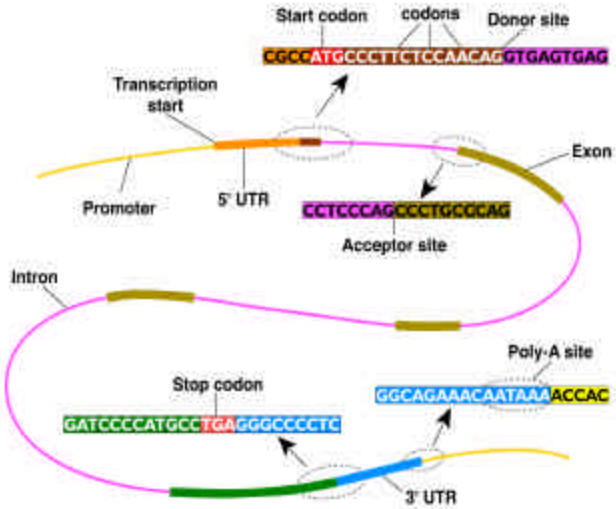
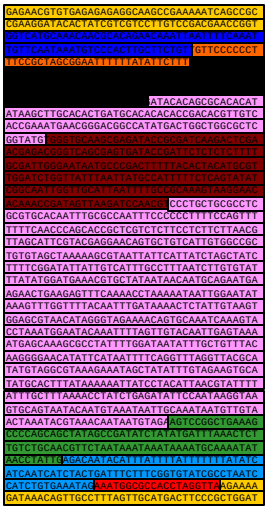
The Hidden Markov Models for sequence parsing

Gene structure in eukaryotes



Gene finding

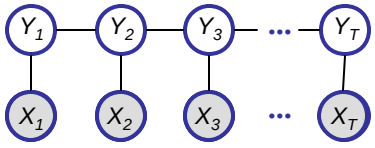
- Given un-annotated sequences,
- delineate:
  - transcription initiation site,
  - exon-intron boundaries,
  - transcription termination site,
  - a variety of other motifs: promoters, polyA sites, branching sites, etc.
- The hidden Markov model (HMM)



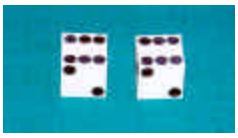
Hidden Markov models

The underlying source:
genomic entities,
dice,

The sequence:
Ploy NT,
sequence of rolls,



Example: The Dishonest Casino



A casino has two dice:

- Fair die
  P(1) = P(2) = P(3) = P(5) = P(6) = 1/6
- Loaded die
  P(1) = P(2) = P(3) = P(5) = 1/10
  P(6) = 1/2

Casino player switches back-&-forth between fair and loaded die once every 20 turns

Game:

1. You bet $1
2. You roll (always with a fair die)
3. Casino player rolls (maybe with fair die, maybe with loaded die)
4. Highest number wins $2

Question # 1 – Evaluation

GIVEN

A sequence of rolls by the casino player

12455264621461461361366616646616366163661636616515615115146123562344

QUESTION

How likely is this sequence, given our model of how the casino works?

This is the **EVALUATION** problem in HMMs

Question # 2 – Decoding

GIVEN

A sequence of rolls by the casino player

1245526462146146136136661664661636616636616361651561511514612356234 4

QUESTION

What portion of the sequence was generated with the fair die, and what portion with the loaded die?

