Machine, Date & Learning

Linear-Programming

Demonstration of linear-programming for solving MDPs.

Team 25 | Tribrid

- Nitin Chandak (2019101024)
- Ayush Sharma (2019101004)

Making `A` matrix

- We created all possible state in the form of 5-tuple i.e. `(pos,mat,arrow,state,health)` and store in `list` as `all_state`. In total `600` states. Their domains are as follows:

```
 POS = [ "W","N","E","S","C" ]    # position of IJ
MAT = [ 0,1,2 ]    # material with IJ
ARROW = [ 0,1,2,3 ]    # number of arrows
STATE = [ "D","R" ]    # ready and dormant state of MM
HEALTH = [ 0,25,50,75,100 ]    # MM’s health
```

- Then we calculated possbile actions of all states and store in `dict` as `state_actions`.
- Created `get_pfsb()` function that will take `(state, action)` as parameters & return possible final state list along with it's probability distribution i.e. list of 6-tuple `(pos,mat,arrow,state,health, prob)`.
- Generated map as `r_state_map` to allocate unique ID to every state.
- Next, we made a 2-D matrix (named it `A`, it is not our final required `A` matrix) with dimension `len(all_state)*len(all_state)`, where each cell have data as a list with entry 0's of length `len(state_actions[all_state[r]])`, where `r` is a row variable ranging from `(0 to len(all_state))`.
- Say, if an action(action index = ai) can take you from state A(rownum = j) to B(rownum = i) with probability P. Then we will do the follwing:

```
 A[j][j][ai] += P
B[i][j][ai] -= P
```

- Basically, add the probability to the value at index of state A and subtract from the value at index of state B in the same column of action(action index = ai).
- Next, to create final `A` matrix (we named it as `LP_A` in our code), by opening the list at each cell of our 2-D `A` matrix.

Finding Policy & Result Analysis

- Before, finding the policy, we obtained utility of `State-Action` pair i.e values `X`. Values `X` is basically the expected number of time an action can be opted in a given state.
- Values `X` has been calculated via linear-programming with a linear objective and inequality constraints. LPP for obtaining `X` is shown:

```
maximize RX | with constraint AX  = alpha, X >= 0
```

- `R` in above LPP is array of rewards for all actions valid in each and evry state.
- `A` is matrix described as in above section (matrix which we named `LP_A`). It represents flow of probability of valid actions for each and every state.
- `alpha` holds initial probability of states. It is list of zeros with value `1` at index corressponding to state `('C',2,3,'R',100)`.
- To determine `policy` for each state, we pick the `action` with highest corressponding value in `X` by iterating to all valid `X` values for each and every state. Following is the code for the same:

```
optimal_policy = {}
cum_i = 0
for i in range(len(all_state)):
    s = all_state[i]
    max_reward = -1000000
    for j in range(len(state_actions[s])):
        a = state_actions[s][j]
        if max_reward <= R[0][cum_i+j]:
            max_reward = R[0][cum_i+j]
            optimal_policy[s] = a
    cum_i += len(state_actions[s])

policy = []
for s in optimal_policy:
    (pos,mat,arrow,state,health) = s
    sublist = []
    state = [pos[0],mat,arrow,state,health]
    action = optimal_policy[s]
    sublist.append(state)
    sublist.append(action)
    policy.append(sublist)
```

Multiple Policies

Can there be multiple policies? Why?

Yes, there can be multiple policies. Since, the `(state, actions)` with the same coressponding value in `X` are interchangeable. Hence, there can be multiple policies.

What changes can you make in your code to generate another policy?

We can have following two changes that can generate another policy:-

1. In our code we have modification in the condition for finding action having max value of 'X'.The former finds the last highest value of x in case many x's have the same largest value. The second one finds the first x with the highest value. Code for both are as follows:-

```
if max_reward <= R[0][cum_i+j]:
    max_reward = R[0][cum_i+j]
    optimal_policy[s] = a
```

```
if max_reward < R[0][cum_i+j]:
    max_reward = R[0][cum_i+j]
    optimal_policy[s] = a
```

- This change won't be affecting `A` matrix (which we name `LP_A` in our code), as it has been formed before performing LP.
- Also, `alpha` and `R` vector will remain unaffected for the same reason.

2. Another way to change the generated `policy` is by changing the order of actions stored in `state_actions` for each state. For example changing `[ 'UP','LEFT','DOWN','RIGHT','STAY' ]` to `[ 'UP','LEFT','STAY','RIGHT','DOWN' ]` will change the last possible highest reward giving action. Say if in current setup all actions have same reward for a state then policy will `STAY` in first case, but after changing the order the policy will be `DOWN`.

- It will surely affect the `A` matrix (which we name `LP_A` in our code), as it has been formed before performing LP, but depends on the order the action has been stored for each states.
- Same is the case for `R` vector as it is with `A` matrix (which we name `LP_A` in our code) for the same reason.
- As, `alpha` vector holds the initial status i.e. probability of the states, which won't get updated during algorithm. Hence, no affect on `alpha` vector.

3. Other way to chane the `policy` is by changing starting variables like changing the start from `('C',2,3,'R',100)` to `('N',0,3,'R',75)` or any other state.

- It will only affect the `alpha` vector as initial probability distribution gets modified. `A` & `R` will remain unaffected.

4. We can change the `STEPCOST` value.

- Only `R` will get affected as it represents the expected reward for each action valid for each and every state, which will get changed due to change in `STEPCOST`. `alpha` & `A` will remain unaffected.

NOTE: Running command `python3 part_3.py` will generate an output file in `outputs/part_3_output.json`. It will contain data of `A` matrix, `R` matrix, `Alpha` matrix, `X` values, `Policy` and `Objective` in following format:

```
{
    "a": [
        [
            1.0,
            0.0,
            0.0,
            0.0
        ],
        [
            0.0,
            1.0,
            0.0,
            0.0
        ],
        [
            0.0,
            0.0,
            1.0,
            0.0
        ]
    ],
    "r": [
        1.0,
        1.0,
        1.0,
        1.0
    ],
    "alpha": [
        0,
        0,
        0,
        1
    ],
    "x": [
        1.724,
        1.784,
        0.645,
        1.854
    ],
    "policy": [
        [
            [W,0,0,D,0], "RIGHT"
        ],
        [
            [W,0,0,D,25], "RIGHT"
        ],
            ...
        [
            [C,2,3,R,75], "HIT"
        ],
        [
            [C,2,3,R,100], "SHOOT"
        ]
    ],
    "objective": -11.54321
}
```