

TODO #1

```
# TODO:1 : write a function that give the probability of choosing arm randomly
def randomize(self, state):
    probs = np.random.random(len(state))
    return probs / np.sum(probs)
```

TODO #2

```
# 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):
    # epsilon decay
    eps = max(start_eps * gamma**t, end_eps)
    if np.random.random() <= eps:
        return self.equal_weights(state)
    else:
        rates = np.array([state[i][1] / state[i][0] if state[i][0] > 0 else 0 for i in range(len(state))])
        best_arm = np.argmax(rates)
        probs = np.zeros(len(state))
        probs[best_arm] = 1.0
        return probs
```

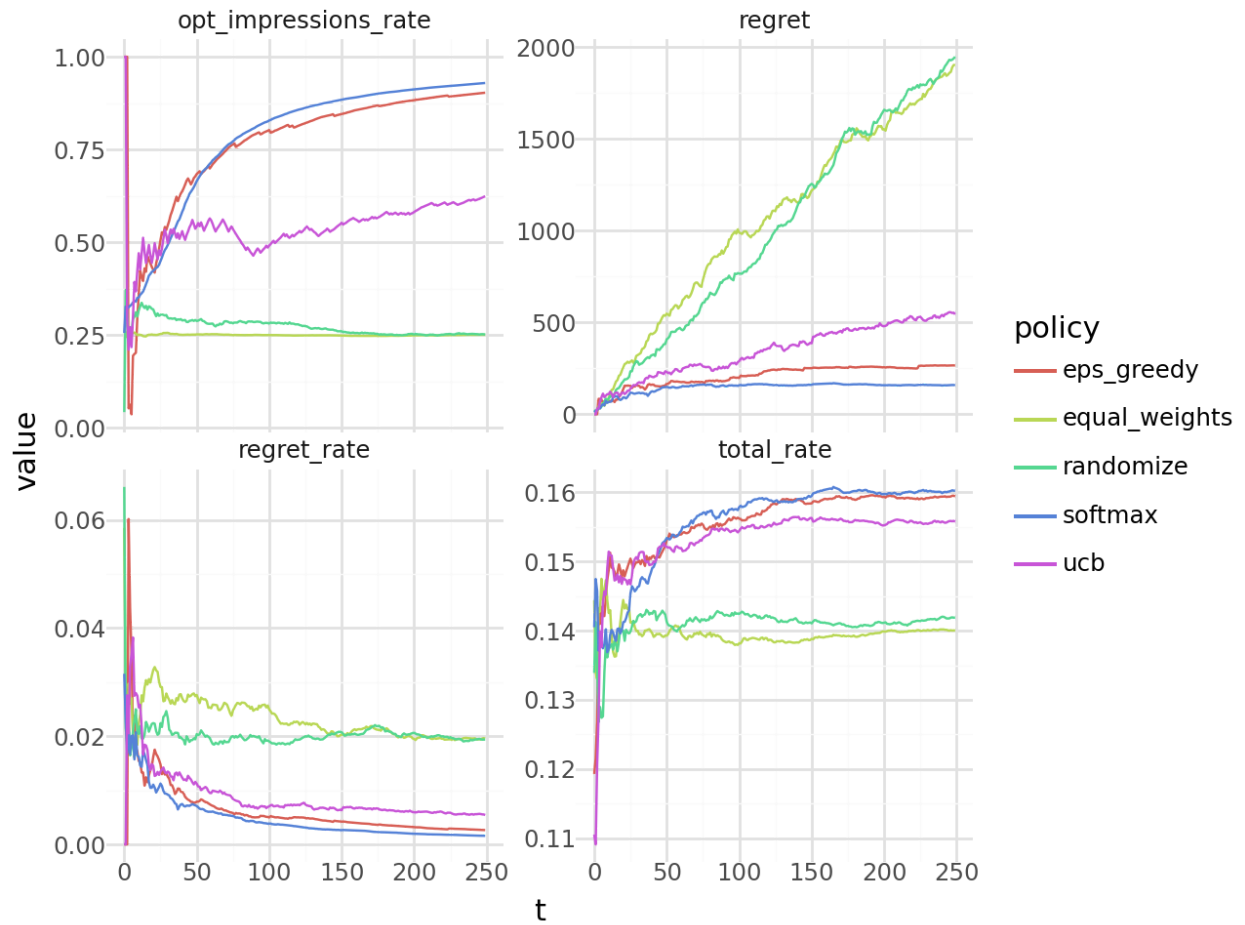
TODO #3

```
# 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(start_tau * gamma**t, end_tau)
    rates = np.array([state[i][1] / state[i][0] if state[i][0] > 0 else 0 for i in range(len(state))])
    if np.all(rates == 0):
        return self.equal_weights(state)
    max_rate = np.max(rates)
    exp_values = np.exp((rates - max_rate) / tau)
    probs = exp_values / np.sum(exp_values)
    return probs
```

TODO #4

```
# TODO:4 : write a function that give the probability of choosing arm based on UCB policy
def ucb(self, state, t):
    ucb_values = [
        (state[i][1] / state[i][0] if state[i][0] > 0 else 0) +
        np.sqrt((2 * np.log(max(t, 1))) / max(state[i][0], 1))
        for i in range(len(state))
    ]
    best_arm = np.argmax(ucb_values)
    probs = np.zeros(len(state))
    probs[best_arm] = 1.0
    return probs
```

TODO #5



From the result, the softmax policy has the best performance, as it has the highest impressions rate and lowest regret