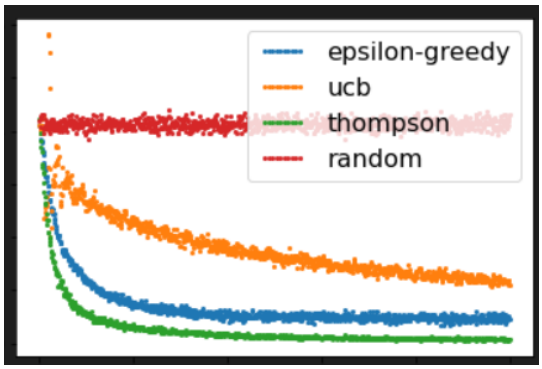
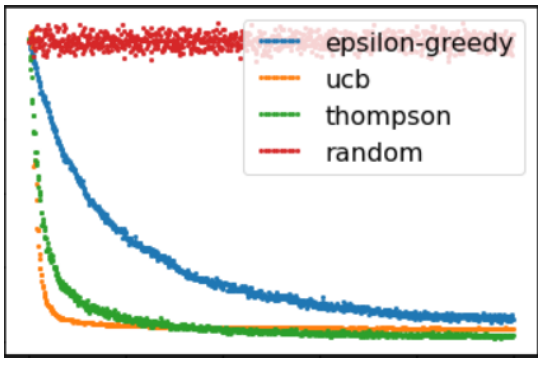
Assignment 1

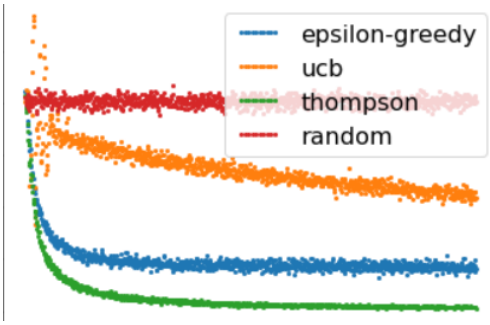
Yucong Feng

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

1. Thompson Sampling: The 'stationary' parameter is very essential because if we have the non-stationarity setting, it will re-samples the action every hundred steps.
2. Random Sampling: No hyperparameter.
3. E-greedy: Epsilon. It decides the ratio that balances exploration and exploitation. If the generated random number is less than epsilon, the agent will try a random machine in this round. If the generated random number is larger than epsilon, the agent will choose the machine that has best performance in the past rounds. According to the experiments, too small epsilon will decrease the speed of convergency, while too large epsilon will make training converge to worse level.
4. UCB: Constant c. According to the formula, it will decide how often the exploitation process is. If C is larger, the agent will be more likely to choose a machine to explore rather than to choose a known best machine to exploit: less C value makes UCB converge faster. If C is too large, the training will be too slow.

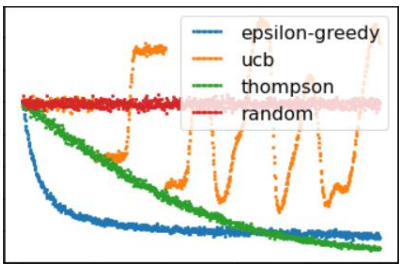


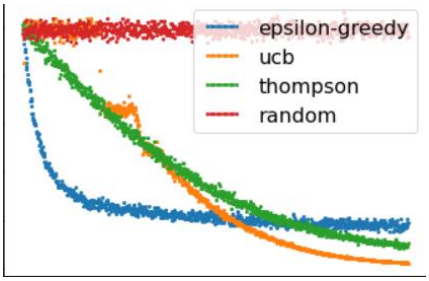


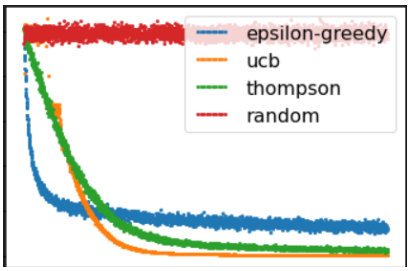


2. How does the action space affect Thompson Sampling, e-greedy, UBC, and random sampling? Show why. (15 Points)

In this setting, we increase the number of actions from 10 to 200, leaving the other parameters equal to our baseline.







All algorithms exhibit negative change as the action space increases.

1. Thompson Sampling: Convergence becomes slower but it can still converge to a lower regret level in the end.
2. Random Sampling: Keeps no-optimization shape, but the regret increasing from 0.4 to 0.5.
3. E-greedy: A fast convergence rate can still be maintained, but its performance is worse than Thompson as the number of training epochs increases.
4. UCB: The algorithm is greatly affected, do not converge at all throughout training, and is extremely unstable. However, as we reduce the value of C, the accuracy of the algorithm is greatly improved, and finally it can perform the best among the three algorithms.

The reason Thompson and UCB are heavily affected is that both require training to make the

beta distribution or bounds accurate. The larger the action space, the harder it is for them to get enough training for each machine. Therefore, they cannot decide to be accurate and correct.

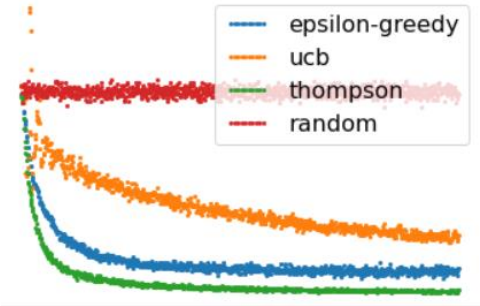
When we increase the number of trainings, the two algorithms both converge successfully and show good accuracy. This is because each machine is trained enough. For E-greedy algorithm, it need only sample a few values and has a high probability of its most favorable action's estimated reward probability being close to the true optimal reward probability, ensuring it a low regret, even if it never chooses the optimal action.

