10-601 Assignment 5

Tanay Gavankar
tgavanka@andrew.cmu.edu
Due: 11/30/12

1: Bayes Net

(a) All Bayes Nets are directed acyclic graphs. So, we can prove that all DAGs must have at least one "root" (no incoming edges) node.

Let $u_1 u_2 ... u_m$ be a path of maximal length in a DAG. Since we know the graph is acyclic, no vertex can occur more than once in any path.

Claim: u_1 has no incoming edges.

If $\overrightarrow{u_0 u_1}$ were an edge in the DAG, then $u_0 u_1 u_2 ... u_m$ would be a path in the graph with length greater than the length of $u_1 u_2 ... u_m$, which contradicts the fact that this path is maximal length.

This shows that all DAGs must have at least one root node, and therefore all DAGs must have at least one root node.

(b) If not all nodes have been sampled, then we know there must be at least one unsampled node. We want to show that this unsampled node either does not have a parent, or all its parents have already been sampled.

Every node is either a free node (no parents) or has parents. If it is a free node, we are done because that means it does not have any parents.

If it has parents, then we claim that all parents have already been sampled. Assume for the sake of contradiction that there exists a parent that was not sampled. This parent node must similarly be a free node or a child node. If it is a free node, it must have been sampled as free nodes are sampled first as per the stochastic inference algorithm, and we have reached a contradiction. If it is a child node, then we can inductively go up the ancestors until we reach a free node, which again must be sampled and gives us a contradiction again. (We know that we will eventually hit a free node because by the properties of DAGs, there cannot be any cycles in a path, and so it is not possible to revisit a node we have already seen in our upwards traversal when searching for the ancestor free node.)

2.a: Hidden Markov Models: Full Likelihood of a Data Set

(1)

$$\begin{aligned} P(X, Z|\Theta) &= P(x_1, ..., x_T, z_1, ...z_T) \\ &= P(z_1) * P(x_1|z_1) * (\prod_{t=2}^{T-1} p(z_t|z_{t-1}) p(x_t|z_t)) * p(z_T|z_{T-1}) \\ &= \pi_{z_1} * \phi_{x_1 z_1} * (\prod_{t=2}^{T-1} a_{z_t z_{t-1}} \phi_{x_t z_t}) * a_{z_T z_{T-1}} \end{aligned}$$

(2)

$$\begin{aligned} P(X|\Theta) &= \sum_Z P(x_1, ..., x_T, z_1, ...z_T) \\ &= \sum_Z P(z_1) * P(x_1|z_1) * (\prod_{t=2}^{T-1} p(z_t|z_{t-1})p(x_t|z_t)) * p(z_T|z_{T-1}) \\ &= \sum_Z \pi_{z_1} * \phi_{x_1 z_1} * (\prod_{t=2}^{T-1} a_{z_t z_{t-1}} \phi_{x_t z_t}) * a_{z_T z_{T-1}} \end{aligned}$$

2.b: Hidden Markov Models: EM for Maximum Likelihood Learning

(1) Since Z has T elements, and $z_t \in \{1..K\}$, this means that there are a total of T^K different Z vectors that have to be inspected. Since we can computer $P(X, Z|\Theta$ in $O(1)$, this means that the total runtime will be $O(T^K)$, bottlenecked by the M step.

(2)

$$\begin{aligned} \xi(z_{t-1}, z_t) &= P(z_{t-1}, z_t|X) \\ &= \frac{P(z_{t-1}, z_t, X)}{P(X)} \\ &= \frac{P(z_{t-1}, z_t, x_1, ...x_{t-1}, x_t, ...x_T)}{P(X)} \\ &= \frac{P(z_t, x_t|x_1, ...x_{t-1}, z_{t-1})P(x_1, ...x_{t-1}, z_{t-1})P(x_t, ..., x_T|z_t)}{P(X)} \\ &= \frac{\alpha(z_{t-1})P(z_t, x_t|x_1, ...x_{t-1}, z_{t-1})P(x_t, ..., x_T|z_t)}{P(X)} \\ &= \frac{\alpha(z_{t-1})P(x_t|z_t)P(z_t|z_{t-1})\beta(z_t)}{P(X)} \end{aligned}$$

Without loss of generality, solve for β in terms alpha, and then plug it into $\sum_j \alpha_t(j)\beta_t(j)$.

