DATA SCIENCE II

HW05 WRITE-UP

March 31, 2019

David Landay
University of Vermont
Graduate Student, Comp. Systems and Data Sci.

0.1 Introduction:

This Homework assignment explores several strategies for learning an optimum "pay out" reward in a stochastic multi armed bandit (MAB) problem. A "one armed bandit" is another name for a slot machine. With every pull of the bandit's arm, there is some probability of receiving a reward. The multi armed bandit can be thought of as multiple one armed bandits, or one bandit with many arms. Simply stated, the multi armed bandit problem is to solve a scenario in which a gambler wants to maximize their reward by exploring different arms to pull, and exploiting the arms for which they most often get the best reward. More generally, the gambler is an agent that explores an environment to acquire knowledge about "where" the largest expected value exists in the state space, and then exploits said knowledge to improve it's overall chances of drawing that value ever after.

The MAB problem has useful real world applications. One example of a MAB in practice is in clinical trials. As a drug trial moves forward, for instance, the length of the study can shorten as the expected success rate of a drug becomes more apparent. In dynamic A/B/n testing, whereby mulitple (n) versions of an A/B test are administered, some kind of strategy may be employed to eliminate or change certain test attributes that yield poor (or sub optimal) results. A classic example of this is a front end web designer who believes there is some correlation between the lack of sales from their online store, and a design attribute of their homepage. The designer may divert traffic to two, or more, different sites which test the hypothesis that the design attribute is affecting online sales. The front end developer may divide site-goers in a statistically appropriate way, and then observe how many sales are generated from each site. After each sale, the designer will (hopefully) have new knowledge of what drives sales from their site. A strategy for exploring/exploiting the sales data can be used to choose the optimal layout. Some such policies will be discussed in this writeup.

For this assignment, the MAB problem is simulated by initializing K arms, or probability distributions, of which an agent has no prior knowledge, and employing some strategy to explore these distributions. The agent will update it's knowledge of the domain space for a given time step (each pull of the arm) and then exploit that knowledge gained to draw a reward from the distribution with the most optimal expected reward henceforth. To be precise, there will be K probability distributions with expected rewards $< \mu_1,...\mu_k >$ which are unknown to the agent. At each time step t the agent will select one of the distributions to draw a value from. The goal for the agent is to leverage finding the distribution that has the

highest expected reward, and gaining as much reward as possible. We will then assess how well each explore/exploit policy is at carrying out the task. To measure how well the agent does, we will calculate the total expected regret and subtract whatever cumulative reward the agent has acquired up to the current time step. The total expected regret is the amount you have lost as a result of not drawing from the optimal distribution. According to [2] Kuleshov et. al, calculating the total expected regret for each time step t in the gamble is the "most popular performance measure". In this paper, we plot several time series of the total expected regret for each of the MAB strategies employed. In addition, we use the expected percentage of pulls from the optimal distribution, as seen in [2], as a second metric of performance. We perform 100 gambles of length $T = 1000$, calculate the number of times the optimal distribution was sampled, and then calculate the percentage of optimal pulls at each time step. Convergence to a particular probability will indicate how often the strategy picks the optimal arm after t steps (pull or time, or however "t" will be interpreted).

0.2 Strategies and Choosing Distributions

0.2.1 Choosing Distributions:

In the `MAB_algorithms.py` library, the `MAB()` class initializes a multi armed bandit. By default, the method `add_beta_arm()` will add 5 arms, each generated from a beta distribution with varying β, α pairs, such that $\beta = 2,3,4,5,6$, and $\alpha = 2,3,4,5,6$. However, this method allows users to define their own beta distributions and bandits with different numbers of arms. Beta distributions were selected because they easily allowed for the creation of several independent distributions whereby the means and variances of each distribution could be fine tuned. These particular default values were assigned to an $Easy\ Bandit$; a set of distributions whose mean expected values were distinct enough for a policy to easily converge to the optimal arm. From the assessment carried out in [2], we know that an "algorithm's performance relative to each other is affected only by the number of bandit arms and the variance of the rewards." Thus, we can attempt to fool each algorithm by initializing arms to distributions with higher variance.

