CS587

PROBABILITY IN COMPUTING

LECTURE NOTES

AJJEN JOSHI

LECTURES GIVEN ON:

OCTOBER 22, 24 2013

TABLE OF CONTENTS

INTRODUCTORY NOTES:

Wenxin (Feng) and I were scheduled to give two lectures on Markov Chains and their applications. We decided to work together on our lectures to provide a flow between the two lectures, instead of preparing separate lectures on separate topics. For our first lecture on October 22nd, Wenxin introduced concepts regarding Markov chains and Hidden Markov Models, whereas I talked about a randomized algorithm for 2-satisfiability, which applied ideas related to Markov chains. For our second lecture of October 24th, I began by talking about Markov Decision Processes and an interesting example that illustrated the concept: the ipod shuffle algorithm by Peter Norvig. To conclude the lecture, I talked about the Gambler's ruin problem which illustrated concepts about classification of states in a Markov chain, which Wenxin had introduced.

The following document contains the lecture notes which I used for the lectures.

A. EXAMPLE OF A MARKOV CHAIN APPLICATION: A RANDOMIZED ALGORITHM FOR 2-SATISFIABILITY

1. GENERAL SATISFIABILITY (SAT) PROBLEM:

- Boolean formula given as the conjunction (AND) of a set of clauses, where each clause is the disjunction (OR) of literals
- Each literal is a Boolean variable or the negation of a Boolean variable
- A solution to an instance of a SAT formula is an assignment of the variables to the values True or false such that all clauses are satisfied
- The general SAT problem is NP-hard
- In a k-SAT problem each clause has exactly k literals
- 2-SAT is solvable in polynomial time
- e.g. $(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \bar{x}_3) \wedge (x_4 \vee \bar{x}_1)$

2. APPROACH TO FINDING A SOLUTION FOR A 2-SAT PROBLEM

- Start with an assignment
- look for a clause that is not satisfied
- Change the assignment so that the clause becomes satisfied
- Because there are two literals in the clause, then there are two possible changes to the assignment that will satisfy the clause

Note: Goto 4

3. RANDOMIZED 2-SAT ALGORITHM

- i. Start with an arbitrary truth assignment
  - ii. Repeat up to $2mn^2$ times, terminating if all clauses are satisfied:
    - * m: integer parameter that determines the probability that the algorithm terminates with a correct answer
    - n: the number of variables in the formula
    - a) Choose an arbitrary clause that is not satisfied
    - b) Choose uniformly at random one of the literals in the clause and switch the value of its variable
  - iii. If a valid truth assignment has been found, return it
  - iv. Otherwise, return that the formula is unsatisfiable

4. EXAMPLE:

$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$

→ Set x_1, x_2, x_3, x_4 to FALSE

$(x_1 \vee x_2)$ = False

→ Set x_1 to True

$(x_4 \vee \overline{x_1})$ = False

Set x_4 to True

∴ Truth assignment = x_1 = True, x_2 = False, x_3 = False, x_4 = True

5. ALGORITHM ANALYSIS

We are interested in the number of iterations of the while loop executed by this algorithm

→ Treat each time the algorithm changes a truth assignment as a step

→ 2-SAT formula has $O(n^2)$ distinct clauses, so each step can be executed in $O(n^2)$

→ let S: a satisfying argument for the n variables

A_i: variable assignment after the i^{th} step of the algorithm

X_i: number of variables in the current assignment A_i, that have the same value as in the satisfying assignment S.

→ Algorithm can terminate before X_i reaches n if it finds another satisfying assignment, but our analysis of the worst case is that the algorithm only stops when $X_i = n$

→ We consider how long it takes before X_i reaches n

6. LEMMA

Assume that a 2-SAT formula with n variables has a satisfying assignment and that the 2-SAT assignment is allowed to run until it finds a satisfying assignment. Then the expected no. of steps until the algorithm finds an assignment is at most n^2.