(3) The priors cannot change with subsequent updates of the EM algorithm because they are considered base cases, and you perform the algorithm until it converges. You do not modify base cases, and so the priors can never change.

2.c: Hidden Markov Models: A Coin Game

(1) An HMM model for this game is as follows. A state is defined by who flipped the last coin and what the last three flips were. There are 2^3 possible coin flips for three coins, and there are two players, which gives us a total of $2^4 = 16$ states. In addition to that, we will need some states to handle when the game is first started (i.e. when there are fewer than 3 coins tossed). Again, there would be symmetrical states for each player, and a total of $2 + 4$ additional states (2 1-toss and 4 2-toss), giving us 12 more states. The transitions for these states are not included in the

table below because they are edge cases and there is a simple pattern, but the same rules apply (just drop the first or first 2 tosses in each state to get the transition probability). First, we define the states $< S, THT >$ and $< B, THT >$ as termination states, with Selen and Brenden being the winners respectively. The transition probability table is given below with the row indicating the current state, and the value in a column of that row giving the probability of transitioning to the column's state.

	BHHH	BHHT	BHTH	BHTT	BTHH	BTHT	BTTH	BTTT	SHHH	SHHT	SHTH	SHTT	STHH	STHT	STTH	STTT
BHHH	0.3	0.3							0.2	0.2						
BHHT			0.3	0.3							0.2	0.2				
BHTH					0.3	0.3							0.2	0.2		
BHTT							0.3	0.3							0.2	0.2
BTHH	0.3	0.3							0.2	0.2						
BTHT			0.3	0.3							0.2	0.2				
BTTH					0.3	0.3							0.2	0.2		
BTTT							0.3	0.3							0.2	0.2
SHHH									0.5	0.5						
SHHT			0.5	0.5												
SHTH											0.5	0.5				
SHTT							0.5	0.5								
STHH													0.5	0.5		
STHT			0.5	0.5												
STTH															0.5	0.5
STTT							0.5	0.5								

The values of 0.3 were obtained from $0.6 * 0.5$, because Brenden has a 0.6 chance of tossing the next coin, and the values of 0.2 were similarly from $0.4 * 0.5$, where Brenden has a 0.4 chance of losing the cointoss.

The values of 0.5 are the fair coin toss by Selen. Since her switch-coin requirement depends on what she tosses, that is encoded in where the transitions are actually present.

(2) Since you are given an observed list of coin flips, let's call that collection $O = [O_1, ..., O_t]$. We want to find the most probable path Q^* such that $P(Q^*|O) = argmax_Q P(Q|O)$. To do this, we want to find $\delta_t(i) = max_{q_1 ... q_{t-1}} p(q_1 ... q_{t-1} \wedge q_t = s_i \wedge O_1 ... O_t)$. To find this, we simply directly apply the Viterbi algorithm (dynamic programming) to get the path defined by $argmax_j \delta_t(j)$, as that will have the path for $P(Q^*|O)$, and will give us the most probable path (and most probable winner).

3: HMM Programming

(1) $\overrightarrow{z} = [1, 1, 2, 2, 2, 2, 2]$

(2) $\overrightarrow{x} = [3, 1, 3, 1, 2, 3, 3]$

(3) $lnp(smallx, \hat{z}) = -11.639429219$
$lnp(allhotx, \hat{z}) = -12.8921921875$

(4) Viterbi Algorithm on bigX returns:
$[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2]$.

Running this with the exhaustive algorithm is not feasible because it would have to explore 2^{33} different paths.

4: Markov Decision Process

---

(1) We can convert the k-order MDP $M = (S, A, T, R)$ to a first-order MDP M' by the following process.

First, we defined the states in M' to be a k-tuple of states from M. This means that each state in S' represents $(s, s_1, ..., s_{k-1})$.

We preserve the actions, so $A' = A$.