This is the **DECODING** question in HMMs

Question # 3 – Learning

GIVEN

A sequence of rolls by the casino player

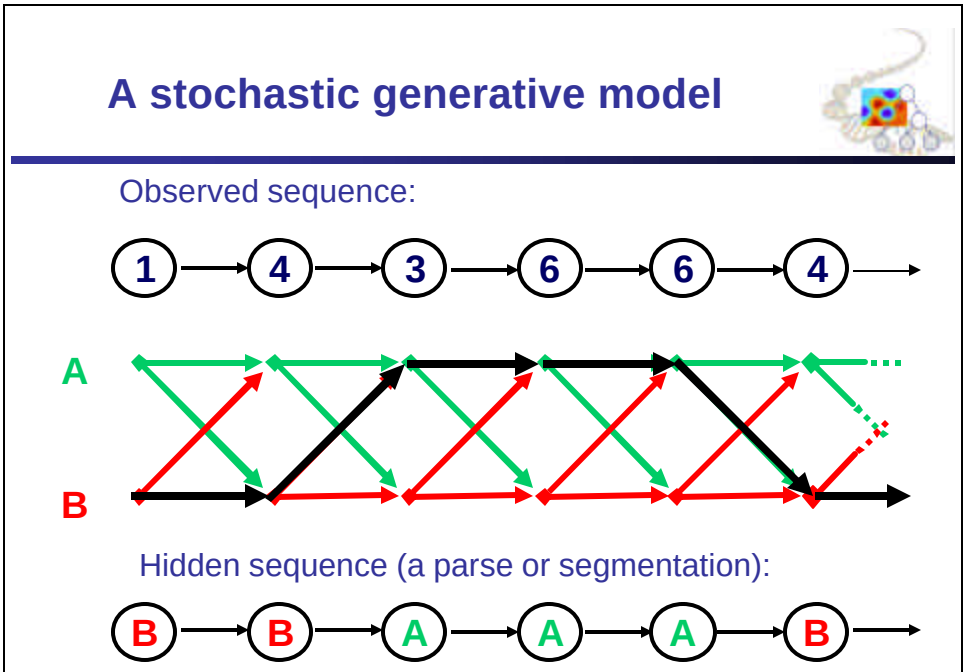
1245526462146146136136661664661636616636616361651561511514612356234

QUESTION

How “loaded” is the loaded die? How “fair” is the fair die? How often does the casino player change from fair to loaded, and back?

This is the **LEARNING** question in HMMs

A stochastic generative model



Definition (of HMM)

Definition: A hidden Markov model (HMM)

- Observation alphabet $\Sigma = \{ b_1, b_2, \ldots, b_M \}$
- Set of hidden states $Q = \{ 1, \ldots, K \}$
- Transition probabilities between any two states

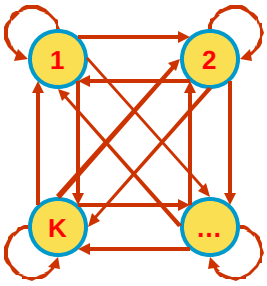
$$a_{i,j} = P(y_t=j | y_{t-1}=i)$$
$$a_{i,1} + \ldots + a_{i,K} = 1, \quad \text{for all states } i = 1 \ldots K$$

- Start probabilities $a_{0,i}$

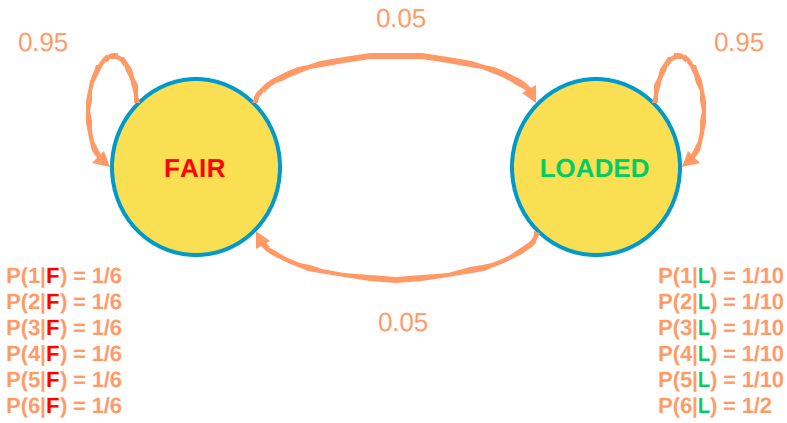
$$a_{0,1} + \ldots + a_{0,K} = 1$$

- Emission probabilities associated with each state

$$e_{ib} = P(x_t = b \mid y_t = i)$$
$$e_{i,b1} + \ldots + e_{i,bM} = 1, \quad \text{for all states } i = 1 \ldots K$$



