Lecture 17: Hidden Markov Models II

• Review:
  - Path $P(x, \pi) = a_0 \pi_1 \pi_2 \ldots \pi_T a_T$.
  - Viterbi $\pi^* = \arg \max \ P(x, \pi)$.
  - Forward $P(x) = \sum \ P(x, \pi)$.
  - Posterior decoding $P(\pi = k \mid x)$.
  - Supervised learning $\max \ P(x, \pi \mid A)$.
  - Unsupervised learning $\max \ P(x \mid A)$.

Notation: path $\pi$, emissions $x$, state $k$, time $i$.

- $P(x, \pi)$ is the (hidden) path
- $a$ is the (observed) sequence

Probability of observing emissions $x$ with known path $\pi$.

- $P(x, \pi) = a_0 \pi_1 \pi_2 \ldots \pi_T a_T$ and $P(x) = \sum \ P(x, \pi)$

Example: the dishonest casino

$$P(x_1, all-fair) = 0.5$$

$$P(x_1, all-loaded) = 0.05$$

$$\pi_x = \frac{1}{10}$$

$$\times 0.95 \times \times 0.05 \times \times 0.95 \times 0.05$$

Transition: $0.95$ (most likely)

Unsupervised learning max

Posterior decoding $P(\pi)$

Example: the dishonest casino

$$P(x_1, all-loaded) = 0.95$$

$$P(x_1, all-fair) = 0.05$$

$$\pi_x = \frac{1}{10}$$

$$\times 0.95 \times \times 0.05 \times \times 0.95 \times 0.05$$

Transition: $0.95$ (most likely)

Unsupervised learning max

Posterior decoding $P(\pi)$

Viterbi decoding - Finding the most likely path

1. Find path $\pi^*$ that maximizes total joint probability $P(x, \pi)$
   - $\pi^* = \arg \max \ P(x, \pi) = a_0 \pi_1 \pi_2 \ldots \pi_T a_T$

2. Scoring $x$, one path
   - $P(x, \pi)$: Prob of a path, emissions
   - $P(x)$: Prob of emissions, over all paths

3. Viterbi decoding
   - $\pi^* = \arg \max \ P(x, \pi)$

Most likely path

Path containing the most likely state at any time point.

4. Posterior decoding
   - $x^* = \{ \pi^* \ | \ \pi^* = \arg \max \ \sum \ P(\pi \mid k, s) \}$

5. Supervised learning, given $\pi$
   - $A^* = \arg \max \ P(x, \pi \mid A)$

6. Unsupervised learning
   - $A^* = \arg \max \ \sum \ P(x, \pi \mid A)$

- Baum-Welch training, best path

Viterbi algorithm: Calculate $\max \ P(x, \pi)$ recursively

Define $V_i(j) = \text{Probability of the most likely path through state } x=k$.

Compute $V_i(i+1)$ recursively as a function of $V_i(i)$. 

$V_i(i+1) = \max \ V_i(i) \times a_{jk}$

- Assume we know all $V_i$ values for previous time step $i-1$.

- Calculate $V_i(i) = \max \ (V_i(i-1) \times a_{jk})$

- In state $j$ at step $i$.

- Transition from state $j$.

- Emissions $e_{x_i}$

- $a_{jk}$
**The Viterbi Algorithm**

Input: \( x = x_1 \ldots x_N \)

**Initialization:**
\[ V_0(0) = 1, V_k(0) = 0, \text{ for all } k > 0 \]

**Iteration:**
\[ V_k(i) = \operatorname{e}^{x_i} \times \max_j a_{jk} V_j(i-1) \]

**Termination:**
\[ P(x, \pi^*) = \max_k V_k(N) \]

**Traceback:**
Follow max pointers back

**In practice:**
Use log scores for computation

**Running time and space:**
Time: \( O(K^2 N) \)
Space: \( O(KN) \)

**Derivation of Viterbi algorithm**

Let \( V_k(i) = \max_{\pi_1, \ldots, i-1} P[x_1 \ldots x_i, \pi_1 \ldots \pi_{i-1}, \pi_i = k] \)

