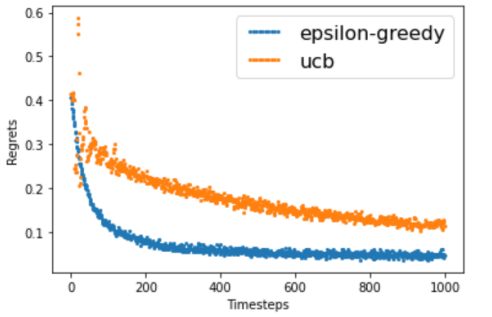
CSYE 7370 Deep Learning and Reinforcement Learning in Game Engineering

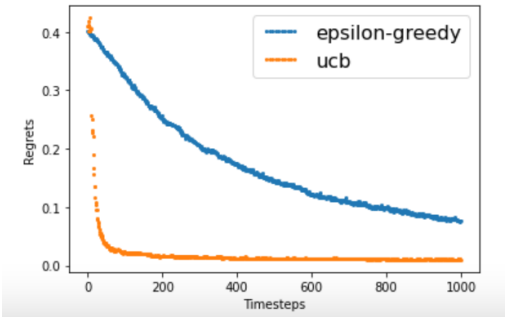
Bandit Problems - Multi-Armed Bandit Problem.
Optimize the Bandit Problem using Thompson Sampling, ε-greedy, UCB, and random sampling.

1. **Which hyperparameters are important for Thompson Sampling, ε-greedy, UCB, and random sampling? Show that they are important (15 Points)**
   Different hyperparameters will make the model different, and we need to set the value before starting the learning process. In deep learning, hyperparameters include learning rate, number of iterations, number of layers, and so on. For these models, the hyperparameter epsilon is to the Epsilon Greedy model and the hyperparameter ucb_c is to the UCB model. The following experiment can see the effect of adjusting the hyperparameters on the results.

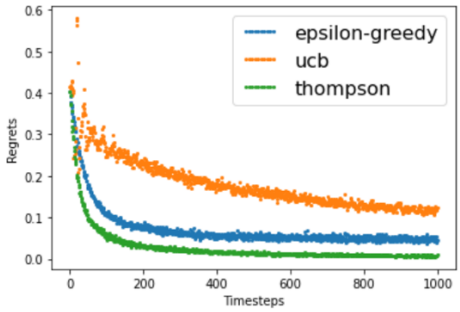
```
# Experiment 1
arm_count = 10 # number of arms in bandit
epsilon = 0.1
ucb_c = 2
stationary=True
experiment(arm_count)
```

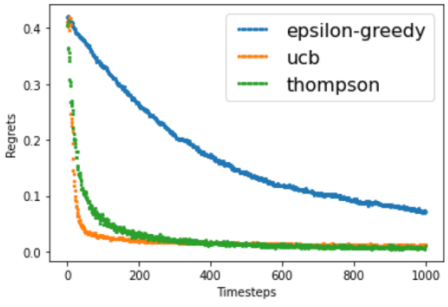



```
# Experiment 2 - change hyperparameters
arm_count = 10 # number of arms in bandit
epsilon = 0.01
ucb_c = 0.2
stationary=True
experiment(arm_count)
```



Using Experiment 1 as the baseline, we can compare the consequences of adjusting the hyperparameters (Experiment 2). When ε is decreasing to 0.01, its regret values have not decreased, but have increased. On the other hand, when ucb_c is decreasing to 0.2, it will quickly converge to a low regret value, which is far better than ε-greedy. At this time, the merit of the UCB model is obvious.



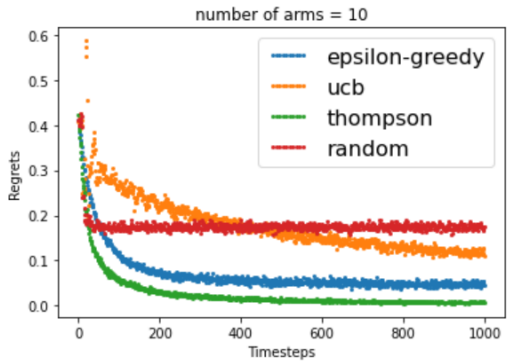


Then I add the Thompson Sampling model for comparison, and we note that that at first it outperforms Thompson. In the long run, although the UCB model has almost approached the Thompson Sampling model, the regret value has never been lower than it.