"Thompson and UCB, on the other hand, with their emphasis on exploration, give a high premium to exploring actions not yet seen. Since there are so many actions, and since the mean rewards are evenly distributed between 0 and 1, there is a high cost of constantly exploring actions with rewards close to 0. Noticeably, the Thompson algorithm is incredibly stable across time, despite the large action space. Due to its sampling nature, it has a lower probability than UCB of constantly taking suboptimal actions."[1]

[1] cited from https://github.com/andrecianflone/thompson/blob/master/thompson.ipynb

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

"We now reset our parameters to the baseline settings: 10-arm bandit, and UCB-c value of 2. However, we now set the environment to being non-stationary. Actually, after every 100 steps, the bandit resamples the action Bernouilli parameters. This places special emphasis on exploration. An algorithm that converges to an action and does not emphasize exploration will perform quite poorly as the true reward distribution changes over time. Below we see a clear jump in regret every time the true bandit parameters are resampled (every 100 steps). While the -greedy and Thompson's regret quickly decrease, on average they increase over time. On the other hand, UCB reacts much more quickly to the bandit non-stationarity, and actually continues to improve in regret over time, no doubt due to its higher emphasis on exploration. We confirm this in the next experiment."[1]



**[1] cited from https://github.com/andrecianflone/thompson/blob/master/thompson.ipynb*

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

1. Thompson Sampling: Thompson sampling stops exploring when the beta distribution of the best machine does not overlap with other machines, which means that other machines cannot produce beta values greater than the best machine. Because of the exploration-exploitation tradeoff of Thompson sampling, it will sample beta values in each round, and if there is overlap, a poorer

machine may produce larger beta values. In this way, the agent will choose the worst machine for this round as exploration.

2. Random Sampling: It will not stop exploration.

3. E-greedy: E-greedy will stop exploring when the epsilon value equals to 0 (usually this parameter will have a rate of decline). It just simply divides exploration and exploitation. When the random number is less than epsilon, the agent chooses a new machine to explore. When the random number is larger than epsilon, the agent chooses the best machine to expoilt it.

4. UBC: UBC stops exploring when the lower confidence bound of the best machine is higher than the upper confidence bounds of the other machines. Exploration stops when the lower confidence bound of the best machine is higher than the upper confidence bound of others.

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

1. Thompson Sampling: Thompson sampling always remembers past actions. Since the algorithm is based on a beta distribution, where alpha records the number of times the machine succeeds and beta records the number of times the machine fails.

2. Random Sampling: It doesn't remember any actions because the algorithm chooses randomly each time.

3. E-greedy: The algorithm doesn't remember the actual actions, but instead calculates the choice it should make next by recording the average reward of all past actions throughout the experiment.

4. UBC: UBC doesn't remember the actual actions, but it will judge the next action by the sum of the current reward mean and the standard deviation of the mean, where the standard deviation is calculated by the current number of all actions and the current number of times the machine has been selected.

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

```
# beta -> normal
class BernThompsonNormal():
    def __init__(self, bandit):
        self.bandit = bandit
        self.arm_count = bandit.arm_count


        self.record = [] # save the result of each round, n * m
        for i in range(0,self.arm_count):
            self.record.append([1]) # initialize every machine win once at the begining


        self.mean = np.ones(self.arm_count) # initialize mean = 1
        self.var = np.zeros(self.arm_count) # var = 0



    def get_reward_regret(self, arm):
        reward, regret = self.bandit.get_reward_regret(arm)
        self._update_params(arm, reward)
```

```
        return reward, regret

    def _update_params(self, arm, reward):
        self.record[arm] = np.append(self.record[arm],reward) # save result
        self.mean[arm] = self.record[arm].mean() # calculate and save mean
        self.var[arm] = self.record[arm].var() # calculate and save var

    @staticmethod
    def name():
        return 'normal-thompson'

    def get_action(self):
        """ Bernouilli parameters are sampled from the normal distribution"""
        theta = np.random.normal(self.mean, self.var) # normal distribution
        return theta.argmax()

def experiment_(arm_count, timesteps=1000, simulations=1000):
    algos = [EpsilonGreedy, UCB, BernThompsonNormal, BernThompson, RandomSampling]
    regrets = []
    names = []
    for algo in algos:
      regrets.append(simulate(simulations, timesteps, arm_count, algo))
      names.append(algo.name())
    multi_plot_data(regrets, names)
```

7. What code is yours and what have you adapted (10 Points)

The part of the Question 6 that replaces the beta distribution with the normal distribution in Thompson Sampling :

1. Improved the code of the experimental part and used the control variable method to discuss different hyperparameters separately, which made the answer more accurate and more in line with the requirements of the question.
2. Added more reasonable comments to the adapted code and the code I wrote to improve the readability of the code.

8. Did I explain my code clearly? (10 Points) Your code review score will be scaled to a range of 0 to 10 and be used for this score.

Yes, I am sure that I explained my code clearly. I've added comments and markdown where needed. You can see them when you browse this jupyter documentation.

9. Did I explain my licensing clearly? (5 Points) Failure to cite a clear license will result in a zero for this section.

Yes, I am sure that I explained my licensing clearly. The following is my MIT License. And I also zipped my License file as an attachment in the folder.