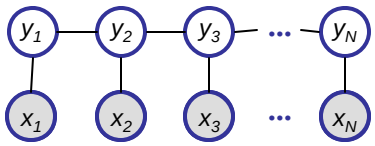
The dishonest casino model



Likelihood of a parse

Given a sequence $x = x_1 \ldots\ldots x_N$
and a parse $y = y_1, \ldots\ldots, y_N$,

To find how likely is the parse:
(given our HMM and the sequence)



$P(x, y) = P(x_1, \ldots, x_N, y_1, \ldots\ldots, y_N)$ (Joint probability)

$= P(y_1)\, P(x_1 \mid y_1)\, P(y_2 \mid y_1)\, P(x_2 \mid y_2) \ldots P(y_N \mid y_{N-1})\, P(x_N \mid y_N)$

$= P(y_1)\, P(y_2 \mid y_1) \ldots P(y_N \mid y_{N-1}) \times P(x_1 \mid y_1)\, P(x_2 \mid y_2) \ldots P(x_N \mid y_N)$

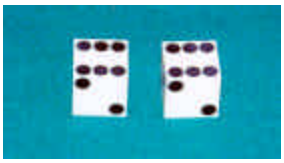
$= P(y_1, \ldots\ldots, y_N)\, P(x_1, \ldots, x_N \mid y_1, \ldots\ldots, y_N)$

$= a_{0,y1}\, a_{y1,y2} \ldots\ldots a_{yN-1,yN}\, e_{y1,x1} \ldots\ldots e_{yN,xN}$

$P(x) = ?_y P(x,y)$ (Marginal probability)

$P(y|x) = P(x, y)/P(x)$ (Posterior probability)

Example: the dishonest casino



Let the sequence of rolls be:

x = 1, 2, 1, 5, 6, 2, 1, 6, 2, 4

Then, what is the likelihood of

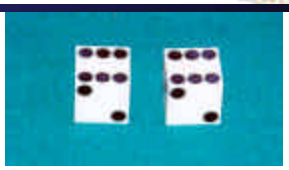
y = Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair?

(say initial probs $a_{0Fair} = ½$, $a_{oLoaded} = ½$)

½ × P(1 | Fair) P(Fair | Fair) P(2 | Fair) P(Fair | Fair) … P(4 | Fair) =

$½ \times (1/6)^{10} \times (0.95)^9 = .00000000521158647211 = 0.5 \times 10^{-9}$

Example: the dishonest casino



So, the likelihood the die is fair in all this run is just 5.21×10^{-9}

OK, but what is the likelihood of

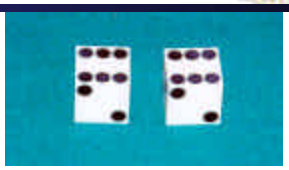
p = Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded?

½ × P(1 | Loaded) P(Loaded | Loaded) … P(4 | Loaded) =

$½ \times (1/10)^8 \times (1/2)^2 (0.95)^9 = .00000000078781176215 = 0.79 \times 10^{-9}$

Therefore, it is after all 6.59 times more likely that the die is fair all the way, than that it is loaded all the way

Example: the dishonest casino



Let the sequence of rolls be:

x = 1, 6, 6, 5, 6, 2, 6, 6, 3, 6

Now, what is the likelihood π = F, F, …, F?

$\frac{1}{2} \times (1/6)^{10} \times (0.95)^9 = 0.5 \times 10^{-9}$, same as before

What is the likelihood

y = L, L, …, L?

