Rationality Enhancement Thesis Assignment Report

Author:
Ayush Jain

February 23, 2020

Contents

1 Problem Statement 1

Let's assume we have a simple environment where the agent repeatedly chooses between two gambles of the form (x1 with probability p and y1 with probability 1-p) vs. (x2 with probability p and y2 with probability 1-p) where p Uniform([0.5,1]) and x1,x2,y1,y2 Uniform([-10,10]). On each trial the probability and the payoffs of both gambles are different and known to the agent. The agent has 1000 time steps to earn as much money as possible and is equipped with two different decision strategies: the Lexicographic heuristic (choose based on the payoff for the most probable outcome) and the Equal-Weight heuristic (which averages the payoffs and chooses the gamble with the higher average payoff). Each heuristic performs a sequence of operations and each operation takes one time step. Executing the Lexicographic heuristic takes less time than executing the Equal-Weight heuristic because it performs fewer operations. Develop a meta-level RL algorithm that learns to adaptively choose between the two heuristics according to the value of p so that the agent obtains as much reward per time-step as possible. Plot learning curves showing how the learners performance changes over time and how their preference for the two heuristics evolves over time for p¿=0.9 versus p¡=0.6.

If you are not sure how to approach this problem, feel free to take a look at our previous work on meta-cognitive reinforcement learning.

1.1 Solution Approach

The goal of the solution is to build an algorithm which can adaptively choose heuristic (strategy) in different problems. Here the problems are different in the sense of possible outcome values and the probability of their occurring. Each heuristic has its pros and cons. While the Lexicographic algorithm is faster than equal weight heuristic, it totally neglects the lower probability outcomes. Intuitively, if p is close to 0.5, equal weight heuristic is expected to work better because chances of both outcomes are high and it makes more sense to choose the gamble with higher average returns. While in case of p close to 1, one outcome has significantly higher chances of occurring and hence lexicographic approach is better for saving on time. We need to model these intuitions in a meta-reinforcement setting.

1.1.1 Modelling of problem

The problem is of the form of contextual multi-armed bandit. It is contextual because there are context specific information like the outcomes and probability which can be exploited for choosing the heuristic more efficiently. Hence it can be modelled according to the following equations [Lieder and Griffiths, 2017] :

$$VOC(s, problem(i)) = U(s(problem(i)) - cost(s, problem(i))$$

where VOC = Value of computation, U = Expected utility of strategy s on problem i, cost = cost of using strategy s for problem(i). Both U and cost can be represented as a lines expression of features, extracted from the problems as follows:

$$VOC = \sum[w_u * features] - \gamma * \sum[w_t * features]$$

where w_u and w_t are learn-able parameters.

The following sections will describe the formulation of utility and cost for our problem.

1.1.1.1 Utility value

We can model the expected utility in terms of simple features like the maximum possible reward, least possible reward, most probable reward and the probability of each reward.

The mathematical model is inspired from the discussion given in [Lieder and Griffiths, 2017]. We can combine them into the following way:

$$RelativeReward(s, o) = V(s, o)/max(a, o)$$

which calculates the relative reward achieved by the strategy s given outcome o. V(s, o) denotes the reward received by strategy s if outcome is o. max(o, a) gives the maximum possible reward given outcome o.

We can combine the relative reward for all possible outcomes:

$$ExpectedRelativeReward = p*RelativeReward(s, o1)+(1-p)*RelativeReward(s, o2)$$

where p is the probability for outcome o1. Naturally (1 - p) will be the probability for outcome o2.

We can convert this Expected Relative Reward into Absolute reward using the following equation:

$$ExpectedAbsoluteReward = min(r_{\min} + (r_{\max} - r_{\min}) * r_{\text{expected\_relative}}, r_{\max})$$

where $r_{\max}$ and $r_{\max}$ are the maximum and minimum reward possible.

