NAME: Maxfield Green
DATE:2/4/2019
Homework: Assignment 3
STAT/CS 387

Problem 1

Implement the Multi-armed Bandit (MAB) model so that you may simulate the problem with algorithms for below. Take the following steps.

- sets up the probability distributions and (true) mean rewards for each of the N arms,
- returns the reward r_i taken from distribution D_i when the arm i is played,
- implements gambles, the sequential playing of arms over T time-steps,
- computes the regret R of a strategy as a function of the time over the course of a gamble.

Describe the implementation of MAB simulations and discuss and motivate your choice of reward distribution.

1.1 Solution

The multi-armed bandit problem addresses resources allocation between competing investment choices when each choice has a partially unknown reward or consequence that will potentially become clearer over time. An example application would be choosing to gamble on one-of-k slot machines or more practically A-B testing in which the optimal effective treatment is unknown. Throughout this assignment report, I use the gambling example referring to different slot machines as "arms" which have different reward distributions. As advised, I used a beta distribution to sample from for each of the 'slot arms'. The beta distribution is advantageous over others as there are two parameters that we can tune, α and β. These parameters allow us to control the shape of the resulting distribution. The PDF of the beta function is given as:

$$\frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)} \tag{1}$$

$$\text{where} B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \tag{2}$$

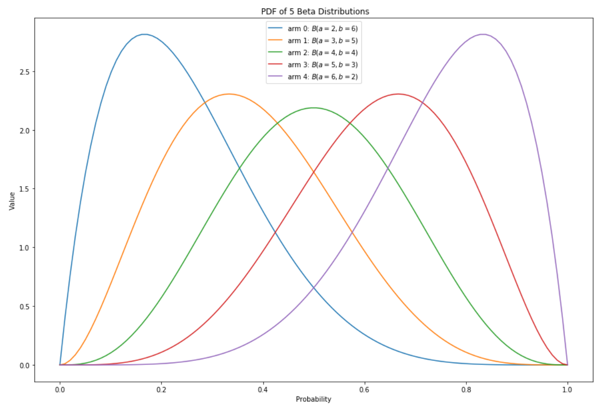

Figure 1: PDF of Beta Distributions

I discuss manipulating the beta distribution further in Section Three when different reward distributions are generated.

2 Problem 2

Implement four strategies for selecting arms during the course of a gamble.

- Random
- (Naive) greedy
- ϵ-first greedy
- greedy

Describe the strategies, describe and document the code, discuss how you ensure your strategies are not "cheating" by seeing information they should not have access to.

2.1 Solution

All four algorithms are fairly simple. I will provide brief pseudo code and comments bellow for each strategy.

For all strategies, regret is calculated as :

Time*Maximum Expected Reward - sum(Rewards from current gamble)

2.2 Random Method

The Random method is quite simple and plays as follows:

- for t in gamble duration:
  - i = random int(1,N) where N is number of bandit arms being considered
  - r = reward from playing arm i
  - Calculate regret from gamble, add to regret history

This method is not very good. An evaluation of its performance is given in part 3 of the report.

2.3 Naive Greedy

The Naive Greedy Method is slightly better sometimes and is outlined below:

- for t in time:
  - i = argmax$\hat{r_i} | i \in (1,..,N)$ where $\hat{r_i}$ is the average reward of arm i
  - r = reward from playing arm i
  - Update $\hat{r_i}$

One note about this algorithm is that I initialize the expected reward for each at 1/2 and randomly choose the first arm to pull. This way, the first pull is unbiased. I choose to update the expected reward $\hat{r_i}$ after each pull, so that if a poor arm is pulled and generates a very good reward by chance, the algorithm can still recover. I also investigated a m = 1 first greedy scenario for the naive greedy method. Both are discussed in the analysis section. I found that initializing the greedy algorithm in different ways really effected how well the algorithm performed. This suggests that performance is highly predicated on initial state and not subsequent decisions. It is not robust to small changes in initial state.

