Aprendizaje Automático

GRADO EN INGENIERÍA INFORMÁTICA

Tutorial 4

Curso 2020/2021

Jorge Rodríguez Fraile, 100405951, Grupo 83, 100405951@alumnos.uc3m.es

Carlos Rubio Olivares, 100405834, Grupo 83, 100405834@alumnos.uc3m.es

Índice

Ejercicio 1

Pregunta 1

Hay 12 estados, 11 accesibles, cada uno correspondiente a cada una de las celdas en la que nos podemos colocar. Puede ejecutar 4 acciones, aunque algunas pueden estar restringidas debido a los muros o a la celda gris del tablero.

Si tuviéramos que implementar este ejercicio mediante aprendizaje por refuerzo, lo ideal sería penalizar todos los ejemplos que hagan pasar al punto azul por la casilla del -1. En el caso contrario aplicaremos un refuerzo positivo. Por lo tanto, lo que tendría que establecer es la dirección que tomar en cada estado del agente para maximizar el refuerzo positivo lo más rápido posible.

Pregunta 2

En cuanto a las funciones de esta clase tenemos las siguientes:

- __**init**__: Función por defecto para inicializar la clase.
- **readQtable**: Esta función lee los datos de la tabla Q almacenados en disco. Para esto crea el array de la tabla y recorre todas las líneas que se encuentran en table_file. Posteriormente recoge cada una de las filas mediante split() y las añade al array de la tabla Q con append().
- **writeQtable**: Se encarga de escribir la tabla en disco, para ello lo que hace es ir obteniendo los elementos de la tabla Q por fila y columnas, entonces los escribe de una manera más visual separándolos por espacios.
- **printQtable**: Imprime los elementos de la tabla Q, básicamente se usa un bucle for y se imprime cada línea separada por un salto de línea.
- __**del**__: función destructora de la clase, cada vez que se desea escribir la tabla Q, se cierra el fichero del que proviene.
- **computePosition**: Calcula la posición que ocupa un determinado estado, como si se representa ese un vector unidimensional, dado en forma de tupla (x, y), el cálculo lo hace sabiendo que son 4 acciones de tal manera que calcula cuántas posiciones se avanzan al bajar filas más lo que se avanza en esa columna.
- **getQValue**: Retorna la posición (i, j) de la tabla. Para esto utiliza la función anterior para obtener la fila del valor deseado y usarlo con la columna, que viene dada por las acciones.
- **computeValueFromQValues**: Retorna el valor máximo de las acciones de un estado, para esto vuelve a utilizar la función computePosition para poder obtener la información del estado que deseamos.

- **computeActionFromQValues**: Calcula cuál es la acción óptima dado un estado y una tabla Q, teniendo en cuenta que, si no se puede mover será None, para el resto de los casos crea un vector con un número de posiciones igual al número de acciones legales en las que almacena su valor de Q para ese estado y acción correspondiente. Por último, la acción escogida será la de mayor valor y en caso de que haya más de una con el mismo valor coge una aleatoriamente.

- **getAction**: Da la opción a realizar en un estado; en el caso de no haber acciones legales se utiliza None, y en el caso de haber probabilidad, se utiliza la variable épsilon y se realiza la función flipCoin().

- **Update**: Se utiliza para dar a la tabla Q nuevos valores en base a la función que debemos codificar en este tutorial.

- **getPolicy**: Retorna la mejor acción dado un estado, para esto se utiliza la función computeActionFromQValues

- **getValue**: Retorna el valor más alto de una acción dado un estado, se realiza llamando a la función computeValueFromQValues.

Pregunta 4

Casa casilla aparece dividida en 4 espacios, donde cada uno de ellos representa el valor en la tabla Q de cada una de las acciones. Por otro lado, las casillas de 1 y -1 aparecen con el valor de refuerzo o penalización que aportan. En este caso al no haber creado nuestra fórmula de refuerzo todos los valores están en 0.0.