A $Hard\ Bandit$ was created from another set of beta distributions generated by `hard_mab.add_arm_beta(alpha=[1.9,2,2.1,2.2,2.3],beta=[4.2])`, with α, β pairs

of $1.9, 2, 2.1, 2.2, 2.3$. The amount of variance among each distribution and low probability density were good indicators that a given policy would have trouble discovering the optimal distribution to exploit. Below are plots of the domain for each bandit:

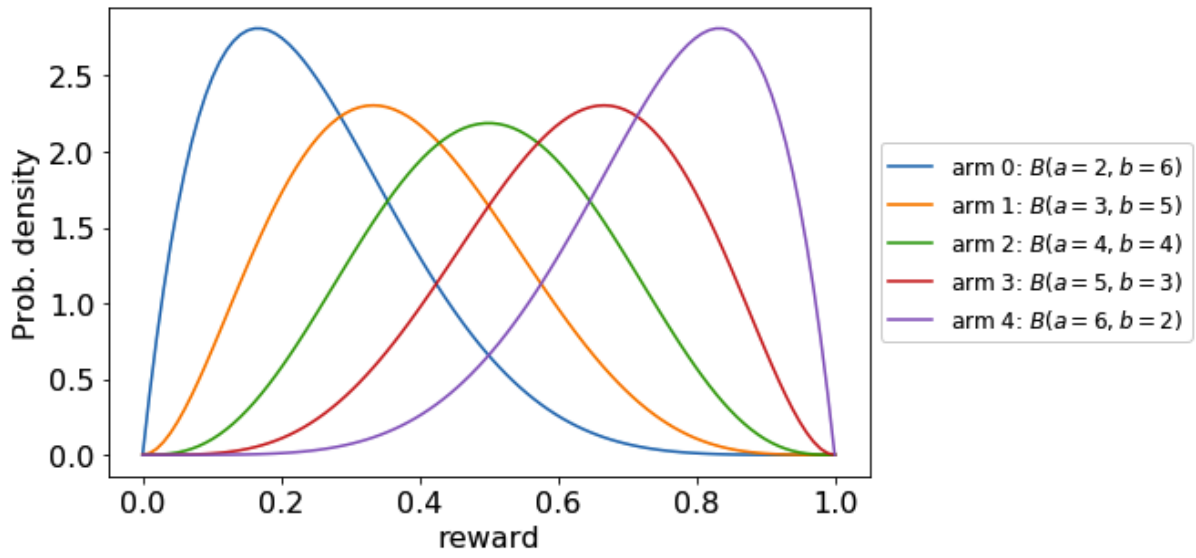

Figure 1: True Reward Distributions for Easy Bandit

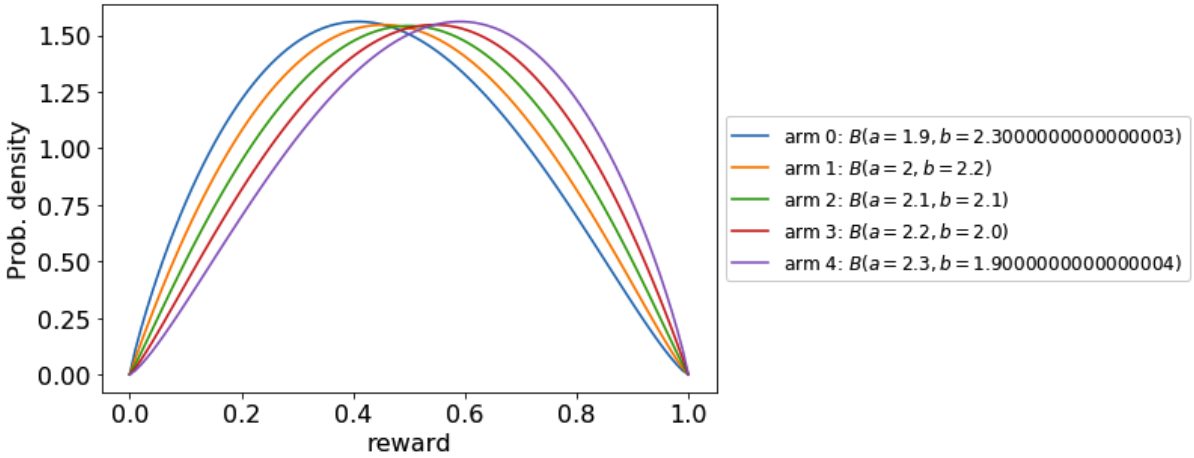

Figure 2: True Reward Distributions for Hard Bandit

0.2.2 Building Policies:

For this study, we implemented five common learning algorithms to act as the pull mechanism in the MAB scenario. The five algorithms that will be looked at in the next section are a completely **random gamble**, in which an agent continuously explores the state space, and will pick from each distribution with equal likelihood at each time step, a pure **greedy** policy that randomly picks an arm and exploits that arm for the remaining (T - t) steps, where T denotes the duration of the gamble, an ϵ-**greedy** strategy that explores with probability ϵ and exploits the arm with the largest expected reward (as it has been calculated thus far) with probability 1- ϵ, an ϵ-**first