2.4 ϵ first greedy

The ϵ-first greedy is different from the naive greedy algorithm in that there is an exploratory phase of length m. Additionally, the expected reward for each are is not updated after each pull. Once the expected best arm has been identified, the algorithm will commit to pulling that arm for the remainder of the gamble. To achieve optimal rewards, the parameter m is fixed at $\frac{2}{\delta^2} log(2NT)$. In class we proved that this is will minimize the regret burden from exploration while maximizing the reward return. However, we can't allow the delta term to enter our m, as this would be cheating!

```
• for t in exploration time:
    – i = argmax $(\hat{r}_i | i \in (1,..,N))$ where $\hat{r}_i$ is the average reward of arm i
    – r = reward from playing arm i
    – Update $\hat{r}_i$
• for t in [exploration time, gamble duration]:
    – r = reward from playing arm i
    – update regret
```

2.5 ϵ Greedy

Lastly, the ϵ greedy algorithm introduces a user parameter ϵ that specifies how likely it is that a random arm will be pulled as opposed to the arm with the largest expected reward. In this case, $\hat{R}_i$ are initialized at $1/2$ for the same reason as above.

```
• for t in gamble duration:
    – $p_{\text{inovate}}$ = rand(0,1)
    – if $p_{\text{inovate}} < \epsilon$:
        * r = reward of playing random arm
    – else:
        * i = argmax $\hat{r}_i | i \in (1,..,N)$
        * r = reward of playing arm i
        * update $\hat{r}_i$
```

To ensure that the strategies are not "cheating" so to speak, I just needed to make sure that non of the "decisions" made were based on the true distribution means. The only place where the algorithms contact those values is in the calculation of the regret, however these do not inform decisions.

3 Problem Three: Performance Evaluation

Use what you have build in problem 1 to implement two different five-armed bandits(i.e., N = 5):

- An easy bandit, where the true rewards are distinct enough that you expect it is possible to identify the optimal arm;
- A hard bandit, where the true rewards are such that strategies are likely to struggle finding which arm is optimal.

Evaluate the performances of the strategies (Part 3) over the course of gambles of length T = 1000. For each strategy, show with plots in your write-up.

- The regret of an individual gamble
- The expected regret averaged over 100 gambles
- Another metric of performance different from regret of your own choice, also averaged over 100 gambles and as a function of time.

3.1 Solution

To generate "easy" and "hard" bandits, I simply tweak the hyper parameters of the underlying reward beta distributions.

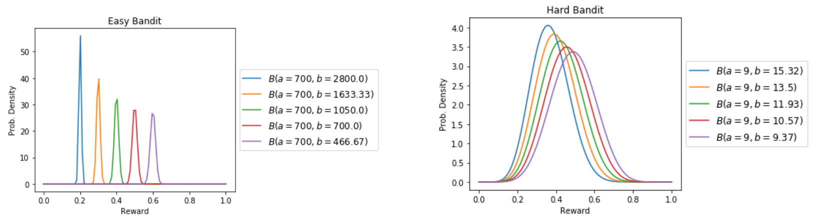

Figure 2: Hard and Easy Bandit Distributions

Initially I choose "easy" reward distributions such that they were fully separated from one another. However, this was not very interesting as most strategies easily found the best arm to pull right away. Now, the "Easy" bandit is built of distributions with unique center and but overlapping spread. The hard bandit alternatively is built of several heavily overlapping distributions with similar but still distinct centers. I was able to generate these two sets of distributions by choosing a set of means that centered the distribution by holding the β constant and manipulating the first expectation of the beta function.

$$\begin{aligned}\mu = E[X] &= \int_0^1 x f(x;\alpha,\beta)dx \\ &= \int_0^1 \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta}dx \\ &= \frac{\alpha}{\alpha+\beta} \\ &= \frac{1}{1+\frac{\beta}{\alpha}}\end{aligned}$$

For all simulations conducted, to easy computational burden I began the simulations with gamble duration T = 100 pulls and ended at T = 1000 pulls. Additionally, I skipped every 10 time steps. For example, The first gamble duration in a single gamble would be 100 pulls, the second 110 pulls and so forth all the way to 1000 pull gamble duration. This greatly reduced the time it took to run 100 gambles to get clear results. For more details on the implementation of the simulations, see the function titled "simulation function" on lines 171 - 205 in the attached script. This function will take in the reward distribution, grading metric and simulation length and can perform an of the desired simulations reported in this assignment. The function below produces plots. First, we will look at the regret from single gambles of duration 1000, and then to reduce noise average regret over 100 gambles. ////

In Figure 3, I present the regret over gambles up to duration T = 1000. We see that ϵ first greedy performs better than all other algorithms on the. It is still quite possible for the algorithms to get stuck on one arm, so we want to see what happens on average as the gamble duration changes.

