# Hidden Markov models redux

ISTA 410 / INFO 510: Bayesian Modeling and Inference

U. of Arizona School of Information

April 28, 2021

## Outline

Last time:

- Intro to hidden Markov models

Today:

- Review of HMM algorithms
- A pattern-recognition example

Note: office hours moved to Friday 1-3PM this week; Tuesday 1-3 PM next week.

## Last HW / Data Diary

Last HW posted (HW 5)

- Bernoulli GLM with covarying slopes and intercepts
- Show that correlation between parameters can be a feature of the model rather than the data

Reflective assignment: statistics diary

- Create an empty text document
- Once a day for 7 days, add to your statistics diary
  - Entries can be anything; slice-of-life observations, research notes, etc.
  - Write as much or as little as you want
- Optional; can substitute for a missed HW

## Temporal and dynamical models

## Temporal and dynamical models

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

- Hidden Markov model: latent state variables evolve according to a Markov chain/Markov process
- Linear dynamical system: latent state variables evolve according to linear dynamics, possibly with added noise

What these have in common: hidden/latent state variables

## Hidden state models

Structure of the model:

$$\begin{tikzcd}
X_0 \arrow[r] \arrow[d] & X_1 \arrow[r] \arrow[d] & X_2 \arrow[r] \arrow[d] & \cdots \arrow[r] & X_{T-1} \arrow[d] \\
\mathcal{O}_0 & \mathcal{O}_1 & \mathcal{O}_2 & \cdots & \mathcal{O}_{T-1}
\end{tikzcd}$$

- 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
  Observations a linear function of the state, “corrupted” by Gaussian noise

## 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, estimate the parameters of the underlying dynamical model.

## Hidden Markov models

## Example: the unfair casino

A casino employee has two 6-sided dice. We'll assume we know their properties:

| Die | $P(1)$ | $P(2)$ | $P(3)$ | $P(4)$ | $P(5)$ | $P(6)$ |
|---|---|---|---|---|---|---|
| fair | 1/6 | 1/6 | 1/6 | 1/6 | 1/6 | 1/6 |
| loaded | 1/2 | 1/10 | 1/10 | 1/10 | 1/10 | 1/10 |

The operator throws a die, but you don't know which one. What is the probability the die is loaded, assuming it lands on 1? What if instead it lands on 3?

## Tracking fairness over time

Let's say we know a little more about this casino employee's habits.

- The employee always starts the game with the fair die
- Every so often, they secretly switch the die
- Note: this is not an independent choice of die per throw

Result of this: streaks of fair/loaded die over time.

If we observe the result of the die rolls, can we infer when each die was in use?

## Hidden Markov models

A hidden Markov model deals with two sequences:

- a sequence of *states*: the un-observed variable, changing over time according to a Markov chain model
- a sequence of *observations*, or *emissions*: the observed variable, with a distribution based on the current state

In our example:

- the state is which die is currently in use
- the emission is the roll of the die

## Simplest case

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

Our simple example 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 $\pi$ 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.

Reference: Stamp, *A Revealing Introduction to Hidden Markov Models*

## Simplest case

Assumptions:

- Initial state is always fair, so

$$\pi = (1, 0)$$

- Transition matrix:

$$A = \begin{pmatrix} 0.95 & 0.05 \\ 0.05 & 0.95 \end{pmatrix}$$

- Observation matrix:

$$B = \begin{pmatrix} 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/2 & 1/10 & 1/10 & 1/10 & 1/10 & 1/10 \end{pmatrix}$$

## Three algorithms

Today: standard algorithms for filtering, smoothing, and fitting:

1. Given a multinomial HMM $\lambda$ and a sequence of observations $\mathcal{O}$, compute the distribution $P(X_t|\lambda, \mathcal{O})$.
2. Given a multinomial HMM $\lambda$ and a sequence of observations $\mathcal{O}$, compute the probability distribution of $X_t$ for some $1 \leq t \leq T$.
3. Given a sequence of observations $\mathcal{O}$, what is the multinomial HMM $\lambda$ that maximizes the marginal likelihood $P(\mathcal{O}|\lambda)$?

## Naïve filtering

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

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

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

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

What's the problem?