$\frac{1}{2} \times (1/10)^4 \times (1/2)^6 \, (0.95)^9 = .00000049238235134735 = 0.5 \times 10^{-7}$

So, it is 100 times more likely the die is loaded

The three main questions on HMMs

1. Evaluation

   GIVEN an HMM ***M***, and a sequence x,
   FIND Prob (x | M)

2. Decoding

   GIVEN an HMM ***M***, and a sequence x,
   FIND the sequence π of states that maximizes, e.g., P(x | π, M)

3. Learning

   GIVEN an HMM ***M***, with unspecified transition/emission probs., and a sequence x,

   FIND parameters $\theta = (e_{ik}, a_{ij})$ that maximize P(x | θ)

Applications of HMMs

Some early applications of HMMs

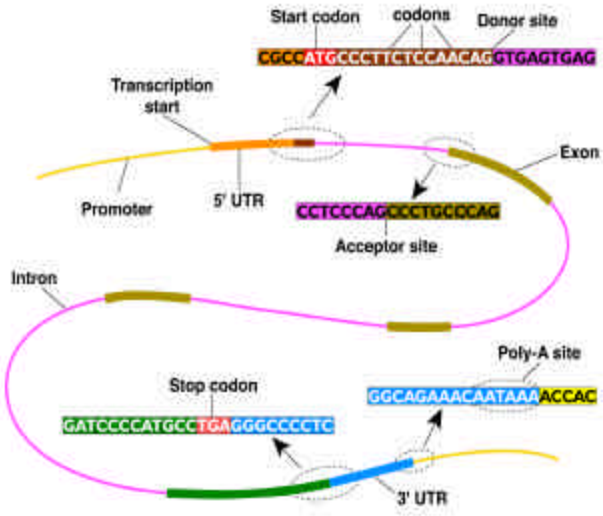
- finance, but we never saw them
- speech recognition
- modelling ion channels

In the mid-late 1980s HMMs entered genetics and molecular biology, and they are now firmly entrenched.

Some current applications of HMMs to biology

- mapping chromosomes
- aligning biological sequences
- predicting sequence structure
- inferring evolutionary relationships
- finding genes in DNA sequence

Typical structure of a gene



Some facts about human genes

- Comprise about 3% of the genome
- Average gene length: ~ 8,000 bp
- Average of 5-6 exons/gene
- Average exon length: ~200 bp
- Average intron length: ~2,000 bp
- ~8% genes have a single exon

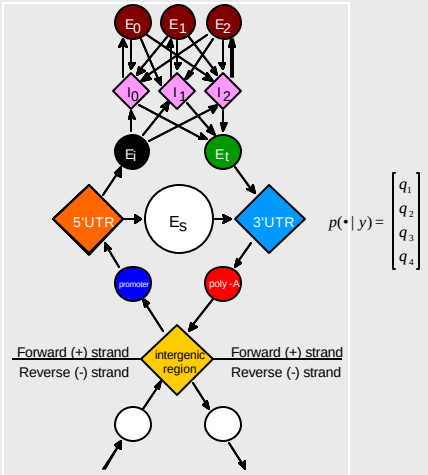
Some exons can be as small as 1 or 3 bp.

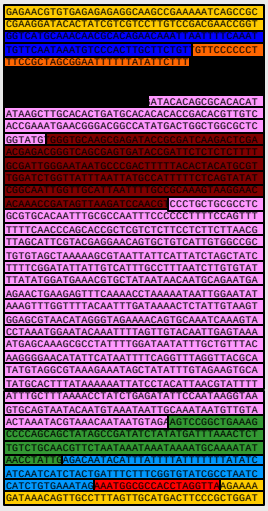
HUMFMR1S is not atypical: 17 exons 40-60 bp long, comprising 3% of a 67,000 bp gene

The idea behind a GHMM genefinder

- **States** represent standard gene features: intergenic region, exon, intron, perhaps more (promotor, 5'UTR, 3'UTR, Poly-A,..).
- **Observations** embody state-dependent base composition, dependence, and signal features.
- In a GHMM, **duration** must be included as well.
- Finally, **reading frames** and **both strands** must be dealt with.