7. PROOF

→ If $X_i = 0$, then for any change in variable value on the next step, we have $X_{i+1} = 1$. Hence, $Pr(X_{i+1} = 1 \mid X_i = 0) = 1$

→ Now that $1 \leq X_i \leq n-1$, at each step we choose a clause that is unsatisfied

→ Since S satisfies the clause, that means A_i and S disagree on the value of at least one of the variables in this clause

→ Because the clause has no more than two variables, the probability that we increase the number of matches is at least $\frac{1}{2}$.

→ The probability that we increase the number of matches would be 1 if we are in the case where A_i and S disagree on the value of both variables in this clause. It follows that the probability that we decrease the number of matches is at most $1/2$. Hence, for $1 \le j \le n-1$

$$\Pr(X_{i+1} = j+1 \mid X_i = j) \ge 1/2 \qquad \Pr(X_{i+1} = j-1 \mid X_i = j) \le 1/2$$

→ The stochastic process $X_0, X_1, X_2 \ldots$ is not necessarily a Markov chain, since the probability that X_i increases could depend on whether A_i and S disagree on one or two variables in the unsatisfied clause the algorithm chooses at that step. This, in turn, might depend on the clauses that have been considered in the past.

→ Consider the following Markov chain:

$$Y_0 = X_0$$

$$\Pr(Y_{i+1} = 1 \mid Y_i = 0) = 1; \quad \Pr(Y_{i+1} = j+1 \mid Y_i = j) = 1/2; \quad \Pr(Y_{i+1} = j-1 \mid Y_i = j) = 1/2;$$

The Markov chain $Y_0, Y_1, Y_2, \ldots$ is a pessimistic version of the stochastic process $X_1, X_1, X_2, \ldots$

→ The expected time to reach n starting from any point is longer for the Markov chain Y than for the process X (we will use this fact)

→ This Markov chain models a random walk on an undirected graph G. The vertices of G are the integers $0, \ldots n$ and for $1 \le i \le n-1$, node i is connected to node $i-1$ & node $i+1$

→ Let h_j: the expected number of steps to reach n when starting from j. For the 2-SAT algorithm h_j is an upper bound on the expected number of steps to fully match S when starting from a truth assignment that matches S in j locations.

$h_n = 0$; $h_0 = h_1 + 1$, since from h_0 we always move to h_1 in 1 step

→ let Z_j: a random variable representing the number of steps to reach n from state j

Consider starting from state j, where $1 \le j \le n-1$

$\Pr(\text{next state is } j-1) = 1/2 \Rightarrow Z_j = 1 + Z_{j-1}$

$\Pr(\text{next state is } j+1) = 1/2 \Rightarrow Z_j = 1 + Z_{j+1}$

$E[Z_j] = E\left[\frac{1}{2}(1 + Z_{j-1}) + \frac{1}{2}(1 + Z_{j+1})\right]$

$E[Z_j] = h_j$

Applying linearity of expectations we get: $h_j = \frac{h_{j-1}+1}{2} + \frac{h_{j+1}+1}{2} = \frac{h_{j-1}}{2} + \frac{h_{j+1}}{2} + 1$

→ We therefore have the following system of equations:

$h_n = 0$

$h_j = \frac{h_{j-1}}{2} + \frac{h_{j+1}}{2} + 1 \quad , 1 \le j \le n-1$

$h_0 = h_1 + 1$

→ We can show inductively that for $0 \le j \le n-1 \Rightarrow h_j = h_{j+1} + 2j + 1$

→ It is true when $j = 0$, since $h_1 = h_0 - 1$

→ from $h_j = \frac{h_{j-1}}{2} + \frac{h_{j+1}}{2} + 1$

$$h_{j+1} = 2h_j - h_{j-1} - 2$$
$$= 2h_j - (h_j + 2(j-1) + 1) - 2 \quad \text{(using the inductive hypothesis)}$$
$$= h_j - 2j - 1$$

$\therefore h_0 = h_1 + 1 = h_2 + 1 + 3 = \ldots = \sum_{i=0}^{n-1} 2i + 1 = n^2$

