sheet06-programming

June 3, 2024

Exercises for the course Artifical Intelligence Summer Semester 2024

G. Montavon Institute of Computer Science Department of Mathematics and Computer Science Freie Universität Berlin

Exercise Sheet 6 (programming part)

[1]:
```
import numpy, utils
```

0.1 Exercise 3: Inferences in a Bayesian Decision Network (60 P)

In this exercise, we will consider a Bayesian decision network that models the decision making related to going picnicking, and the utility of the various possible actions. The Bayesian decision network is shown below.

Like for the previous programming exercise, the variables indices start at zero. Also, we use different indices for actions and variables because they will occupy distinct dimensions in the tensor representation. Specifically, an action can be seen as a variable that the agent can set to any desired value. The utility nodes do not require indices as they are simple measurements and not variables. We also assume that the agent's overall utility is to the sum of the two utility nodes.

The factors (matrices of conditional probability values and utility scores) associated to the network above can be found in the module `utils`. Like in the previous programming exercise, they are already casted into 5-dimensional tensors in order to ease the subsequent computations.

[2]:
```
P02  = utils.P02   # P(X2|A0)
P1   = utils.P1    # P(X1)
P13  = utils.P13   # P(X3|X1)
```

[3]:
```
U0   = utils.U0    # U(A0)
U234 = utils.U234  # U(X2,X3,A4)
```

For convenience, we also provide a textual description for the values of some variables and actions of interest.

[4]:
```
N0 = utils.N0; print('A0: %s'%N0)
N1 = utils.N1; print('X1: %s'%N1)
N4 = utils.N4; print('A4: %s'%N4)
```

```
A0: ['no shelter', 'rent shelter']
X1: ['dry forecast', 'rain forecast']
A4: ['stay at home', 'go picnicking']
```

0.1.1 (a) Utility of different actions (20 P)

We would like to perform inferences in this Bayesian decision network. First, we will compute the overall expected utility for the agent if no variable X_1, X_2 and X_3 are observed, and for each possible pair of actions the agent chooses. The function below performs the corresponding calculations. the procedure can be decomposed in three steps: (1) The joint probability distribution is computed, (2) the utility of each scenario is weighted by the probability distribution and summed to arrive at the expected utility, and (3) one selects the utility associated to the actions the agent has provided as arguments.

[5]:
```python
def get_expected_utility(rent_shelter,go_picnicking):

    # build the joint probability distribution
    P = P02 * P1 * P13

    # compute expected utility
    U = (P * (U0 + U234)).sum((1,2,3),keepdims=True)

    # choose actions
    U = U.take([rent_shelter],0).take([go_picnicking],4)

    return U
```

We would now like to compute the expected utility of each possible pair of actions. Although not especially efficient, this can be achieved by calling function above multiple times with the appropriate arguments.

[6]:
```python
def print_expected_utility(get_expected_utility):
    actions = [(0, 0), (0, 1), (1, 0), (1, 1)]
    utilities = {}
    for rent_shelter, go_picnicking in actions:
        utility = get_expected_utility(rent_shelter, go_picnicking)
        utilities[(rent_shelter, go_picnicking)] = utility[0, 0, 0, 0, 0]

    for (rent_shelter, go_picnicking), utility in utilities.items():
        print(f"{N0[rent_shelter]}, {N4[go_picnicking]} -> {utility:.2f}")

print_expected_utility(get_expected_utility)
```

```
no shelter, stay at home -> 0.00
no shelter, go picnicking -> 9.74
rent shelter, stay at home -> -12.00
rent shelter, go picnicking -> 10.52
```

These calculations indicate that the action that gives the highest utility is to rent a shelter and go

picnicking. Close second is the action of going picnicking without renting shelter.

0.1.2 (b) Decision Making Conditioned on Rain Forecast (20 P)

We now assume that we have access to the rain forecast, i.e. we have evidence for the variable X_1. We would like to infer what is the best possible action (and the associated expected utility) for the case where the forecast indicates no rain i.e. dry weather ($X_1 = 0$), and for the case where the forecast indicates rain ($X_1 = 1$). The function below receives a forecast as input and should return a triplet consisting of the best action pair and the corresponding expected utility.

[7]:
```
def get_expected_utility_given_forecast(rent_shelter, go_picnicking, X1):
    # building the joint probability distribution conditioned on X1
    P = P02 * P13.take([X1], 1)

    # computing the expected utility
    U = (P * (U0 + U234)).sum((1, 2, 3), keepdims=True)

    # choosing actions
    U = U.take([rent_shelter], 0).take([go_picnicking], 4)

    return U


def get_best_action_and_expected_utility(X1):
    actions = [(0, 0), (0, 1), (1, 0), (1, 1)]
    best_action = None
    best_utility = float('-inf')

    for rent_shelter, go_picnicking in actions:
        utility = get_expected_utility_given_forecast(rent_shelter,␣
 ↪go_picnicking, X1)
        utility_value = utility[0, 0, 0, 0, 0]
        if utility_value > best_utility:
            best_utility = utility_value
            best_action = (rent_shelter, go_picnicking)

    A0, A4 = best_action
    U = best_utility
    return A0, A4, U
```

The following code tests your implementation. It prints for the two possible forecasts the action pair and expected utility returned by the function above.

[8]:
```
for X1 in [0,1]:

    A0,A4,U = get_best_action_and_expected_utility(X1)

    print('%s -> %s, %s -> %.2f'%(N1[X1],N0[A0],N4[A4],U))
```

```
dry forecast -> no shelter, go picnicking -> 24.04
rain forecast -> no shelter, stay at home -> 0.00
```

We can observe that the best actions depend on the forecast, and the pairs of actions are also different from the optimal one in absence of evidence (cf. above). In particular, renting a shelter is no longer recommended.

0.1.3 (c) Expected Utility of a Decision Policy (20 P)

Part (b) of this exercise can be interpreted as having built a decision policy. We would like to hardcode that policy into an extended Bayesian network for the purpose of evaluating the policy's performance. This can be achieved by transforming action nodes into actual variables. In particular, inspection of the optimal decision policy above, we find that the action A_0 can also seen as a variable X_0 always set to 0, and the action node A_4 can be seen as a variable X_4 that should be set to 1 when $X_1 = 0$, and to 0 when $X_1 = 1$. The extended Bayesian network is depicted below.

Build a function that performs inferences in this extended Bayesian network. In particular, construct the missing matrices of probabilities `P0` and `P14`, and then compute the expected utility when no evidence is provided for any of the variables.

[9]:
```
def get_expected_utility_new():
    P0 = numpy.array([1, 0])[:, numpy.newaxis, numpy.newaxis, numpy.newaxis,␣
 ↪numpy.newaxis]

    P14 = numpy.array([[0, 1], [1, 0]])

    P14 = P14[numpy.newaxis, :, numpy.newaxis, numpy.newaxis, :]

    # joint probability distribution
    P = P02 * P1 * P13 * P0 * P14

    # expected utility
    U = (P * (U0 + U234)).sum((0, 1, 2, 3, 4), keepdims=True)

    return U.item()
```

We now call the function we have just implemented to compute the expected utility of our forecast-informed decision policy.

[10]:
```
print('expected utility %.2f'%get_expected_utility_new())
```

```
expected utility 16.83
```

We observe that the expected utility is higher than any of the static policies that have been evaluated in part (a). Hence, looking at the evidence from the weather forecast allows for more targeted actions that improve the overall agent's utility.