greedy** policy whereby the agent explores the state space for some $N*\epsilon$ steps while keeping track of the expected reward for each arm at time t, it then exploits the arm with highest expected reward for the remaining $N*\epsilon - 1$ steps, for some $N \in T$ (** be greedy after learning*), and the **UCB1** algorithm as described in [2][3].
For the next section, we will discuss the first four algorithms and return to discuss **UCB1** in a later section.

0.3 Results of One Gamble of Duration T

We will start by discussing the results for each of the "one gamble" plots for each Bandit.

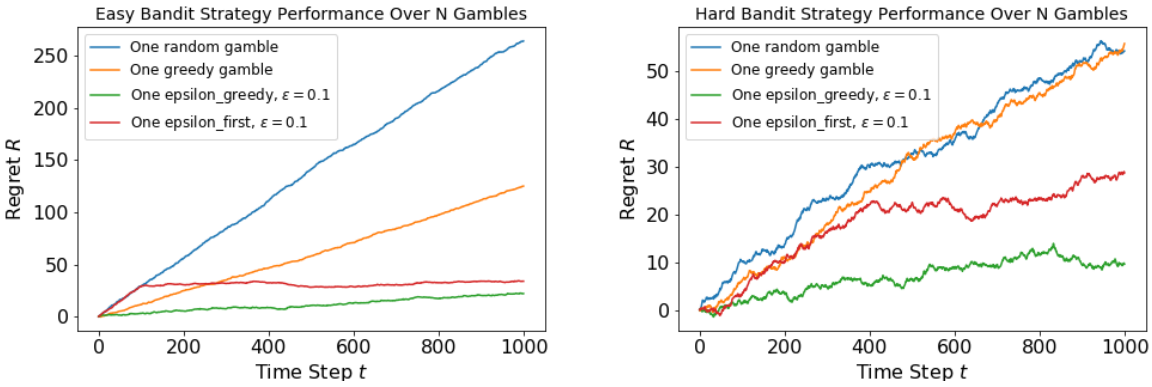

Figure 3: One Gamble of Duration 1000

It should be noted that these plots could look different, given that the `MAB()` objects had initialized to some other distribution in each domain. Hence, these plots are not

truly representative of algorithm performance. However, notice the effects of large variance in the set of $Hard\ Bandit$ distributions introduces a lot of uncertainty to the policies, and is clearly visible in this demonstration. To get a truer sense of what is happening, we must find the mean expected regret at each time step t. To do this, we can perform gambles of length T 100 times and take the average regret at each time step. We do this here:

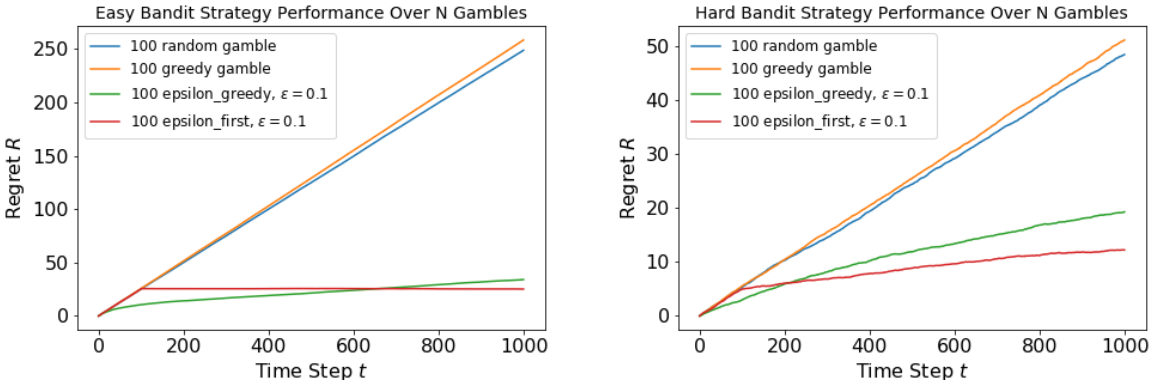

Figure 4: 100 Gambles of Duration 1000

Notice that each plot is roughly the same. After averaging at each time step, it is revealed what actions we can expect an agent to take given ample time to explore the space over multiple trials. Additionally, the plots show which, on average, converge faster to the optimal distribution, as seen by plateauing regret. Notice that on average, a purely greedy decision will do as well, if not worse than random. Those actions are actually bounded linearly, whereas more sophisticated policies plateau.

One thing to notice is the scale of the y-axis for each plot. The total expected regret is different as a result of each bandit representing a separate distribution space. Hence, we still do not have a definitive comparison between each algorithm for each MAB. To better understand whether the algorithms have been successful, we should normalize by scaling each set of regrets to unit length. Before we do this, I will introduce the fifth learning strategy in the next section.

0.4 UCB1 and a Second Metric:

0.4.1 UCB1:

UCB stands for upper confidence bound, and the **UCB1** algorithm can be explained by the "principle of optimism in the face of uncertainty"[1], which is simply the process that has been described thus far to explain how every learning strategy goes about searching for the most optimal arm to play: they will make an "optimistic bet" as to what the expected reward is for a given arm, and then use that knowledge to find the arm that yields the highest expected reward given uncertainty about every arm. In the case of the **UCB1** strategy, optimism comes in the form of an upper bound on the probability that a pull of an arm will deviate from the expected reward associated with that arm. Essentially, as arm_i is pulled, a confidence bound can be determined about the reward drawn to estimate the expected reward for that arm. As arm_i is pulled more, the uncertainty shrinks and the confidence bound will better determine which arms to explore or exploit. Hence, more variance in the distributions will affect convergence time because arms will need to be explored more, and distributions with similar true expected rewards will confuse the algorithm because at each time t it chooses the arm with the maximum estimate of that expected reward, and will likely find multiple arms with equal estimates.

