B(E)3M33UI—Exercise PM03: Hidden Markov Models II.

Petr Pošík

May 4, 2018

In this exercise we shall continue with the development of the HMM inference algorithms from the previous exercise. The goal of this exercise is

- to implement smoothing via forward-backward algorithm, and
- to find most likely sequence of states via Viterbi algorithm.

Forward, forward-backward and Viterbi algorithms constitute the foundation for the second semestral project.

1 Inference for HMM

In the previous exercise, you implemented the forward algorithm which computed

$$P(X_t|e_1^t),$$

i.e. the filtered belief distribution over the states at time t given the evidence from time 0 to time t.

1.1 Smoothing

In smoothing, we want to compute the smoothed belief distribution

$$P(X_k|e_1^t) \text{ for some } k \in (0, t),$$

i.e. for the case, when we have not only some past evidence, e_1^k, up to the current time (as in filtering), but also some future evidence, e_{k+1}^t. In the lecture, we have shown that

$$P(X_k|e_1^t) = \alpha f_k \times b_k,$$

where f_k and b_k are the *forward* and *backward* messages, respectively. Forward messages f_k are computed during a forward pass in the forward algorithm; they are just the filtered beliefs.

The meaning of the backward messages b_k is

$$b_k = P(e_{k+1}^t|X_k).$$

Note that on contrary to forward messages f_t, *backward messages are not probability distributions*, i.e. they don't have to sum up to 1. They are just a collection of probabilities, one for each state, but coming from several different distributions.

The backward messages are computed recursively in a backward pass:

$$b_k = \sum_{x_{k+1}} P(e_{k+1}|x_{k+1})P(x_{k+1}|X_k)b_{k+1}.$$

Task 1: In `hmm_inference.py`, fill in function `backward1()` which shall implement the above update equation for the backward message.

Hints:

- You should return a new `Counter` object, not just the modified input argument.
- Similarly to function `forward1()`, it realizes matrix-vector multiplication, i.e. you will probably use 2 nested `for`-loops.
- For the default `WeatherHMM` and $b(+rain) = 0.6, b(-rain) = 0.7$, a correct implementation shall work like this:

```
>>> from weather import WeatherHMM
>>> from hmm_inference import backward1
>>> from collections import Counter
>>> wtr = WeatherHMM()
>>> b = Counter({'+rain': 0.6, '-rain': 0.7})
>>> e = '+umb'
>>> backward1(b, e, wtr)
Counter({'+rain': 0.42, '-rain': 0.26})
```

Now, we are ready to implement the following forward-backward algorithm:

Algorithm 1: FORWARD-BACKWARD(e_1^t, P_0) returns a vector of prob. distributions

Input : e_1^t – a vector of evidence values for steps $1, \ldots, t$
P_0 – the prior distribution on the initial state
Local : f_0^t – a vector of forward messages for steps $0, \ldots, t$
b – the backward message, initially all 1s
s_1^t – a vector of smoothed estimates $P(X_k|e_1^t)$ for steps $k = 1, \ldots, t$
Output: a vector of prob. distributions, i.e. the smoothed estimates s_1^t
begin
  $f_0 \leftarrow P_0$
  for $i = 1$ *to* t **do**
    $f_i \leftarrow$ FORWARD1(f_{i-1}, e_i)
  $b \leftarrow \mathbf{1}$ // Vector of all 1s, for each state one.
  for $i = t$ *downto* 1 **do**
    $s_i \leftarrow$ NORMALIZED($f_i \times b$)
    $b \leftarrow$ BACKWARD1(b, e_i)
  return s_1^t

Note that the whole first loop is just the forward algorithm, so you can just call function `forward()` to get $f_0, \ldots, f_t$. Also note that the above algorithm does not store the individual b_ts; it just updates a single collection of probabilities b. If you need all the b_ts, store them in a list and return them together with the smoothed beliefs.

Task 2: In `hmm_inference.py`, fill in function `forwardbackward()` such that it implements the above algorithm.

Task 3: In `PM03.py`, execute function `run_fb()`. Observe the results and try to explain them. Try longer observation sequences, e.g. by repeating the default sequence $(+u, +u, -u, +u, +u)$ several times. How do the estimates for observation $-u$ differ in individual repetitions?