The utility function could also have been learn in a Bayesian approach. The reason I chose the above approach was that I believe that the above equations have managed to extract important information out of the features. Hence I believe learning the weights might not have improved the results drastically. Though it can be done for a useful comparison. Please let me know if my assumption here is flawed.

1.1.1.2 Approximating strategy time cost

The cost of choosing can be represented as :

$$Cost = \gamma * T$$

where γ is the opportunity cost per unit time and T is the expected time required for choosing a given strategy. For our problem, I have assumed $\gamma = 7$ \$ / hour or 0.002 \$ / sec as done in Lieder and Griffiths [2017]. This is a reasonable estimate since the distribution of rewards is uniform between [-10, 10], whose expected value is 0. But to include the effect of time of each heuristic, a small positive value is chosen.

Time of execution of each heuristic is the number of elementary operations which the heuristic has to do to obtain a decision, with time for each elementary operation being 1s. The elementary operations relevant to our problem are: Read, Add, Compare, Result [Johnson and Payne, 1985]. The equal weight heuristic requires 4 Read operations, 2 Add operations (for average), 1 Compare operation for selecting the largest and 1 Result operation. Hence the execution time of Equal Weight Heuristic is 8 sec. Similarly for Lexicographic heuristic, 2 Read operations (of most probable outcome), 1 Compare Operation and 1 Result Operation is required. Hence the cost of execution of Lexicographic heuristic is 4 sec.

Please note that I have not learned the parameters w_t as described before. The reason is that the heuristic computation time is independent of the problem structure in our case.

1.1.2 Implementation

The program is generating a gamble and probability p using the uniform distribution. Then expected VOC is calculated using the features and methodology as described earlier. Using this VOC and epsilon-greedy approach, we select the heuristic to use. Using this heuristic, the corresponding reward is returned. The time for computation

of each heuristic is added into the running count of time, which simulates the delayed return of rewards.

1.2 Results and Discussion

I ran the simulation 5 times for each probability range p ¡= 0.6 and p ¿= 0.9. The final total reward is returned by the program. The program also returns the count of each heuristic utilised. The plot of the experiments can be seen below:

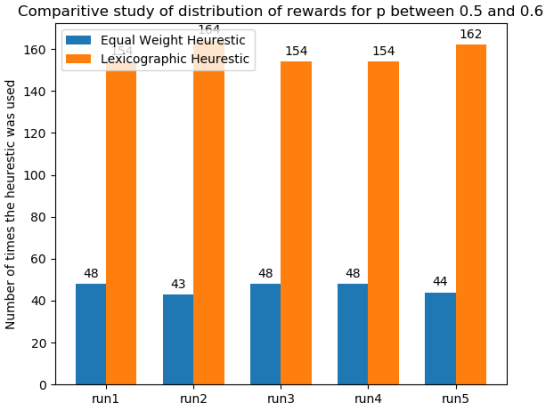

Figure 1: Simulation results for $p <= 0.6$

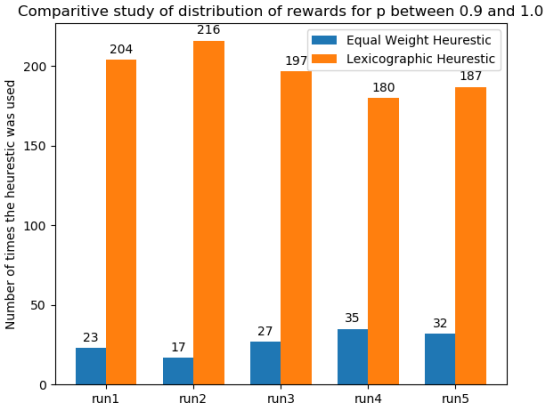

Figure 2: Simulation results for $p >= 0.9$

As we can infer from Figure 1 and Figure 2, when the probability is between 0.5 and 0.6, the model tends to use equal weight heuristic more as compared to the case with high probability ($p >= 0.9$). This captures the intuitive understanding that we need not invest more time in thinking about the average outcome if the probability

of one outcome is very skewed. We should rather look to save on the important time cost.