## Naïve filtering

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

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

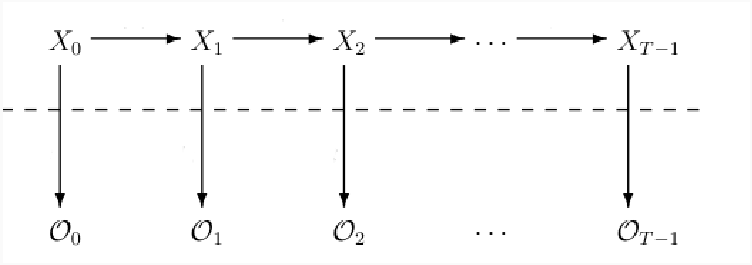
So, naïvely, we could compute this for all sequences of states, and then

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

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

## The forward algorithm

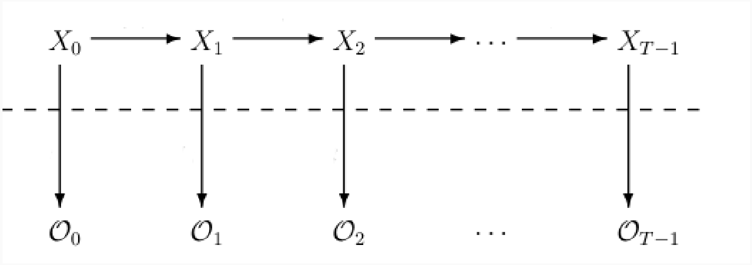
This problem can be solved by the *forward algorithm*, which exploits the Markov property to marginalize recursively on the fly:



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

## The forward algorithm

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

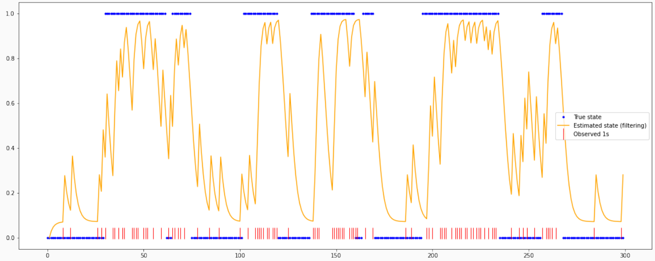


Let $\alpha_t(x_i) = P(X_t = X_i, \mathcal{O}|\lambda)$

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

## Filtering result

To test, generate a sequence of states and observations, and run the forward algorithm:



## 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:

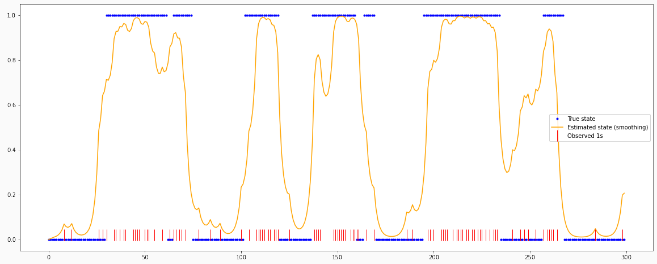
$$P(X_t = x_i | \mathcal{O}, \lambda) = \frac{\alpha_t(x_i)\beta_t(x_i)}{P(\mathcal{O}|\lambda)}$$

Where can we get the normalizing constant?

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

## Smoothing result

To test, generate a sequence of states and observations, and run the forward-backward algorithm:



## Aside: Viterbi algorithm

The forward-backward algorithm computes

$$P(X_t = i | \mathcal{O}, \lambda)$$

for each $1 \leq t \leq T$.

- Slightly different: MAP estimate of the state sequence
- Most probable sequence X may not equal the most probable state at each time step
- Viterbi algorithm (described in Stamp's article as *dynamic programming*) gives MAP estimate for the state sequence

## 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 probable 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 $X_t = x_i$.

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

## HMM fitting

## Fitting parameters

The fitting problem gives a new challenge:

- Given a fixed state space $\{0, 1, \ldots, n\}$ and a sequence $\mathcal{O}$ of observations, find the model parameters that best fit the sequence $\mathcal{O}$
- i.e., tune $A$ (transition matrix), $B$ (observation matrix), and $\pi$ (initial state distribution)
- Target: maximize $P(\mathcal{O}|A, B, \pi)$

This is a form of unsupervised learning.

