CS6700 Written Assignment - II

Sabarinath N P *(CS10B020)*

Department of Computer Science and Engineering
IIT Madras

September 2013

1.1 Exercise 1

Question: In the comparison shown (between ϵ-greedy and greedy method), which method will perform best in the long run in terms of cumulative reward and cumulative probability of selecting the best action? How much better will it be?

Solution: The three policies under consideration are greedy, 0.01-greedy, 0.1-greedy. For the average reward obtained at each step, it is clear that 0.01-greedy dominates other two policies. Similar pattern is observed for % Optimal Action choosen.

When it comes to Cumulative measures, we can argue that the value of cumulative average reward point $F(x)$ after x steps is directly proportional to the area under the average reward plot $f(x)$ from X = 0 and X = x. That is,

$$F(x) \propto \int_0^x f(x)\,dx.$$

Now using the above proportionality and the plot $f(x)$, one might argue that for 0.1-greedy policy still outperforms 0.01-greedy. One important reason is that $\epsilon = 0.1$ policy does more exploration and plays global optimal moves more often than $\epsilon = 0.01$ polciy in the first few steps. This might affect the relative performance of $\epsilon = 0.01$ in-terms of cumulative measures. But when the number of games $n \to \infty$, $\epsilon = 0.01$ policy explores less often and plays the global optimal moves more often and thus the area under the plot $f(x)$ continuous to grow more than that of $\epsilon = 0.1$. This can be attributed by the fact that "Only for finite number of steps $\epsilon = 0.1$ outperforms $\epsilon = 0.01$". Also, we know that "Area under a curve $f(x)$ will be less than that of any infinitely increasingly dominating curve $F(x)$". This is the exact case after both the policies (or their $f(x)$) converge.

1.2 Exercise 2

Question: Consider a class of simplified supervised learning tasks in which there is only one situation (input pattern) and two actions. One action, say a, is correct and the other, b, is incorrect. The instruction signal is noisy: it instructs the wrong action with probability p; that is, with probability p it says that b is correct. You can think of these tasks as binary bandit tasks if you treat agreeing with the (possibly wrong) instruction signal as success, and disagreeing with it as failure. Discuss the resulting class of binary bandit tasks. Is anything special about these tasks? How does the supervised algorithm perform on these tasks?

Solution: Given that with probability p, the instructor returns wrong action b. In the bandit problem case, it is given that agreement of agent's action selection with the instructor's signal is considered success and disagreement is considered failure. So, when it chooses action a, there is $(1 - p)$ probability that it is a success. And similarly when it chooses action b, there is p probability that is is a success. Hence, we can say easily infer from above argument that irrespective of the value of p this problem falls in into lower-right or uppper-left quandrants which correspond to easy problems for which supervised learning might work well.

Also the given binary bandit problem is special in way that probability of success of one action equals to probability of failure of the other action. Hence resulting into set of easy problems.

1.3 Exercise 3

Question: If the step-size parameters, $\alpha_k(a)$ are not constant, then the estimate Q_k is a weighted average of previously received rewards with a weighting different from that given by

$$Q_k = (1-\alpha)^k Q_0 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} r_i$$

What is the weighting on each prior reward for the general case?

Solution: In non-stationary case, the step-size parameter $\alpha_k(a)$ varies with k. With this, the estimate of Q_k in general scenario can be obtained as follows,

$$Q_k = Q_{k-1} + \alpha_k[r_k - Q_{k-1}]$$

$$Q_k = (1-\alpha_k)Q_{k-1} + \alpha_k r_k$$

$$Q_k = \alpha_k r_k + (1-\alpha_k)[\alpha_{k-1} r_{k-1} + (1-\alpha_{k-1})Q_{k-2}]$$

$$Q_k = \alpha_k r_k + \alpha_{k-1}(1-\alpha_k) r_{k-1} + \alpha_{k-2}(1-\alpha_{k-1})(1-\alpha_k) r_{k-2} + ... + Q_0 \prod_{i=1}^{k}(1-\alpha_i)$$

$$Q_k = \alpha_k r_k + \sum_{i=0}^{k-1} \alpha_i r_i \prod_{j=i+1}^{k}(1-\alpha_j) + Q_0 \prod_{i=1}^{k}(1-\alpha_i)$$

The above formula can be used for general weighted update of the estimate, based on the prior rewards, by carefully choosing the variation in α_k values such that the following condition holds,

$$\alpha_k + \sum_{i=0}^{k-1} \alpha_i \prod_{j=i+1}^{k}(1-\alpha_j) + \prod_{i=1}^{k}(1-\alpha_i) = 1$$

Also the weight of each prior reward is given by,

$$W(r_i) = \begin{cases} \alpha_i & \text{if } i \text{ is k} \\ \alpha_i \prod_{j=i+1}^{k}(1-\alpha_j) & \text{if } 0 < i < k \end{cases}$$

1.4 Excercise 4

Question: The results shown in Figure should be quite reliable because they are averages over 2000 individual, randomly chosen 10-armed bandit tasks. Why, then, are there oscillations and spikes in the early part of the curve for the optimistic method? What might make this method perform particularly better or worse, on average, on particular early plays?