We define the transition function as:
$T'((s, s_1, ..., s_{k-1}), a, s'') = P(s', a, s, s_1, ..., s_{k-1})$ if $s'' = (s', s, s_1, ..., s_{k-2})$, or 0 otherwise.
This transition function comes from the fact that when transitioning to the new state, the history must be preserved correctly via shifting. The 0 transition probability is used for cases where the history is not correctly preserved.

To convert the reward function, R, we note that R is a function of the current state only in S, and we have that information in the representation of S', so we can say that $R'((s_{k-1}, ..., s_1)) = R(s)$.

There is a bijection between M and M'. So, we have demonstrated how to convert a k-order MDP to a first-order MDP. This conversion is similar to the process of converting NFAs (non-deterministic finite automata) to DFAs (deterministic automata), and so both suffer from the issue that the state space grows exponentially during conversion.

(2.a) Since we are only concerned with iterations 1 and 2, we can simply perform the value iteration and populate the table.

t	Jt(S1)	Jt(S2)	Jt(S3)	Jt(S4)
0	0	0	0	0
1	3	5	4	6
2	7	9.64	4	10.8

These were the recursive formulas used:
$J_t(S1) = 3 + 0.8 * max(J_{t-1}(S2), J_{t-1}(S3))$
$J_t(S2) = 5 + 0.8 * max(0.7 * J_{t-1}(S3) + 0.3 * J_{t-1}(S2), 0.8 * J_{t-1}(S4) + 0.2 * J_{t-1}(S2))$
$J_t(S3) = 4$
$J_t(S4) = 6 + 0.8 * J_{t-1}(S4)$
$J_0(s) = 0$

Thus, the requested values are:
$J_1(S2) = 5$
$J_2(S2) = 9.64$

(2.b) There are two action decision points (two actions available on S1 and two actions available on S2). This gives us a total of 4 different "action routes" that can be taken, and so to maximize $J^*(S2)$, we must calculate $J^*(S2)$ with all 4 of the combination routes. We start by creating a system of equations.

This route will be with the action $S1 \rightarrow S2$ and $S2 \rightarrow S3$ being chosen.

$$\begin{aligned} J^*(S1) &= 3 + 0.8 * (1.0 * J^*(S2)) \\ J^*(S2) &= 5 + 0.8 * (0.3 * J^*(S2) + 0.7 * J^*(S3)) \\ J^*(S3) &= 4 \\ J^*(S4) &= 6 + 0.8 * (1.0 * J^*(S4)) \end{aligned}$$

Solving this system gives $J^*(S2) = 9.52632$.

This route will be with the action $S1 \rightarrow S3$ and $S2 \rightarrow S3$ being chosen.

$$\begin{aligned} J^*(S1) &= 3 + 0.8 * (1.0 * J^*(S3)) \\ J^*(S2) &= 5 + 0.8 * (0.3 * J^*(S2) + 0.7 * J^*(S3)) \\ J^*(S3) &= 4 \\ J^*(S4) &= 6 + 0.8 * (1.0 * J^*(S4)) \end{aligned}$$

Solving this system gives $J^*(S2) = 9.52632$.

This route will be with the action $S1 \rightarrow S2$ and $S2 \rightarrow S4$ being chosen.

$$\begin{aligned} J^*(S1) &= 3 + 0.8 * (1.0 * J^*(S2)) \\ J^*(S2) &= 5 + 0.8 * (0.2 * J^*(S2) + 0.8 * J^*(S4)) \\ J^*(S3) &= 4 \\ J^*(S4) &= 6 + 0.8 * (1.0 * J^*(S4)) \end{aligned}$$

Solving this system gives $J^*(S2) = 28.8095$

This route will be with the action $S1 \rightarrow S3$ and $S2 \rightarrow S4$ being chosen.

$$\begin{aligned} J^*(S1) &= 3 + 0.8 * (1.0 * J^*(S3)) \\ J^*(S2) &= 5 + 0.8 * (0.2 * J^*(S2) + 0.8 * J^*(S4)) \\ J^*(S3) &= 4 \\ J^*(S4) &= 6 + 0.8 * (1.0 * J^*(S4)) \end{aligned}$$

Solving this system gives $J^*(S2) = 28.8095$

Thus, the optimal value of $J^*(S2) = 28.8095$. (This also matches the converged value after 63 iterations of a value iteration table.)