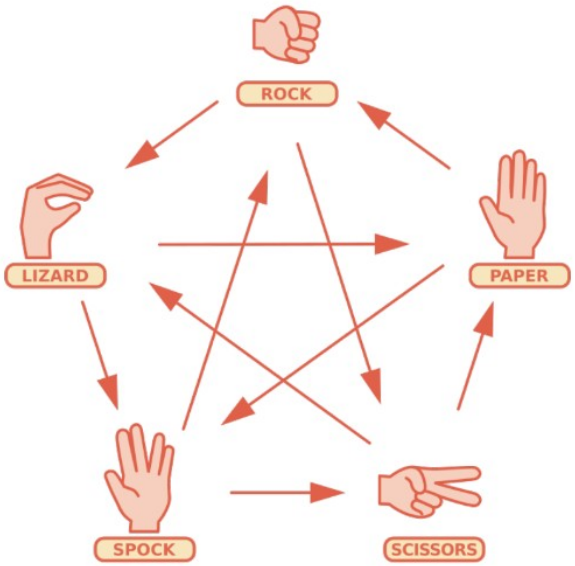
Assignement 1

Introduzione problema

- Rock, Paper, Scissor, Lizard, Spock
- Si sceglie quale azione giocare tra di esse
- L’avversario gioca casualmente
- Reward +1 per la vincità e -1 per la sconfitta
- L’obiettivo è massimizzare i reward



Avversario

- Sono stati implementati 3 tipi di avversario:
  - Gioca tutte le mosse con probabilità uguale

    Ha una leggera preferenza verso una mossa
  - Cambia preferenza ogni N mosse (problema non stazionario)

- Il bandito ha una mappa per le azioni vincenti

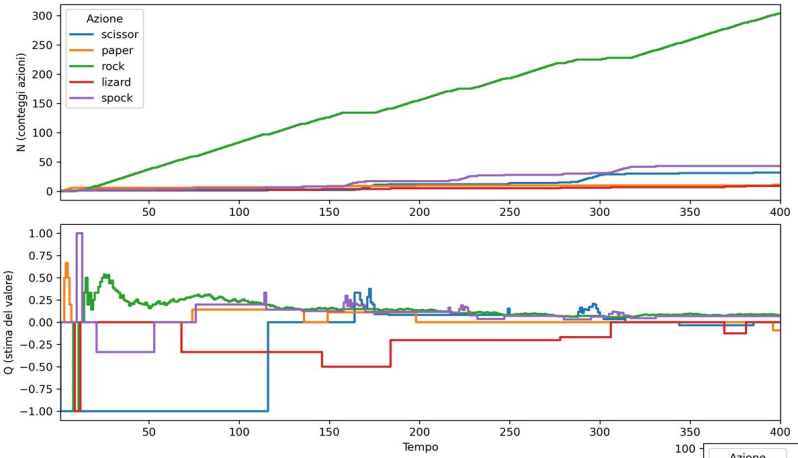
```
def stationary_player():  3 usages  gronz
    if np.random.random() < eps_player:
        a = np.random.randint( low: 0,dim_A)
    else:
        a = 2 #rock
    return a
def non_stationary_player(t):  3 usages  gronz *
    if np.random.random() < eps_player:
        a = np.random.randint( low: 0,dim_A)
    else:
        a = int(t/N) % 5
    return a
```

