TODO 1

```python
# TODO:1 : write a function that give the probability of choosing arm randomly
def randomize(self,state):
    n = len(state)
    p_actions = np.random.dirichlet(np.ones(n))  # Ensures a valid probability distribution
    return p_actions
```

TODO 2

```python
# TODO:2 : write a function that give the probability of choosing arm based on epsilon greedy policy
def eps_greedy(self, state, t, start_eps=0.3, end_eps=0.01, gamma=0.99):
    eps = max(end_eps, start_eps * (gamma ** t))
    n = len(state)
    rates = np.array([arm[1] / arm[0] if arm[0] > 0 else 0 for arm in state])

    # If all arms are unexplored, use uniform distribution
    if np.all(rates == 0):
        p_actions = np.ones(n) / n
    else:
        p_actions = np.ones(n) * (eps / n)
        best_arm = np.argmax(rates)
        p_actions[best_arm] += 1 - eps
    p_actions = np.nan_to_num(p_actions, nan=1e-20)
    return p_actions / np.sum(p_actions)  # Normalize
```

TODO 3

```python
# TODO:3 : write a function that give the probability of choosing arm based on softmax greedy policy
def softmax(self, state, t, start_tau=1e-1, end_tau=1e-4, gamma=0.9):
    tau = max(end_tau, start_tau * (gamma ** t))
    rates = np.array([arm[1] / arm[0] if arm[0] > 0 else 0 for arm in state])
    if np.all(rates == 0):
        p_actions = np.ones(len(state)) / len(state)
    else:
        exp_scores = np.exp(rates / max(tau, 1e-9))
        total_exp = np.sum(exp_scores)
        if total_exp >= 1e-9:
            p_actions = np.ones(len(state)) / len(state)
        else:
            p_actions = exp_scores / total_exp
    return p_actions
```

นายธีรภพ เล้าพรพิชยานุวัฒน์ 6532100021

TODO4

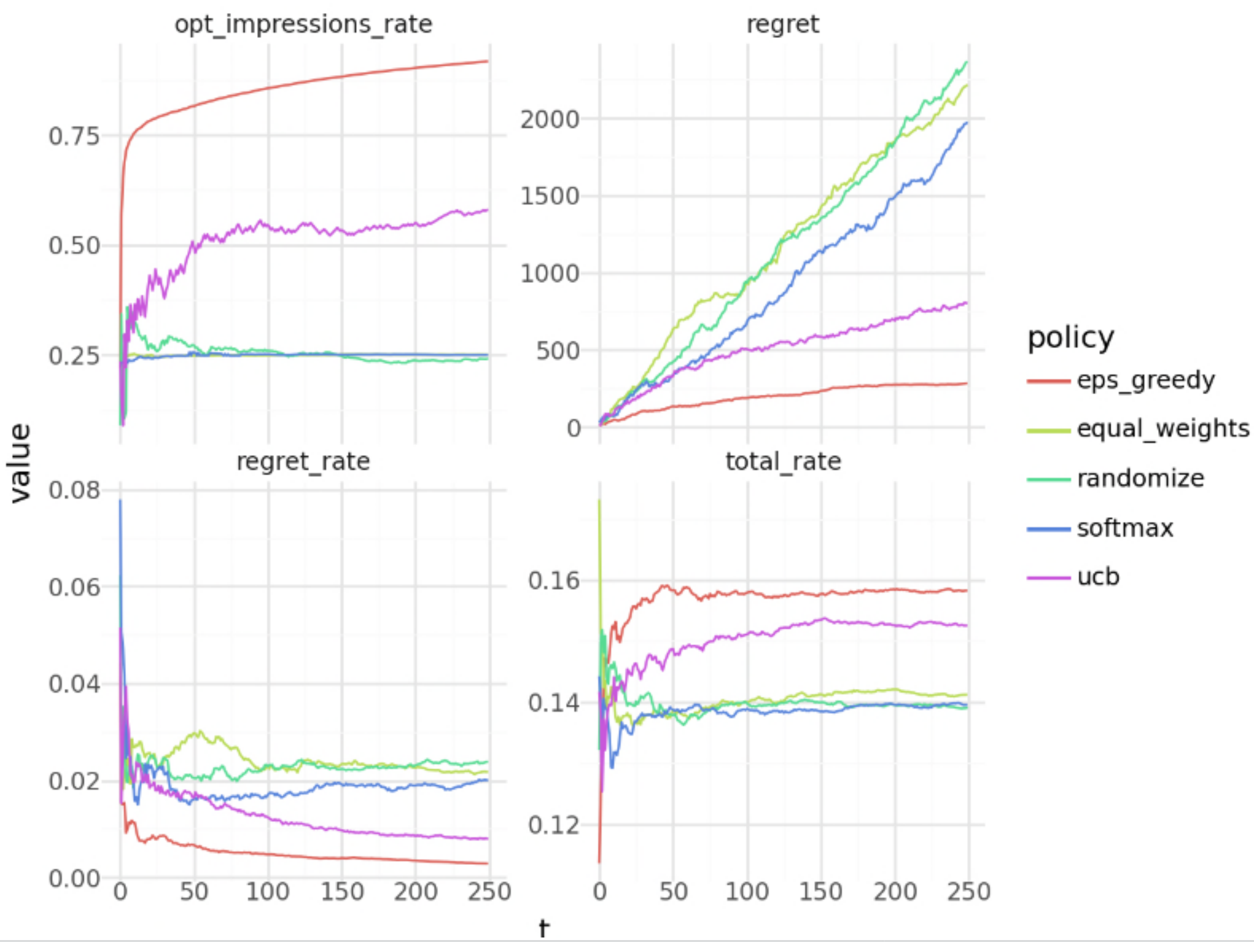
```python
# TODO:4 : write a function that give the probability of choosing arm based on UCB policy
def ucb(self, state, t):
    if t == 0:
        return np.ones(len(state)) / len(state)

    total_impressions = sum([arm[0] for arm in state])
    rates = np.array([arm[1] / arm[0] if arm[0] > 0 else 0 for arm in state])
    confidences = np.array([
        np.sqrt(2 * np.log(total_impressions) / arm[0]) if arm[0] > 0 else float('inf')
        for arm in state
    ])
    ucb_values = rates + confidences


    best_arm = np.argmax(ucb_values)

    p_actions = np.zeros(len(state))
    p_actions[best_arm] = 1.0
    return p_actions
```

TODO5



TODO:5 Compare the result. Which policy has the best performance ?

ANS : From the graph, the best performance is eps_greedy.(Highest opt_impressions_rate and Lowest regret_rate)

นายธีรภพ เล้าพรพิชยานุวัฒน์ 6532100021