Calculate \( V_k(i+1) \) recursively as a function of \( V_j(i) \):
\[ V_l(i+1) = \max_{\pi_1, \ldots, i} P[x_1 \ldots x_i, \pi_1, \ldots, \pi_i, x_{i+1}, \pi_{i+1} = l] \]

from definition
\[ = \max_{\pi_1, \ldots, i} P(x_{i+1}, \pi_{i+1} = l | \pi_i) P[x_1 \ldots x_i, \pi_1, \ldots, \pi_i] \]
from Markov property (no memory)
\[ = \max_k P(x_{i+1}, \pi_{i+1} = 1 | \pi_i = k) \max_{\pi_1, \ldots, i} P[x_1 \ldots x_i, \pi_1 \ldots \pi_i, x_{i+1}, \pi_{i+1} = k] \]
from commutativity of multiplication, max
\[ = a_l(x_{i+1}) \max_k a_{lk} V_k(i) \]

from recursive definition of \( V_k \) variable

**2. Model evaluation:**
Total \( P(x|M) \), summed over all paths

Forward algorithm

**Learning**

1. Supervised learning, given \( \pi \)
\[ \Lambda^* = \max_{\pi} P(x, \pi | \Lambda) \]

2. Unsupervised learning
\[ \Lambda^* = \max_{\pi} \arg \max_{\pi} \Sigma P(x, \pi | \Lambda) \]

Viterbi training, best path

Baum-Welch training, over all paths

**Simple: Given the model, generate some sequence \( x \)**

Given a HMM, we can generate a sequence of length \( n \) as follows:
1. Start at state \( \pi_1 \) according to \( \operatorname{prob} a_{01} \)
2. Emit letter \( x_1 \) according to \( \operatorname{prob} e_{\pi_1}(x_1) \)
3. Go to state \( \pi_2 \) according to \( \operatorname{prob} a_{\pi_1 \pi_2} \)
4. \ldots until emitting \( x_n \)

We have some sequence \( x \) that can be emitted by \( \pi \). Can calculate its likelihood. However, in general, many different paths may emit this same sequence \( x \). How do we find the total probability of generating a given \( x \), over any path?

**Complex: Given \( x \), was it generated by the model?**

Given a sequence \( x \),
What is the probability that \( x \) was generated by the model (using any path)?

\[ P(x) = \sum_{\pi} P(x, \pi) \]

- Challenge: exponential number of paths
- (cheap) alternative:
  - Calculate probability over maximum (Viterbi) path \( \pi^* \)
- (real) solution
  - Calculate sum iteratively using principles of dynamic programming
Calculate total probability $\sum P(x,\pi)$ recursively

- Assume we know $f_j$ for the previous time step ($i-1$)
- Calculate $f_k(i) = e_k(x_i) \times \sum_{j} f_j(i-1) a_{kj}$

### The Forward Algorithm – derivation

Define the forward probability:

$$f_l(i) = P(x_1, x_2, ..., x_i, \pi_i = l)$$

$$= \sum_{l=1}^{L} P(x_1, x_2, ..., x_i, \pi_1 = l, ..., \pi_{i-1} = l, \pi_i = l) = \sum_{l=1}^{L} f_l(i-1) a_{kl} e_l(x_i)$$

### The Backward Algorithm

Input: $x = x_1, ..., x_N$

- **Initialization:** $b_k(N) = a_{0k}$
- **Iteration:** $b_k(i) = \sum_l e_l(x_{i+1}) a_{kl} b_l(i+1)$
- **Termination:** $P(x) = \sum_l a_{0l} e_l(x_1) b_l(1)$