ε-greedy

- Scelta azione con ε-greedy
- Aggiornamento basato sulla stazionarietà

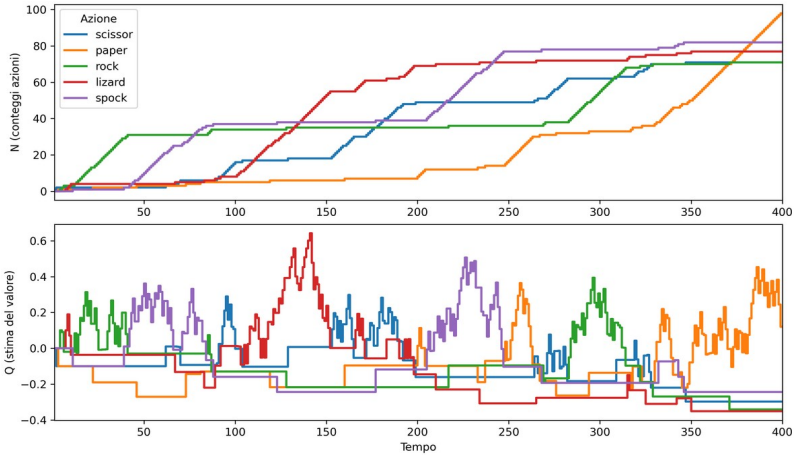
```
def play_eps(sta=True):  2 usages  gronz *
    Q = np.zeros((dim_A,T))
    N = np.zeros((dim_A,T))
    alp=0.1
    for t in range(T-1):
        a_agent = epsilon_greedy(Q[:,t])
        a_player = stationary_player() if sta else non_stationary_player(t)
        r = bandit(a_agent,a_player)
        N[:,t+1] = N[:,t]
        N[a_agent,t+1] = N[a_agent,t]+1
        #aggiorna usando media empirica iterativa
        Q[:,t+1] = Q[:,t]
        if sta:
            Q[a_agent,t+1] = Q[a_agent,t] + 1/N[a_agent,t+1]*(r-Q[a_agent,t])
        else:
            Q[a_agent,t+1] = Q[a_agent,t] + alp*(r-Q[a_agent,t+1])
```

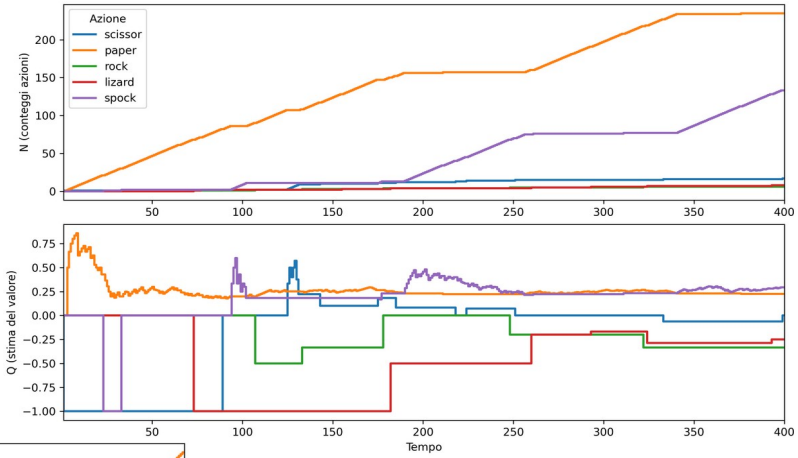
ε-greedy



Uniforme

Non stazionario





Stazionario

Upper confidence bound

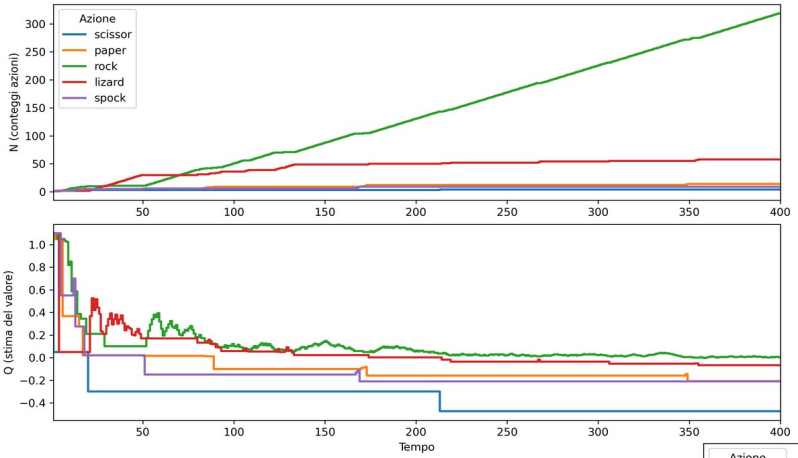
- Scelta azione UCB

$$A_t = \arg\max_a \left\{ Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right\}$$

