CS 7180: Behavioral Modeling and Decision-making in AI

Learning Probabilistic Graphical Models

Prof. Amy Sliva
October 31, 2012
Hidden Markov model

- Stochastic system represented by three matrices

<table>
<thead>
<tr>
<th>Description</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N) = number of states</td>
<td>(Q = {q_1, \ldots, q_T})</td>
</tr>
<tr>
<td>(M) = number of observations</td>
<td>(O = {o_1, \ldots, o_T})</td>
</tr>
<tr>
<td>(A) = transition model</td>
<td>(a_{ij} = P(q_{t+1} = j \mid q_t = i))</td>
</tr>
<tr>
<td>(B) = observation model</td>
<td>(b_j(k) = P(o_1 = k \mid q_t = j))</td>
</tr>
<tr>
<td>(\pi) = prior state probabilities</td>
<td>Probability distribution that state (q) is the start state</td>
</tr>
</tbody>
</table>

Environmental context—sequence of states from times \(1-T\)
Sequence of evidence the agent observes at times \(1-T\)
State transition probability matrix
Probability distribution over observations (probability of seeing observation \(o\) in state \(q\))

- Full HMM is a triple \(\lambda = (A,B,\pi)\)
- **First-order Markov** transition model
- **Stationary** transition and observation model
Graphical representation of HMMs

- Assume each state is represented by a **single random variable**
  - Same as Bayesian networks
  - If states have more than one variable, use state space representation
    - Megavariable for state equal to tuple of all values of the individual variables
Matrix representation of HMM

- HMM is a triple $\lambda = (A, B, \pi)$
Three basic HMM problems

**Evaluation**
- Given observation sequence $O = \{o_1, \ldots, o_T\}$ and an HMM $\lambda = (A, B, \pi)$ how do we compute the probability of $O$ given the model
- $P(O \mid \lambda)$—**Forward** and **Backward** algorithms

**Decoding**
- Given observation sequence $O = \{o_1, \ldots, o_T\}$ and an HMM $\lambda = (A, B, \pi)$ how do we find the state sequence $Q = \{q_1, \ldots, q_T\}$ that best explains the observations
- $\arg\max_Q P(Q \mid O, \lambda)$—**Viterbi** algorithm

**Learning**
- How do we adjust the model parameters $\lambda = (A, B, \pi)$ to best fit the sequence
- $\arg\max_{\lambda} P(O \mid \lambda)$
Learning the parameters of an HMM

- Previously assumed underlying model $\lambda = (A,B,\pi)$

- Where do these parameters come from?
  - Estimate transition, observation, and prior probabilities from annotated training data

- Drawbacks of this parameter estimation?
Learning the parameters of an HMM

- Previously assumed underlying model $\lambda = (A,B,\pi)$

- Where do these parameters come from?
  - Estimate transition, observation, and prior probabilities from annotated training data

- Drawbacks of this parameter estimation?
  - Annotation is difficult and/or expensive
  - Training data is different from the current data
Learning the parameters of an HMM

• Previously assumed underlying model $\lambda = (A,B,\pi)$

• Where do these parameters come from?
  • Estimate transition, observation, and prior probabilities from annotated training data

• Drawbacks of this parameter estimation?
  • Annotation is difficult and/or expensive
  • Training data is different from the current data

• Want to maximize the parameters w.r.t. the current data,
  • i.e., Looking for model $\lambda'$, such that $\lambda' = \arg\max_\lambda P(O \mid \lambda)$
  • Want to train HMM to encode observation sequence $O$ s.t. similar sequence will be identified in the future
How can we find the maximal model?

• Unfortunately, no known way to analytically find a **global maximum**
  • i.e., The model $\lambda'$, such that $\lambda' = \arg\max_\lambda P(O | \lambda)$

• But it is possible to find a **local maximum**

• Given initial model $\lambda$, we can always find a model $\lambda'$, s.t.

\[ P(O | \lambda') \geq P(O | \lambda) \]

• Process called **parameter re-estimation**
Parameter re-estimation in HMMs

• Three parameters need to be re-estimated:
  • Initial state distribution: $\pi_i$
  • Transition probabilities: $a_{ij}$
  • Emission (observation) probabilities: $b_i(o_t)$

• Use **forward-backward** (or **Baum-Welch**) algorithm
  • Hill-climbing algorithm
Parameter re-estimation in HMMs

• Three parameters need to be re-estimated:
  • Initial state distribution: $\pi_i$
  • Transition probabilities: $a_{ij}$
  • Emission (observation) probabilities: $b_i(o_t)$

• Use **forward-backward** (or **Baum-Welch**) algorithm
  • Hill-climbing algorithm

