Assignment 9

In this homework you will be using bandit algorithms to trade stocks (*Because Computational Geometry problems are just not fun to code).* Please don't let the fancy looking math scare you. The algorithms themselves are truly very simple and intuitive. In what follows, we are using the notation "$\sim$" to mean "sampled from" i.e. $x \sim \mathcal{D}$denotes a value x randomlysampled from a distribution $\mathcal{D}$.

Multi-Armed Bandits

The multi-armed bandit problem is a quintessential reinforcement learning problem which illustrates the exploration/exploitation tradeoff dilemma. In this problem, a fixed limited set of resources must be allocated between competing (alternative) choices in a way that maximizes some expected gain. Crucially, each choice's properties (reward(s)) are only partially known at the time of allocation, and may become better understood as time passes or by allocating resources to the choice.

The classic example is that of a gambler at a casino, allocating money to k slot machines, each of which has some (unknown) underlying reward distribution, say $\mathcal{N}(\mu_i, 1)$, where each μ_i is fixed, but unknown to the gambler. Through playing the machines and analyzing the empirical distribution of rewards, the gambler can hope to maximize her gain (equivalently, minimize her loss) by applying a policy which exploits those machines with higher expected reward μ_i. The dilemma is that, the more the bandit expplores (by allocating resources to different machines), the less resources they will have to exploit the empirically best machine later on.

Formally, the problem is defined as a tuple $(\mathcal{A}, \mathcal{R})$, where $\mathcal{A} = \{a_i\}_{i \in [k]}$ is a set of actions, and $\mathcal{R}$ is an unknown distribution of rewards parameterized by the chosen action. We use $\mathcal{R}(a, t)$ to denote the expected reward given action $a \in \mathcal{A}$ at a time $t = 1..T$. Over the course of T time steps, a *bandit* applies a policy $\pi_a := \pi_a^{(t)} = \Pr[a \mid t]$ to select an action $a^{(t)} \in \mathcal{A}$ at time t based on, previous actions, and their associated rewards. Once an action is taken, the agent receives a reward $r^{(t)} \sim \mathcal{R}\left(a^{(t)}, t\right)$.[1] Success for a bandit algorithm is framed in terms of the rewards gained by the bandit vs. the maximum reward the bandit *could have received*. The difference in these two quantities is called the *regret*. The agent's goal is to maximize

$$R = \sum_{t=1}^{T} r^{(t)} \tag{1}$$

which is equivalent to minimizing the regret

$$\rho = \sum_{t=1}^{T} r^{*(t)} - \sum_{t=1}^{T} r^{(t)} \tag{2}$$

where $r^{*(t)} \in \max_{a \in \mathcal{A}} \mathbb{E}[\mathcal{R}(a, t)]$ is the reward associated with a best action at time t. Importantly, this notion of success is relative to what is possible to obtain, rather than what is actually obtained by the bandit.

As has been the case with almost all of the algorithmic techniques and ideas we have seen thus far in the class, the multi armed bandit problem can be decomposed into a small number of exhaustive and irreducible factors. In fact, bandit algorithms differ only in two components : The policy employed for choosing a next action, $\pi := \pi^{(t)}$, and the underlying model for the reward distribution $\mathcal{R}(a)$. As the programmer, you often have no control over the number of available actions, nor do you have knowledge of the underlying distribution $\mathcal{R}$at the onset of the problem. If information on this is available before hand, then this may affect your choice for modeling $\mathcal{R}$ and choosing a policy π.

Reward Distributions

Typically, for each action $a \in \mathcal{A}$, we fix a reward distribution $\mathcal{R}(a, t) = \mathcal{R}(a)$ and update the distribution as information is revealed. The result (reward) of choosing a particular action is then a sample from this distribution. As discussed, the goal of the bandit is to balance the exploration/exploitation tradeoff - which is done by modeling the (apriori unknown) reward distribution for each action. In our setting, we will assume independence of rewards over both rounds and actions.

[1] Recall that $\mathcal{R}$ is a distribution, and thus the reward for each action is not necessarily constant – further, in the real world, the distribution $\mathcal{R}$ may be changing over time.

Policy

A policy is a randomized algorithm A which, given the history of chosen actions and associated rewards, determines an action $a^{(t)} \in \mathcal{A}$ for round t. The hope is that the policy chooses actions with a good associated reward.

Bandit algorithms - Balancing the Exploration / Exploitation tradeoff

You will implement (at a minimum) two bandit algorithms

1. $\epsilon-$Greedy
2. UCB

ϵ-Greedy (7 points)

In this algorithm, the policy is paramaterized by a number $\epsilon \in [0,1]$, which is meant to balance exploration and exploitation. The policy is given by

$$\pi^{(t)} = \begin{cases} \text{choose the action with the highest empirical reward} & \text{w.p. } 1-\epsilon \\ \text{choose an action at random} & \text{w.p. } \epsilon \end{cases}$$

so our policy exploits w.p. $1-\epsilon$ and explores with probability ϵ. rewards can be modeled however you'd like, but I'd suggest starting with

$$\mathcal{R}(a) := \mathcal{R}^{(t)}(a) = \frac{\sum_{i=1}^{t} r^{(i)}(a) \cdot \mathbf{1}_{a^{(i)}=a}}{n_a^{(t)}}$$

or

$$\mathcal{R}(a) := \mathcal{R}^{(t)}(a) = \frac{\sum_{i=1}^{t} r^{(i)}(a)}{t}$$

where $n_a^{(t)}$ is the number of times action a has been chosen, and $\mathbf{1}_{a^{(i)}=a}$ is 1 iff $a^{(i)}$equals a.