- Optimistic initialization

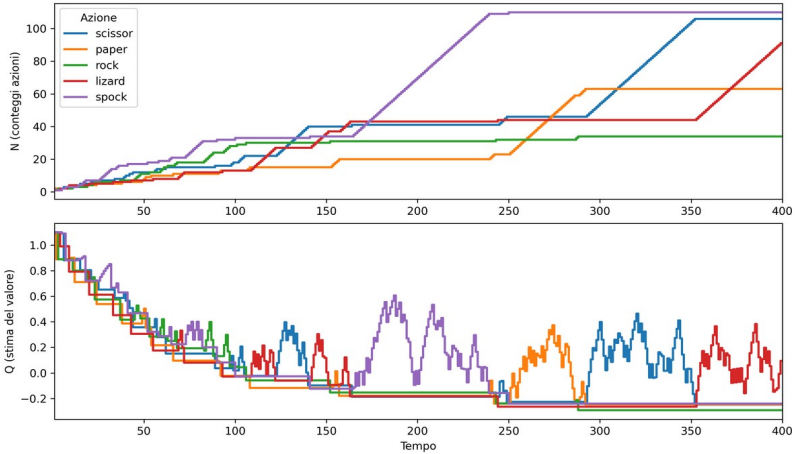
```
def play_ucb(sta=True):  2 usages  gronz
    Q = np.ones((dim_A,T))*1.1
    N = np.ones((dim_A,T))
    c = 0.3
    alp = 0.1
    for t in range(T-1):
        a_agent = np.argmax(np.add(Q[:,t],c*np.sqrt(np.log(t+1)/N[:,t])))
        a_player = stationary_player() if sta else non_stationary_player(t)
        r = bandit(a_agent,a_player)
        N[:,t+1] = N[:,t]
        N[a_agent,t+1] = N[a_agent,t]+1

        Q[:,t+1] = Q[:,t]
        if sta:
            Q[a_agent,t+1] = Q[a_agent,t] + 1/N[a_agent,t+1]*(r-Q[a_agent,t])
        else:
            Q[a_agent,t+1] = Q[a_agent,t] + alp*(r-Q[a_agent,t])
```

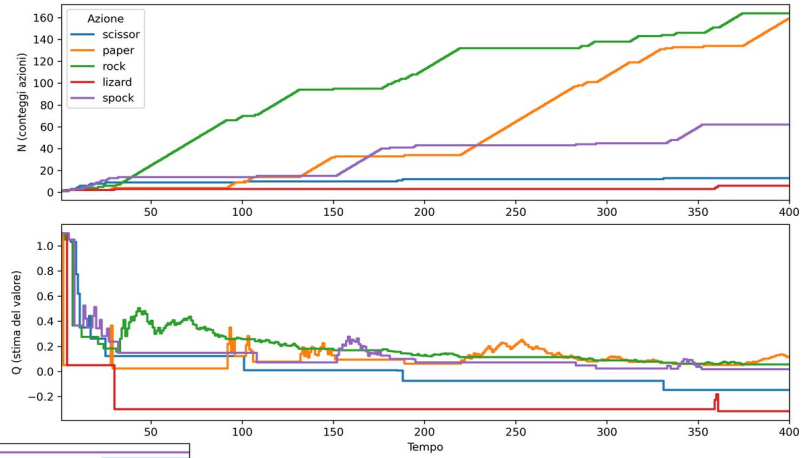
Upper confidence bound



Uniforme

Non stazionario





Stazionario

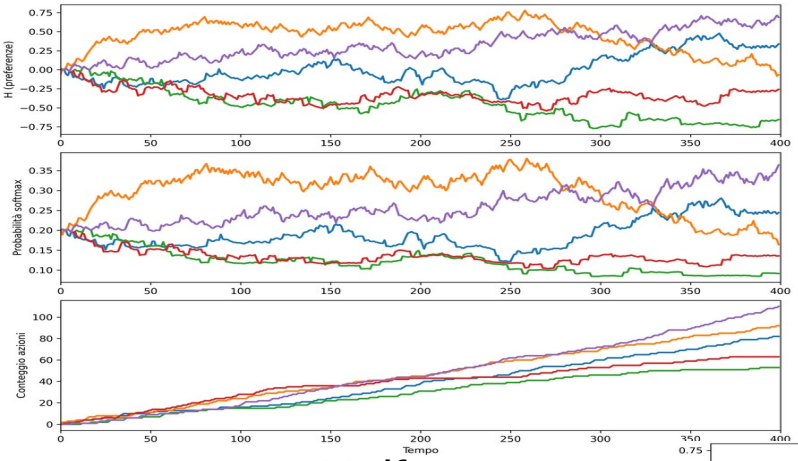
Preference updates

- Scelta azione tramite softmax
- Aggiornamento preferenze

$$\begin{cases} H_{t+1}(A_t) = H_t(A_t) + \alpha \left(R_t - \overline{R}_t\right)\left(1 - \pi_t(A_t)\right), & \text{and} \\ H_{t+1}(a) = H_t(a) - \alpha \left(R_t - \overline{R}_t\right)\pi_t(a), & \forall a \neq A_t, \end{cases}$$