### The Backward Algorithm – derivation

Define the backward probability:

$$b_k(i) = P(x_i, ..., x_N | \pi_i = k) = \sum_{l=1}^{L} a_{lk} b_l(i+1)$$

$$= \sum_{l=1}^{L} b_l(i+1) - \sum_{l=1}^{L} b_l(i+1) a_{lk} e_l(x_i)$$

$\sum_{l=1}^{L} b_l(i+1) = d_i(x_{i+1})$ for all $i$
4. Decoding, all paths

Find the likelihood an emission \( x_i \) is generated by a state

\[
\text{Calculate } P(\pi_7 = \text{CpG}+ | x_7 = \text{G})
\]

- With no knowledge (no characters)
  - \( P(\pi_i = k) = \text{most likely state (prior)} \)
  - Time spent in Markov chain states
- With very little knowledge (just that character)
  - \( P(\pi_i = k | x_i = \text{G}) = (\text{prior}) \ast (\text{most likely emission}) \)
  - Emission probabilities adjusted for time spent
- With knowledge of entire sequence (all characters)
  - \( P(\pi_i = k | x = \text{AGCGCG...GATTATCGTCGTA}) \)
  - Sum over all paths that emit ‘G’ at position 7
  - Posterior decoding

Combining the Forward and Backward algorithms

We want to compute:

\[
P(\pi_i = k | x_i), \text{ the probability distribution on the } i^{th} \text{ position, given } x
\]

We can expand it into its forward and backward components:

\[
P(\pi_i = k, x) = P(x_1, \ldots, x_i, \pi_i = k) P(x_{i+1}, \ldots, x_N | x_1, \ldots, x_i, \pi_i = k)
\]

- Forward, \( f_k(i) \)
- Backward, \( b_k(i) \)

Putting it all together: Posterior decoding

For classification, more informative than Viterbi path \( \pi^* \)
- More refined measure of “which hidden states” generated \( x \)
- However, it may give an invalid sequence of states
- Not all \( j \rightarrow k \) transitions may be possible

Summary this far

- Generative model: Hidden states, observed emissions.
  - Generate a random sequence
    - Choose random transition, choose random emission (#0)
  - Scoring the likelihood of a sequence
    - Calculate likelihood of annotated path and sequence (#1)
    - Without specifying a path, total probability of generating \( x \)
    - Sum probabilities over all paths (#2)
  - Decoding: Finding the most likely path, given a sequence
    - What is the most likely path generating entire sequence?
    - Viterbi algorithm (#3)
    - What is the most probable state at each time step?
    - Forward + backward algorithms, posterior decoding (#4)
  - Next: Learning (#5 and #6)

One path

- Scoring \( x \), one path
  \[
P(x, \pi)
\]
  Prob of a path, emissions

- Viterbi decoding
  \[
x^* = \text{argmax}_\pi P(x, \pi)
\]
  Most likely path

- Superseded learning, given \( \pi \)
  \[
  \Lambda^* = \text{argmax}_\Lambda P(x, x|\pi)
  \]
  Viterbi training, best path

All paths

- Scoring \( x \), all paths
  \[
P(x) = \sum \pi P(x, \pi)
\]
  Prob of emissions, over all paths

- Posterior decoding
  \[
x^* = \{ \pi_i | \pi_i = \text{argmax}_\pi \sum \pi_i P(x, \pi|\pi_i) \}
\]
  Path containing the most likely state at any time point.

- Unsupervised learning
  \[
  \Lambda^* = \text{argmax}_\Lambda \sum \pi_i P(x, x|\pi)
  \]
  Baum-Welch training, over all paths

Putting it all together: Posterior decoding

```
<table>
<thead>
<tr>
<th>State</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
```
Two types of learning: Supervised / Unsupervised

5. Supervised learning
infer model parameters given labeled training data

- GIVEN:
  * a HMM \( M \), with unspecified transition/emission probs.
  * labeled sequence \( x \),
- FIND:
  * parameters \( \theta = (E_i, A_{ij}) \) that maximize \( P(x, \pi | \theta) \)

Simply count frequency of each emission and transition, as observed in the training data

