Multi Arm Bandit

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 / 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 = start\_eps \* gamma\*\*t          if eps < end\_eps:              eps = end\_eps          if np.random.random() <= eps:              # explore              return self.equal\_weights(state)          else:              # exploit              probs = np.zeros(len(state))              # state = [impressions, actions]              # choose the arm with the highest rate              probs = np.array([state[i][1] / state[i][0] if state[i][0] > 0 else 0 for i in range(len(state))])                maxVal = max(probs)              probs = np.array([1 if probs[i] == maxVal else 0 for i in range(len(probs))])              return probs / sum(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 decay          tau = start\_tau \* gamma\*\*t          if tau < end\_tau:              tau = end\_tau          # softmax          probs = np.array([(state[i][1] / state[i][0]) if state[i][0] > 0 else np.inf for i in range(len(state))])          if np.isinf(probs).any():              # if there is an arm that has not been pulled yet, pull it              probs = np.array([1 if state[i][0] == 0 else 0 for i in range(len(state))])          else:              maxVal = max(probs)              probs = np.exp((probs - maxVal) / tau)          return probs / sum(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          probs = np.zeros(len(state))          # state = [impressions, actions]          # choose the arm with the highest UCB          probs = np.array([state[i][1] / state[i][0] + np.sqrt(2 \* np.log(t) / state[i][0]) if state[i][0] > 0 else np.inf for i in range(len(state))])          maxVal = max(probs)          probs = np.array([1 if probs[i] == maxVal else 0 for i in range(len(probs))])          return probs / sum(probs) |

TODO #5

|  |
| --- |
| From the result, the best performance is SoftMax policy. (Highest Impressions rate and Lowest regret) |