UCB (10 points)

This algorithm takes a slightly more sophisticated approach to balancing the exploration/exploitation tradeoff. the policy is given by

$$\pi^{(t)} = \arg\max_{a \in \mathcal{A}} \left\{ \mu_a + \sqrt{\frac{2 \cdot \ln(t)}{n_a^{(t)} + 1}} \right\}$$

where, again, $n_a^{(t)}$ is the number of times action a was chosen up to time t. Intuitively, the $\sqrt{\frac{2 \cdot \ln(t)}{n_a^{(t)}}}$ measures our uncertainty in the reward associated with action $a-$ if $n_a^{(t)}$ is small, then we are less certain in the reward. In addition, we also hope to exploit good actions, so we add the emipirical mean of the action to help balance this. For this strategy, the reward distribution simply needs to measure the empirical mean.

Your job

You must apply the two algorithms described above to stock trading and compare their performance with the code provided. Actions correspond to choosing a stock to buy, and rewards correspond to the price difference between the close and open price for a given day if 1$ of the share was bought and sold at the open/close price. An example is given in the assignment where a policy that chooses a stock (action) with probability proportional to its mean empirical day change. Please compare the two algorithms in terms of the expected mean reward and variance (code is already provided for this).

Extra Credit (optional)

Please include a brief write up of any extra credit implementations and remark on any changes in performance against the baselines.

- (5 points) Try modeling the rewards with a "fancier" reward distribution, such as a gaussian process, or with a kernel regression (explained below). You may wish to weight actions corresponding time steps closer to the current time more heavily than those that occurred earlier. It's up to you on how you'd like to model this.
- (2 points) (if you have not already done this) Try revealing/updating the outcomes/rewards associated with each stock at each time step and updating your distributions accordingly. That is, don't just update the data for your chosen stock/action, but update the distributions for all stocks after a given time step. This should lead to more accurate modeling and better results
- (5 points) Add context to your decision, and analyze the effect on your average reward. This is explained below.
- (3 points) add the option for bandits to sell a stock - i.e. earn a positive reward if the stock goes down.

Reward Modeling

Kernel regression

Suppose you believe that your reward distribution has smoothness properties, and the property that nearby actions have similar expected rewards (as is the case with most problems). In this case, Kernel Regression is a great candidate for modeling your reward distribution. Kernel Regression is a non-parametric method (i.e. does not require you to calculate parameters) used to estimate the conditional expectation of a random variable.

We assume that

$$\mathbb{E}\left[Y|X\right] = \Phi\left(X\right)$$

where Φ is some unknown function.

To be slightly more concrete, suppose we want to mode the reward based on the similarity of different actions. Let $\phi : \mathcal{A} \times \mathcal{A} \to \mathbb{R}$ be some function that measures the similarity in its inputs a and $a^{'}$. e.g. $\phi\left(a, a^{'}\right) = 1 + corr\left(a, a'\right)$ - one plus the pearson correlation between rewards associated with actions a and $a^{'}$. We call ϕ a kernel, and given a list of previous actions and associated rewards $\left\{\left(a^{(t)}, r^{(t)}\right)\right\}_{t=1..T}$, we can try to predict the expected reward of a new action a at time t as

$$\mathcal{R}\left(a\right) \sim \Phi^{(t)}\left(a\right) := \frac{\sum_{i=1}^{t-1} \phi\left(a^{(i)}, a\right) r^{(i)}}{\sum_{t=1}^{t-1} \phi\left(a^{(i)}, a\right)}$$

As you can see, Φ is a very simple function. It simply estimates the (unknown) reward of an action a by taking a locally weighted average with weights given by the kernel ϕ. The rewards of actions that are more similar to a will be weighted proportionally more than those that are dissimilar. We then divide by the sum of the weights to normalize.

Naadaraya-Watson Kernel Regression

This technique simply uses $\phi\left(x, x^{'}; \sigma\right) = \exp\left[-\|x - x^{'}\|_2^2/\sigma^2\right]$. i.e. It assigns a gaussian like weighting to the underlying distribution. This is best used on contextual bandits (explained below).

Context

Contextual Bandits use a conditional reward distribution $r^{(t)} \sim \mathcal{R}\left(a \mid \chi^{(t)}\right)$, where $\chi^{(t)}$ holds some "extra" contextual information. In the case of stocks, $\chi^{(t)}$ could be, e.g. a vector representing the price change for each stock on the previous trading day. For example, if you have two stocks a and b and stock $a/b's$ open and close price on day $t-1$ was $(1, 1.1)/(2.3, 2.1)$ respectively, then the corresponding context for day t would be $\chi^{(t)} = (0.1, -0.2)$. You could then use the Naadaraya-Watson Kernel Regression to find

$$\mathbb{E}\left[\mathcal{R}\left(a\right)|\chi^{(t)}\right] = \frac{\sum_{i=1}^{t-1} \exp\left[-\|\chi^{(i)} - \chi^{(t)}\|_2^2/\sigma^2\right] r^{(i)}\left(a\right)}{\sum_{i=1}^{t-1} \exp\left[-\|\chi^{(i)} - \chi^{(t)}\|_2^2/\sigma^2\right]}$$

where $r^{(i)}\left(a\right)$ is the reward you would have received from choosing actoin a at time i. If the formula looks troubling - note that it's literally just taking an average reward for action a over earlier time steps.when taking this average, we give days i higher weight if $\chi^{(i)}$ is similar to $\chi^{(t)}$, and lower weight otherwise. This is essentially saying -

"what is the average reward for this stock when something *similar* hapened the day before?" Our notion of *similar* is $\exp\left[-\|\chi^{(i)} - \chi^{(t)}\|_2^2/\sigma^2\right]$, and tuning σ controls how close two price changes need to be to be deemed similar (simply setting σ to be the vector of empirical standard deviations should work fine).