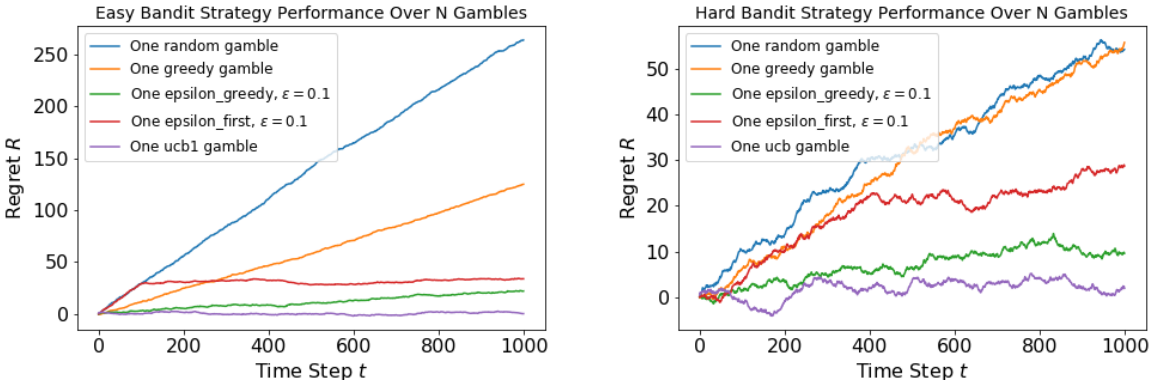

Figure 5: One Gamble of Duration 1000 With UCB1

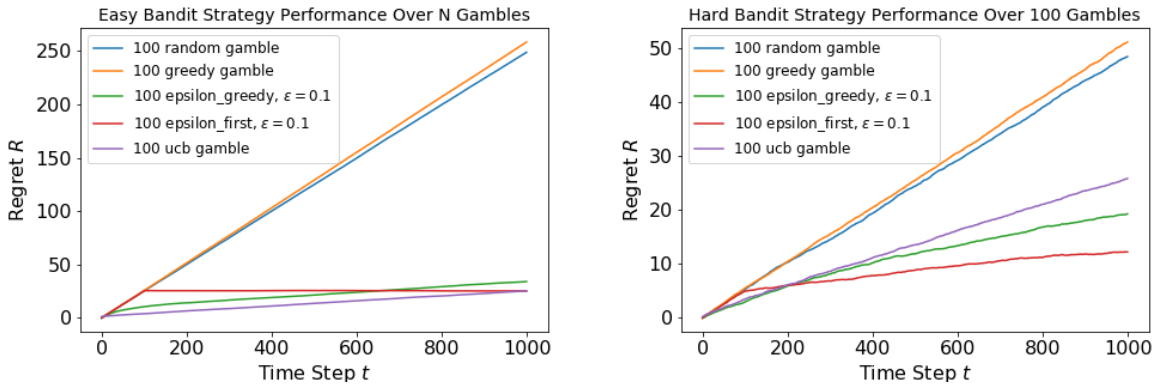

Figure 6: 100 Gambles of Duration 1000 With UCB1

In Figure 5., we see that for individual gambles, the **UCB1** algorithm appears to perform the best out of all strategies. However, as we average over 100 gambles, we see a downside to the algorithm's design. The left most plot of **figure 6.** shows the algorithm working optimally over distributions with low variance and very distinct expected values. However, our *Hard Bandit* was designed to trick algorithms by introducing high variance. Since all of the distributions of the *Hard Bandit* slightly offset expected values with overlapping and high variances, we will be less confident about our upper bounds until we explore the distribution space a significant number of times. So, the spread of the distribution affects the speed of convergence to the optimal arm, which results in higher regret compared to the *Easy Bandit*.

