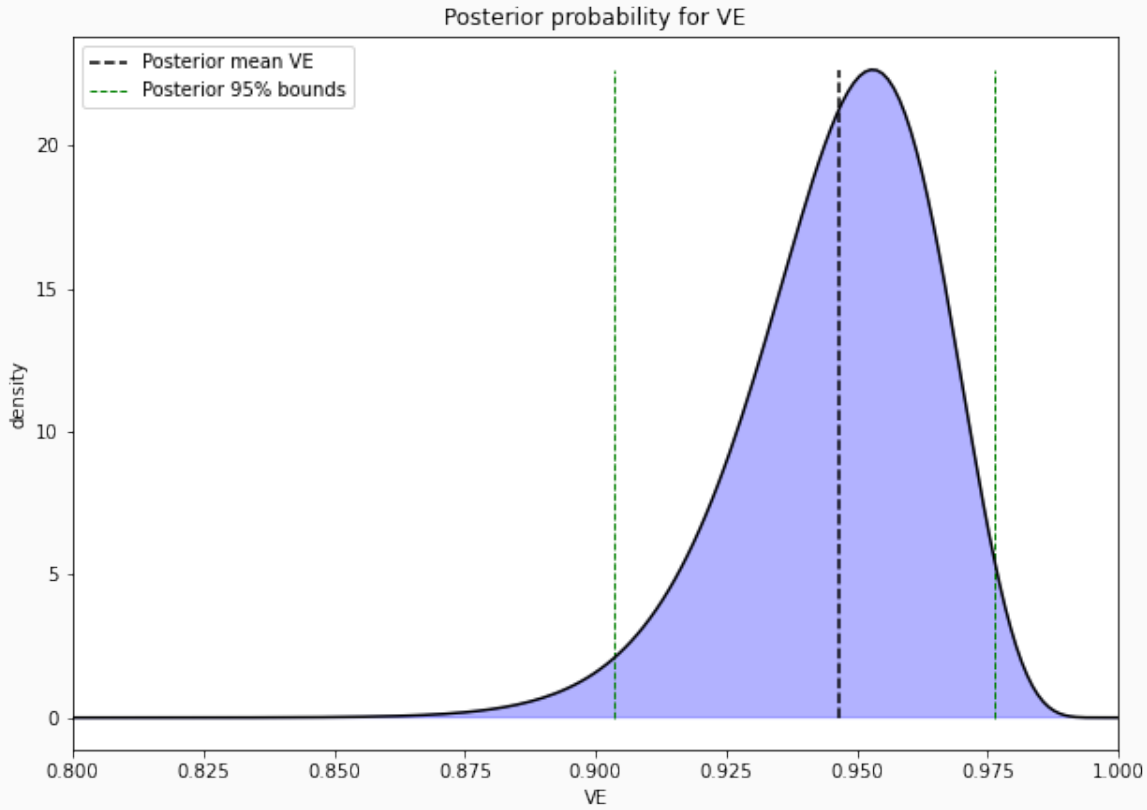
Update:

- New data from Pfizer
- 162 cases in placebo, 8 in vaccine
- Seems like the second reverse engineering from last time (91 cases in placebo, 3 in vaccine) was correct

Posterior distribution



Posterior distribution



Outline

Last time:

- Current events interlude
- Brief overview of plate models

Today:

- Overview of temporal models
- Intro to hidden Markov models

Temporal and dynamical models

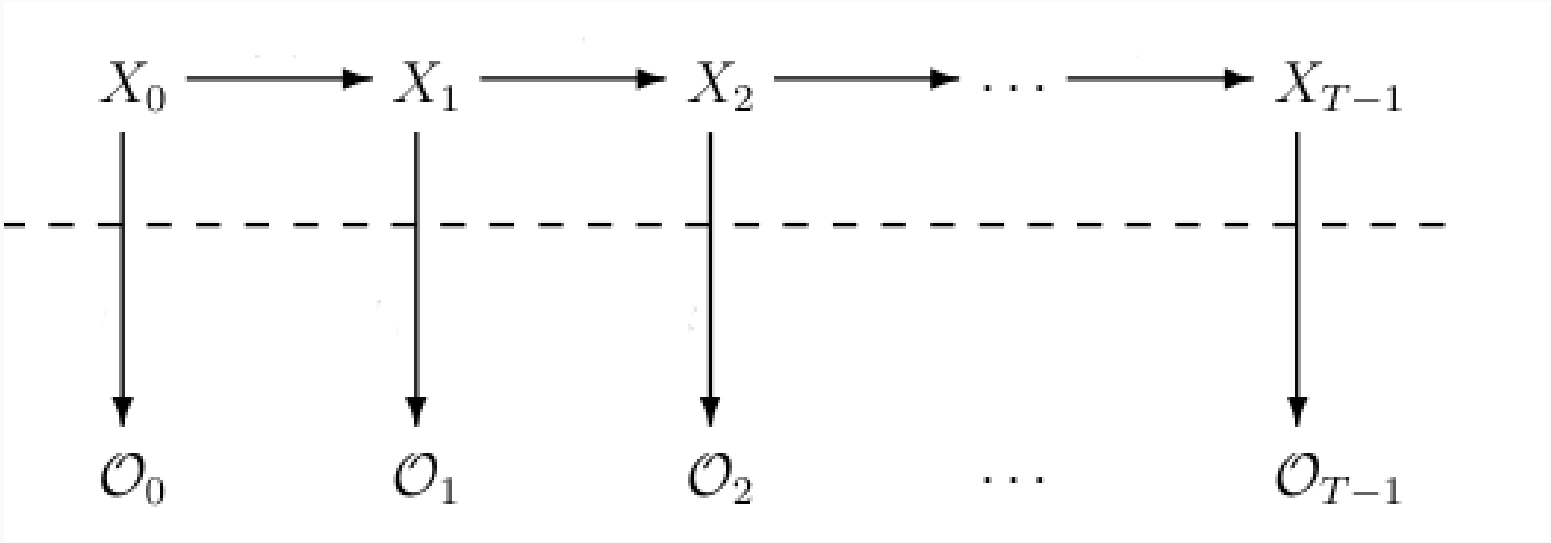
Temporal and dynamical models

The models we'll look at next model are used to model sequential data, especially time series.

These models are intended to be usable in an on-line fashion – that is, incorporating data in real time as it comes.

Hidden state models

We're going to focus on hidden state models, which have a general structure similar to below:



- Latent/hidden system state
- Observations based on system state

Two state-observation models

Two common models:

- Hidden Markov model
  Hidden states evolve according to a Markov process
  Observations typically Gaussian or multinomial
- Linear Gaussian dynamical system
  States evolve according to linear dynamics
  Noise is multivariate normal

Typical inference problems

Typical problems we want to solve, given a sequence of observations $\mathcal{O}$ of time length T:

- Filtering: find the distribution of X_T – that is, the distribution of the current state, accounting for all observations up to now.
- Prediction: find the distribution of X_t for some $t > T$.
- Smoothing: find the distribution of X_t for some $1 \leq t < T$. This looks very similar to filtering, but differs in that we can take the observations after time t into account.
- MAP or best-explanation: find the sequence (X_i) maximizing $P(\mathcal{O}, X)$.
- Fitting: Given a sequence of observations, find the model parameters that maximize $P(\mathcal{O}|\lambda)$.

Hidden Markov models

Simplest case

In a HMM, the underlying states are governed by a Markov process.

The simplest case, which we'll start with, is a finite state, multinomial HMM:

- Underlying state X_t follows a Markov chain with N states
- Observed values $\mathcal{O}_t$ follow a multinomial distribution conditional on X_t

So the model is described by two matrices, A (transition matrix), and B (observation matrix).

To do calculations, we also need to assume a certain distribution π on the initial state X_1. As a shorthand, I'll use the notation $\lambda = (A, B, \pi)$ to represent a choice of these parameters.

Example

Suppose we want to determine the average annual temperature for a particular location over a series of years in the distant past.

For simplicity, we will group years into "hot" and "cool" years, denoted H and C, and assume that the hot/cool states are determined by a Markov process with transition matrix:

$$A = \begin{pmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{pmatrix}$$

Classifying tree rings as "small", "medium", or "large", modern evidence suggests that hot and cool years produce different sized rings with the following probabilities:

$$B = \begin{pmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{pmatrix}$$

Three problems to solve

Today we'll focus on two main calculations:

1. Given a multinomial HMM λ and a sequence of observations $\mathcal{O}$, compute the distribution $P(X_t|\lambda, \mathcal{O})$. (This is the *filtering* task from before.)
2. Given a multinomial HMM λ and a sequence of observations $\mathcal{O}$, compute the probability distribution of X_t for some $1 \leq t \leq T$. (This is the *smoothing* task from before.)

Naïve filtering

It is clear that we can compute the joint probability of a particular sequence of states:

$$P(X, \mathcal{O}|\lambda) = \pi_{X_1} \prod_{t=1}^{T} A_{X_{t-1},X_t} B_{X_t,\mathcal{O}_t}$$

So, naïvely, we could compute this for all sequences of states, and then

$$P(X_t = x_i) = \sum_{\text{sequences with } X_t = x_i} P(X, \mathcal{O}|\lambda)$$

What's the problem?

Naïve filtering

It is clear that we can compute the joint probability of a particular sequence of states:

$$P(X, \mathcal{O}|\lambda) = \pi_{X_1} \prod_{t=1}^{T} A_{X_{t-1},X_t} B_{X_t,\mathcal{O}_t}$$