6. Unsupervised learning
infer model parameters given unlabeled training data

- GIVEN:
  * a HMM \( M \), with unspecified transition/emission probs.
  * unlabeled sequence \( x \),
- FIND:
  * parameters \( \theta = (E_i, A_{ij}) \) that maximize \( P(x | \theta) \)

Viterbi training:
guess parameters, find optimal Viterbi path (\#2), update parameters (\#5), iterate

Baum-Welch training:
guess parameters, sum over all paths (\#4), update parameters (\#5), iterate

Two learning scenarios

Case 1. When the right answer is known

Example:
GIVEN: a genomic region \( x = x_1…x_{1,000,000} \) 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

Case 2. When the right answer is unknown

Example:
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 \( \theta \) of the model to maximize \( P(x | \theta) \)

Case 1. When the right answer is known

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 \( \theta \) is unreasonable
0 probabilities – VERY BAD

Example:
Given 10 casino rolls, we observe \( x = 2, 1, 5, 6, 1, 2, 3, 6, 2, 3 \)

\( \pi = \pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7, \pi_8, \pi_9, \pi_10 \)

Then:

\( a_{11} = 1, a_{21} = 0, a_{31} = 3, a_{41} = 2, a_{51} = 1, a_{61} = 0, a_{71} = 4, a_{81} = 2, a_{91} = 1 \)

\( e_{11} = .2, e_{12} = .3, e_{13} = .1, e_{14} = 0, e_{15} = e_{16} = .1 \)

5: Supervised learning

Estimate model parameters based on labeled training data

Case 1. When the right answer is known

Given \( x = x_1…x_n \) for which the true \( \pi = \pi_1…\pi_n \) is known,

Define:

\[ a_{ki} = \text{ # times } k \rightarrow l \text{ transition occurs in } \pi \]
\[ e_k(b) = \text{ # times state } k \text{ in } \pi \text{ emits } b \text{ in } x \]

We can show that the maximum likelihood parameters \( \theta \) are:

\[ a_{kl} = \frac{A_{kl}}{\sum_i A_{ki}} \quad \quad e_k(b) = \frac{E_k(b)}{\sum_c E_k(c)} \]

Pseudocounts

Solution for small training sets:

Add pseudocounts

\[ a_{kl} = \# \text{ times } k \rightarrow l \text{ transition occurs in } \pi + r_{kl} \]
\[ e_k(b) = \# \text{ times state } k \text{ in } \pi \text{ emits } b \text{ in } x + r_k(b) \]

\( r_{kl} \) and \( r_k(b) \) are pseudocounts representing our prior belief

Larger pseudocounts \( \Rightarrow \) Strong prior belief

Small pseudocounts (\( \epsilon < 1 \)): just to avoid 0 probabilities
Pseudocounts

Example: dishonest casino

We will observe player for one day, 500 rolls

Reasonable pseudocounts:

\[ r_{RF} = r_{FL} = r_{LF} = r_{FR} = 1; \]
\[ r_{FF} = r_{LL} = r_{TT} = r_{TT} = 1; \]
\[ r_{F1} = r_{F2} = \ldots = r_{F6} = 20 \quad \text{(strong belief fair is fair)} \]
\[ r_{F1} = r_{F2} = \ldots = r_{F6} = 5 \quad \text{(wait and see for loaded)} \]

Above #s pretty arbitrary – assigning priors is an art

6: Unsupervised learning

Estimate model parameters based on unlabeled training data

Learning case 2. When the right answer is unknown

We don’t know the true \( A_{kl}, E_q(b) \)

Idea:

• We estimate our “best guess” on what \( A_{kl}, E_q(b) \) are

• We update the parameters of the model, based on our guess

• We repeat

Estimating new parameters

To estimate \( A_{kl} \):