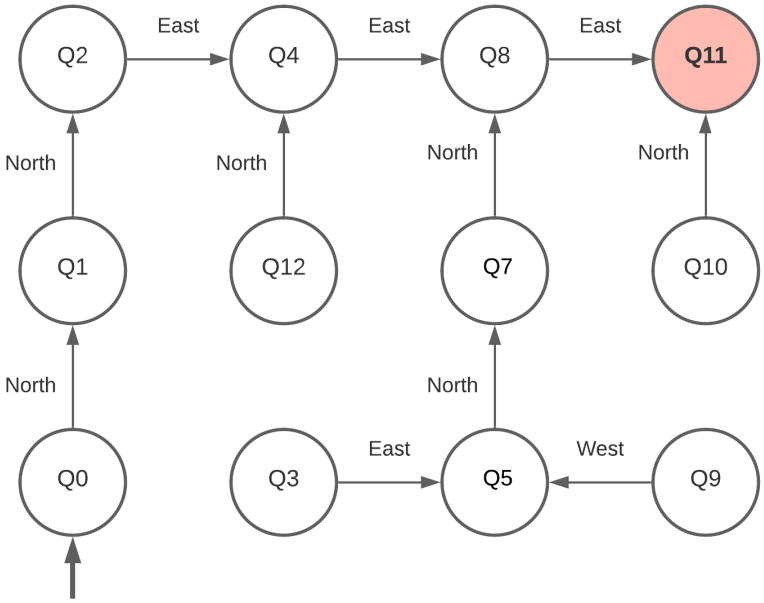
En todo momento por terminal se muestra el estado del agente en forma de dos coordenadas, la acción que ejecuta, el estado al que se transita y el refuerzo de la acción, que se emplea para actualizar la tabla Q que se muestra también. Además, nos indica que posición y columna corresponde actualizar.

Pregunta 5

Este agente se ejecuta de manera aleatoria mediante la variable épsilon mencionada anteriormente, que genera aleatoriamente los movimientos que se realizarán. Este comportamiento viene determinado por la función getAction explicado en la pregunta 2.

Pregunta 6

Este es el MDP determinista que realizaría la política óptima en la que en cada estado tiene la certeza de realizar la acción optima.



Pregunta 7

En principio habría dos posibles políticas óptimas. Una de ellas es la que se muestra en el anterior esquema MDP, y la otra es igual, es decir, el estado Q10, pero yendo por el camino determinado por Q0-Q3-Q5-Q7-Q8-Q11, los estados restantes apuntarían al mismo estado que en la pregunta 6.

Pregunta 9

Presenta el mismo escenario que en la pregunta 4, solo que esta vez, se generan nuevos valores de la tabla Q en cada iteración, cada vez que una acción se utiliza para llegar al estado final en el estado se aumenta la probabilidad de tomarlo mediante el refuerzo. Tras 100 iteraciones en las que llega al estado final para y muestra el valor en la tabla Q de cada una de las acciones de los estados.

Pregunta 10

El fichero qtable.txt contiene la tabla Q, donde se refleja cómo de buena es cada una de las acciones para los distintos estados. A diferencia de cuando no habíamos escrito la función, donde la tabla eran todos ceros, es que esta vez va teniendo en cuenta sus decisiones anteriores para aprender qué acción realizar para obtener refuerzo positivo, por lo que en este caso se toman valores no nulos.

Ejercicio 2

Pregunta 1

Lo que sucede es que cuando tratamos de realizar una acción no ocurre con certeza, sino que tiene un 0.3 de ruido, por lo que no siempre ocurre lo deseado. Esto puede hacer que se tarde más iteraciones de lo debido en obtener el camino correcto, aunque con el suficiente tiempo el agente será capaz de llegar correctamente al estado final.

Pregunta 4

Se acaba generando la política óptima, la que se establece en la pregunta 6 como principal, aunque como hemos explicado anteriormente, tarda más iteraciones que en el caso determinista debido al comando -n 0.3, que hace que las acciones pasen a ser no deterministas.

Implementación de la función update

En la función update que es la que se ejecuta en cada interacción, se encarga de imprimir por terminal la información sobre el estado del aprendizaje y debería actualizar la Tabla Q, por lo que hemos implementado la función de actualización no determinista de los Q valores, que se encontraba descrita en un comentario.

Lo que hemos hecho es actualizar la Tabla Q en la posición correspondiente al estado del agente y la acción que va a ejecutar, el valor que se calcula es el que corresponde con $(1-\alpha)\cdot Q(s,a)+\alpha\cdot\left[r+\gamma\cdot\max_{a'}Q(s',a')\right]$.

La primera parte de la fórmula la obtenemos directamente multiplicando el Q valor antes de actualizarlo por el opuesto de la constante de aprendizaje alfa del agente.

Para la segunda parte, sumamos el refuerzo de ese estado que es un parámetro de update con el producto de la constante de descuento por el valor que se obtiene mediante computeValueQValues del siguiente estado (el máximo para todas las acciones de los Q valores del estado dado), y todo esto multiplicado por la constante de aprendizaje.

Por último, sumamos ambas partes y ya podemos actualizar esa posición de la Tabla Q.

De esta manera hemos podido actualizar la tabla Q en cada iteración, para conseguir que según vaya progresando en el mismo escenario repetidas veces aprenda las acciones que le llevan más directo al refuerzo positivo evitando el negativo.