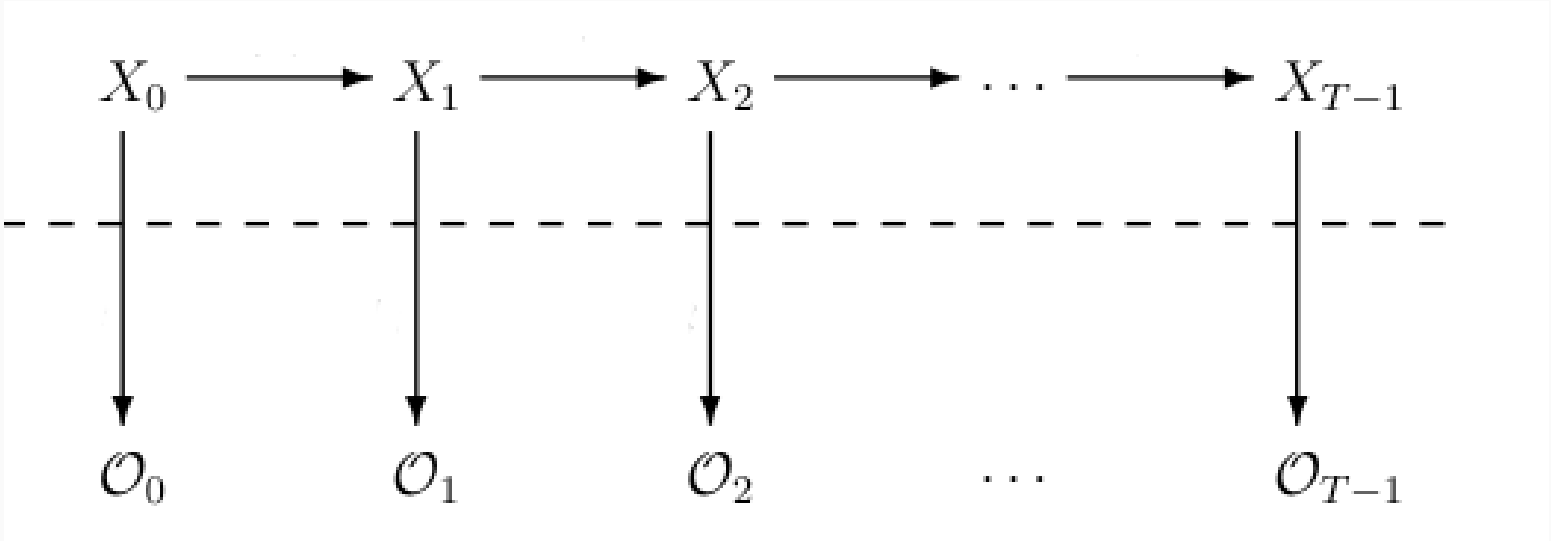
So, naïvely, we could compute this for all sequences of states, and then

$$P(X_t = x_i) = \sum_{\text{sequences with } X_t = x_i} P(X, \mathcal{O}|\lambda)$$

What's the problem? N^T sequences – computationally infeasible for all but short sequences.

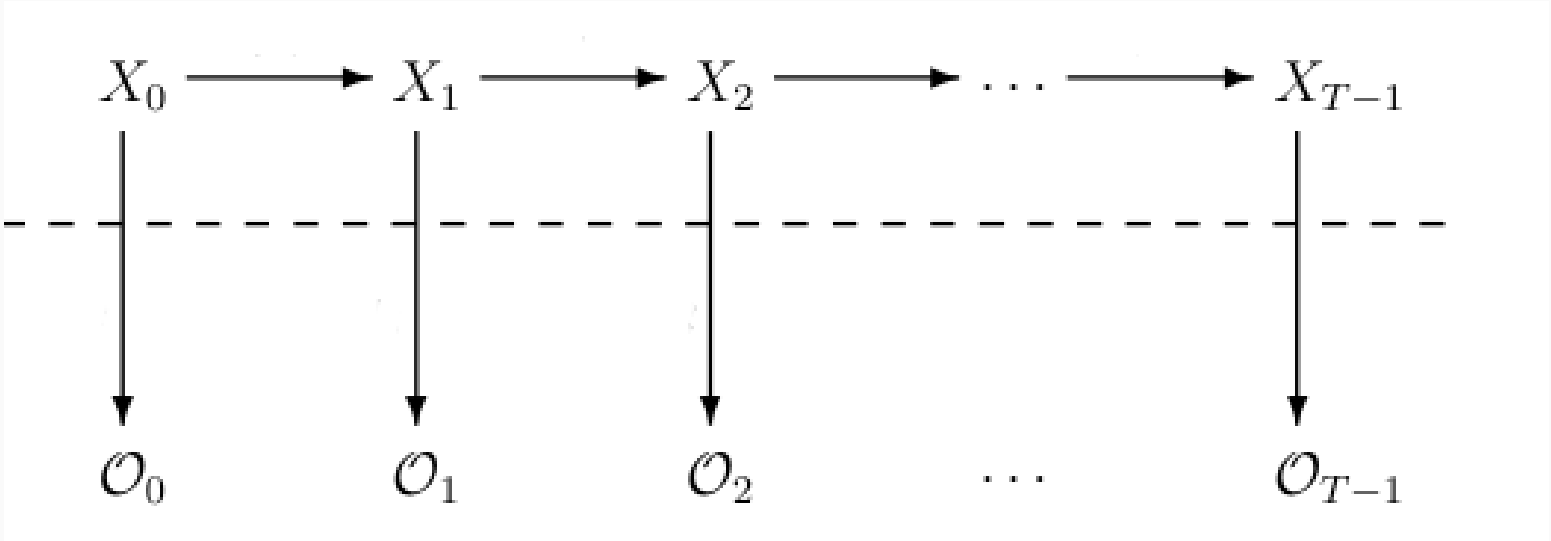
The forward algorithm

This problem can be solved by the *forward algorithm*, which exploits the Markov property to marginalize on the fly:



The forward algorithm

This problem can be solved by the *forward algorithm*, which exploits the Markov property to marginalize on the fly:

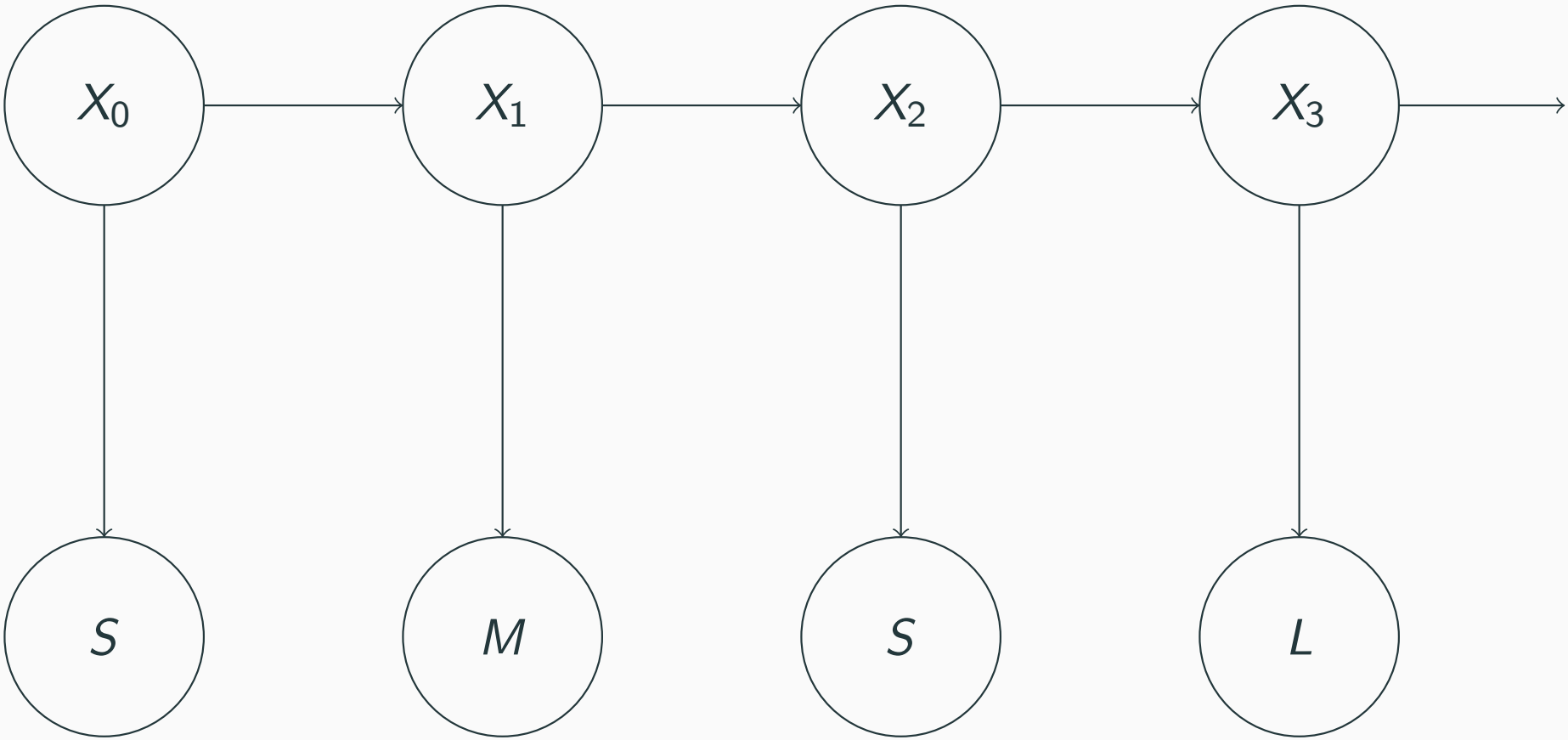


Let $\alpha_t(x_i) = P(X_t = X_i, \mathcal{O}|\lambda)$; then,

$$\alpha_t(x_i) = B_{x_i,\mathcal{O}_i} \sum_{j=1}^{N} \alpha_{t-1}(x_j) A_{x_j,x_i}$$

Hidden Markov models

Suppose we observe a sequence of rings like S, M, S, L.



The backward pass

The smoothing problem asks us to calculate $P(X_t = x_i, \mathcal{O})$ for some $t < T$. We could just solve the filtering problem by running the forward algorithm up to time t, but we would lose the information from future states.

Solution: do a backward pass too.

Let $\beta_t(x_i) = P(\mathcal{O}_{t:T}|X_t = x_i)$; that is, the probability of the "remaining" observations from time t to the end, given $X_t = x_i$. Then,

$$\beta_t(x_i) = \sum_{j=1}^{N} A_{x_i,x_j} B_{x_i,\mathcal{O}_t} \beta_{t+1}(x_j)$$

so we can recursively calculate from the end of the sequence, letting $\beta_T(x_j) = 1$ for each j.