GENSCAN (Burge & Karlin)





The HMM algorithms

Questions:

1. **Evaluation**: What is the probability of the observed sequence? Forward
2. **Decoding**: What is the probability that the state of the 3rd position is Bk, given the observed sequence? Backward
3. **Decoding**: What is the most likely die sequence? Viterbi
4. **Learning**: Under what parameterization are the observed sequences most probable? Baum-Welch (EM)

The Forward Algorithm

We want to calculate

P(x) = probability of x, given the HMM

Sum over all possible ways of generating x:

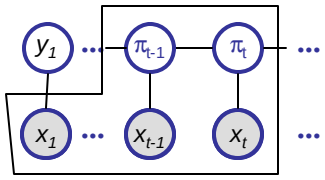
$$P(x) = \Sigma_{\pi} P(x, \pi) = \Sigma_{\pi} P(x \mid \pi)\, P(\pi)$$

To avoid summing over an exponential number of paths π, define

$$f_k(t) = P(x_1 \ldots x_t, \pi_t = k) \qquad \text{(the forward probability)}$$

The Forward Algorithm – derivation

Compute the forward probability:



$$f_k(t) = P(x_1 \ldots x_{t-1}, x_t, \pi_t = k)$$

$$= \Sigma_{\pi t-1} P(x_1 \ldots x_{t-1}, x_t, \pi_{t-1}, \pi_t = k)$$

$$= \Sigma_{\pi t-1} P(x_1 \ldots x_{t-1}, \pi_{t-1}) P(\pi_t = k | \pi_{t-1}, x_1 \ldots x_{t-1}) P(x_t | \pi_t = k, \pi_{t-1}, x_1 \ldots x_{t-1})$$

$$= \Sigma_{\pi t-1} P(x_1 \ldots x_{t-1}, \pi_{t-1}) P(\pi_t = k | \pi_{t-1}) P(x_t | \pi_t = k)$$

$$= \Sigma_i P(x_1 \ldots x_{i-1}, \pi_{i-1} = i) P(\pi_t = k | \pi_{t-1} = i) P(x_t | \pi_t = k)$$

$$= e_k(x_t) \Sigma_i f_i(t-1) a_{ik}$$

The Forward Algorithm

We can compute $f_k(t)$ for all k, t, using dynamic programming!

Initialization:

$f_0(0) = 1$

$f_k(0) = 0$, for all $k > 0$

Iteration:

$f_l(t) = e_l(x_t) \Sigma_k f_k(t\text{-}1) a_{kl}$ (a_{0k} is a vector of initial probability)

Termination:

$P(x) = \Sigma_k f_k(T)$

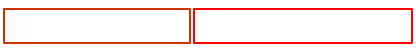
The Backward Algorithm

We want to compute $P(\pi_t = k \mid x)$,

the probability distribution on the t^{th} position, given x

We start by computing

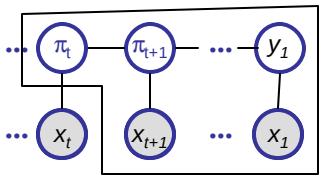
$$P(\pi_t = k, x) = P(x_1 \ldots x_t, \pi_t = k, x_{t+1} \ldots x_N)$$
$$= P(x_1 \ldots x_t, \pi_t = k)\, P(x_{t+1} \ldots x_N \mid x_1 \ldots x_t, \pi_t = k)$$
$$= P(x_1 \ldots x_t, \pi_t = k)\, P(x_{t+1} \ldots x_N \mid \pi_t = k)$$



Forward, $f_k(i)$ **Backward, $b_k(i)$**

The Backward Algorithm – derivation

Define the backward probability:



$b_k(t) = P(x_{t+1} \ldots x_N \mid \pi_t = k)$

$= \Sigma_{\pi t+1} P(x_{t+1}, x_{t+2}, \ldots, x_N, \pi_{t+1} \mid \pi_t = k)$

$= \Sigma_i P(\pi_{t+1} = i \mid \pi_t = k) P(x_{t+1} \mid \pi_t = k) P(x_{t+2}, x_{t+2}, \ldots, x_N \mid \pi_{t+1} = k)$

$= \Sigma_i a_{k,i} e_i(x_{t+1}) P(x_{i+2}, \ldots, x_N \mid \pi_{i+1} = i)$

$= \Sigma_i a_{k,i} e_i(x_{t+1}) b_i(t+1)$

The Backward Algorithm

We can compute $b_k(t)$ for all k, t, using dynamic programming

Initialization:

$b_k(T) = 1$, for all k

Iteration:

$b_k(t) = \Sigma_l e_l(x_{t+1}) a_{kl} b_l(t+1)$

Termination:

$P(x) = \Sigma_l a_{0l} e_l(x_1) b_l(1)$

Posterior Decoding

We can now calculate

$$P(\pi_t = k \mid x) = \frac{f_k(t)\, b_k(t)}{P(x)}$$

Then, we can ask

What is the most likely state at position t of sequence x:

Define π^* by Posterior Decoding:

$$\pi^*_t = \text{argmax}_k\, P(\pi_t = k \mid x)$$

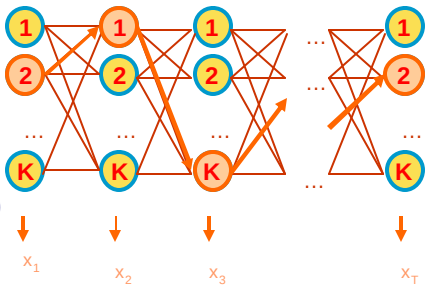
Decoding

GIVEN $x = x_1 x_2 \ldots\ldots x_T$

We want to find $\pi = \pi_1, \ldots\ldots, \pi_T$, such that $P(\pi|x)$ is maximized

$\pi^* = \text{argmax}_\pi\, P(\pi|x) = \text{argmax}_\pi\, P(\pi,x)$

We can use dynamic programming!



Let $V_k(t) = \max_{\{\pi 1, \ldots, \pi i-1\}} P(x_1 \ldots x_{t-1}, \pi_1, \ldots, \pi_{t-1}, x_t, \pi_t = k)$

= Probability of most likely sequence of states ending at state $\pi_t = k$

Decoding – main idea

Given that for each possible state k,
and for a fixed position t,

$V_k(t) = \max_{\{\pi 1,\ldots,\pi t-1\}} P(x_1 \ldots x_{t-1}, \pi_1, \ldots, \pi_{t-1}, x_t, \pi_t = k)$

What is $V_l(t+1)$?

From definition,

$V_l(t+1) = \max_{\{\pi 1,\ldots,\pi t\}} P(x_1 \ldots x_t, \pi_1, \ldots, \pi_t, x_{t+1}, \pi_{t+1} = l)$

$= \max_{\{\pi 1,\ldots,\pi t\}} P(x_{t+1}, \pi_{t+1} = l \mid x_1 \ldots x_t, \pi_1, \ldots, \pi_t)\, P(x_1 \ldots x_t, \pi_1, \ldots, \pi_t)$

$= \max_{\{\pi 1,\ldots,\pi t\}} P(x_{t+1}, \pi_{t+1} = l \mid \pi_t)\, P(x_1 \ldots x_{t-1}, \pi_1, \ldots, \pi_{t-1}, x_t, \pi_t)$

$= \max_k P(x_{t+1}, \pi_{t+1} = l \mid \pi_t = k) \max_{\{\pi 1,\ldots,\pi t-1\}} P(x_1 \ldots x_{t-1}, \pi_1, \ldots, \pi_{t-1}, x_t, \pi_t = k)$

$= e_l(x_{t+1}) \max_k a_{kl}\, V_k(t)$

The Viterbi Algorithm

Input: $x = x_1 \ldots\ldots x_N$

Initialization:

$V_0(0) = 1$ (0 is the imaginary first position)

$V_k(0) = 0$, for all $k > 0$

Iteration:

$V_j(t) = e_j(x_t) \times \max_k a_{kj}\, V_k(t-1)$

$Ptr_j(t) = \text{argmax}_k\, a_{kj}\, V_k(t-1)$

Termination:

$P(x, \pi^*) = \max_k V_k(T)$

Traceback:

$\pi_T^* = \text{argmax}_k\, V_k(T)$

$\pi_{t-1}^* = Ptr_{\pi t}(t)$

Viterbi Algorithm – a practical detail

Underflows are a significant problem

$P(x_1,\ldots., x_t, \pi_1, \ldots, \pi_t) = a_{0\pi 1} a_{\pi 1\pi 2}\ldots\ldots a_{\pi t-1,\pi t} e_{\pi 1}(x_1)\ldots\ldots e_{\pi t}(x_t)$

These numbers become extremely small – underflow

Solution: Take the logs of all values

$V_i(t) = \log e_k(x_t) + \max_k [V_k(t-1) + \log a_{ki}]$

Computational Complexity

What is the running time, and space required, for Forward, and Backward?

