CS747: Foundations of Intelligent and Learning agents
Assignment 2

Guduru Manoj
200050044

October 12, 2022

Contents

1 Task 1

To implement this task, I had written a class `planner` to enclose all the functions required for this class. I had used a variable `mdp` variable which is a dictionary variable to store various attributes like the reward function, transition function, discount factor etc of the mdp. Various methods of the class are `init, extractmdp, valueiteration, linear programming, evalppolicy, howardpolicyiteration, printoutput`

1.1 Implementation Details

1.1.1 __init__

This is the standard constructor of the class. It extracts the mdp by calling extractmdp function and then calls the required algorithm's function. Then finally it calls printoutput function to print the value function and policy in the required format. It takes arguments $path, algorithm, policy$ where each represent their respective names with default policy being `None`. The class variables are $mdp, policy$ and $valfunc$

1.1.2 extractmdp

This function takes in argument the `path` of the mdp input file and parses the file and extracts the data in the file to fill in the mdp variable. Each attribute of the mdp is stored with its corresponding name, $noofstates$ for number of states in the mdp, $noofactions$ for number of actions in the mdp, $transitionfunc$ to store the transition function of the mdp, it is a 3-dimensional numpy array, $rewardfunc$ to store the reward function of the mdp, it is also a 3-dimensional numpy array, $discountfactor$ to store the discount factor γ of the mdp and $mdptype$ to store the type of the mdp

1.1.3 printoutput

This function is used to print output in the desired manner. It prints the value function's values rounded off to 6 decimal places and the corresponding action space seperated.

1.1.4 valueiteration

This function computes the optimal value function using value iteration. It is written as taught in class. I had made use of the vectored representation of the data to code for this question. To implement it

- I had used two variables `v, vdash` where v is the initial value function and $vdash$ is the policy function after applying the operator
- It is executed in a while(True) loop with the breaking condition being every state's value not changing upto the precision of 8 (took 8 because we should be printing till 6 digits)

- To calculate `vdash`, I had first directly calculated the expression (the one involving the $\sum$) by normal elementwise multiplication and addition and took the sum along the s' axis. Then I took the maximum value of the sum among the different actions possible. `policy` would simply be the index of the maximum value instead of the value itself

1.1.5 linearprogramming

I had first initialised a Linear Program solver and then had initialised the variables for the Linear program. I had named the variables as v_i where i ranges from 0 to $noofstates$. I had kept all these variables in a numpy array which makes it easy to access them whenever needed and to perform operations as well.
Then I had added the objective function and the nk conditions required. Now I had made a call to the solver to solve. Finally, I had accessed the values of the variables using the `pulp` api provided by `pulp`

1.1.6 evalpolicy

This function evaluates the value function for the given policy. As stated in the class, this involves solving the n-linear equations which comprise of n-variables which are the values for each state.
Since we know the action at each state, I had first reduced the transition and reward function from 3-dimensional to 2-dimensional. Now, I had written a vectorised code to make the computation faster. Value function for the given policy will be
$(I_{nxn} - \gamma T)^{-1} \cdot sum(TR,$ along the s' dimension)

1.1.7 howardpolicyiteration

This function calculates the optimal value function using Howard Policy Iteration. To implement it

- I had first created two variables `policy, policydash` which are numpy arrays. These are randomly initialised to suitable values
- To get the value function for the current policy, I used the evalpolicy function.
- Now, policydash is calculated using the value function obtained from above step. It is run in a while loop with the breaking condition being `policy` and `policydash` being identical.

1.2 Observations

Among the three algorithms, value iteration is the easiest to implement and is relatively quicker to output the solution compared to linear program solver and howard policy iteration.

2 Task 2

2.1 MDP formulation

I had formulated the game as an MDP which has $2 \cdot runs \cdot balls + 2$ states where the 2 states correspond to win or loss state and twice the number of $runs \cdot balls$ correspond to states of the form $bbrr0$ or $bbrr1$ where 0 corresponds to the middle order batsman and 1 corresponds to the tailender being on strike.
Total possible actions are 5 as stated in the question. The states corresponding to the middle order batsman have the transitions for each action according to the given parameters whose total possible outcomes are 7 and the transitions for the tailender batsman have the same 3 possible outcomes for every action.

2.2 Implementation Details

I had created classes namely `encoder, decoder` to do the work of encoder and decoder as stated. I had created functions as done in planner's part. Class *enocder* has `__init__, extractparameters, extractstates, encodeasmdp, printoutput` and class *decoder* has `__init__, extractvalpol, extractstates, printoutput` where the names are self-explanatory of what their functionality. The `main` in both files contains parsing the arguments