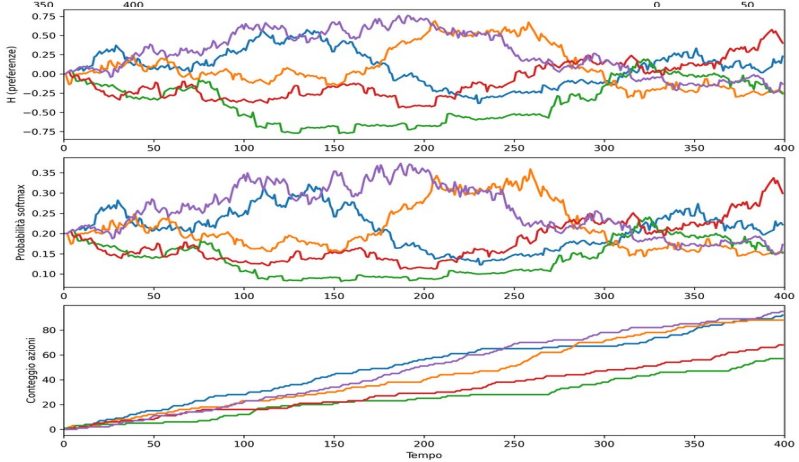
```python
def preference_update(sta = True):  2 usages  gronz *
    H = np.zeros(dim_A)
    historyH = np.zeros((dim_A, T))
    historyProb = np.zeros((dim_A, T))
    historyN = np.zeros((dim_A, T + 1))
    bR=0
    beta=0.1
    alpha=0.1

    for t in range(T):
        expH = np.exp(H - np.max(H))  # stabilità numerica
        Prob = expH / np.sum(expH)



        a = np.searchsorted(np.cumsum(Prob), np.random.rand())
        a_player = stationary_player() if sta else non_stationary_player(t)
        r = bandit(a,a_player)

        # Aggiornamento reward medio
        bR = bR + beta * (r - bR)
        temp = H[a]
        # Aggiornamento preferenze
        H = H - alpha * (r - bR) * Prob
        H[a] = temp + alpha * (r - bR) * (1 - Prob[a])



        historyH[:, t] = H
        historyProb[:, t] = Prob
        historyN[a, t] += 1
        historyN[:, t + 1] = historyN[:, t]
```

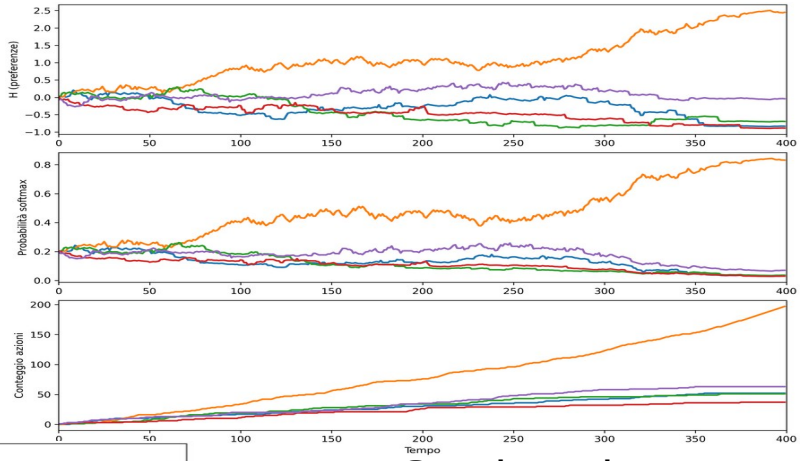
Preference updates



Uniforme

Non stazionario





Stazionario

FINE