HW4.A

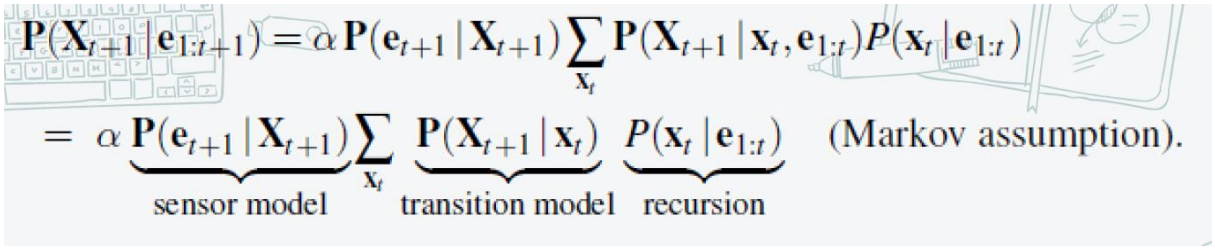
ML | spring 1401 | Dr.Abdi

Student name : **Amin Fathi**

Student id : **400722102**

HMM برای به دست اوردن احتمال حضور عامل در یک خانه با توجه به دو ماتریس وضعیت انتقال و وضعیت مشاهده به کار می رود. همانطورکه از نام آن ها پیداست ماتریس وضعیت انتقال بررسی میکند وضعیت قبلی وفعلی را و ماتریس مشاهده هم برای تقریب احتمال مشاهدات و اساسا تعریف مشاهدات است .

طبق آنچه در اسلاید درسی آمده است:



HMM اساسا می‌گوید که احتمال اینکه در وضعیت xt+1 باشم، به شرط مشاهدات ۱ تا t. که در واقع اگر رابطه ریاضی را باز کنیم ما اساس موقعیت این سنسور قصد تخمین زدن موقعیت فعلی را داریم .

در کد ابتدا کلاس grid را باید در نظر گرفت ، در این کلاس در واقع ما یک محیط m*n را می سازیم که m و n را به عنوان متغیر ورودی به کد می دهیم و با استفاده از خط کد زیر یک مکان تصادفی برای ربات در ابتدا تعیین میکنیم:

```
    self.height = height
    self.robot_location = random.randint(0, width - 1), random.randint(0, height - 1)
```

تابع زیر توسط کلاس sensor برای برگردان یکی از همسایه های خانه بکار میرود و این خانه تصادفی رابه صورت تاپل لوکیشنی خروجی میدهد :

```
def robot_adj_location(self):
    """
    Called by sensor to return an adjacent location.
    :return: Tuple containing a random adjacent location to the robot, or None if that particular location
             is out of bounds.
    """
    x, y = self.robot_location

    l_s = [(x - 1, y - 1), (x - 1, y), (x - 1, y + 1), (x, y - 1), (x, y + 1), (x + 1, y - 1), (x + 1, y),
           (x + 1, y + 1)]

    adj = l_s[random.randint(0, 7)]

    # wall check
    if adj[0] >= self.width or adj[1] >= self.height or adj[0] < 0 or adj[1] < 0:
        return None
    else:
        return adj
```

تابع زیر هم یک همسایگی در شعاع ۲ را همانند تابع بالایی برمیگرداند به صورت تصادفی :

```
def robot_adj2_location(self):
    """
    Called by sensor to return an adjacent location 2 spaces away.
    :return: Tuple containing a random adjacent location 2 away from the robot, or None if that particular location
             is out of bounds.
    """
    x, y = self.robot_location

    ls_2 = [(x - 2, y - 2), (x - 2, y - 1), (x - 2, y), (x - 2, y + 1), (x - 2, y + 2), (x - 1, y - 2),
            (x - 1, y + 2), (x, y - 2), (x, y + 2), (x + 1, y - 2), (x + 1, y + 2), (x + 2, y - 2), (x + 2, y - 1),
            (x + 2, y), (x + 2, y + 1), (x + 2, y + 2)]

    adj = ls_2[random.randint(0, 15)]

    # wall check
    if adj[0] >= self.width or adj[1] >= self.height or adj[0] < 0 or adj[1] < 0:
        return None
    else:
        return adj
```

تابع زیر برای آنکه مشخص کند که ایا ربات به دیوار خورده است یا خیر به کار می رود که در صورت برخورد True و در غیر این صورت False:

```
def robot_faces_wall(self):
    """
    Called by sensor to determine is robot faces wall.
    :return: Boolean whether or not robot is facing a wall.
    """
    x, y = self.robot_location

    # NORTH, EAST, SOUTH, WEST
    next_locations = [(x, y + 1), (x + 1, y), (x, y - 1), (x - 1, y)]
    next_coord = next_locations[self.robot_dir]
    if next_coord[0] >= self.width or next_coord[1] >= self.height or next_coord[0] < 0 or next_coord[1] < 0:
        return True
    else:
        return False
```

در تابع زیر هم حرکت ربات ها را مدیریت میکنیم که در واقع با احتمال ۰.۳ تغییر جهت می دهند و حواسشان هست که روبه رویاشن هم دیوار نباشد و سپس به یکی از همسایه ها در شمال جنوب یا شرق و غرب رفته اکشن بعدی و ...

```python
def move_robot(self):
    """
    Move robot according to strategy.
    """
    # 30% Chance to change direction.
    rand = random.random()
    if rand <= 0.3:
        self.robot_dir = Direction.random(self.robot_dir)
    # Changes direction until robot doesn't face wall.
    while self.robot_faces_wall():
        self.robot_dir = Direction.random(self.robot_dir)

    x, y = self.robot_location

    # Moves forward in robot's direction.
    # NORTH, EAST, SOUTH, WEST
    next_locations = [(x, y + 1), (x + 1, y), (x, y - 1), (x - 1, y)]
    self.robot_location = next_locations[self.robot_dir]
```