The forward-backward algorithm

The forward-backward algorithm solves the smoothing problem for HMMs:

$$P(X_t = x_i|\mathcal{O}, \lambda) = \frac{\alpha_t(x_i)\beta_t(x_i)}{P(\mathcal{O}|\lambda)}$$

Where can we get the normalizing constant?

$$P(\mathcal{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(x_i)$$

Summary

Today:

- Temporal models, especially hidden Markov models

Going forward:

- More HMM
- Linear Gaussian dynamical systems and the Kalman filter

Viterbi algorithm

In problems of decoding or recognition (such as speech recognition) it's often necessary to compute the most likely sequence of states corresponding to a sequence of observations.

At first glance, it seems that this is the same as the smoothing problem; find the distribution of X_t for each t and choose the most probable state. However, it turns out this doesn't (necessarily) give the most probable sequence.

Instead, we can use the *Viterbi algorithm.*

Viterbi algorithm

Like the forward algorithm, the Viterbi algorithm exploits the Markov property to express the calculation recursively:

- Assume we can calculate the most probable sequences that end in states x_i at time $t-1$
- The most probable sequence that ends in state x_j at time t must be an extension of one of the N most probable sequences up to time $t-1$
- The most probable sequence overall must be the most probably sequence ending in state x_i at time T, for some i

Viterbi algorithm

Essentially, we modify the forward algorithm to replace a sum with a max. Let:

$$\tilde{\alpha}_t(x_i) = P(X_{1:t} = \tilde{X}_{1:t,i}, \mathcal{O}_{1:t}|\lambda)$$

where $\tilde{X}_{1:t-1,i}$ is the most probable sequence of steps given $\mathcal{O}$ such that $X_t = x_i$.

$$\tilde{\alpha}_t(x_i) = \max_j \left(A_{x_j,x_i} B_{x_i,\mathcal{O}_t} \tilde{\alpha}_{t-1}(x_j)\right)$$

Let's step through this graphically with our tree ring model, using an observed sequence M, M, S, S, L.

Viterbi algorithm

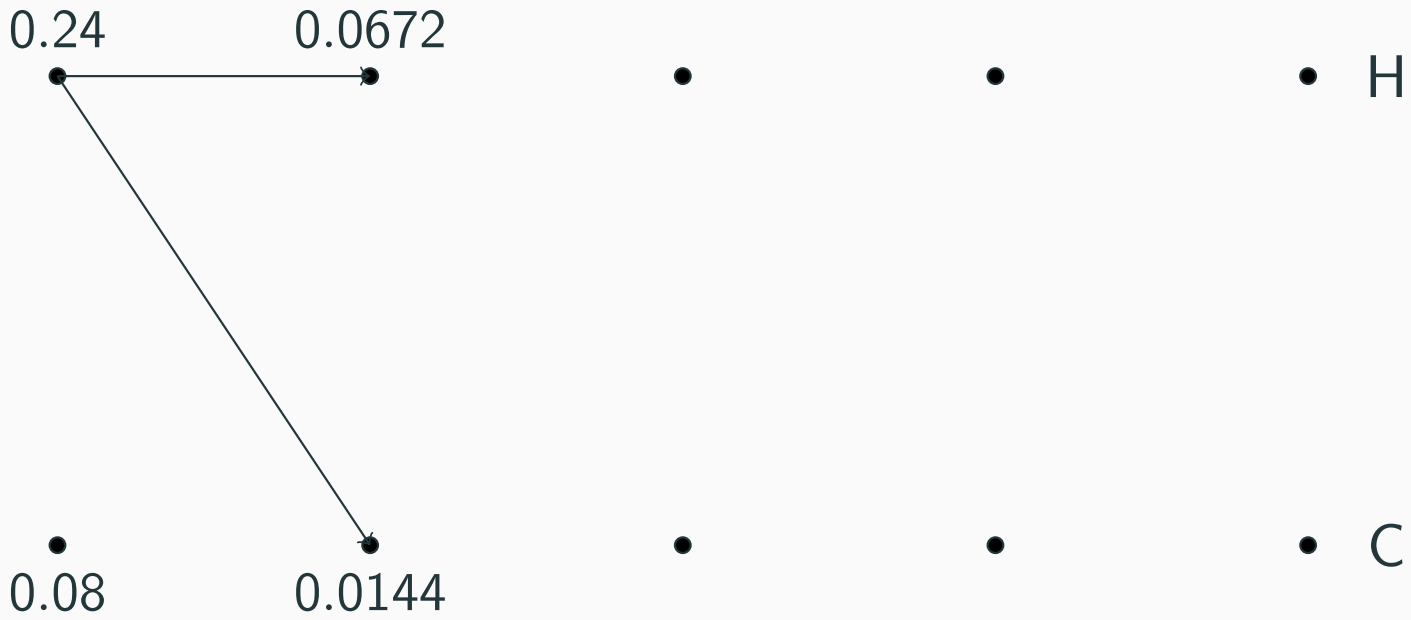
Step 1: calculate probability that time 0 is H or C:

Viterbi algorithm

Step 2: find most likely paths that end in each of C or H at time 1

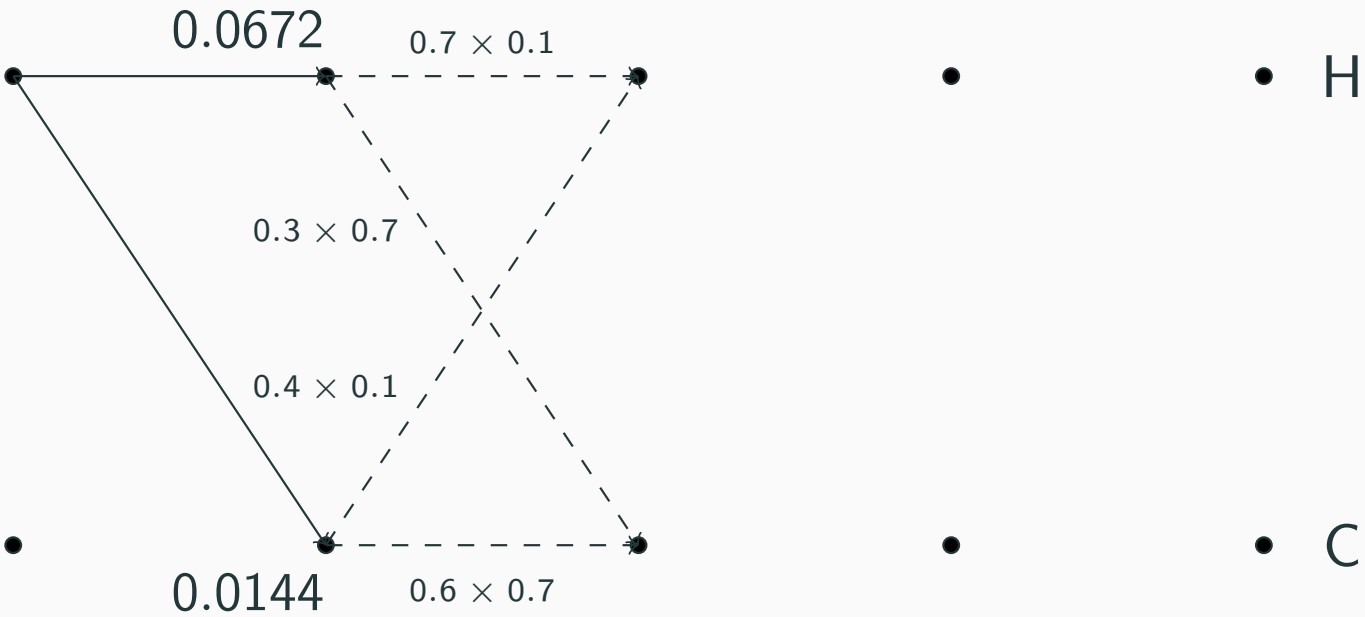
Viterbi algorithm

Step 2: find most likely paths that end in each of C or H at time 1



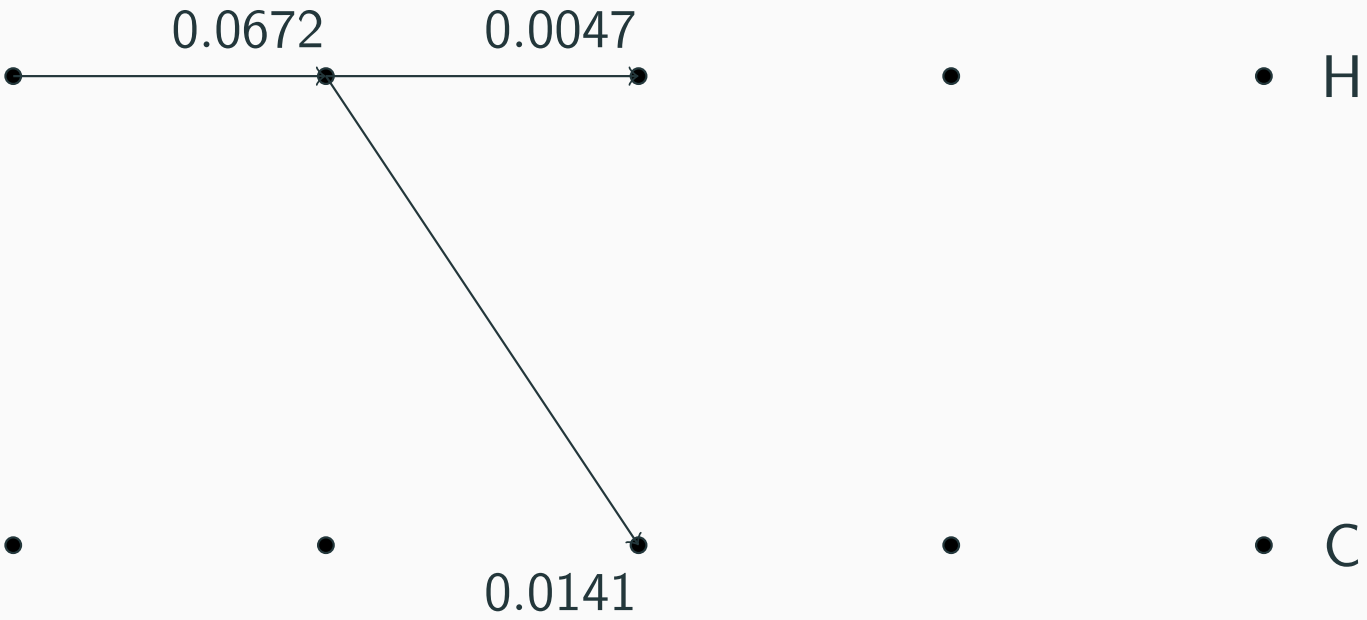
Viterbi algorithm

Step 3: find most likely paths that end in each of C or H at time 2



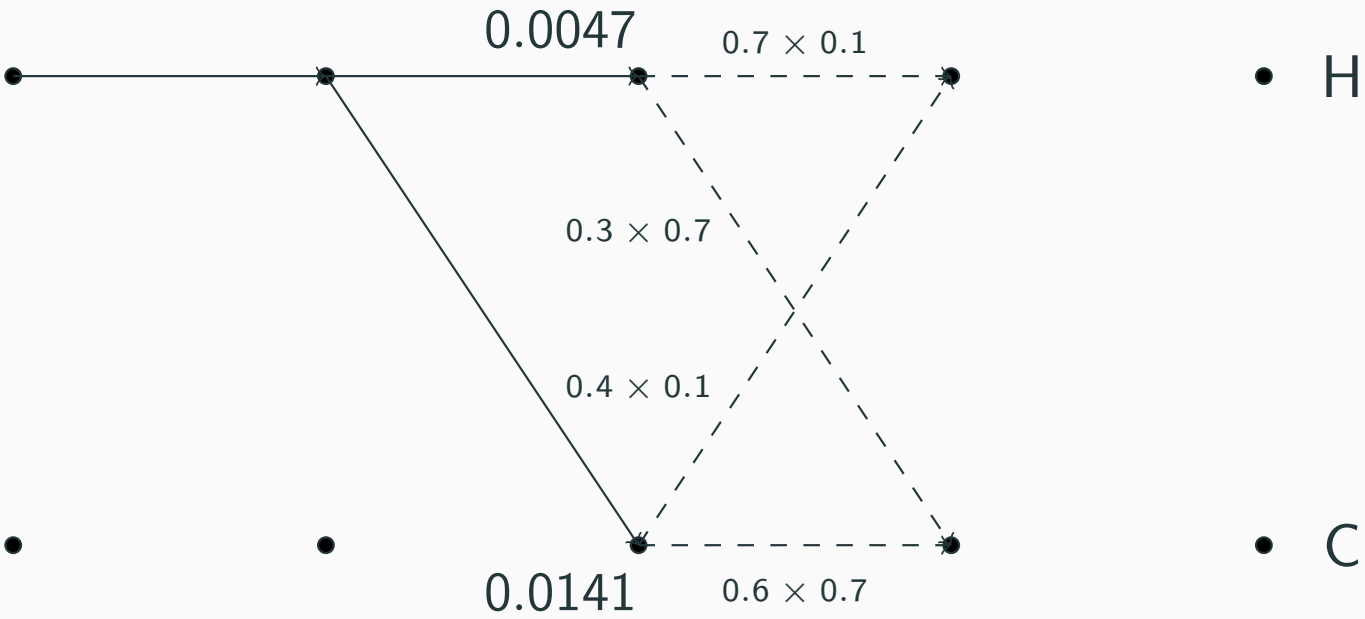
Viterbi algorithm

Step 3: find most likely paths that end in each of C or H at time 2