At each position \( i \) of sequence \( x \),

Find probability transition \( k \rightarrow l \) is used:

\[ P(\pi_i = k, \pi_{i+1} = l | x) = \frac{Q}{P(x)} \times \frac{P(\pi_i = k, \pi_{i+1} = l, x_1, \ldots, x_N)}{P(x)} = \frac{Q}{P(x)} \times \frac{P(\pi_i = k, \pi_{i+1} = l, x, x_{i+2}, \ldots, x_N)}{P(x)} = \frac{Q}{P(x)} \times \frac{P(\pi_{i+1} = l, x_{i+2}, \ldots, x_N | \pi_i = k)}{P(x)} = \frac{Q}{P(x)} \times \frac{b_{k}(i+1) e_l(x_{i+1}) a_{kl}}{P(x)} \]

So:

\[ P(\pi_i = k, \pi_{i+1} = l | x, \theta) = \frac{\zeta(i) a_{kl} e_l(x_{i+1}) b_{k}(i+1)}{P(x | \theta)} \]

(For one such transition, at time step \( i \rightarrow i+1 \))

Estimating new parameters

Case 2. When the right answer is unknown

Starting with our best guess of a model \( M \), parameters \( \theta \):

Given \( x = x_1 \ldots x_N \)

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

We can get to a provably more likely parameter set \( \theta \)

Principle: EXPECTATION MAXIMIZATION

1. Estimate \( A_{kl}, E_q(b) \) in the training data
2. Update \( \theta \) according to \( A_{kl}, E_q(b) \)
3. Repeat 1 & 2, until convergence

Estimating new parameters

(\text{Sum over all } k \rightarrow l \text{ transitions, at any time step } i)

So,

\[ A_{kl} = \sum_{i} P(\pi_i = k, \pi_{i+1} = l | x, \theta) = \sum_{i} \frac{\zeta(i) a_{kl} e_l(x_{i+1}) b_{k}(i+1)}{P(x | \theta)} \]

Similarly,

\[ E_q(b) = \frac{1}{P(x)} \sum_{i} f(i) a_{kb} \]
Estimating new parameters

If we have several training sequences, \(x^1, \ldots, x^M\), each of length \(N\),

\[
f_k(i) = a_{kl} e_k(x_{i+1}) b_{l(i+1)}
\]

\[
A_{kl} = \sum_x \sum_i P(\pi_i = k, \pi_{i+1} = l | x, \theta) = \sum_x \sum_i \frac{P(x \mid \theta)}{P(x)}
\]

Similarly,

\[
E_k(b) = \sum_x \left( \frac{1}{P(x)} \right) \sum_{\{i \mid x_i = b\}} f_k(i) b_k(i)
\]

(Sum over all training seqs, all \(k \rightarrow l\) transitions, all time steps \(i\))

The Baum-Welch Algorithm

Initialization:

Pick the best-guess for model parameters

(or arbitrary)

Iteration:

1. Forward
2. Backward
3. Calculate \(A_{kl}\), \(E_k(b)\)
4. Calculate new model parameters \(a_{kl}\), \(e_k(b)\)
5. Calculate new log-likelihood \(P(x \mid \theta)\)

GUARANTEED TO BE HIGHER BY EXPECTATION-MAXIMIZATION

Until \(P(x \mid \theta)\) does not change much

The Baum-Welch Algorithm – comments

Time Complexity:

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

- Guaranteed to increase the log likelihood of the model

\[P(\theta \mid x) = \frac{P(x, \theta)}{P(x)} = \frac{P(x \mid \theta)}{P(x) P(\theta)}\]

- Not guaranteed to find globally best parameters

Converges to local optimum, depending on initial conditions

- Too many parameters / too large model: Overtraining

The Baum-Welch Algorithm – comments

Time Complexity:

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

- Guaranteed to increase the log likelihood of the model

\[P(\theta \mid x) = \frac{P(x, \theta)}{P(x)} = \frac{P(x \mid \theta)}{P(x) P(\theta)}\]

- Not guaranteed to find globally best parameters

Converges to local optimum, depending on initial conditions

- Too many parameters / too large model: Overtraining

Alternative: Viterbi Training

Initialization:

Same

Iteration:

1. Perform Viterbi, to find \(\pi^*\)
2. Calculate \(A_{kl}\), \(E_k(b)\) according to \(\pi^*\) + pseudocounts
3. Calculate the new parameters \(a_{kl}\), \(e_k(b)\)

Until convergence

Notes:

- Convergence is guaranteed – Why?
- Does not maximize \(P(x \mid \theta)\)
- In general, worse performance than Baum-Welch

What have we learned?

