Laboratorio 01: MRP, MDP, Value Function

Julio Ticona Quispe

December 4, 2023

1 Pregunta 1: [6pt, SO1]

Sea un MDP con factor de descuento $0 < \gamma < 1$ y sin estado terminal. Esto quiere decir que siempre actua el agente. La función de valor óptima original es V_1^* y la política óptima es π_1^*.

1. Se decide colocar una pequeña bonificación c al premio en todas las transiciones del MDP. La función de premios es ahora $\hat{r}(s,a) = r(s,a) + c, \forall s, a$, donde $r(s,a)$ es la función de premios original.

   - **Indique cual sería la expresión de la nueva función de valor.**

     La ecuación de Bellman para la función de valor óptima $V_1^*(s)$ en un MDP sin estado terminal y con factor de descuento γ es:

     $$\hat{V}_a^*(s) = \max_a r(s,a) + \gamma \sum P(s'|s,a) V_a^*(s') \tag{1}$$

     Ahora, con la nueva función de premios $\hat{r}(s,a) = r(s,a) + c$, la ecuación de Bellman se convierte en:

     $$\hat{V}_a^*(s) = \max_a \hat{r}(s,a) + \gamma \sum P(s'|s,a) V_a^*(s') \tag{2}$$

     Sustituyendo la expresión de $\hat{r}(s,a)$

     $$\hat{V}_a^*(s) = \max_a ((r(s,a) + c) + \gamma \sum P(s'|s,a) V_a^*(s')) \tag{3}$$

   - **Cambiará la política óptima ante esta nueva configuración?**

     Añadir una bonificación c en la función de premios no debería cambiar la política óptima si la bonificación es un valor adicional y constante para todas las transiciones. La política óptima π_1^* continuara siendo igual, ya que está determinada por la función de valor óptima V_1^*, y al añadir una bonificación no afectara las comparaciones relativas entre las acciones.

2. Si en vez de aplicar una bonificación pequeña al premio, ahora se multiplica los premios por una constante $c \in R$ cualquiera. La nueva función de premio es ahora $\hat{r}(s,a) = c.r(s,a)$, $\forall s$, donde $r(s,a)$ es la función de premio original.

   - **Podrán existir casos en que la nueva política continué siendo π_1^* y la función de valor pueda expresarse en función de c y V_1^*. en caso afirmativo indiquen la expresión resultante explicando para qué valores de c se cumple y justifique. En caso negativo, también justifique.**

     Cuando multiplicamos los premios por una constante c, la nueva función de premio es $\hat{r}(s,a) = c.r(s,a)$, hay un escenario específico en el cual la política óptima original π_1^* podría permanecer igual. Este caso especial ocurre cuando c es una constante positiva no negativa ($0 \leq c$).

     Esto se debe a que al multiplicar, estamos escalando las recompensas, y la relación entre las acciones sigue siendo la misma. Las acciones que eran probables anteriormente seguirán siendo probables después de la operación de multiplicación. Dado que la política óptima se basa en estas relaciones, no cambia.

La nueva función de valor $\hat{V}_1^*$ en función de c y la función de valor original V_1^* se expresaría de la siguiente manera:

$$\hat{V}_a^*(s) = \max_a((c.r(s,a)) + \gamma \sum P(s'|s,a)V_a^*(s')) \tag{4}$$

- **Podrán existir casos en que la nueva política cambiaría?, en caso afirmativo colocar un valor de c y una descripción de por qué cambiaría la política óptima. También justifique el caso negativo.**

  Si c fuera negativo, la situación cambiaría significativamente, ya que se podria invertir la dirección de las probabilidades y afectariamos la política óptima original. Por otro lado, si $c = 0$, la política óptima original no sería válida, ya que anularía todas las recompensas y no habría una recompensa que incentive al agente.

  Por ejemplo, si originalmente una acción a_1 tenía una recompensa positiva mayor que una acción a_2, después del escalado, si c es negativo, la acción a_1 tendría una recompensa negativa mayor que $c.\hat{r}(s,a)$. Esto podría hacer que a_2 sea preferido sobre a_1 en términos de maximizar la recompensa esperada, lo que cambiaría la política óptima.