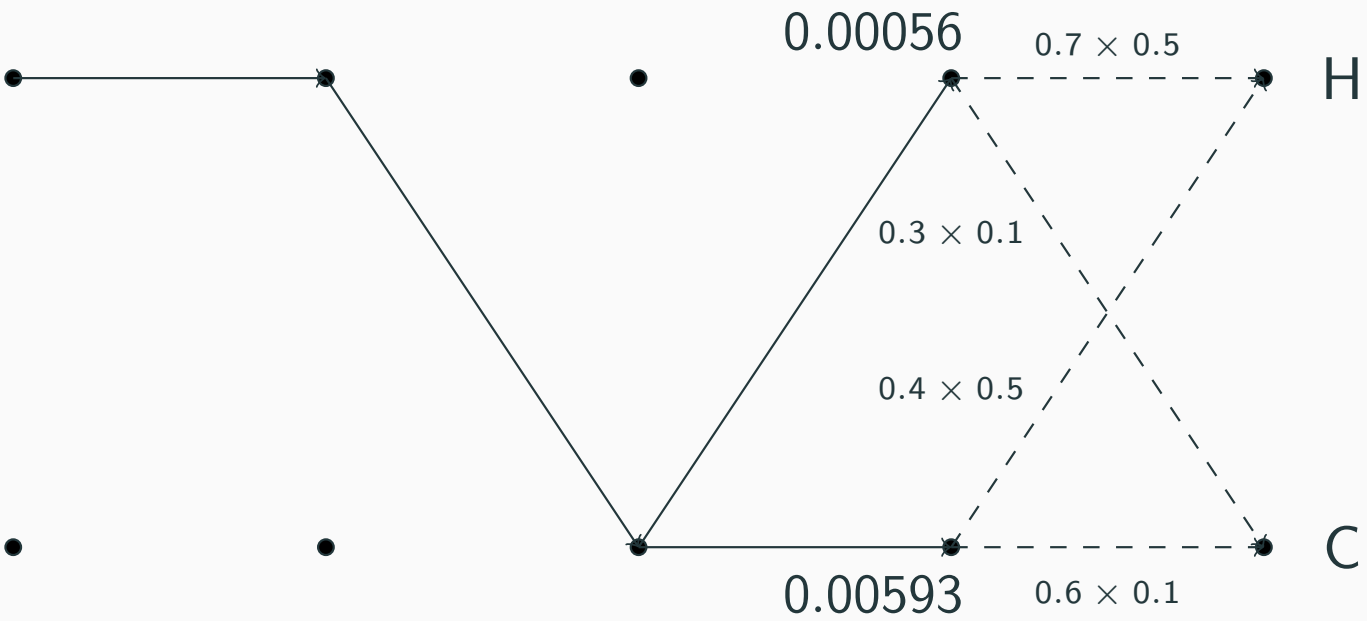
Viterbi algorithm

Step 4: find most likely paths that end in each of C or H at time 3



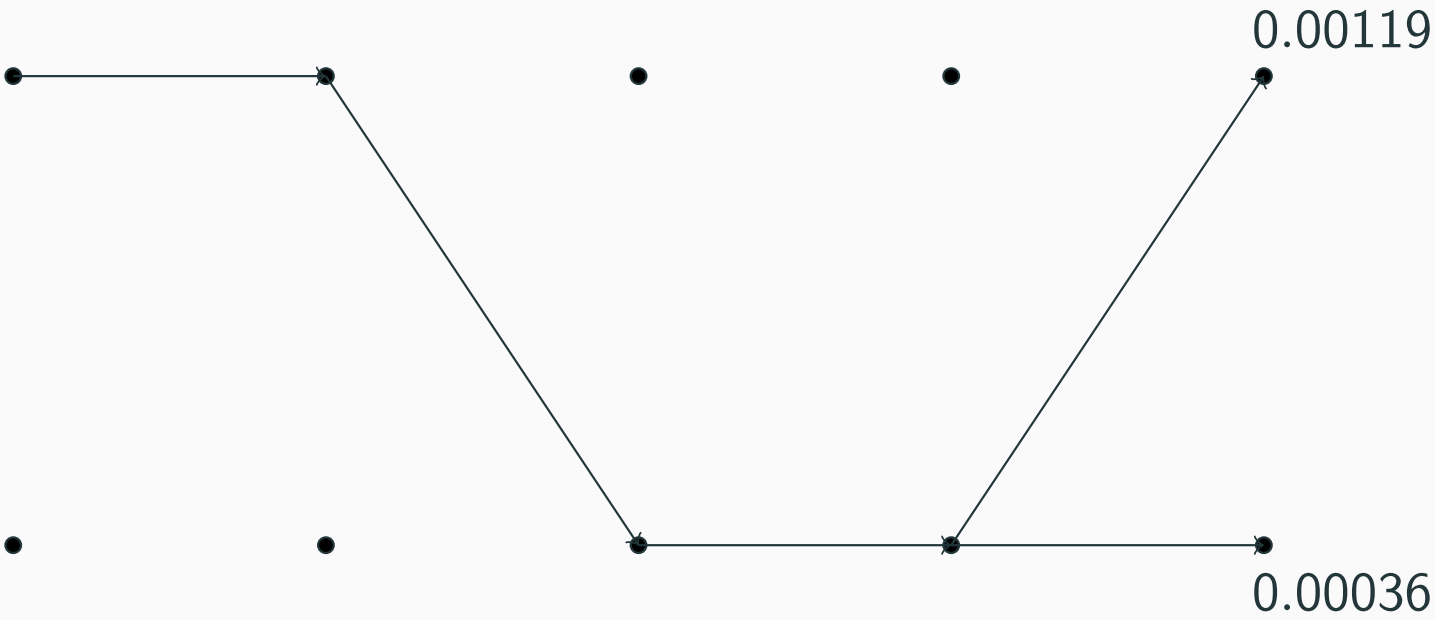
Viterbi algorithm

Step 5: find most likely paths that end in each of C or H at time 4



Viterbi algorithm

Step 5: find most likely paths that end in each of C or H at time 4



Summary

A summary of the algorithm:

- Keep track of the most probable path that ends in each state i at time t (and its non-normalized probability)
- At time $t + 1$, for each state j:
  - For each i, calculate the probability of following the best path to state i and then transitioning to j
  - Choose the highest of those probabilities to conclude the most probable path that ends in state j at time $t + 1$
  - Update the record of most probable paths
- Once we reach the final time step, choose the end state with the best probability