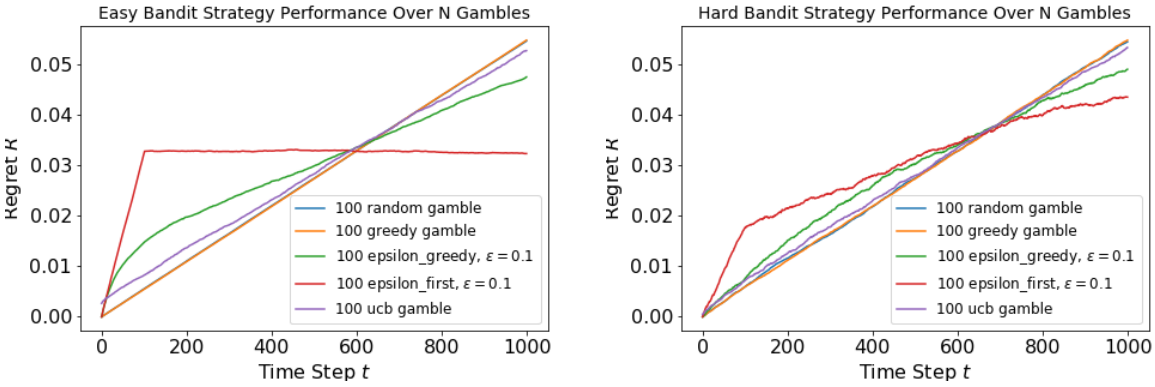

Figure 7: 100 Gambles of Duration 1000 Scaled to Unit Length

Now that we have normalized the data, we can adequately compare the two distribution states. In the right most **figure 7** It would appear that the ϵ-**first-greedy** strategy will

explore heavily and then clearly converge to the optimal solution in the $Easy\ Bandit$. While it also appears to be the optimal solution for the $Hard\ Bandit$, in the left most **figure 7**, the effects of a high variance state space are clearly visible. ϵ-**first-greedy** does sub-optimally among both distributions, but is the most optimal choice. The ϵ-**greedy** policy will do sub-optimally in both cases, which has also been shown to perform better in a less ambiguous distribution spaces [2]. Given further experimentation, randomly selected distributions for a $Hard\ Bandit$ tuned for even higher variances might yield clearer, distinguishable, results.

0.4.2 Percentage of Optimal Arm played at t:

For the second metric, I took inspiration from [2], where for each time step, a record of whether the optimal arm was pulled is kept in memory. For 100 gambles, the percentage of times the optimal arm was played at time t is then calculated and plotted as a function of t. These plots not only serve as a measure of algorithm performance, but act as good visual aids for understanding the learning each strategy is doing.