3. Si el MDP pasa a ser episódico (tiene uno o más estados terminales s_T):

   - **Cambiaría la respuesta de la pregunta a. En caso afirmativo, dar un un ejemplo de MDP donde se muestre como difieren las respuestas en el caso de la pregunta a. y en este caso episódico.**

     Para los estados terminales, la nueva función de valor es simplemente la bonificación c, ya que no hay transiciones a otros estados desde los estados terminales:

$$\hat{V}_1^*(s_{terminal}) = c \tag{5}$$

2 Pregunta 2: [14pt, SO5, SO6]

1. Se debe implementar tanto iteración de valor y la iteración de política para el ambiente Frozen Lake de Gymnasium cuya versión original está en OpenAI Gym.

   policy_evaluation En cada iteración, realiza un bucle para recorrer sobre todos los estados (nS). Para cada estado, calculamos un nuevo valor utilizando la política especificada y la nueva matriz de transición. Actualizamos la función de valor para cada estado con el nuevo valor calculado. Luego se verifica si la diferencia máxima entre las funciones de valor en esta iteración y la iteración anterior es menor que la tolerancia permitida en los parámetros. Si es así, se termina el bucle y retornamos la función de valor calculada.

```
1   def policy_evaluation(P, nS, nA, policy, gamma=0.9, tol=1e-3):
2
3       while True:
4           delta = 0
5
6           for s in range(nS):
7               new_value = 0
8
9               for prob, next_state, reward, done in
                    P[s][policy[s]]:
10                  new_value += prob * (reward + gamma *
                        value_function[next_state])
11
12              delta = max(delta, np.abs(new_value -
                    value_function[s]))
13
14              value_function[s] = new_value
```

```
15
16              if delta < tol:
17                  break
18          return value_function
```

Listing 1: Función policy_evaluation

policy_improvement

La función **policy_improvement** utiliza la función de valor calculada desde la política actual para seleccionar las acciones óptimas en cada estado, construyendo una nueva política que mejorara el rendimiento (maximizar del valor esperado acumulado).

```
1     def policy_improvement(P, nS, nA, value_from_policy, policy,
          gamma=0.9):
2
3         new_policy = np.zeros(nS, dtype="int")
4
5         for s in range(nS):
6             q_values = np.zeros(nA)
7             for a in range(nA):
8                 q_values[a] = sum(prob * (reward + gamma *
                      value_from_policy[next_state])
9                                   for prob, next_state, reward, _
                                      in P[s][a])
10
11            new_policy[s] = np.argmax(q_values)
12        return new_policy
```

Listing 2: Función policy_improvement

policy_iteration

Utilizamos **policy_iteration** para encontrar una política óptima. En esta función se combina con la función **policy_evaluation** para obtener la función de valor de la política actual con **policy_improvement** para actualizar la política basada en la función de valor. Este proceso es iterativo, se repite hasta que la política no cambie significativamente entre iteraciones, lo cual indicara que se ha alcanzado una política óptima.

```
1     def policy_iteration(P, nS, nA, gamma=0.9, tol=1e-3):
2         value_function = np.zeros(nS)
3         policy = np.zeros(nS, dtype=int)
4
5         check_policy = 1
6
7         while check_policy>0:
8             value_function = policy_evaluation(P, nS, nA, policy,
                  gamma, tol)
9
10            improved_policy = policy_improvement(P, nS, nA,
                  value_function, policy, gamma)
11
12            check_policy = np.linalg.norm(improved_policy -
                  policy, ord=1)
13
14            policy = improved_policy
15        return value_function, policy
```

Listing 3: Función policy_iteration

value_iteration

Combinamos la **value_iteration** con **policy_improvement** para aprender la función de valor y la política óptima para un entorno dado. Esta función nos permitirá converger hacia una política óptima que maximiza el valor esperado acumulado en cada estado.

```
    def value_iteration(P, nS, nA, gamma=0.9, tol=1e-3):
        value_function = np.zeros(nS)
        policy = np.zeros(nS, dtype=int)

        while True:
            delta = 0
            for s in range(nS):
                v = value_function[s]
                q_values = [sum(prob * (reward + gamma *
                    value_function[next_state])
                                for prob, next_state, reward, _
                                    in P[s][a])
                            for a in range(nA)]
                value_function[s] = max(q_values)
                delta = max(delta, np.abs(v - value_function[s]))

            if delta < tol:
                break

        # Policy Improvement
        policy = policy_improvement(P, nS, nA, value_function,
            np.zeros(nS, dtype=int), gamma)
        return value_function, policy
```

Listing 4: Función value_iteration