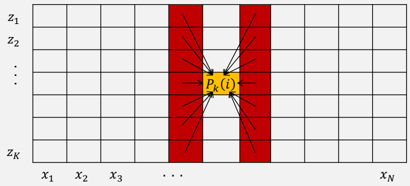
1

BTRY 4840/6840, CS 4775
Computational Genetics and Genomics



September 20, 2018

2

Announcements

- Problem set 2 due on Tuesday
- My office hours now 4:30-5:30pm on Mondays
  - Must come before 5pm because 102 door locks at that time
  - TAs still Tuesdays 4:20-5:20pm

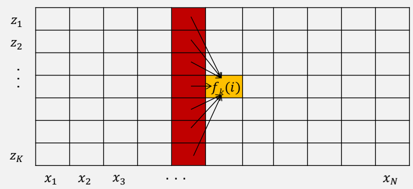
3

Today's lecture

- Continuing Hidden Markov models (HMMs)
  - Finish backward algorithm
  - Forward/backward algorithm
  - Posterior state probabilities and posterior decoding
  - Supervised and one form of unsupervised learning
- Log probabilities for numerical stability

4

Recall: Forward probability calculation



- Initialization: $f_k(1) = a_{0k} e_k(x_1) = P(z_1 = k) P(x_1 | z_1 = k)$
- Iteration: $f_k(i) = e_k(x_i) \cdot \sum_{j=1}^{K} f_j(i-1) \cdot a_{jk}$
- Final value: $P(\boldsymbol{x}) = \sum_{k=1}^{K} f_k(N)$

No trace back: not about one path

- Runtime, Space complexity: $O(K^2 N)$, $O(KN)$, respectively

5

Hidden Markov models
What is $P(z_i = k|\boldsymbol{x})$?

Forward-backward algorithm

6

Computing $P(z_i = k|\boldsymbol{x})$

- Want $P(z_i = k|\boldsymbol{x})$
- By definition of conditional probability, we have

$$P(z_i = k|\boldsymbol{x}) = \frac{P(\boldsymbol{x}, z_i = k)}{P(\boldsymbol{x})}$$

  - Forward probability gives $P(\boldsymbol{x})$
  - How do we compute the numerator?

$$\begin{aligned} P(\boldsymbol{x}, z_i = k) &= P(x_1, \dots, x_i, z_i = k) P(x_{i+1}, \dots, x_N | x_1, \dots, x_i, z_i = k) \\ &= P(x_1, \dots, x_i, z_i = k) P(x_{i+1}, \dots, x_N | z_i = k) \\ &= f_k(i) \cdot \underbrace{P(x_{i+1}, \dots, x_N | z_i = k)}_{b_k(i)} \end{aligned}$$

7

Backward probability

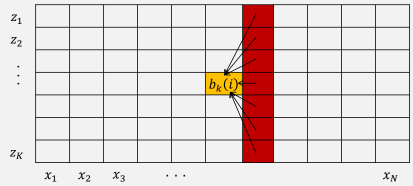
- Similar to forward probability, but calculated using observed data *after* a given step i
- Let $b_k(i) = P(x_{i+1}, \ldots, x_N | z_i = k)$: backward probability
- Given $b_k(i+1)$ at some step, the following holds:

$$b_k(i) = \sum_{l=1}^{K} a_{kl} \cdot e_l(x_{i+1}) \cdot b_l(i+1)$$

8

Backward probability calculation

States



- Initialization: $b_k(N) = a_{k0}$ (typically 1)
- Iteration: $b_k(i) = \sum_{l=1}^{K} a_{kl} \cdot e_l(x_{i+1}) \cdot b_l(i+1)$
- Note: $b_k(i)$ does not include emission probability for step i
- Final value: $P(\boldsymbol{x}) = \sum_{k=1}^{K} a_{0k} e_k(x_1) b_k(1)$