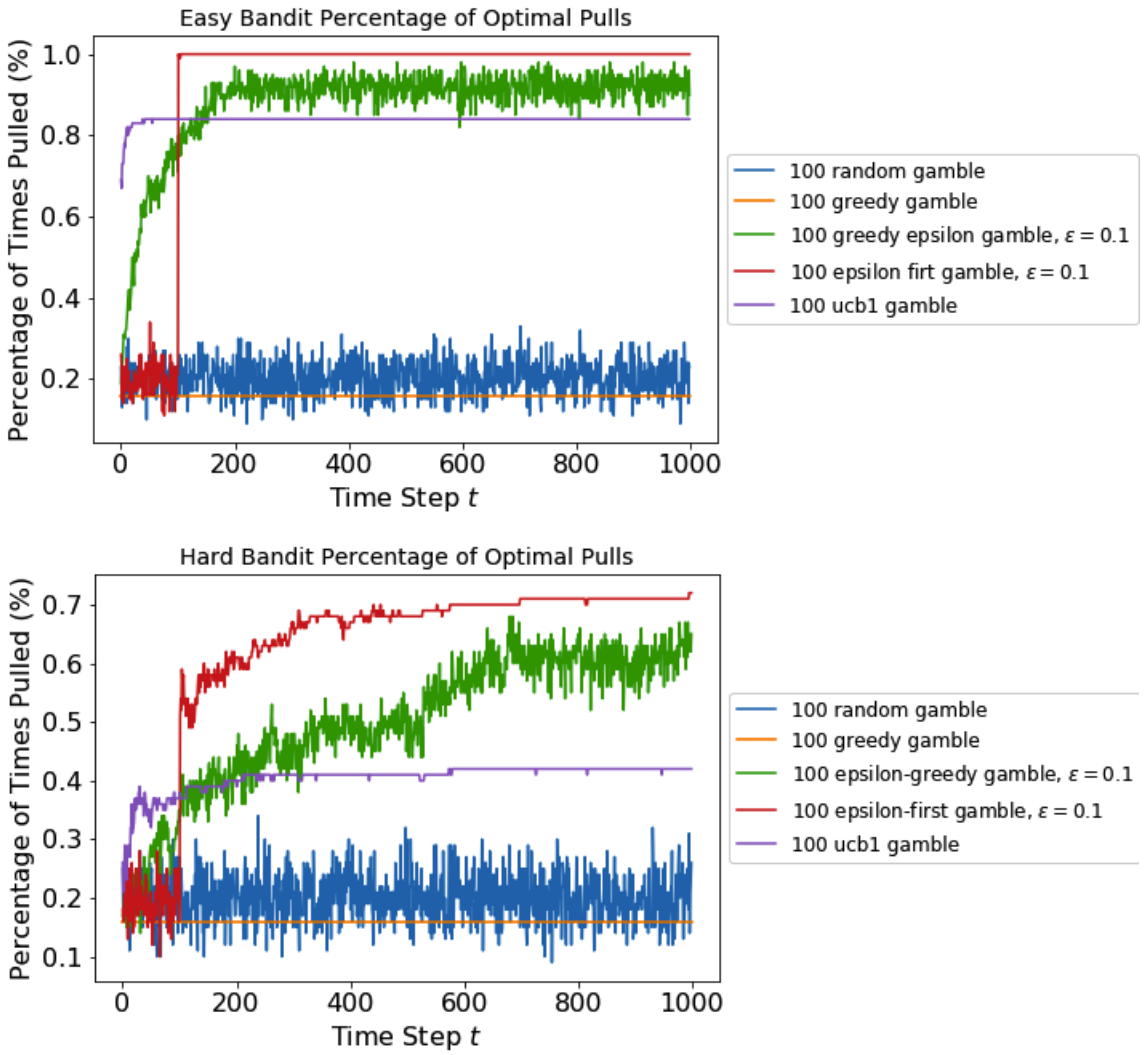

Figure 8: 100 Gambles of Duration 1000

The plots are nice representations of what we expect would happen for every pull of the arm for a given strategy. For example, a pure **greedy** method will plot as a straight line because once an arm is randomly selected, it is played indefinitely after. Given many gambles, you will expect to play the correct arm only 1 in K times, where K is the number of arms that the bandit has. Therefore, by playing in a pure greedy way, we can expect to perform worse than if we had played each arm at random for every step in the gamble. This is evidenced by the variability we can observe in the blue line denoting a random gamble scheme.

What's nice about this plot is that we can observe the exploration/exploitation

phases for each strategy, and how quickly each strategy converges. For instance, notice that the ϵ-**first-greedy** method is a step function, where after N steps of random search, the algorithm can use it's prior knowledge to converge after just 100 trials. On the other hand, **UCB1** and ϵ-**greedy** will leverage exploring with exploiting to converge to the optimal solution faster, but pick the optimal solution less frequently as a result of their architectures. considering this, plotting regrets seems to be a stronger metric of performance because it more accurately describes the algorithm that compounded the most reward. However, this is a good metric to show why that particular outcome occurred; because it shows which converges fastest.

Bibliography

[1] j2kun. Optimism in the Face of Uncertainty: the UCB1 Algorithm, October 2013.

[2] Volodymyr Kuleshov and Doina Precup. Algorithms for multi-armed bandit problems. *arXiv:1402.6028 [cs]*, February 2014. arXiv: 1402.6028.

[3] Joannès Vermorel and Mehryar Mohri. Multi-armed Bandit Algorithms and Empirical Evaluation. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, João Gama, Rui Camacho, Pavel B. Brazdil, Alípio Mário Jorge, and Luís Torgo, editors, *Machine Learning: ECML 2005*, volume 3720, pages 437–448. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.