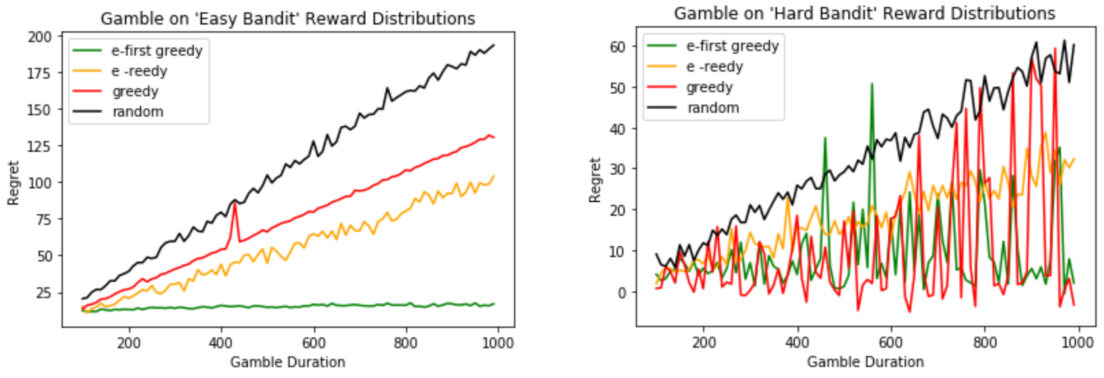

Figure 3: Single and Averaged Regrets On Easy Bandit

These are nice and we can certainly see the ranking of the strategies however, on the hard bandit, there is a lot of noise. To strengthen the signal, I average over 100 gambles.

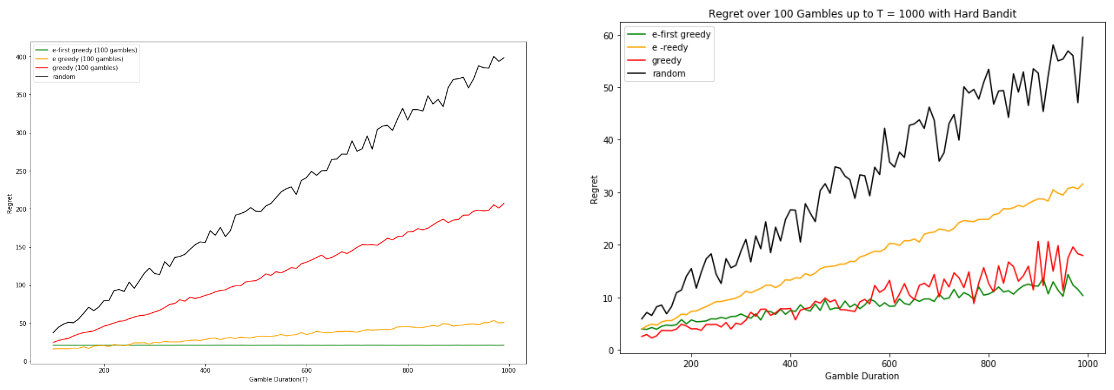

Figure 4: Average Regret over 100 Gambles on Hard and Easy Bandit

We see that for both easy and hard bandits, the ϵ - first greedy strategy is the optimal for large gamble size. In class we proved that as the gamble duration approaches infinity, the relative regret approaches zero for the ϵ - first greedy strategy. Our MAB agrees with the theoretical conclusion.

