Answer the following questions

**1. Which hyperparameters are important for** **Thompson Sampling,** **e-greedy, UBC, and random sampling? Show that they are important (15 Points)**

The parameter - ‘stationary’ is important for all the algorithms (except random sampling) because if we have the Non-stationarity setting, it will resamples the action Bernouilli parameters every 100 streps. This places special emphasis on exploration because the if the algorithm keep the old guess its regret will dramatically increase:

图表

描述已自动生成

The parameter - ‘arm\_count’ is also important for all the algorithms (except random sampling) because if we increase the number of choices we can take, it will need more time to explore the best choice. In this case, as we can see in below graph, the greedier setting will have better result:

图表, 直方图

描述已自动生成

For Thompson Sampling, we don’t need to tune any special hyperparameters.

For e-greedy, we need to tune the ‘epsilon’ parameter. This parameter decides how greedy (less exploration) the algorithm is. The smaller value will have a higher probability of keep the previous best choice and don’t make new exploration.

For UBC, we need to tune the ‘c’ parameter. This parameter will easily change the weight of the bound 图示

描述已自动生成 This parameter also decides how greedy (less exploration) the algorithm is. With smaller ‘c’ value, we will easily be sure about the q-value of an action and make less exploration about it.

For random sampling, we don’t need to tune any special parameters since the choice is made randomly in every turn.

**2. How does the action space affect Thompson Sampling, e-greedy, UBC, and random sampling? Show why. (15 Points)**

As we can see the difference between these 2 experiments, the first experiment only has 10 choices while the second experiment has 200 choices. All the other parameter for the algorithms is unchanged. If we increase the number of choices we can take, it will need more time to explore the best choice. In this case, as we can see in the graphs, the greedier setting will have better result, while the algorithm that emphasizes exploration will spend more time to find the better choice. There is no impact for random sampling.

***# Experiment 1***

**arm\_count = 10 *# number of arms in bandit***

图形用户界面

描述已自动生成

***# Experiment 3***

**arm\_count = 200 *# number of arms in bandit***

图表, 直方图

描述已自动生成

**3. How does stationary affect Thompson Sampling, e-greedy, UBC, and random sampling? Show why. (15 Points)**

The parameter - ‘stationary’ is important for all the algorithms because if we have the non-stationarity setting, it will resamples the action Bernouilli parameters every 100 streps. This places special emphasis on exploration, because the if the algorithm keep the old best choice its regret will dramatically increase after the resampling. However, if we use the random sampling method, there will be no effect.

图表

描述已自动生成

In this case, UCB with c-value 2 has better result after the resampling since it have a better chance of exploration. If we change the c-value to 0.1, its performance will very like other algorithms.

**4. When do** **Thompson Sampling, e-greedy, UBC, and random sampling stop exploring? Explain why. Explain the** **exploration-exploitation tradeoff (15 Points)**

For the Thompson Sampling if the is no overlap in this beta distribution graph, that is the point it stop exploring, since the blue one will always has a bigger beta value than the red one.

图表, 折线图

描述已自动生成

For the e-greedy, if will never stop exploring unless the ‘epsilon’ parameter is not equal to 0.

For UBC, if the bound 图示

描述已自动生成 is becoming smaller, that means the algorithm is more sure about the q-value we get. So in that case, if one choice always has the biggest 图示, 示意图

描述已自动生成 value, it will stop exploring and always choose that option.

The random sampling will never stop exploring since it always takes random option.

Exploration-exploitation tradeoff is the most important part for us to solve multi-armed bandit problem. That is because we don’t know the probability of winning for every choice at the beginning, but we want to get the optimal result in a shorter time. In this case, we need to explore other choices because they may have a better reward. But on the other hand, we also need to be greedy and exploit the best choice we have for now. A good algorithm to solve multi-armed bandit problem should consider both sides.

**5. How long do Thompson Sampling, e-greedy, UBC, and random sampling remember the past actions? Explain your answer. (10 Points)**

The Thompson Sampling, e-greedy, UBC will remember their past experiences. They will record the number of times the action wins and the total number of all actions. The e-greedy and UBC will calculate q-value every turn, which equal to ‘action wins times / total times of all actions. And for Thompson Sampling, it will also store the past result to calculate the beta distribution. For random sampling, we don’t need to record past result for random sampling.

However, since we only count the number of action but don’t record exactly how it got there. If the simulation is non-stationary, we can see how quickly the algorithm responds to the new data generating process and forget the old one in this graph of Experiment 6:

图表

描述已自动生成

In this case, the algorithm with higher emphasis on exploration will have better performance. It can quickly ‘forget’ the past actions and find the best new option.

**Thompson Sampling with non-Beta distribution (5 Points) Modify the Thompson Sampling to run with a different distribution (e.g. Parteo, Normal, etc)**

图片包含 图示

描述已自动生成

**What code is yours and what have you adapted? (10 Points)**

**You must explain what code you wrote and what you have done that is different. Failure to cite ANY code will result in a zero for this section.**

The base code is from the notebook by Andre Cianflone - Thompson sampling <https://github.com/andrecianflone/thompson/blob/master/thompson.ipynb>

I made some changes based on it:

* + I add a way to manually input the action Bernouilli parameters like “#self.thetas = np.array([0.5, 0.2, 0.3, 0.1, 0.25, 0.7, 0.1, 0.3, 0.1, 0.25])”. So, this simulation can be used on various situations. For example, should I try a new restaurant or not? If I try new one, will I happy?
  + I added the code for Thompson Sampling with normal distribution.

*# Thompson Sampling with normal distribution***class** ThompsonNormal(BetaAlgo):  
 **def** \_\_init\_\_(self, bandit):  
 super().\_\_init\_\_(bandit)  
  
 @staticmethod  
 **def** name():  
 **return 'thompson with normal distribution'  
  
 def** get\_action(self):  
 theta = np.random.normal(self.alpha, self.beta)  
 **return** theta.argmax()

* + I made some comments to explain the code clear, for example, how to calculate the q-value.
  + For the UCB algorithm, I changed the ln() to log() according to professor’s equation. “log\_timestep = np.log10(np.full(self.arm\_count, self.timestep))”
  + I added the code for random sampling algorithms and these new algorithms into the experiment list.

*# random sampling algorithm:***class** RandomSampling():  
  
 **def** \_\_init\_\_(self, bandit):  
 self.bandit = bandit  
 self.arm\_count = bandit.arm\_count  
  
 @staticmethod  
 **def** name():  
 **return 'random-sampling'  
  
 def** get\_action(self):  
 action = np.random.randint(0, self.arm\_count)  
 **return** action  
  
 **def** get\_reward\_regret(self, arm):  
 reward, regret = self.bandit.get\_reward\_regret(arm)  
 **return** reward, regret

* + I added simulation for restaurants and the explanation for the simulation parameters.

*# Simulation of restaurants:  
#  
# Situation: I come to a new town which has some restaurants, but I am not familiar with these restaurants.  
# I need to stay in this town for some days and every day I need to choose one restaurant to eat dinner.  
# The result can be: I think today's dinner is delicious / not delicious.  
# I want to maximize happy days (days I think restaurant is delicious).  
#  
# Parameters:*days = 1000 *# how many days I will stay in the town and have dinner.*simulations = 1000 *# run this simulation for how many times.*restaurants = 10 *# the number of restaurants*epsilon = 0.1 *# hyperparameter for e-greedy, 10% chance try new restaurant*ucb\_c = 2 *# hyperparameter for UBC*stationary=**True** *# Is the probability constant or change in every 100 days*experiment(restaurants, days, simulations)