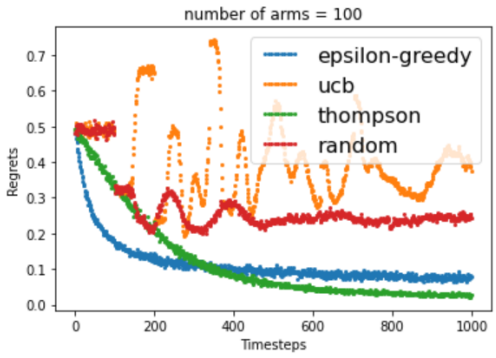
2. **How does the action space affect Thompson Sampling, ε-greedy, UCB, and random sampling? Show why. (15 Points)**

   When all other condition factors are the same, we can observe that UCB model becomes very unstable and divergent after increasing the action space. It's also because Bernoulli reward probabilities are close to each other. And random sampling is unstable at the beginning but tends to be stable later. Then, we can notice that the ε-greedy result has hardly changed after adjusting the action space. Thompson's early performance is not as good as ε-greedy model, but we can find that its regrets are gradually reduced and it's worse than ε-greedy at about timestep 300. In this case, I believe that Thompson's is optimal, because the Thompson algorithm is very stable in time in the long run.

```
# Experiment 3
arm_count = 10 # number of arms in bandit
epsilon = 0.1
ucb_c = 2
stationary=True
plt.title("number of arms = 10")
experiment(arm_count)
```




```
# Experiment 4
arm_count = 100 # number of arms in bandit
epsilon = 0.1
ucb_c = 2
stationary=True
plt.title("number of arms = 100")
experiment(arm_count)
```



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

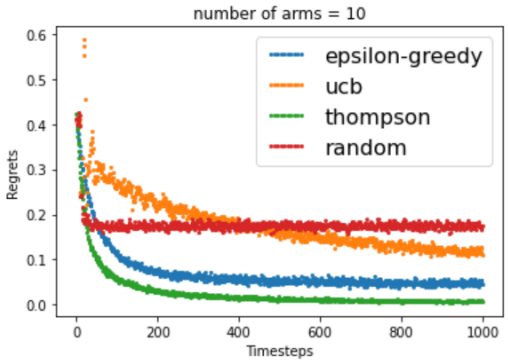
   If the machine is the same, and the distribution of rewards sampled at each time step remains unchanged, this is called stationary. But if there is a deviation, it will be a non-stationary problem. Now we simulate this situation and set its environment to be non-stationary.

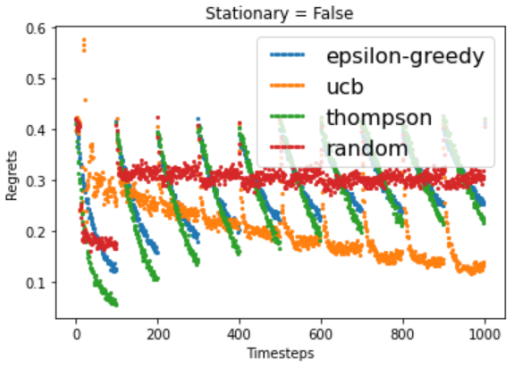
   From the figure below, we can see that every 100 steps, regret will increase

significantly. Moreover, the UCB model reacts much faster to bandit non-stationarity, much better than other models, and in fact continues to improve in regret over time, which is due to its high emphasis on exploration.

```
# Experiment 3
arm_count = 10 # number of arms in bandit
epsilon = 0.1
ucb_c = 2
stationary=True
plt.title("number of arms = 10")
experiment(arm_count)
```

```
# Experiment 5 - Change its stationary to False
arm_count = 10 # number of arms in bandit
epsilon = 0.1
ucb_c = 2
stationary=False
plt.title("Stationary = False")
experiment(arm_count)
```

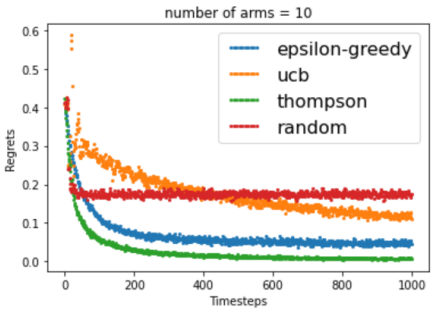


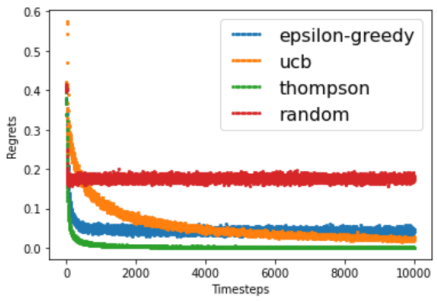


4. **When do Thompson Sampling, ε-greedy, UCB, and random sampling stop exploring? Explain why. Explain the exploration-exploitation tradeoff (15 Points)**
   The regrets on a game with 10000 rounds for each model and run across 1000 simulations. In the graph below, increasing the timesteps to 10000 confirms that Thompson and UCB closely converge. We see that Thompson sampling greatly outperforms the other models. Also, UCB catch up to ε-greedy at around timesteps 3000. https://gdmarmerola.github.io/ts-for-bernoulli-bandit/

```
# Experiment 3
arm_count = 10 # number of arms in bandit
epsilon = 0.1
ucb_c = 2
stationary=True
plt.title("number of arms = 10")
experiment(arm_count)
```

```
# Experiment 6
arm_count = 10 # number of arms in bandit
epsilon = 0.1
ucb_c = 2
stationary=True
experiment(arm_count, timesteps=10000, simulations=1000)
```