- Generative model. Hidden states, observed emissions.
  - Generate a random sequence
    - Choose random transition, choose random emission \((0)\)
  - Scoring: Finding the likelihood of a given sequence
    - Calculate likelihood of annotated path and sequence
    - Multiply emission and transition probabilities \((01)\)
    - Without specifying a path, total probability of generating \(x\)
    - Sum probabilities over all paths
    - Forward algorithm \((03)\)
  - Decoding: Finding the most likely path, given a sequence
    - What is the most likely path generating entire sequence?
    - Viterbi algorithm \((02)\)
    - What is the most probable state at each time step?
    - Forward + backward algorithm, posterior decoding \((04)\)
  - Learning: Estimating HMM parameters from training data
    - When state sequence is known
      - Simply compute maximum likelihood \(A\) and \(E\) \((05)\)
    - When state sequence is not known
      - Baum-Welch: Iterative estimation of all paths / frequencies \((06)\)
      - Viterbi training: Iterative estimation of best path / frequencies \((08)\)
### The main questions on HMMs

<table>
<thead>
<tr>
<th>1. Scoring $x$, one path</th>
<th>2. Scoring $x$, all paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>= Joint probability of a sequence and a path, given the model</td>
<td>= Total probability of a sequence, summed across all paths</td>
</tr>
<tr>
<td><strong>Given:</strong> a HMM $M$, a path $\pi$, and a sequence $x$</td>
<td><strong>FIND:</strong> the total probability $P(x</td>
</tr>
<tr>
<td><strong>EXAMPLE:</strong> “all fair” vs. “all loaded” comparisons</td>
<td><strong>Forward algorithm,</strong> sum score over all paths (same result as backward)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3. Viterbi decoding</th>
<th>4. Posterior decoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>= Parsing a sequence into the optimal series of hidden states</td>
<td>= Total prob that emission $x_i$ came from state $k$, across all paths</td>
</tr>
<tr>
<td><strong>Given:</strong> a HMM $M$, a sequence $x$</td>
<td><strong>FIND:</strong> the total probability $P(\pi_i=k</td>
</tr>
<tr>
<td><strong>FIND:</strong> the sequence $\pi^*$ of states that maximize $P(x,\pi</td>
<td>M)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5. Supervised learning</th>
<th>6. Unsupervised learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>= Optimize parameters of a model given training data</td>
<td>= Optimize parameters of a model given training data</td>
</tr>
<tr>
<td><strong>Given:</strong> a labeled $x$, with unspecified transition/emission probs., labeled sequence $x$</td>
<td><strong>Given:</strong> a labeled $x$, with unspecified transition/emission probs., labeled sequence $x$</td>
</tr>
<tr>
<td><strong>FIND:</strong> parameters $\theta = (e_i, a_{ij})$ that maximize $P(x</td>
<td>\theta)$</td>
</tr>
<tr>
<td><strong>EXAMPLE:</strong> Viterbi training: guess parameters, find optimal Viterbi path ($\pi^<em>$); update parameters ($\pi^</em>$); iterate</td>
<td><strong>EXAMPLE:</strong> Baum-Welch training: guess, sum over all emissions/transitions ($\pi^<em>$), update ($\pi^</em>$), iterate</td>
</tr>
</tbody>
</table>

### Decoding

- **One path**
  - $\pi^* = \arg\max_{\pi} P(x,\pi)$
  - Most likely path

- **All paths**
  - $\pi^* = \{\pi_i \mid \pi_i = \arg\max_k \sum \pi P(\pi_i=k|x)\}$
  - Path containing the most likely state at any time point.