• Choose arbitrary **initial parameter instantiation**
  • Forward-backward algorithm **iteratively** re-estimates parameters
  • **Improves probability** that the given observations are generated by the new parameters
General Baum-Welch algorithm

1. Initialise: $\lambda_0$

2. Compute new model $\lambda$, using $\lambda_0$ and observed sequence $O$

3. Set $\lambda_0 \leftarrow \lambda$

4. Repeat steps 2 and 3 until:

$$\log P(O | \lambda) - \log P(O | \lambda_0) < d$$
Re-estimating the transition probabilities

• What is the probability of being in state $i$ at time $t$ and going to state $j$, given the current model and parameters?

$$\xi(i,j) = P(q_t = i, q_{t+1} = j \mid O, \lambda)$$

• Use the **forward** and **backward** probabilities to compute the re-estimated transition probability
Re-estimating the transition probabilities

- **Estimated** transition probability: \( \xi(i,j) = P(q_t = i, q_{t+1} = j \mid O, \lambda) \)

\[
\xi(i,j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_i(o_{t+1}) \beta_{t+1}(j)}
\]
Re-estimating the transition probabilities

- **Estimated** transition probability: \( \xi(i,j) = P(q_t = i, q_{t+1} = j \mid O, \lambda) \)

\[
\xi(i,j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_i(o_{t+1}) \beta_{t+1}(j)}
\]

Forward probability of seeing observations so far, being in state \( i \), and transitioning to state \( j \)
Re-estimating the transition probabilities

- **Estimated** transition probability: \( \xi(i,j) = P(q_t = i, q_{t+1} = j \mid O, \lambda) \)

\[
\xi(i,j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_i(o_{t+1}) \beta_{t+1}(j)}
\]

**Probability of seeing observation** \( o_{t+1} \) **in state** \( j \) **and the backward probability of seeing the rest of the sequence from** \( j \) **to the end**
Re-estimating the transition probabilities

- **Estimated** transition probability: \( \xi(i,j) = P(q_t = i, q_{t+1} = j \mid O, \lambda) \)

\[
\xi(i,j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_i(o_{t+1}) \beta_{t+1}(j)}
\]

Sum over all states \( i \) and \( j \) of seeing observations so far, transitioning to \( j \), and seeing rest of observations to the end
Estimating each transition probability

• **Intuition** behind re-estimation of transition probabilities

\[ \hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to } j}{\text{expected number of transitions from state } i} \]

• **Formally**

\[ \hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j'=1}^{N} \xi_t(i,j')} \]
Estimating each transition probability

• Define state probability of being in state $i$ given the complete observation sequence $O$

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$$

• Sums over forward probability of $i$ and all possible transitions and backward probabilities

• Simplify transition re-estimation:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$
# Review of probabilities

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Forward</strong> probability</td>
<td>$\alpha_t(i)$ Probability of being in state $i$, given the partial observation $o_1,...,o_t$</td>
</tr>
<tr>
<td><strong>Backward</strong> probability</td>
<td>$\beta_t(i)$ Probability of being in state $i$, given the partial observation $o_{t+1},...,o_T$</td>
</tr>
<tr>
<td><strong>Transition</strong> probability</td>
<td>$\xi_t(i,j)$ Probability of going from state $i$, to state $j$, given the complete observation $o_1,...,o_T$</td>
</tr>
<tr>
<td><strong>State</strong> probability</td>
<td>$\gamma_t(i)$ Probability of being in state $i$, given the complete observation $o_1,...,o_T$</td>
</tr>
</tbody>
</table>
Re-estimating the observation probabilities

- **Observation/emission probabilities** re-estimated as

  \[ \hat{b}_i(k) = \text{expected number of times in state } i \text{ where observation is } k \]

  \[ \text{expected number times in state } i \]

- **Formally**

  \[ \hat{b}_i(k) = \frac{\sum_{t=1}^{T} \delta(o_t, k) \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \]

- Where \( \delta(o_t, k) = 1 \text{ iff } o_t = k \), and 0 otherwise

- **Note**: \( \delta \) here is the Kronecker delta function and is not related to the \( \delta \) in the discussion of the Viterbi algorithm!!
Re-estimating initial state probabilities

• **Initial state distribution** $\pi_i$—probability that $i$ is start state

• Re-estimation is easy and intuitive

  $\hat{\pi}_i = \text{expected number of times in } i \text{ at time } 1$

• **Formally** $\hat{\pi}_i = \gamma_1(i)$
Updating the model

- Coming from $\lambda = (A,B,\pi)$ we get to $\lambda' = (A',B',\pi')$ using the following update rules:

\[
\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}
\]

\[
\hat{b}_i(k) = \frac{\sum_{t=1}^{T} \delta(o_t,k) \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}
\]

\[
\hat{\pi}_i = \gamma_1(i)
\]
Expectation maximization

• Baum-Welch is example of **expectation maximization (EM)**

• Expectation maximization has two steps:

1. E step—use current $\lambda$ to estimate the state sequence

   • Compute the expected value for the state at time $t$: $\gamma_t(i)$
   • Compute the expected value of state transitions from $i$ to $j$: $\xi_t(i,j)$
   • Both computed using the forward and backward probabilities

2. M step—compute new parameters $\lambda'$ as expected values given the state sequence estimated in E step

   • Update $\hat{a}_{ij}$, $\hat{b}_i(k)$, and $\hat{\pi}_i$ using rules
   • Maximum likelihood (ML) estimation—maximizes $P(O \mid \lambda')$
What about the structure of the model?