* Hale lemma

8. THEOREM

The 2-SAT algorithm always returns a correct answer if the formula is unsatisfiable. If the formula is satisfiable, then with probability at least $1 - 2^{-m}$, the algorithm returns a satisfying assignment. Otherwise, it incorrectly returns that the formula is unsatisfiable

9. PROOF

If no satisfying assignment

→ then the algorithm correctly returns that the formula is unsatisfiable

If formula is satisfiable

→ Divide the execution of the algorithm into segments of $2n^2$ steps each

→ By lemma, the expected time to find a satisfying argument is bounded by n^2

→ Given that no satisfying assignment was found in the first $(i-1)$ segments, let Z be the no. of steps from the start of segment i until the algorithm finds a satisfying assignment.

Applying Markov's inequality

$$\Pr(Z > 2n^2) \le \frac{n^2}{2n^2} = \frac{1}{2}$$

$\therefore$ The probability that the algorithm fails to find a satisfying assignment after m segments is bounded by $\left(\frac{1}{2}\right)^m$

B. MARKOV DECISION PROCESS EXAMPLE : IPOD SHUFFLE BY PETER NORVIG

1. MARKOV DECISION PROCESS

MDPs provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying a wide range of optimization problems via dynamic programming and reinforcement learning.

2. DIFFERENCE WITH MARKOV CHAINS

MDPs are an extension of Markov chains; the difference is the addition of actions (allowing choice) and rewards (giving motivation). MDPs reduce to Markov chains if only one action exists for each state and all rewards are the same (e.g. zero)

3. THE IPOD SHUFFLE

→ Problem: How to find a song in an ipod shuffle

→ Algorithm: Assuming the songs are sorted alphabetically.

Naive: Press the next button until you come to the desired song

Randomized algorithm: 1) Listen to the current song long enough to identify it

2) If it is alphabetically close you press the 'next' or 'previous' song button in sequential or non-shuffle mode until you arrive at your target

3) If the current song is far away go into shuffle mode and hit next (randomly jumping to another song) until you do get close. Then switch to non-shuffle mode

4. ALGORITHM ANALYSIS

→ How close do you have to get before you switch to non-shuffle mode?

→ How long will it take, on average to find a song with this approach?

→ We need a policy for when to hit the shuffle button and when to switch to the sequential button. Can we determine the optimal policy when we don't know where we'll end up? We can if we treat this as a Markov Decision Process

→ We define the following for the MDP:

States: The state is just the current song. If there are N songs, there are N states.
for a 1GB ipod shuffle, assume N = 250

Actions: We define two actions: Shuffle and Sequential

Sequential: moving all the way to the target (single action consisting of multiple button presses)

Shuffle: pressing next in shuffle mode i.e. jumping to a random song

Transitions: For each (action, state) pair, we enumerate the possible states the model might transition to. for sequential we always transition to the target. for shuffle we transition to each of the other states with equal probability.

Costs: for each transition there is an associated cost. We'll measure the cost in seconds

Sequential costs: 1 second per button press

Shuffle costs: T seconds (takes longer because you have to stop to identify the song and remember where it is in alphabetical order)

→ Basic idea for finding the optimal policy in an MDP is simple:
For each state of the problem, choose the action that minimizes the sum of the cost of the action and the expected cost of getting from the resulting state to the target.

→ Let: $V[s]$ be the value or cost of a state and t be the target state

Assume that the target song is $N/2$. (We can do this without loss of generality because the songs are arranged in a circle and not a line, so the numbering is arbitrary)

Then, the cost of a state is the minimum of

i) the cost of sequentially moving to the target: $|s-t|$

ii) the cost of shuffling and then finding the way to the target: $T + \underbrace{\frac{1}{N}\sum_r V[r]}_{\text{average cost of wherever we end up shuffling}}$

$\therefore V[s] = \min\left(|s-t|, T + \frac{1}{N}\sum_r V[r]\right)$

→ Solve this equation using the value iteration algorithm

→ start with an initial guess for all $V[s]$

e.g. $V[s] = |s-t|$ for all s

→ update the guesses repeatedly, until there are no more changes (or until all changes are smaller than epsilon)

→ to update the value for a state, we check to the see if we could do better by switching to the Shuffle strategy.