2.3 Analysis

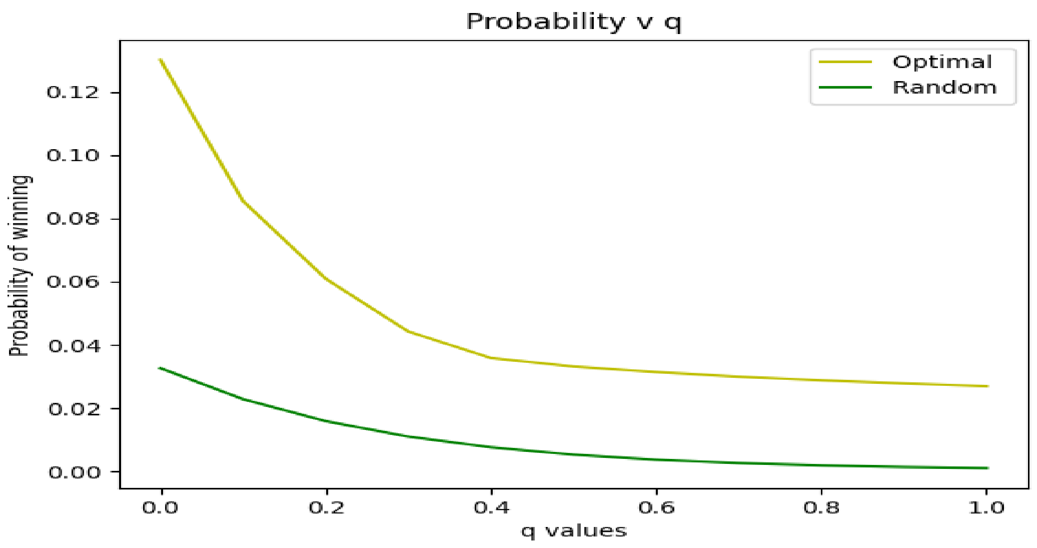

Figure 1: Plot for the probability values v B's weakness

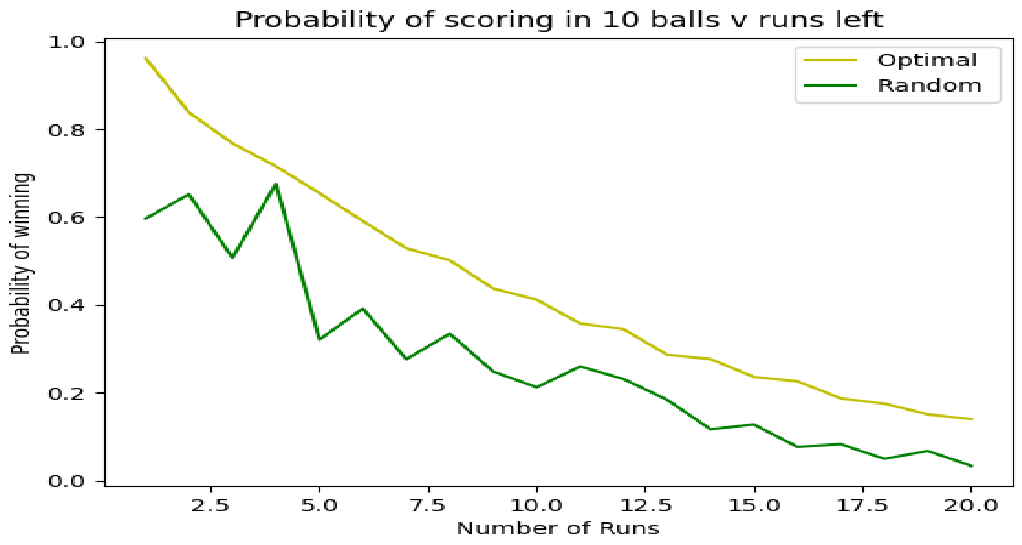

Figure 2: Plot for the probability values v Runs to score

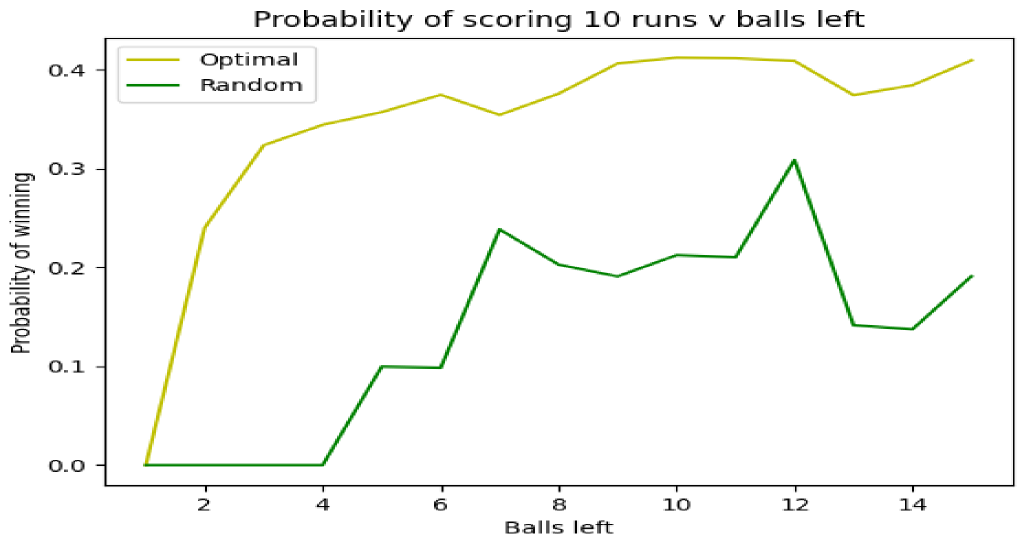

Figure 3: Plot for the probability values v Runs to score

The optimal policy does perform better in almost all cases as desired because it is better than the random policy as the optimal policy always picks the optimal value possible