## Baum-Welch algorithm

The Baum-Welch algorithm iteratively improves the fit of the model parameters in a two-step process:

- Do a smoothing step, estimating the probability distributions of the hidden states $X_t$
- Re-adjust the model parameters to better fit this estimated distribution
- Score the model by the log-probability of the observed sequence
- Continue until log-probability change is negligible

End result: MAP estimate of model parameters

## Idea behind BW algorithm

Intuitively:

- The smoothing step allows us to estimate the probability that the underlying chain is in each state $x_i$ at time $t$
- We can use this to count the estimated probability of transitions from state $x_i$ to state $x_j$
- We can use this, together with the observation sequence, to estimate the probability of each observation from state $x_i$

## Estimating the transition matrix

Smoothing gives us:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(\mathcal{O}|A, B, \pi)}$$

which estimate the probability that the chain was in state $x_i$ at time $t$. We extend this to:

$$\gamma_t(i,j) = \frac{\alpha_t(i)A_{ij}B_{j,\mathcal{O}_t},\beta_{t+1}(j)}{P(\mathcal{O}|A, B, \pi)}$$

which estimates the probability that the chain was in state $x_i$ at time $t$ *and* state $x_j$ at time $t+1$.

Then, we re-estimate the transition probability $A_{ij}$ as:

$$A_{ij} = \frac{\sum_t \gamma_t(i,j)}{\sum_t \gamma_t(i)}$$

## Estimating the rest

Similarly, we can re-estimate the observation probability $B_{ij}$ as

$$B_{ij} = \frac{\sum_{t,\mathcal{O}_t=j} \gamma_t(i)}{\sum_t \gamma_t(i)}$$

the expected proportion of the time spent in state $i$ that produces observation $j$.

## Estimating the rest

Similarly, we can re-estimate the observation probability $B_{ij}$ as

$$B_{ij} = \frac{\sum_{t,\mathcal{O}_t=j} \gamma_t(i)}{\sum_t \gamma_t(i)}$$

the expected proportion of the time spent in state $i$ that produces observation $j$.

The estimate of the initial state vector is just:

$$\pi_i = \gamma_0(i)$$

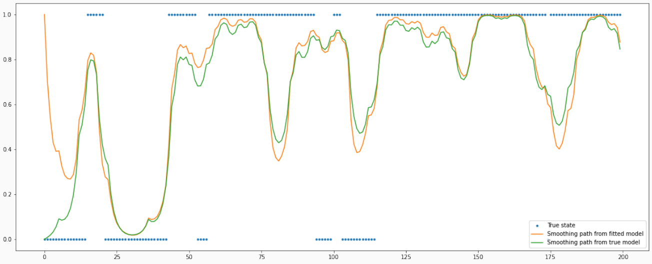
## Testing the algorithm

Let's test the algorithm on the unfair casino problem:

- Generate 1200 observations from the "true" model
- Initialize a HMM with the correct number of states, but randomly initialized parameters
- Fit the model; test its performance on a smoothing problem

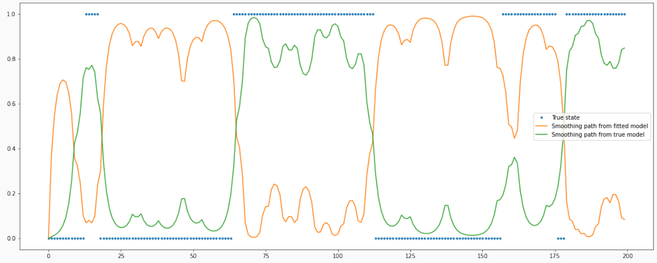
## Testing the algorithm

Now: 200 new states and observations



## Testing the algorithm

Another run of this experiment produced this:



Is this a failure of the fitting process?

## Other applications

Because of their ability to find and recognize patterns in sequence data without supervision, HMMs find applications in, among other places:

- speech recognition
- cryptanalysis (codebreaking) and malware detection
- activity recognition
    - identify what an animal is doing based on GPS data

## Text analysis example

Imagine you're an alien with no knowledge of human language, but you gain access to a sample of English text, and you would like to extract some information about the relationships between characters.

Simplifying assumptions:

- No cases – everything is lowercase
- No digits or punctuation; only characters are letters and spaces

Idea:

- different characters play different roles in the written language.
- fit a hidden Markov model with $k$ different states to a large sample of text, and see if any patterns can be seen.

Let's take a look at the results for $k = 2$.

## Expectation-maximization algorithms

## EM algorithms

The Baum-Welch algorithm we saw before is an example of a much wider class of algorithms called *expectation-maximization* algorithms.

These are applicable when the observed data depends on hidden/latent state variables as well as model parameters. Roughly, the idea is:

- Expectation step: compute the distribution of hidden state variables, given current model parameters
- Maximization step: compute the model parameters that maximize (log) likelihood given the state parameters from the expectation step

Repeat until done – score model by total log-likelihood of the data.

Section 13.4-13.6 in BDA has another presentation of EM algorithms in a different context.