The additional metric that I choose to report is the percentage of pulls in a gamble that are made on the optimal arm. This would be a good indicator of how well an algorithm stabilizes over time and whether or not it will "learn" which arm to pull. This was really easy to implement as it only requires the number of times the optimal are was pulled for each gambling period. The code to produce this statistic is interwoven into each of the strategy functions. I used a Boolean parameter to indicate whether or not I wanted to calculate the regret or number of optimal arm pulls of a given gamble. See code chunks referred too in section 2 of the assignment. We see the results from a single gamble on both easy and hard bandits in Figures 5 and 6.

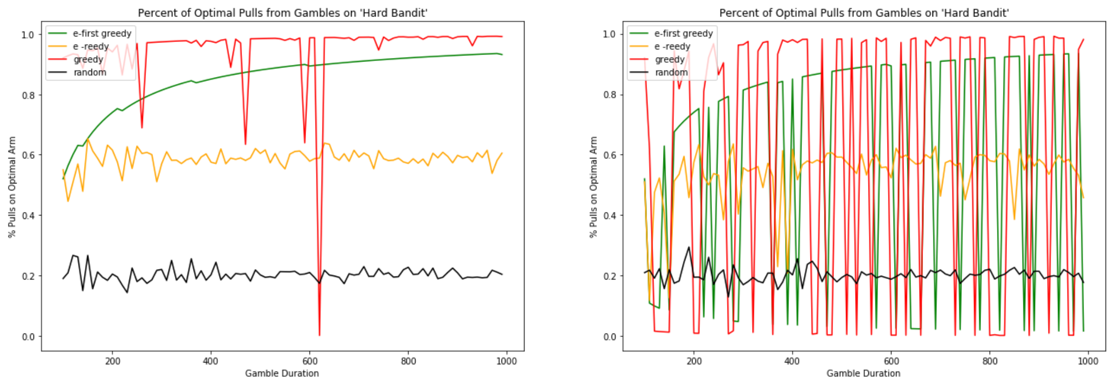

Figure 5: Percent of Optimal Arm Pulls From Gamble on Hard and Easy Bandit

Again, this not very useful unless we average over many different gambles. However, on the easy bandit we can see that the greedy method is able to get to the best arm easily without sacrificing for cost of exploration like we are seeing in ϵ first greedy. Additionally, as a sanity check, we see that random is converging to 0.2, as it should if there are five arms being pulled. This ensures that the optimal arm metric is working correctly. Now, we examine In figure 6 we see some trends beginning to emerge.

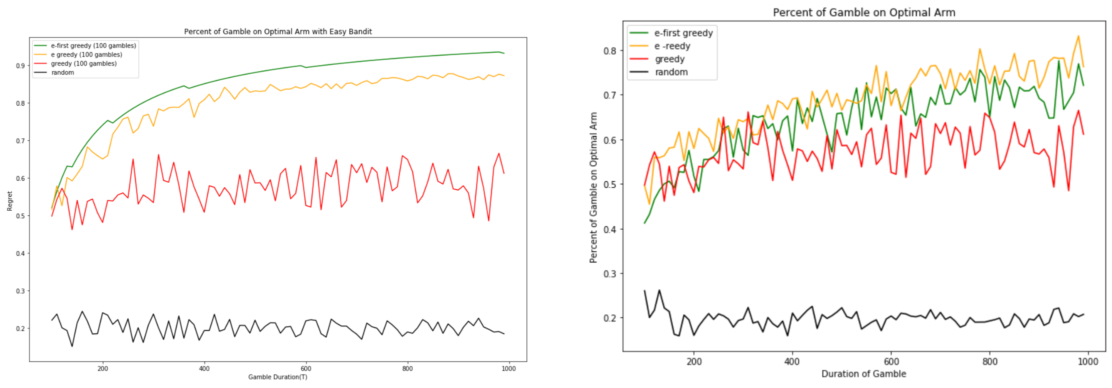

Figure 6: Average Percent of Optimal Arm Pulls over 100 Gambles on Hard and Easy Bandit

This additional metric delivers similar results as the original regret term. We see that ϵ first greedy performs the best over time. However in the case of the hard bandit, its a close tie between first and ϵ first greedy. I suspect that over time they will both converge to the 1. Additionally, if we averaged over more gambles, we would see a smoother regime allowing for stronger rank conclusions for hard reward distributions.

