TDT4171 Methods in AI

Assignment 2

Written Spring 2022 by Ø. Solbø

Task 1: Time and Uncertainty

The umbrella world is a model describing the probability of rain, given the observation of umbrella on the different days. This means that the measurement (observable variable) is the detection of an umbrella at a given time-step, while rain being the unobservable state in the system.

Using same notation for the models as used in TTK4250. The system (transition) model and the measurement model is respectively given as

$$\text{System (transition) model: } \mathbf{X_{k+1}} = \mathbf{F}\mathbf{X_k}$$
$$\text{Measurement (sensor) model: } \mathbf{Z_k} = \mathbf{H}\mathbf{X_k}$$

where

$$\mathbf{F} = \begin{bmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{bmatrix},$$
$$\mathbf{H}|(\mathbf{X_k} = \mathbf{true}) = \begin{bmatrix} 0.9 & 0 \\ 0 & 0.2 \end{bmatrix},$$
$$\mathbf{H}|(\mathbf{X_k} = \mathbf{false}) = \mathbf{I_{2x2}} - \mathbf{H}|(\mathbf{X_k} = \mathbf{true}) = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.8 \end{bmatrix}.$$

The choice of the measurement-matrix $\mathbf{H}$ (denoted $\mathbf{O}$ in [1]), depends on the measurement $\mathbf{Z_k}$ (denoted $\mathbf{E_t}$ in [1]) for a given day. If an umbrella is observed in the measurement/evidence, $\mathbf{H}|(\mathbf{X_k} = \mathbf{true})$ must be used. If no umbrella is detected, $\mathbf{H}|(\mathbf{X_k} = \mathbf{false})$ is used. See the implementation in appendix A for details.

As the guard could only estimate the state (raining or not) by observation of the directors umbrella, it is assumed that the guard has no other method in obtaining information regarding the weather. This implies that the guard does not look up weather-reports, does not hear talk about the weather, does not deduce the likelihood of weather based on the clothing, cannot step or look outside at all and is stuck inside at all times, ++.

The derivation of the equations assumes that a first order Markov process is valid for the transition model, while a sensor Markov assumption is valid for the measurement model.

During the derivation of the models, it is assumed that the system can be reduced into linear models with known matrices. In reality, both the models and values must be estimated, and can fluctuate depending on other hidden variables. For example, the season will affect the transition model, $\mathbf{F}$. Such effects will in reality require nonlinear and adaptive models to accurately depict the probability of rain.

Task 2: Inference in Temporal Models - Filtering

The forward algorithm implemented in appendix B, is developed using equation (14.12) in [1, p. 492]. The system matrices, measurements and prior information is defined in appendix A.

The normalized forward-calculated messages are given in table 1. One can see that $\mathbf{P}(\mathbf{X}_2|\mathbf{Z}_{1:2}) = 0.88335704 \approx 0.883$, which is the expected value according to the task.

Using the normalized forward messages in table 1, one gets that $\mathbf{P}(\mathbf{X}_5|\mathbf{Z}_{1:5}) = 0.86733889$.

$f_{1:k}$	True	False
$f_{1:1}$	0.81818182	0.18181818
$f_{1:2}$	0.88335704	0.11664296
$f_{1:3}$	0.19066794	0.80933206
$f_{1:4}$	0.730794	0.269206
$f_{1:5}$	0.86733889	0.13266111

Table 1: Normalized forward-messages for task 2.

Task 3: Inference in Temporal Models - Smoothing

The backward algorithm implemented in appendix C, is developed using equation (14.13) in [1, p. 492]. The system matrices, measurements and prior information is defined in appendix A.

The forward-backward algorithm is implemented as shown in appendix D, with the system matrices, measurements and prior information defined in appendix A. There are two tables showing the implementation results, namely table 2 and table 3. The former table shows the results when just using the two first measurements. Based on table 2, one can see that $\mathbf{P}(\mathbf{X_1}|\mathbf{Z_{1:2}}) = [0.8673389, 0.11664296]^T$. The latter table, shows the forward-backward results when using the entire measurement space. From table 3, one can see that $\mathbf{P}(\mathbf{X_1}|\mathbf{Z_{1:5}}) = 0.8673389$.

$fb_{k\|1:2}$	True	False
$fb_{1\|1:2}$	0.88335704	0.11664296
$fb_{2\|1:2}$	0.88335704	0.11664296

Table 2: Normalized messages from the forward-backward algorithm developed for task 3. This table only includes the first couple of values, and is only used for testing.

$fb_{k\|1:5}$	True	False
$fb_{1\|1:5}$	0.86733889	0.13266111
$fb_{2\|1:5}$	0.82041905	0.17958095
$fb_{3\|1:5}$	0.30748358	0.69251642
$fb_{4\|1:5}$	0.82041905	0.17958095
$fb_{5\|1:5}$	0.86733889	0.13266111

Table 3: Normalized messages from the forward-backward algorithm developed for task 3 for $k \in [1, 5]$.

The not-normalized and normalized backward-messages are given in tables 4 and 5 respectively.