Hints:

- The expected output from `run_fb()` is as follows:

```
Comparison of filtering and smoothing
Initial distribution: Counter({'+rain': 0.5, '-rain': 0.5})
Observation at time 1 : +umb
Filtered: Counter({'+rain': 0.8181818181818181, '-rain': 0.18181818181818182})
Smoothed: Counter({'+rain': 0.8673388895754847, '-rain': 0.13266111042451528})
Observation at time 2 : +umb
Filtered: Counter({'+rain': 0.883357041251778, '-rain': 0.1166429587482219})
Smoothed: Counter({'+rain': 0.8204190536236754, '-rain': 0.17958094637632463})
Observation at time 3 : -umb
Filtered: Counter({'-rain': 0.8093320602764746, '+rain': 0.19066793972352525})
Smoothed: Counter({'-rain': 0.6925164239933823, '+rain': 0.30748357600661774})
Observation at time 4 : +umb
Filtered: Counter({'+rain': 0.7307940045849821, '-rain': 0.2692059954150179})
Smoothed: Counter({'+rain': 0.8204190536236753, '-rain': 0.17958094637632457})
Observation at time 5 : +umb
Filtered: Counter({'+rain': 0.8673388895754848, '-rain': 0.13266111042451526})
Smoothed: Counter({'+rain': 0.8673388895754848, '-rain': 0.13266111042451526})
```

1.2 Most likely sequence of states

In this section, our goal is to find

$$\arg\max_{x_1^t} P(x_1^t|e_1^t) = \arg\max_{x_1^t} P(x_1^t, e_1^t).$$

Similarly to other HMM inference algorithms, we shall use a recursive algorithm. We shall first propagate forward the so-called *max message* defined as

$$m_t = \max_{x_1^{t-1}} P(x_1^{t-1}, X_t, e_1^t),$$

which can be recursively computed from a previous message as

$$m_t = P(e_t|X_t) \max_{x_{t-1}} P(X_t|x_{t-1}) m_{t-1}.$$

This is the same as in the forward update (implemented in `forward1()`), only the summation over x_{t-1} is replaced by maximization over x_{t-1}. We shall also keep track about the best predecessors of each state x_t, i.e. for which x_{t-1} gave the computation of $m_t(x_t)$ the maximal value.

Task 4: In `hmm_inference.py`, fill in the function `viterbi1()` such that it implements the max message update as above.

Hints:

- You should return a new `Counter` object, not just the modified input argument.
- The algorithm needs to iterate over the current and previous states, i.e. you will probably use 2 nested `for`-loops.
- For the default `WeatherHMM` and $m(+rain) = 0.45, m(-rain) = 0.1$, a correct implementation shall work like this:

```
>>> from weather import WeatherHMM
>>> from hmm_inference import backward1
>>> from collections import Counter
>>> wtr = WeatherHMM()
>>> m = Counter({'+rain': 0.45, '-rain': 0.1})
>>> e = '+umb'
>>> m, pred = viterbi1(m, e, wtr)
```

```
>>> m
Counter({'+rain': 0.28350000000000003, '-rain': 0.027000000000000003})
>>> pred
{'-rain': '+rain', '+rain': '+rain'}
```

Now, we are ready to implement the Viterbi algorithm.

Task 5: In `hmm_inference.py`, fill in function `viterbi()` which propagates the max message through all the time slices and constructs the maximum likelihood sequence of states by following the best state predecessors from the final state back to the start.

Hints:

- Decompose the function to further functions, as you see fit. E.g., there could be a function for constructing the ML sequence after you propagate the max message forward.
- The recursion should be started with $m_1 = \text{FORWARD1}(P(X_0), e_1)$.

Task 6: In `PM03.py`, understand and execute function `run_viterbi()`, and observe the outputs.

Hints:

- It uses uniform initial distribution over states, the default `WeatherHMM`, and observation sequence with umbrella on days 1, 2, 4, 5 and no umbrella on day 3.
- If implemented properly, you shall observe the following output:

```
Viterbi
Initial distribution: Counter({'+rain': 0.5, '-rain': 0.5})
+umb Max msg: Counter({'+rain': 0.45, '-rain': 0.1})
+umb Max msg: Counter({'+rain': 0.28350000000000003, '-rain': 0.027000000000000003})
-umb Max msg: Counter({'-rain': 0.06804, '+rain': 0.019845})
+umb Max msg: Counter({'+rain': 0.0183708, '-rain': 0.0095256})
+umb Max msg: Counter({'+rain': 0.011573604, '-rain': 0.001333584})
ML seq of states: ['+rain', '+rain', '-rain', '+rain', '+rain']
```

2 Homework

As usual, finish the exercise as a homework, ask questions on the forum, and upload the solution via BRUTE!

As another part of homework, you shall work on the HMM semestral project where you will use the implemented algorithms to estimate the location of a robot in a maze.

3 Have fun!