• Baum-Welch assumes we know the **structure** of the HMM
  • Hidden states
  • Observations
  • Conditional independence relationships

• How can we learn the structure?
  • Fully observable domains—find **maximum likelihood** model for data
  • Partially observable domains—use EM

• Applicable to all **probabilistic graphical** models
Structure learning of graphical models

- Search thru the space of possible network structures!
  - For now, assume we observe all variables

- For each structure, learn parameters using EM

- Pick the one that fits observed data best
  - Caveat—won’t we end up fully connected?
  - Add a penalty for model complexity

- Problems?
Structure learning of graphical models

- Search thru the space of possible network structures!
  - For now, assume we observe all variables

- For each structure, learn parameters using EM

- Pick the one that fits observed data best
  - Caveat—won’t we end up fully connected?
  - Add a penalty for model complexity

- Problems?
  - Exponential number of networks!
  - And we need to learn parameters for each!
  - Exhaustive search out of the question!

- So now what?
Learn structure with local search

- Start with some network structure $\lambda$
  - Make a change to the structure i.e., add, delete, or reverse an edge
  - See if the new network is any better, i.e., is $P(O | \lambda') \geq P(O | \lambda)$ ?

- Same principle as using EM for parameter learning

- What should the initial state be?
Learn structure with local search

• Start with some **network structure** $\lambda$
  
  • Make a change to the structure i.e., add, delete, or reverse an edge
  
  • See if the new network is any better, i.e., is $P(O | \lambda') \geq P(O | \lambda)$ ?

• Same principle as using EM for parameter learning

• What should the initial state be?
  
  • Uniform distribution over initial networks
  
  • Network that reflects domain expertise
Process of learning network structure

Prior Network

Improved Network(s)

Data
Unknown structure, full observability

- Local search over structures
  - Search space—possible states, connections
  - Scoring—maximum likelihood, but penalize complex models

- **Maximum likelihood** computation using **Bayes rule**

\[
P(G \mid D) = \frac{P(D \mid G)P(G)}{P(D)}
\]

- Give higher priors for **simple models**
- For **parameters** \( \Theta \) (i.e., CPTs in Bayes nets, \( \lambda \) in HMMs)

\[
P(D \mid G) = \int_{\Theta} P(D \mid G, \Theta) P(\Theta \mid G)
\]
Unknown structure, full observability

- **Maximum likelihood** computation using **Bayes rule**

\[
P(G \mid D) = \frac{P(D \mid G)P(G)}{P(D)}
\]

- Give higher priors for **simple models**

- For **parameters** Θ (i.e., CPTs in Bayes nets, λ in HMMs)

\[
P(D \mid G) = \int_\Theta P(D \mid G, \Theta) \, P(\Theta \mid G)
\]

- Conditional **independence** and **Markov** property allow decomposition of parameters

\[
P(D \mid G) = \prod_i \int_\Theta P(X_i \mid \text{Parents}(X_i), \Theta_i) \, P(\Theta_i) \, d\Theta_i
\]
Learning structure with hidden variables

- The disease variable is hidden
- Can we just learn without it?
End up with fully connected network...

- Sure! But not the best network for our domain

- With 708 parameters? Much harder to learn
Chicken or the egg problem

- If we had **fully observable data** (i.e., knew that a training instance (patient) had the disease...)
  - It would be easy to learn $P(\text{symptom} \mid \text{disease})$
  - But we can’t observe disease, so we don’t have this information

- If we knew the **parameters** (i.e., $P(\text{symptom} \mid \text{disease})$) then it would be easy to estimate if a patient had the disease
  - But we don’t know these parameters!
Use EM to learn the parameters

• Assume we DO know the parameters
  • Initialize *randomly*

• E step: ?
• M step: ?

• Iterate until *convergence*!
Use EM to learn the parameters

• Assume we DO know the parameters
  • Initialize randomly

• E step:
  • Compute probability of instance (in data) having each possible value of the hidden variable
  • Approximate likelihood by computing expected value

• M step:
  • Treating each instance as fractionally having both values compute the new parameter values

• Iterate until convergence!