$b_{k+1:5}$	True	False
$b_{2:5}$	0.06611763	0.04550767
$b_{3:5}$	0.090639	0.150251
$b_{4:5}$	0.4593	0.2437
$b_{5:5}$	0.69	0.41
$b_{6:5}$	1.0	1.0

Table 4: Backward-messages developed for task 3 for $k \in [1, 5]$. Note that these are NOT normalized.

$b_{k+1:5}$	True	False
$b_{2:5}$	0.5923176	0.4076824
$b_{3:5}$	0.37626718	0.62373282
$b_{4:5}$	0.65334282	0.34665718
$b_{5:5}$	0.62727273	0.37272727
$b_{6:5}$	1.0	1.0

Table 5: Normalized backward-messages developed for task 3 for $k \in [1, 5]$.. These are normalized versions of the values in table 4.

References

[1] S. Norvig, P. Russel, *Artificial Intelligence - A modern approach.* Pearson, 2022.

A Parameters

```
from dataclasses import dataclass
import numpy as np
import matplotlib.pyplot as plt

from numpy import ndarray

@dataclass
class UmbrellaWorldParameters:
  # Using the models:
  # x_{k+1} = Fx_{k}
  # z_{k} = Hx_{k}
  F = np.array(
    [
      [0.7, 0.3],
      [0.3, 0.7]
    ])
  H_given_rain = np.array(
    [
      [0.9, 0],
      [0,   0.2]
    ])
  H_given_no_rain = np.eye(2) - H_given_rain
  H = np.array([H_given_no_rain, H_given_rain])

  # Initial value
  x0 = np.array([[0.5], [0.5]])

  # Observations
  z = np.array([1, 1, 0, 1, 1])
```

B Forward algorithm

```
import numpy as np
import matplotlib.pyplot as plt

from numpy import ndarray
from parameters import UmbrellaWorldParameters as uw_param

def forward(curr_idx: 'int', min_idx=0)->ndarray:
  if curr_idx >= uw_param.z.shape[0] or curr_idx < min_idx:
    return np.zeros_like(uw_param.x0)

  O = uw_param.H[uw_param.z[curr_idx]]
  OFT = O @ uw_param.F.T

  if curr_idx == min_idx:
    prod = OFT @ uw_param.x0
  else:
    prod = OFT @ forward(curr_idx - 1)
  return prod / prod.sum(axis=0)

if __name__ == '__main__':
  # Verification using the first two values
  xf = np.zeros((2, 2))
  for idx, _ in enumerate(xf):
    xf[idx,:] = forward(curr_idx=idx, min_idx=0).T
  print(xf)

  # Full measurement space
  xf = np.zeros((uw_param.z.shape[0], 2))
  for idx, _ in enumerate(xf):
    xf[idx,:] = forward(curr_idx=idx, min_idx=0).T
  print(xf)
```

C Backward algorithm

```
import numpy as np
import matplotlib.pyplot as plt

from numpy import ndarray
from parameters import UmbrellaWorldParameters as uw_param


def backward(curr_idx: 'int', max_idx: 'int')->ndarray:
  if curr_idx > max_idx or curr_idx < 0 or max_idx > uw_param.z.shape[0]:
    return np.ones_like(uw_param.x0)

  O = uw_param.H[uw_param.z[curr_idx]]
  FO = uw_param.F @ O

  if curr_idx == max_idx:
    return FO @ np.ones_like(uw_param.x0)
  return FO @ backward(curr_idx + 1, max_idx)

if __name__ == '__main__':
  xb = np.ones((uw_param.z.shape[0], 2))
  for idx, _ in enumerate(xb):
    xb[idx,:] = backward(idx, uw_param.z.shape[0] - 1).T
  print(xb)
```

D Forward-backward algorithm

```
import numpy as np
import matplotlib.pyplot as plt

from numpy import ndarray
from parameters import UmbrellaWorldParameters as uw_param
from forward import forward
from backward import backward

def forward_backward(min_idx: 'int', max_idx: 'int')->'ndarray':
  if min_idx > max_idx or min_idx < 0 or max_idx >= uw_param.z.shape[0]:
    return np.empty((uw_param.z.shape[0], 2))

  # Initiating memory
  xf = np.zeros((max_idx+1, 2))
  for i in range(uw_param.x0.shape[1]):
    xf[0,i] = uw_param.x0[i]

  xs = np.zeros_like(xf)
  xb = np.ones_like(xf)

  for idx, _ in enumerate(xf):
    # Calculating forward and backward pass
    xf[idx,:] = forward(curr_idx=idx, min_idx=min_idx).T
    xb[idx,:] = backward(curr_idx=idx+1, max_idx=max_idx).T

    xf_xb = np.array([xf[idx, i] * xb[idx, i] for i in range(xf.shape[1])]).T
    xs[idx] = xf_xb / xf_xb.sum(axis=0)

  return xs

if __name__ == '__main__':
  # Initial testing
  xs = forward_backward(min_idx=0, max_idx=1)
  print(xs)

  # Full probability table
  xs = forward_backward(min_idx=0, max_idx=uw_param.z.shape[0]-1)
  print(xs)
```