Time: $O(K^2N)$
Space: $O(KN)$

Useful implementation technique to avoid underflows

Viterbi: sum of logs

Forward/Backward: rescaling at each position by multiplying by a constant

Model learning: two scenarios

1. **Supervised learning**: estimation when the "right answer" is known

   Examples:

   GIVEN: a genomic region $x = x_1 \ldots x_{1,000,000}$ where we have good (experimental) annotations of the CpG islands

   GIVEN: the casino player allows us to observe him one evening, as he changes dice and produces 10,000 rolls

2. **Unsupervised learning**: estimation when the "right answer" is unknown

   Examples:

   GIVEN: the porcupine genome; we don't know how frequent are the CpG islands there, neither do we know their composition

   GIVEN: 10,000 rolls of the casino player, but we don't see when he changes dice

QUESTION: Update the parameters θ of the model to maximize $P(x|\theta)$

Supervised ML estimation

Given $x = x_1 \ldots x_N$
for which the true $\pi = \pi_1 \ldots \pi_N$ is known,

Define:

A_{kl} = # times $k \rightarrow l$ transition occurs in π
$E_k(b)$ = # times state k in π emits b in x

We can show that the maximum likelihood parameters θ are:

$$a_{kl} = \frac{A_{kl}}{\sum_i A_{ki}} \qquad e_k(b) = \frac{E_k(b)}{\sum_c E_k(c)}$$

(Homework!)

Supervised ML estimation

Intuition: When we know the underlying states,
Best estimate is the average frequency of transitions & emissions that occur in the training data

Drawback:
Given little data, there may be **overfitting**:
$P(x|\theta)$ is maximized, but θ is unreasonable
0 probabilities – VERY BAD

Example:
Given 10 casino rolls, we observe
x = 2, 1, 5, 6, 1, 2, 3, 6, 2, 3
π = F, F, F, F, F, F, F, F, F, F
Then: $a_{FF} = 1$; $a_{FL} = 0$
$e_F(1) = e_F(3) = .2$;
$e_F(2) = .3$; $e_F(4) = 0$; $e_F(5) = e_F(6) = .1$

Pseudocounts

Solution for small training sets:

Add pseudocounts

A_{kl} = # times k→l transition occurs in π + r_{kl}
$E_k(b)$ = # times state k in π emits b in x + $r_k(b)$

r_{kl}, $r_k(b)$ are pseudocounts representing our prior belief

Larger pseudocounts ⇒ strong prior belief

Small pseudocounts (ε < 1): just to avoid 0 probabilities --- smoothing

Unsupervised ML estimation

Given $x = x_1 \ldots x_N$

for which the true $\pi = \pi_1 \ldots \pi_N$ is unknown,

EXPECTATION MAXIMIZATION

0. Starting with our best guess of a model M, parameters θ:
1. Estimate A_{kl}, $E_k(b)$ in the training data
2. Update θ according to A_{kl}, $E_k(b)$
3. Repeat 1 & 2, until convergence

This is called the Baum-Welch Algorithm

We can get to a provably more (or equally) likely parameter set θ each iteration

The Baum-Welch algorithm -- comments

Time Complexity:

iterations $\times$ $O(K^2N)$

- Guaranteed to increase the log likelihood of the model

  $P(\theta \mid x) = P(x, \theta) / P(x) = P(x \mid \theta) / (P(x) P(\theta))$

- Not guaranteed to find globally best parameters

  Converges to local optimum, depending on initial conditions

- Too many parameters / too large model: Overt-fitting