Solution: By the definition of the optimistic method, it is clear that this policy might choose to play more exploratory moves than the greedy moves in the first few steps. This is attributed by the fact that initial estimates are probably higher than the highest reward attainable. So, the rewards obtained after each action tends to lower the average estimate. Hence the oscillations and spikes arise in the plot of % optimal action.

The spike corresponds to initial step in which relatively large (compared to other first few steps) number of the bandit games choose to play optimal action. But due to poor reward compared to its initial estimate, the bandits never play that optimal action until it explored all the moves at-least once and updated the their estimates.

This method performs same as greedy (which is relatively better for initial steps only) if the initial estimates are not bigger the actual rewards. Also it might perform worse in the early stages if initial estimates are very low compared to the actual rewards. That is, it might choose to play bad moves repeatedly and will never converge to optimal action (as the optimal action's initial estimate might be very low when compared to the reward of an non-optimal action).

In addition to above variations, the algorithm can decrease the estimate of all the actions based on the rewards it has seen after the 1^{st} play of each move in the initial few steps . Thus in this case, it might not perform the as bad as normal optimistic approach as this lowers the estimate of all the actions so that the optimal action will get to be chosen more often in the initial steps.

1.5 Exercise 5

Question: The reinforcement comparison methods described here have two step-size parameters, α and . Could we, in general, reduce this to one parameter by choosing $\alpha = \beta$? What would be lost by doing this?

Solution: In many problems, the agent might desire to increase the probability of choosing a good action more than the reference reward after each step. That is, it might be desired to have rate by which preference for choosing optimal action increasing more than the rate at which reference reward reaches the optimal reward. Also, in the algorithm β is defined to any positive constant. If $\alpha = \beta$, then the value of beta is constrained to $(0, 1)$. This might affect the convergence rate.

Hence, in the case of $\alpha = \beta$ the agent might converge to the optimal action slowly especially when α chosen is very low. But faster convergence rate could have been achieved if two different step-size parameters were employed. But in general, there in nothing harm in choosing the parameters such that $\alpha = \beta$.

1.6 Exercise 6

Question: An ϵ-greedy method always selects a random action on a fraction of the time steps. How about the pursuit algorithm? Will it eventually select the optimal action with probability approaching 1?

Solution: The pursuit algorithm plays exploratory moves in the initial steps of the game. With number of steps $n \to \infty$ the algorithm increases the probability of choosing the optimal greedy action by decreasingly small amounts and also decreases the probability of choosing other actions by decreasingly small amounts. The increase or decrease is monitored by the step-size parameter β as given in the following equations,

$$\pi_{t+1}(a^*_{t+1}) = \pi_t(a^*_{t+1}) + \beta[1 - \pi_t(a^*_{t+1})]$$

$$\pi_{t+1}(a) = \pi_t(a) + \beta[0 - \pi_t(a)] \quad \text{if a} \neq a^*_{t+1}$$

Here the probability of choosing greedy optimal action $\pi_t(a) \to 1$ as $t \to \infty$. This is because the greedy action is chosen as follows,

$$a^*_{t+1} = argmax_a Q_{t+1}(a)$$

And whenever the optimal action is played its corresponding estimate is also increased thus leading to increase in the frequency of playing the optimal action. Thus optimal action will be played infinitely morethan other action as the number of steps $n \to \infty$ Hence, the optimal action will eventually be choosen with probability 1 as the other action probabilities tend to 0 as number of steps $n \to \infty$ given that the environment is stationary.

1.7 Exercise 7

Question: The pursuit algorithm described above is suited only for stationary environments because the action probabilities converge, albeit slowly, to certainty. How could you combine the pursuit idea with the ϵ-greedy idea to obtain a method with performance close to that of the pursuit algorithm, but that always continues to explore to some small degree?

Solution: The main reason behind good performance of pursuit algorithm is that the probability of choosing optimal action tends to 1. So, to get similar performance, introduction of randomness should not quite affect the convergence of probabilities.

The idea of ϵ-greedy can be introduced in the following way. At each step with some probability ϵ the agent chooses to make random move. And with probability (1-ϵ) it chooses to play according to the probabilities defined by the pursuit algorithm. In this way, the the pursuit algorithm always continuous to explore irrespective of the probabilities of the actions and also gives the performance comaparable to the actual pursuit algorithm. In few cases where the environment is non-stationary, this variation might outperform the original algorithm.

1.8 Exercise 8

Question: Suppose you face a binary bandit task whose true action values change randomly from play to play. Specifically, suppose that for any play the true values of actions and are respectively 0.1 and 0.2 with probability 0.5 (case A), and 0.9 and 0.8 with probability 0.5 (case B). If you are not able to tell which case you face at any play, what is the best expectation of success you can achieve and how should you behave to achieve it? Now suppose that on each play you are told if you are facing case A or case B (although you still don't know the true action values). This is an associative search task. What is the best expectation of success you can achieve in this task, and how should you behave to achieve it?

