hw6_report

December 11, 2022

1 MAS Final Homework Assignment

Daniel Engbert, Dec 2022

[ ]:
```python
# helper function for use in code code below
import random
import numpy as np

def set_seed(seed=42):
  rng = random.Random()
  if seed is None:
    seed = random.randint(0, 9999)
  rng.seed(seed)
  np.random.seed(seed)
```

1.1 1. Monte Carlo Estimation of Shapely Value

How to fairly split a taxi fare across $\{1, ...N\}$ players can be calculated using Shapely values, which treats this scenario as a cooperative game. For this problem we assume that all players live on the way of the Nth player, and that player A lives at distance of 1, player B at a distance of 2, etc.

1.1.1 Computing Shapely values for $n = 4$:

The code below computes all permutations (a.k.a. "coalitions") of the $N = 4$ players $\{A, B, C, D\}$, then for each permutation, `perm`, it assigns each player a share off the total payoff (the fare of total cost 4), by assuming player `perm[0]` showed up first at the taxi, followed by players `perm[1]`, `perm[2]`, `perm[3]`.

For example, for the permutation `perm = ['A', 'C', 'B', 'D']`, the respective payoffs (fare splits) will be `{'A': 1, 'B': 0, 'C': 2, 'D': 1}` (here player B pays 0 because by the time player he arrives at the taxi, it was already going to go by the his house anyways due to player C.)

The final computed Shapely values are simply the average payoffs of each player across all possible permutations.

[ ]:
```python
def get_perms(arr):
    """returns a list of the possible permutations of the entries in the␣
↪provided array."""
    all_perms = []
```

```python
    for p in arr:
        other_elems = sorted(list(set(arr) - set([p])))
        sub_perms = get_perms(other_elems)
        if len(sub_perms) == 0:
            all_perms.append([p])
        else:
            all_perms = all_perms + [[p] + perm for perm in sub_perms]
    return all_perms

def get_shapely(N):
    player_vals = {chr(ord('A') + n): n+1 for n in range(0, N)}
    #player_vals = {'A': 6, 'B': 12, 'C': 42} # should result in shapely values␣
 ↪{2, 5, 35}
    print("player_vals = ")
    print(player_vals)
    players = list(player_vals.keys())

    # list of permutations of coalitions of size len(players)
    perms = get_perms(players)
    print(f"there are {len(perms)} total permutations of {len(players)} players:
 ↪")
    display(perms[:5]) # print first few rows
    print('(only first 5 rows of permutations are shown above)')

    running_payoffs = {p: 0 for p in players}
    total_payoff = max(player_vals.values())
    for perm in perms:
        cur = {p: 0 for p in players}
        for p in perm: # compute share of payoff for each player in this␣
 ↪permutation
            cur[p] = max(0, player_vals[p] - sum(cur.values()))
        running_payoffs = {k: v+cur[k] for (k,v) in running_payoffs.items()}

    shapely_values = {k: v/len(perms) for (k,v) in running_payoffs.items()}
    print(f"\nshapely_values: (for N = {N})")
    print(shapely_values)

    #print('percent of payoff:')
    #print({k: v/total_payoff for (k,v) in shapely_values.items()})

get_shapely(4)
```

```
player_vals =
{'A': 1, 'B': 2, 'C': 3, 'D': 4}
there are 24 total permutations of 4 players:
```

```
[['A', 'B', 'C', 'D'],
 ['A', 'B', 'D', 'C'],
```

```
 ['A', 'C', 'B', 'D'],
 ['A', 'C', 'D', 'B'],
 ['A', 'D', 'B', 'C']]
```

```
(only first 5 rows of permutations are shown above)

shapely_values: (for N = 4)
{'A': 0.25, 'B': 0.5833333333333334, 'C': 1.0833333333333333, 'D':
2.0833333333333335}
```

As can be seen from the output, the shapely values for $n = 4$ players is `{'A': 0.25, 'B': 0.583, 'C': 1.083, 'D': 2.083}`

1.1.2 Estimating Shapely values for $n = 100$

Here we use Monte Carlo sampling to approximiate the Shapely values for the taxi fare problem when $n = 100$.

[ ]:
```
# TODO
```

1.2 2. Monte Carlo Tree Search (MCTS)

[ ]:
```
import networkx as nx
tree = nx.DiGraph()

nx.draw(tree, with_labels=True, node_size=2000)
set_seed()

def edit_distance(add1: str, add2: str) -> int:
    assert len(add1) == len(add2)
    diff = 0
    return sum([int(a1 != a2) for a1, a2 in zip(add1, add2)])

edit_distance("RRLL", "RLRR")
```

[ ]:
```
3
```

1.3 RL: SARSA and Q-Learning

[ ]:
```
import numpy as np

EMPTY = 0
WALL = 1
SNAKES = 2
TREASURE = 3

def create_world():
  world = np.zeros((9,9), dtype=int)
  world[1, 2:7] = WALL
  world[1:5, 6] = WALL
  world[7, 1:5] = WALL
  world[6:5] = SNAKES
  world[-1, -1] = TREASURE
  return world

world = create_world()
```

```
display(world)
```

```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 3]])
```

```
[ ]:
```