4 Problem Four: UCB Performance

Implement the UCB1 algorithm discussed in the reading. Evaluate the performance of the UCB1 algorithm to the optimal strategy discovered in question 3.

4.1 Solution

The Upper Confidence Bounds (UCB1) algorithm is very similar to the Naive Greedy algorithm in implementation,however instead of choosing an arm based on the local reward history, a unique quantity is calculated and used to make a decision. At a given time, the algorithm will pull the arm with the largest quantity J(t) such that:

$$J(t) = \text{argmax}_{i=1..k}(\mu_i + \sqrt{\frac{2\ln(t)}{n_i}}) \tag{3}$$

Where n_i is the number of times arm i has been pulled on a given gamble. The psuedo code for the UCB1 Algorithm is given below

- Pull each arm once, save reward
- for t in gamble duration:
  - i = argmax$J(t)_i | i \in (1,..,N)$ where $\hat{r_i}$ is the average reward of arm i
  - r = reward from playing arm i
  - Update $\hat{r_i}$

This algorithm is defined between lines 156 and 172 in the attached script. This is very clever because it considers how deep into the gamble the theoretic "player" is as well as weighting the term by the number of pulls on the given arm. This way over time we should be picking the arm with the highest expected value but also the arm that is not getting played a lot, it protects the the player from getting stuck playing the wrong arm which is possible in the greedy methods. Additionally as reported in Kulushev (2014), Auer, Cesa-Bianchi Fisher (2002) show that at turn t, the expected regret from this strategy is bounded by:

$$8 \sum_{i:\mu_i<\mu^*} \frac{\ln(t)}{n_i} + (1 + \frac{\pi^2}{3}) \sum_{i=1}^{k} \Delta \tag{4}$$

However, the Kulushev paper does not give a good way to determine what t should be in order to achieve this ideal bounding. The paper simply states that each arm should be played once, and then the algorithm should greedily choose the arm that maximizes q(t). To assess how well this method works in comparison with the other strategies, I average regret over 100 gambles and compare the results. Additionally, I test the algorithm on the optimizing the other metric considered, the percents of optimal arm pulls in a gamble. In Figure 7, we see the regret after a single gamble plotted against gamble duration. We see that the regrets from both strategies are quite low with respect to the competing methods. However, the ϵ first greedy is still beating out the new comer, UCB1.

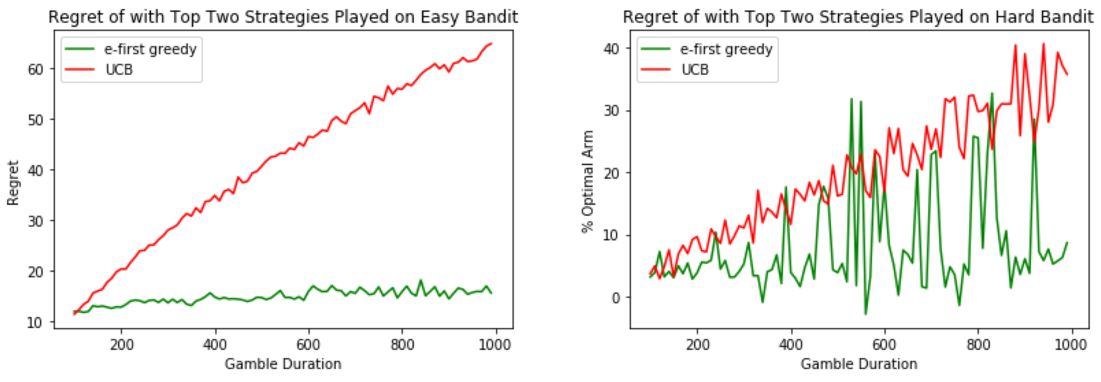

Figure 7: Regret from Gambling using UCB and ϵ - first greedy on Hard and Easy Bandits

To reduce noise, average over 100 gambles to produce Figure 8 below.