## EM algorithms

Formally:

- $\theta$: model parameters
- X: hidden variables
- Y: observations
- $L(\theta|X, Y)$: likelihood function

1. E-step: compute $Q(\theta|\hat{\theta}) = E_{X|Y,\hat{\theta}}[\log L(\theta|X, Y)]$
2. M-step: compute $\theta^{\mathrm{new}} = \arg\max_{\theta} Q(\theta|\hat{\theta})$

## BW algorithm as EM

Recall the Baum-Welch algorithm has two steps:

- Perform smoothing to estimate the distribution of each $X_t$, given current transition/observation matrix values
- Update parameter values by counting transitions/observations given distributions of $X_t$

Although we don't explicitly calculate expectations of log-likelihoods, the smoothing step is an E step and the update step is an M step.

## A few comments

## Alternative approaches to parameter estimation

Other approaches to fitting HMM parameters:

- MCMC – suitable for smaller systems
    - Helmut Strey has a PyMC3 implementation
    - `https://github.com/hstrey/Hidden-Markov-Models-pymc3`
- Particle filter (aka sequential Monte Carlo) methods
    - Brief overview next Wednesday, time permitting

## Gaussian HMM

The most common alternative to the multinomial distribution for HMMs is the Gaussian (normal) distribution. In this model:

- $X_t$ still evolves according to a Markov chain with transition matrix $A$
- $\mathcal{O}_t \sim \mathrm{MVNormal}(\mu_{X_t}, \Sigma_{X_t})$
- Result: the observation distributions are Gaussian mixtures

## Filtering and smoothing in Gaussian HMM

What has to change for our filtering and smoothing algorithms?

## Filtering and smoothing in Gaussian HMM

What has to change for our filtering and smoothing algorithms?

- Only change: $P(\mathcal{O}_t = j | X_t = x_i)$ is no longer given by a matrix entry $B_{ij}$
- Instead, we have $p(\mathcal{O}_t = \mathrm{y} | X_t = x_i) = \mathrm{MVNormal}(\mu_i, \Sigma_i)$ for a certain mean vector $\mu_i$, covariance matrix $\Sigma$

## EM for Gaussian HMM

To fit the Gaussian HMM, we only need to make the following modifications to the M step:

- Replace $B_{ij}$ with $\boldsymbol{\mu}_i, \Sigma_i$
- Replace the update of $B_{ij}$ with a maximum-likelihood estimate for a Gaussian, weighted by the estimated state probabilities (from smoothing):

$$\boldsymbol{\mu}_i^{\mathrm{new}} = \frac{\sum_t P(X_t = i)\boldsymbol{y}_t}{\sum_t P(X_t = i)}$$
$$\Sigma_i^{\mathrm{new}} = \frac{\sum_t P(X_t = i)(\boldsymbol{y}_t - \boldsymbol{\mu}_i^{\mathrm{new}})(\boldsymbol{y}_t - \boldsymbol{\mu}_i^{\mathrm{new}})^T}{\sum_t P(X_t = i)}$$

  where $\mathrm{y}_i$ are the observations.

## Missing data

Suppose we have an incomplete sequence of observations:

$$(\mathcal{O}_t) = (\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_T)$$

where some $\mathcal{O}_t$ are unobserved (NA).

We can still perform forward and backward algorithms for filtering/smoothing; however, steps where $\mathcal{O}_t = NA$ only involve transition probabilities, no observation.

Result: estimated distribution of hidden states relaxes toward the stationary distribution of the MC during “gaps” in observations

## Continuous time

Continuous-time Markov chains do exist, so we could build a HMM on top of one of those.

- Applications:
- How a CT-MC works:
    - Each state $x_i$ has an associated *holding time* – an exponential random variable
    - Chain stays in current state for the holding time and then undergoes a transition according to a transition matrix
- Challenge: transition times are unobserved, and may not correspond to the observation times

## Continuous time

Reduction to discrete HMM:

- The continuous time chain can be expressed in terms of a *transition rate matrix* $Q$
- Each entry $q_{ij}$ gives the rate parameter for an exponential random variable; transitions from state $i$ are determined by the minimum of the exponential random variables
- Can reduce to a discrete-time Markov chain with transition matrix dependent on the time interval between two observations: $P(t) = \exp(Qt)$

Details: Liu et al., “Efficient Learning of Continuous-Time Hidden Markov Models for Disease Progression” (2015)

## Summary

Today:

- Applications of hidden Markov model fitting

Next week:

- Linear Gaussian dynamical systems and the Kalman filter
- Forward filtering/backward sampling algorithms
- Particle filter algorithms (maybe?)