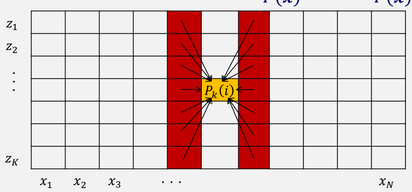
Analogous to forward
- Computes $P(\boldsymbol{x})$
- No trace back

9

Forward-backward calculation

- From before, have: $P(z_i = k|\boldsymbol{x}) = \frac{P(\boldsymbol{x}, z_i = k)}{P(\boldsymbol{x})} = \frac{f_k(i) b_k(i)}{P(\boldsymbol{x})}$

States



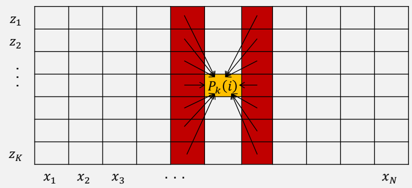
- Note: $e_k(x_i)$ is not in $b_k(i)$ but is included in $f_k(i)$
  - Backward $b_k(i)$ gives probability of reaching state k at step i, but not anything before that, including emitting x_i
- Denominator $P(\boldsymbol{x}) = \sum_{k=1}^{K} P(\boldsymbol{x}, z_i = k) = \sum_{k=1}^{K} f_k(i) b_k(i)$

10

Posterior state probability

- Now have $P(z_i = k|\boldsymbol{x})$ for all i, k
  - This is what is reported for many problems

States

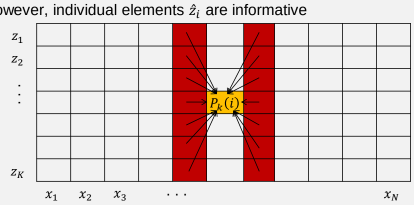


11

Posterior decoding

- Can also get posterior decoding: most likely state at each step i:
  - $\hat{z}_i = \mathrm{argmax}_k\, P(z_i = k|\boldsymbol{x})$
- Unique features of posterior decoding:
  - May not be a possible state path / may not be very likely as a path
  - However, individual elements $\hat{z}_i$ are informative

States

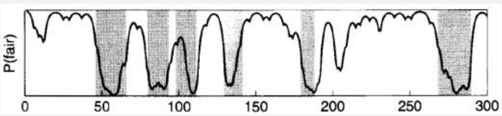


12

Rationale for forward-backward

- Viterbi decoding gives most likely path
- Why do we want $P(z_i = k|\boldsymbol{x})$ or posterior decoding?
  - Can have many high probability paths
  - Way of quantifying how certain we are of any state

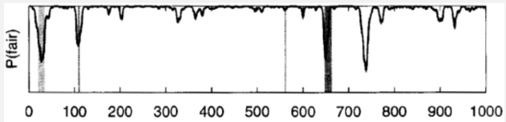
Example: dishonest casino posterior probability



Shaded areas: truth is loaded die

Alternate dishonest casino model

- Slight variation on dishonest casino:
  P(fair → loaded) = 0.01 (not 0.05)
  1,000 rolls



Shaded areas: truth is loaded die

- Viterbi decoding ⇒ no roll is from loaded die
  - Probabilistic treatment more informative

Hidden Markov models

Simulating from Model

HMMs: generative models, can simulate

- Can randomly sample from HMM (or Markov chain)
  - Used to simulate data: gives you both $\boldsymbol{x}$ and hidden states $\boldsymbol{z}$
  - Approach to sampling from HMM – very fast:
    1. Sample first state k from initial probabilities a_{0k}
    2. Given state k, sample emission b from $e_k(b)$
       [reason for "emission" name: generative model]
    3. Given state k, sample state l from transition probabilities a_{kl}
    4. Repeat 2,3 until sufficient data generated
- With simulated data, can evaluate your inference method
  - Common practice to simulate and infer with same model
  - Ideally should generate data from different model
    - Not always practical: may only have one reasonable model

Desired uses of HMMs (highlighted done)