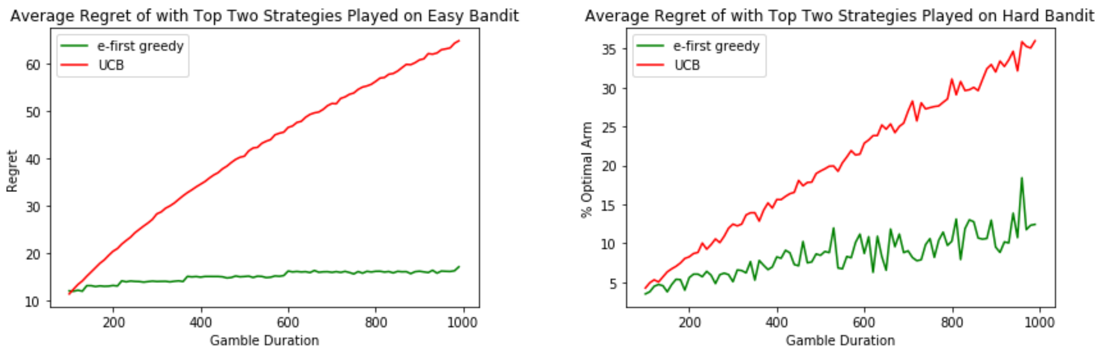

Figure 8: Average Regret from 100 Gambles on Hard and Easy Bandit

We see that UCB performs worse on both cases. This is certainly surprising. In theory, UCB should correct for pull the wrong arm with the weighted term being added to the average. We next evaluate the performance of the two best algorithms using the secondary metric that measures the proportion of pulls on the best arm during a gamble. See Figure 9 for the results averaged over 100 gambles on both the easy and hard bandits.

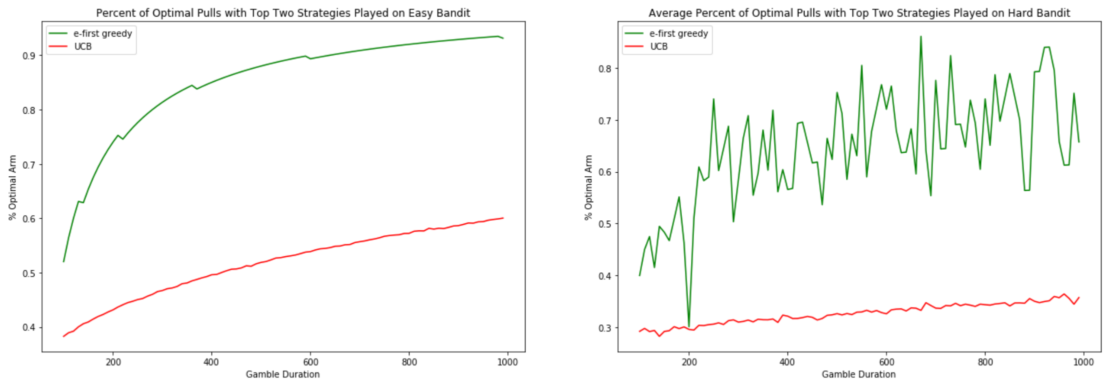

Figure 9: Average Regret from 100 Gambles on Hard and Easy Bandit

As expected, the ϵ first Greedy algorithm captures a higher proportion of optimal arm pulls. We see that over time, the proportion approaches 1. There is an upward slope for both algorithms, however the growth for ϵ first greedy is expectantly logarithmic. This is less clear but still suggested on gambles from the hard bandits reward distribution. On very strange thing that I noticed throughout generating these plots is some regret scaling factor. At this time, I cannot reason why sometimes, there is a break in scale between two different regret histories from two different simulations. This may be due to a bug in my code. However, while both UCB and ϵ first greedy appear to be excellent choices for completing this task, I conclude from this analysis that ϵ first greedy is more effective as it consistently outperforms UCB in both minimizing regret and pulling the optimal arm during a gamble. I would like to review more literature to learn how to tune time suggested t parameter to achieve better results from UCB.

Sources Referenced

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multi armed bandit problem. Machine Learning, 47((2-3)), 2002.