5. VALUE ITERATION ALGORITHM

```
def valueiteration(N, T, ε = 0.001):
    t = N/2
    states = range(N)
    V1 = [abs(s-t) for s in states]
    V2 = [0.0 for s in states]
    while max([abs(V2[s] - V1[s]) for s in states]) > epsilon:
        sequentialCost = abs(s-t)
        shufflecost = T + avg([V1[r] for r in states])
        for s in states:
            V2[s] = min(abs(s-t), shufflecost)
        V1, V2 = V2, V1
    return V2
```

6. ANALYTIC SOLUTION

→ What is the best cutoff?

Let cost(i) : average cost of finding a song that has distance i from current song

$cost(0) = 0$

$cost(i) = \min(1 + cost(i-1), T + P)$

P is the average cost of a position resulting from a shuffle

for N is an even number

$$P = \frac{2}{N}(cost(0) + cost(1) + \dots + cost(N/2))$$

for N is an odd number

$$P = \frac{2}{N}\left(cost(0) + cost(1) + \dots + cost(N/2) + \frac{1}{2}\,cost\left(\frac{N+1}{2}\right)\right)$$

The strategy is to shuffle when i is bigger than a threshold τ, that is

for $i \le \tau$: $cost(i) = 1 + cost(i-1) = i$

for $i > \tau$: $cost(i) = T + P$

for even and odd N

$$N/2 \cdot P = (0 + 1 + \dots + \tau) + (N/2 - \tau) \cdot (T + P)$$

$$N/2 \cdot P = (\tau+1) \cdot \tau/2 + (N/2 - \tau) \cdot (T + P)$$

$$P \cdot (N/2 - (N/2 - \tau)) = (\tau+1) \cdot \tau/2 + (N/2 - \tau) \cdot T$$

$$P \cdot \tau = (\tau+1) \cdot \tau/2 + (N/2 - \tau) \cdot T$$

$$P = (\tau+1)/2 + (N/2 - \tau)\frac{T}{\tau} \quad \text{or} \quad P = \frac{(\tau+1)}{2} + \frac{N}{2}\frac{T}{\tau} - T$$

∴ for $i \le \tau$: $cost(i) = i$

$i > \tau$: $cost(i) = \frac{\tau+1}{2} + \frac{N}{2}\frac{T}{\tau} - T$

→ the threshold that minimizes P is the average cost of the strategy

$$dP/d\tau = 0$$

$$\text{or, } \frac{1}{2} - \frac{NT}{2\tau^2} = 0$$

$$\text{or, } \tau = \sqrt{NT} \quad \text{(neglecting the fact that } \tau \text{ should be an integer)}$$

* P can be calculated from the value of τ

C. CLASSIFICATION OF STATES OF A MARKOV CHAIN EXAMPLE: The GAMBLER'S RUIN

1. When a Markov chain has more than one class of recurrent states, we are interested in the probability that the process will enter and thus be absorbed by a given communicating class

2. → Consider a sequence of independent, fair gambling games between two players. In each round a player wins a dollar with probability $\frac{1}{2}$ or loses a dollar with probability $\frac{1}{2}$
   → The state of the system at time t is the number of dollars won or lost by player 1. The initial state is 0
   → The game ends when it reaches the states $-l_1$ (gambler 1 loses all his money) or l_2 (gambler 1 wins all of gambler 2's money). At this point one of the gamblers is ruined.
   → Therefore, this gives us a Markov chain with two absorbing, recurrent states
   → $-l_1$ and l_2 are recurrent states. All the other states are transient, since there is a nonzero probability of moving from each of those states to either state $-l_1$ or state l_2

3. → What is the probability that player 1 wins l_2 dollars before losing l_1 dollars?
   If $l_2 = l_1$, then by symmetry this probability must be $\frac{1}{2}$.