Solution: In the first scenario, when the information about the bandit game at each step is unspecified. Let us assume that the agent knows the knowledge that there can be many games and also that there only two actions. After each action, the agent receives some reward r_i. If the reward is highly different from the average rewards seen before, the agents attributes it to a new game. So, in this way the agent can learns that there are two games each with probability close to 0.5 and after sufficient number of steps it plays each action assuming the current game is of case A with probability ~ 0.5 and case B with probability ~ 0.5. Thus the expected reward after large number of steps is

$$E(r_i) = 0.5 * (0.1 + 0.2) + 0.5 * (0.9 + 0.8)$$

$$E(r_i) = 10.0$$

This because if it plays each action assuming that it is case A, then there is equal probability that the game might be case B. And also the optimal action of the cases are opposite.

In the second scenario, when the agent knows the exact case which it is playing, then it can follow associative learning algorithms to get better performance. In this way it can learn to play optimal moves in both the games. Thus increasing the expected reward. Say it eventually learns to plays optimal move in both the games (using pursuit algorithm), then

$$E(r_i) = 0.2 + 0.9$$

$$E(r_i) = 11.0$$

1.9 Exercise 9

Question: Consider a variation of the Median Elimination Algorithm(MEA), where in each round, the bottom quartile(the lowest 25% instead of half) of the arms are eliminated. In order to achieve the same ϵ-delta guarantees, how many times should each arm be played and how should epsilon and delta be cooled in each round? How does the total complexity compare to that of MEA?

Solution: For the modified algorithm to satisfy the same PAC guarantees at iteration l,

$$P[max\ p_j \leq max\ p_i + \epsilon_l] \geq 1 - \delta_l$$

it must obey the following condition,

$$P[\hat{p_1} \leq p_1 - \epsilon_1/2] + P[\hat{p_j} \geq p_i \mid \hat{p_1} \geq p_1 - \epsilon_1/2] \leq \delta_1$$

where p_j is bad arm.

For this condition to be satisfied when $n_l = \frac{n_{l-1}}{4}$, rewriting the above formula we have,

$$P[\#bad \geq \frac{3n}{4} \mid \hat{p_1} \geq p_1 - \epsilon_1/2] \leq \frac{\frac{3n\delta_1}{7}}{\frac{3n}{4}} = \frac{4\delta_1}{7}$$

The above value of $\frac{4\delta}{7}$ is obtained using the following expression

$$\frac{\delta_1}{x} + \frac{n\delta_1}{\frac{3nx}{4}} = \delta_1$$

$$x = \frac{3}{7}$$

Thus it satisfies the PAC guarantees. Let us consider the initial values as $\epsilon_1 = \frac{\epsilon}{4}$ and $\delta_1 = \frac{\delta}{2}$. Now the reduction in the parameters can be done as follows, $\epsilon_l = \frac{3\epsilon_{l-1}}{4}$ and $\delta_l = \frac{\delta_{l-1}}{2}$ Also, the number of times each arm must be sampled L is given by

$$e^{-(\frac{\epsilon_l}{2})^2 L} = \frac{3\delta_1}{7}$$

Therefore number of times each arm must be sampled in an iteration is given by

$$L = \frac{1}{(\frac{\epsilon_l}{2})^2} \log_2(\frac{7}{3\delta_1})$$

The overall sample complexity can be found out as follows, the number of iterations is $N = \log_4(n)$. The number of arm samples in l-th round is $n_l L$ So, *sample complexity* is

$$\sum_{l=1}^{\log_{4/3}(n)} \frac{n_l \log(\frac{7}{3\delta_l})}{(\frac{\epsilon_l}{2})^2} = 4 \sum_{l=1}^{\log_{4/3}(n)} \frac{\frac{n}{2^{l-1}} \log_2(\frac{2^l 7}{3\delta})}{((\frac{3}{4})^{l-1}\frac{\epsilon}{4})^2}$$

$$Samplecomplexity = 64 \sum_{l=1}^{\log_{4/3}(n)} n(\frac{8}{9})^{l-1}(\frac{\log(\frac{1}{\delta})}{\epsilon^2} + \frac{\log(\frac{7}{3})}{\epsilon^2} + \frac{l\log(2)}{\epsilon^2})$$

$$Samplecomplexity \leq 64\frac{n\log(\frac{1}{\delta})}{\epsilon^2} \sum_{l=1}^{\infty} (\frac{8}{9})^{l-1}(lC' + C)$$

$$Samplecomplexity = \mathrm{O}\big(\frac{n\log(\frac{1}{\delta})}{\epsilon^2}\big)$$

Hence the asymptotic complexity remains the same though the number of arm samples in each round is different from that of *Medium Elimination Algorithm*

1.10 Exercise 10

Question: What is the optimal percentage of arms that should be dropped in terms of total complexity?

Solution: When considering only the sample complexity of the algorithm, the number of arm samples in each round is inversely related to how large the drop percentage of arms is. Also, number of rounds performed is directly proportional to $\log_{\frac{1}{1-x}}$ when x is the fraction of arms dropped in each round.