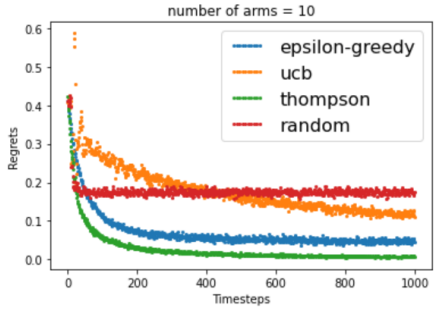




In term of exploration-exploitation tradeoff, for the ε-greedy strategy, it is necessary to adjust the hyperparameters to tradeoff exploration and exploitation. But this is not an ideal method because it may be difficult to adjust. The UCB pick the arm that

has the maximum value at the upper confidence bound to balance exploration and exploitation. For Thompson Sampling, we can see that it performs efficient exploration, quickly ruling out fewer promising arms.

5. **How long do Thompson Sampling, ε-greedy, UCB, and random sampling remember the past actions? Explain your answer. (10 Points)**
   Any action either succeeds or fails when performed. The success probabilities are unknown to the agent, but are fixed over time; therefore, they can be learned by experimentation. The goal is to minimize the regret value over entire timestep and maximize the number of successes. In Thompson Sampling model, it tends to converge, and the regret value is already close to 0.0 after 300 timesteps. In ε-greedy model, it is difficult to see progress after 400 timesteps, and the regret value remains between 0.0 and 0.1 until the end. In UCB model, its learning ability in this case is not as good as the previous two algorithms over time. Although he was slow to reduce regret, it was not lower than 0.1 before the end.



6. **Thompson Sampling with non-Beta distribution (5 Points) Modify the Thompson Sampling to run with a different distribution (e.g. Parteo, Normal, etc)**
   Please find the Notebook Line 210
   I modified the Thompson sampling of the normal distribution. We know that sample θi(t) from the Beta(Si + 1, Fi + 1) in Beta distribution, then if rt = 1, then Si(t) = Si(t) + 1, else Fi(t) = Fi(t) + 1. On the other hand, sample θi(t) independently from the N (ˆμi,1/ki+1) in Gaussian distribution, and then setting ˆμi(t) =(ˆμi(t)ki(t)+rt) /ki(t)+1, ki(t) = ki(t) + 1 to update its parameter.

```
def _update_params(self, arm, reward):
  """ Gaussian parameters return a value from the the posterior normal distribution """
  self.μ_0[arm] = ((self.τ_0[arm] * self.μ_0[arm]) + (self.N[arm] * self.Q[arm]))/(self.τ_0[arm] + self.N[arm])
  self.τ_0[arm] += 1
```

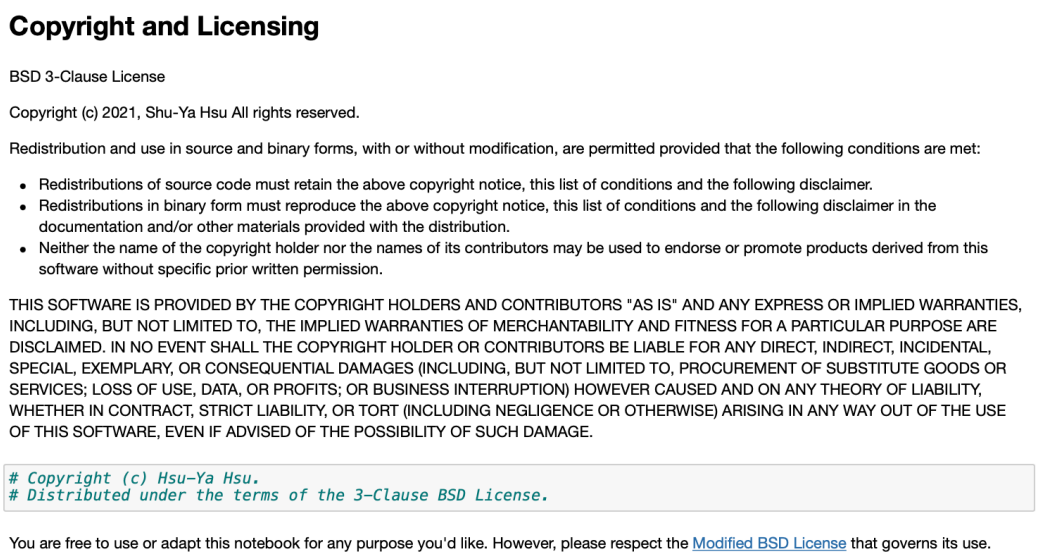
7. **What code is yours and what have you adapted? (10 Points)**

For details, see the section of References at the bottom of the notebook

GitHub for my notebook: https://github.com/leahsu/CSYE7370

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

   The license is at the bottom of the notebook. The following screenshot for reference.



Copyright and Licensing

BSD 3-Clause License

Copyright (c) 2021, Shu-Ya Hsu All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

```
# Copyright (c) Hsu-Ya Hsu.
# Distributed under the terms of the 3-Clause BSD License.
```

You are free to use or adapt this notebook for any purpose you'd like. However, please respect the Modified BSD License that governs its use.