4. → What is the probability for the general case
   → Let P_i^t be the probability that, after t steps, the chain is at state i.
   For $-l_1 < i < l_2$, the state i is transient and so $\lim_{t\to\infty} P_i^t = 0$
   → Let q be the probability that the game ends with player 1 winning l_2 dollars, so that the chain was absorbed into state l_2.
   → Then $1-q$ is the probability the chain was absorbed into state $-l_1$.
   By definition $\lim_{t\to\infty} P_{l_2}^t = q$
   → Since each round of the gambling game is fair, the expected gain of player 1 in each step is 0. Let W^t be the gain of player 1 after t steps.
   Then $E[W^t] = 0$ for any t by induction

   and $E[W^t] = \sum_{i=-l_1}^{l_2} i P_i^t = 0$

   and $\lim_{t\to\infty} E[W^t] = l_2 q - l_1(1-q) = 0$

   $$\therefore \quad q = \frac{l_1}{l_1 + l_2}$$

   → The probability of winning (or losing) is proportional to the amount of money a player is willing to lose (or win)

D. PERSONAL REMARKS

Overall, I thought that the talks went well. For the first lecture on the 2-SAT problem, it took me a few minutes longer than I anticipated while I also felt there were concerns about the legibility of my handwriting on the board. For the 2nd lecture, I made sure to prepare some handouts so that students could focus more on my lecture than on trying to copy the notes from the board, and therefore get a better level of engagement from students.

All in all, this was a great learning experience for me; this exercise not only increased my knowledge about Markov chains and Markov processes but also helped me organize fairly complex material into a (hopefully) articulate lecture.

E. HANDOUT 1

Value Iteration Algorithm for the iPod Shuffle Markov Decision Process

```
def valueiteration(N, T, epsilon=0.001):
    t = N/2
    states = range(N)
    V1 = [abs(s-t) for s in states]
    V2 = [0.0 for s in states]
    while max([abs(V2[s]-V1[s]) for s in states]) > epsilon:
        shufflecost = T + avg([V1[r] for r in states])
        for s in states:
            V2[s] = min(abs(s-t), shufflecost)
        V1, V2 = V2, V1
    return V2
```

HANDOUT 2

MARKOV DECISION PROCESS: IPOD SHUFFLE ALGORITHM

What is the best cutoff? (Given N (no. of songs) and T (time to identify a song) how close do you have to be to the target song to switch from shuffle to sequential mode?)

Let, cost(i) : average cost of finding a song that has distance i from the current song

$cost(0) = 0$

$cost(i) = \min(1 + cost(i-1), T+P)$

P is the average cost of a position resulting from a shuffle

$P = \frac{2}{N}\left(cost(0) + cost(1) + \ldots + cost\left(\frac{N}{2}\right)\right)$ —— (a)

The strategy is to shuffle when i is bigger than a threshold x, that is

for $i <= x$: $cost(i) = 1 + cost(i-1) = i$

for $i > x$: $Cost(i) = T+P$

From (a) $\frac{N}{2} \cdot P = (0 + 1 + \ldots + x) + \left(\frac{N}{2} - x\right) \cdot (T+P)$

$$\text{or, } \frac{N}{2} \cdot P = \frac{x(x+1)}{2} + \left(\frac{N}{2} - x\right)P + \left(\frac{N}{2} - x\right)T$$

$$\text{or, } P\left(\frac{N}{2} - \left(\frac{N}{2} - x\right)\right) = \frac{x(x+1)}{2} + \frac{NT}{2} - xT$$

$$\text{or, } Px = \frac{x(x+1)}{2} + \frac{NT}{2} - xT$$

$$\text{or, } P = \frac{x+1}{2} + \frac{NT}{2x} - T$$

The threshold that minimizes P can be found by

$$\frac{dP}{dx} = 0$$

$$\text{or, } \frac{1}{2} - \frac{NT}{2x^2} = 0$$

$$\text{or, } \frac{NT}{2x^2} = \frac{1}{2}$$

$$\text{or, } x = \sqrt{NT} \quad \text{and } P = \frac{\sqrt{NT}+1}{2} + \frac{NT}{2\sqrt{NT}} - T$$