در کلاس DIRECTION به جهت های جغرافیایی اندیس می دهیم و در لیست dirs ذخیره میکنیم تا بتوانیم از آن استفاده کنیم در ادامه پروژه:

```python
class Direction:
    NORTH, EAST, SOUTH, WEST = range(4)
    DIRS = [NORTH, EAST, SOUTH, WEST]

    def __init__(self):
        pass

    @classmethod
    def random(cls, exempt_dir=None):
        dirs = [cls.NORTH, cls.EAST, cls.SOUTH, cls.WEST]
        if exempt_dir:
            dirs.remove(exempt_dir)
            return dirs[random.randint(0, 2)]
        else:
            return dirs[random.randint(0, 3)]
```

در کلاس Sensor س یک تابع به نام sense_location وجود دارد که در از توابع کلاس Grid استفاده می‌کند برای تشخصی مختصات فعلی . احتمالات ذکر شده در صورت سوال برای تشخیص مکان صحیح توسط سنسور در صورت سوال را هم پیاده کرده و ابتدا یک عدد به‌صورت تصادفی تولید شده و با احتمالات بیان شده مقایسه می‌شود و با توجه به دسته‌بندی که در آن قرار می‌گیرد، تابع مناسب فراخوانی می‌شود:

```
class Sensor:
    # Settings for robot sensor.
    def __init__(self, grid):
        self.grid = grid

    L = 0.1
    L_s = 0.05
    L_s2 = 0.025

    def sense_location(self):
        """
        Uses sensor to detect current coordinates.
        :return: Tuple of coordinates based on sensing probabilities.
        """
        rand = random.random()
        if rand <= self.L:
            return self.grid.robot_location
        elif rand <= self.L + self.L_s * 8:
            return self.grid.robot_adj_location()
        elif rand <= self.L + self.L_s * 8 + self.L_s2 * 16:
            return self.grid.robot_adj2_location()
        else:
            return None
```

در کلاس HMM و در تابع CREATE_PRIORS احتمال حضور ربات در هر خانه را نشان می دهد

```
def create_priors(self):
    length = self.width * self.height * 4
    priors = [float(1) / length] * length
    return np.array(priors)
```

تابع CREATE_T_MATRIX همانطور که از نامش پیداست برای ساختن ماتریس انتقال به کار می رود . که یک ماتریس با ابعاد (width * height * 4, width * height * 4) برای آن درنظر می‌گیرد. سپس بر روی هر یک از حالت‌ها یک حلقه می‌زند و احتمال قرار گیری ربات در هر خانه را به دست آورد. برای این کار از تابع probable_transitions استفاده می‌کند و آن را فراخوانی می‌کند. این تابع نیز احتمال‌ها را براساس آنچه برایش تعریف شده است، به دست می‌آورد :

```
def create_t_matrix(self):
    width = self.width
    height = self.height
    t = np.array(np.zeros(shape=(width * height * 4, width * height * 4)))

    for i in range(width * height * 4):
        x = int(i // (height * 4))
        y = int((i // 4) % height)
        heading = i % 4
        prev_states = self.probable_transitions((x, y, heading))
        for (xcoord, ycoord, direction), probability in prev_states:
            t[i, int(xcoord * height * 4 + ycoord * 4 + direction)] = probability
    return t
```

در انتها هم تابع start_robot را فراخوانی کرده و و محیط و عامل و ... را پیاده کرده با توجه به کتابخانه هایی که گفتیم .

```python
def start_robot(width, height):
    """
    Sets up the world and robot and enters endless loop of guessing.
    :param size:
    """
    # create grid that contains world state
    grid = Grid(width, height)
    # create sensor, which queries the grid
    sensor = Sensor(grid)
    # create robot, which guesses location based on sensor
    hmm = HMM(width, height)
    robot = Robot(sensor, hmm)

    moves = 0
    guessed_right = 0
    while True:
        # move robot
        grid.move_robot()
        moves += 1
        print("\nRobot is in: ", grid.robot_location)
        guessed_move, probability = robot.guess_move()
        if guessed_move == grid.robot_location:
            guessed_right += 1
        man_distance = abs(guessed_move[0] - grid.robot_location[0]) + abs(guessed_move[1] - grid.robot_location[1])
```

```python
        guessed_move, probability = robot.guess_move()
        if guessed_move == grid.robot_location:
            guessed_right += 1
        man_distance = abs(guessed_move[0] - grid.robot_location[0]) + abs(guessed_move[1] - grid.robot_location[1])
        print("Manhattan distance: ", man_distance)
        print("Robot has been correct:", float(guessed_right) / moves, "of the time.")
        sleep(1)


if __name__ == '__main__':
    """
    The main function called when main.py is run from the command line.

    Example usage:
    > python main.py --width 10 --height 10
    First argument defines the width of the grid, second, the height.
    """
    # handle arguments
    # import argparse
    #
    # parser = argparse.ArgumentParser()
    # parser.add_argument("--width", type=int, required=True)
    # parser.add_argument("--height", type=int, required=True)
    # args = parser.parse_args()
    # if args.width <= 1 or args.height <= 1:
    #     raise argparse.ArgumentTypeError("Values must be greater than one.")
    #
    start_robot(10, 10)
```

که البته ۲ وردودی هم میگیرد و عرض و ارتفاع محیط است .

منابع :

GitHub - nickbirnberg/HMM-localisation: AI Project

Tutorial- Robot localization using Hidden Markov Models (dtransposed.github.io)