- Evaluation:
  - **Given**: observed $\boldsymbol{x}$ and HMM specification
    Question: what is the joint probability of $\boldsymbol{x}$ and a given $\boldsymbol{z}$?
    Question: what is the likelihood of $\boldsymbol{x}$ based on the HMM?
- Decoding:
  - **Given**: observed $\boldsymbol{x}$ and HMM
    Question: what sequence of hidden states produced $\boldsymbol{x}$?
  - Viterbi decoding: most likely hidden state sequence
  - Posterior probability of hidden states: probability of each state z_i producing each x_i
    - Get posterior decoding from these posterior probabilities
- Learning:
  - → **Given**: observed $\boldsymbol{x}$ and HMM without complete probabilities
    Question: what emission, transition probabilities produced $\boldsymbol{x}$?

Hidden Markov models

How do we learn transition, emission probabilities from **labeled** data?

Supervised learning

Problem: infer parameters with labeled data

- Given:
  - HMM specification without parameters:
    - Have definition of states, but no emission / transition probabilities
  - Labeled training data:
    - Training data: data we use to fit our models
    - Labeled: hidden states specified
- Question: what should the parameters be?
  - Will maximize $L(\boldsymbol{\theta}|\boldsymbol{x}) = P(\boldsymbol{x}|\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represents the parameters $(a_{kl}, e_k(x))$
- Note: defining HMM states an art: no "right" answer
  - Specific to the problem at hand: will give more examples

Examples of labeled data

- Labeled data for CpG island problem:
  - Very long sequence with annotated CpG islands (say 10 million sites)
- Labeled data for dishonest casino:
  - We watch the casino player change dice and roll for a long period of time (say 10,000 rolls)

Setting the parameters

- Determining $\boldsymbol{\theta}$ from labeled data: fairly simple
- Given $x_1, x_2, \dots, x_N$ with correct $z_1, z_2, \dots, z_N$
- Let

  $A_{kl} = \#\text{ times transition } z_i = k \rightarrow z_{i+1} = l \text{ occus for all } i$

  $E_k(b) = \#\text{ times } z_i = k \text{ emits } b \text{ for all } i$
- Can show that MLE for the parameters are

$$a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}} \text{ and } e_k(b) = \frac{E_k(b)}{\sum_{b'} E_k(b')}$$

What if we have very little training data?

- MLEs rely heavily on data to determine parameters
  - Vulnerable to overfitting if we have only small amounts of data
- Another issue:
  - If we never observe certain transitions/emissions, have undefined or 0 valued probabilities
- Example: we only observe 10 die rolls

  $\boldsymbol{x} = 1,2,1,6,4,5,2,1,4,4$

  $\boldsymbol{z} = F, F, F, F, F, F, F, F, F, F$

  Gives $a_{FF} = 1, a_{FL} = 0, \quad a_{LL} = ?, a_{LF} = ?$

  $e_F(1) = e_F(4) = .3, \; e_F(2) = .2, \; e_F(5) = e_F(6) = .1, \; e_F(3) = 0$
- 0 probabilities ⇒ broken model in most instances

Solution: pseudocounts

- Pseudocounts: added to observed data counts
- With pseudocounts:

  $A_{kl} = (\#\text{ times transition } z_i = k \rightarrow z_{i+1} = l \text{ occurs for all } i) + r_{kl}$

  $E_k(b) = (\#\text{ times } x_i = k \text{ emits } b \text{ for all } i) + r_k(b)$
- Not a hack: represent prior beliefs
  - For CpG island HMM, have prior belief that $a_{C^+G^+} > a_{C^-G^-}$
  - Can be the same for all values (corresponds to uniform prior)
  - Can use any value, e.g., in count data:
    - $r_{kl} < 1$ avoids 0 probability, but heavily relies on data
    - $r_{kl} > 1 \Rightarrow$ priors count more than any one observation
  - Corresponds to Dirichlet distribution prior
    - Multivariate generalization of Beta distribution: very flexible