Another interesting thing to observe is that in the 1st case also, model tend to use lexicographic approach more often than equal weights heuristic. This shows that fast decision making is still the priority and the opportunity cost is higher than the hopeful marginal return of using equal weight heuristic.

1.3 Limitations

I think that there needs to have an update mechanism, which updates our model after doing the epsilon greedy approach. I was not able to figure it out. The research paper [Lieder and Griffiths, 2017] uses Thompson sampling for doing the exploration. I further tried to read this paper [May et al., 2012], but could not comprehend how I can do Thompson sampling in non binomial case, where rewards are not 0 or 1. I would be happy if you could let me know how the Thompson sampling be used and further how can the reward be incorporated in model's parameters.

2 Problem Statement 2

Write a WebPPL program (http://webppl.org) that computes the posterior probability that any given set of strings (e.g., aab, bbaa, aaaaab) was generated by Program 1 versus Program 2. Here each candidate program is represented by a probabilistic context-free grammar. Write your code in such a way that it can be easily extended to handle a larger space of potential programs. Test whether your code is working and be prepared demonstrate and explain the result.

- P(Program 1)=0.7
- P(Program 2)=0.3
- Program 1:
- P(S–¿aSb)=0.5
- P(S–¿bSa)=0.2
- P(S–¿a)=0.1
- P(S–¿b)=0.2
- Program 2:
- theta Beta(0.1,0.1)

- P(S–¿Xa)=theta
- P(S–¿bY)=1-theta
- P(Y–¿aY)=0.4
- P(Y–¿b)=0.6
- P(X–¿Xb)=0.2
- P(X–¿a)=0.8

2.1 Solution Approach

The code is saved in webppl.txt. The graphs for posterior are shown as below:

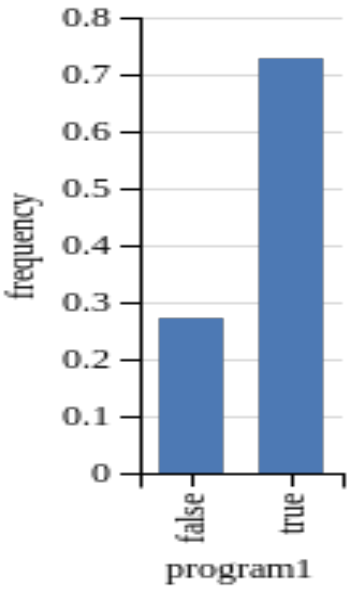

Figure 3: Posterior Distribution of Program 1

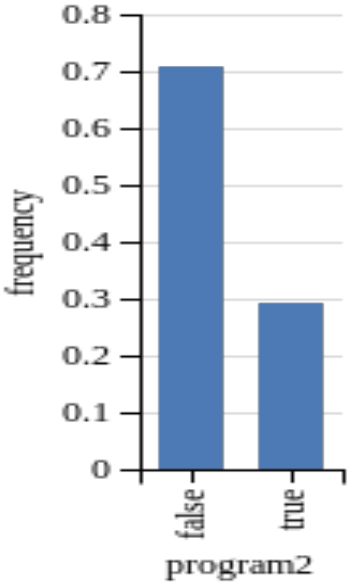

Figure 4: Posterior Distribution of Program 2

References

Eric J Johnson and John W Payne. Effort and accuracy in choice. *Management science*, 31(4):395–414, 1985.

Falk Lieder and Thomas L Griffiths. Strategy selection as rational metareasoning. *Psychological Review*, 124(6):762, 2017.

Benedict C May, Nathan Korda, Anthony Lee, and David S Leslie. Optimistic bayesian sampling in contextual-bandit problems. *Journal of Machine Learning Research*, 13(Jun):2069–2106, 2012.