Numerical stability in evaluating HMMs

Issue: computers store numbers in finite space

- To analyze HMMs, we multiply many numbers < 1
  - Can produce underflow: a number too small for the computer to store
- Example:
  - Want to analyze a sequence of length 10,000 bases
  - $a_{kl} = .1$ for all k, l $\quad e_k(b) = 1$ for some k, b pair
  - Viterbi path score: $10^{-10,000}$
  - ➢ Way too small for standard floating point representations

Underflow example

- Underflow example in Python:

```
>>> prob = 1
>>> for i in range(0,10000):
...   prob *= .1
...
>>> prob
0.0
```

- Solution: log probabilities
  - Effective way to compute using very small numbers
  - Have $\log(x^y) = y\log(x)$ – much easier number to represent for large positive/negative exponent y

Example: no underflow using log probabilities

```
>>> from math import log
>>> logProb = 0
>>> logTransitionProb = log(.1)
>>> for i in range(0,10000):
...   logProb += logTransitionProb
...
>>> logProb
-23025.850929942502
>>> logProb / log(.1)
10000.00000000089
```

Notes:
- `log()` is expensive
  - ➢ Should precompute probabilities of HMM parameters
- Log probabilities are often more efficient because we sum instead of multiply

Issue: need to be able to add probabilities

- Forward & backward algorithms sum probabilities
  - Inconvenient: $\log(x) + \log(y) \neq \log(x + y)$
- Can try converting to normal space
- Approach 1:
  `log( exp(a) + exp(b) )`
  (Here a, b are log probabilities)

Mathematically correct, but
- Can underflow: `exp(-1000) == 0.0`
- Expensive: two `exp()` and one `log()` calls

Improved sum of log probabilities

- Let a, b be log probabilities, then
  $$\exp(a) + \exp(b) = \exp(a) \cdot \left(1 + \frac{\exp(b)}{\exp(a)}\right) = \exp(a) \cdot (1 + \exp(b - a))$$
  So: $\log(\exp(a) + \exp(b)) = \boxed{a + \log(1 + \exp(b - a))}$
- Better numerical stability:
  - Worst case for approach 1: `exp(a) + exp(b) == 0.0`, so get `log(0.0)`: $-\infty$ or undefined
  - Worst case for above approach: `exp(b-a) == 0.0`, so get `a + log(1.0 + 0.0) == a` ← **much** better
- Better computation:
  - Now only one `exp()` and one `log()` call

Even better sum of log probabilities

- If $\exp(b - a)$ is very small (e.g., 10^{-20}), can have `1+exp(b-a) == 1.0`, and thus `log(1) == 0`
- Solution: `log1p(x)`
  - Computes log(1+x) at higher precision:

```
>>> log1p(1e-20)
9.9999999999999995e-21
>>> log(1+1e-20)
0.0
```

- This is related to the lower.tail=FALSE option in R:
  - One-tailed p-value definition: $p = 1 - F(x)$, F the CDF: $P(X \leq x)$

```
pnorm(7) == 1.0  ⇒  1-pnorm(7) == 0.0
pnorm(7, lower.tail=FALSE) == 1.279813e-12
```

Python implementation to sum log probabilities

```
from numpy import log1p
from math import exp
def sumLogProb(a, b):
  if a > b: return a + log1p(exp(b − a))
  else:     return b + log1p(exp(a − b))
```

- If statement:
  - Why do we prefer max of `a` and `b`?
    - Trying to sum the corresponding probabilities: is $\geq \max(a, b)$
- For efficiency, Durbin *et al.* suggest generating a table for log(1 + exp(b-a))
  - Idea: `b−a` often close to 0, so don't need large table
  - Use linear interpolation between table values

Final notes

- Summary:
  - Hidden Markov models: very general framework for analyzing data from a given model
    - Can infer:
      - Most likely path of hidden states (Viterbi)
      - Probability of hidden state at a position (Forward-Backward)
    - Supervised learning of parameters
    - Sampling from posterior distribution
  - Should use log probabilities for numerical stability