B(E)3M33UI—PM02: Hidden Markov Models I.

Petr Pošík

April 17, 2018

The goals of this exercise:

- get acquainted with the HMM interface we will use
- implement belief updates by time step and by observation
- implement filtering by the forward algorithm

1 Representation of probabilities

In HMM exercises and project, we need to represent discrete probability distributions (and other collections of probabilities), i.e. we need to store a probability for each state or observation, such that we can easily query any of them.

We chose to represent the probabilities by `collections.Counter` objects. `Counter` is a subclass of dictionary, specialized such that

- the values must be numbers, and
- the query for a non-existent key returns 0 (and not `KeyError` as in case of dictionary).

Sampling from a distribution represented by a `Counter` can be done using a helper function `utils.weighted_random_choice`.

```
>>> from collections import Counter
>>> pX = Counter()
>>> pX['rain']
0
>>> pX['rain'] = 0.4
>>> pX['sun'] = 0.6
>>> pX
Counter({'rain': 0.4, 'sun': 0.6})
>>> from utils import weighted_random_choice
>>> for i in range(5):
...     print(weighted_random_choice(pX), end=', ')
sun, rain, rain, sun, rain,
```

2 Hidden Markov model

2.1 General interface

We have specified for you a general interface of an HMM. It is implemented as an abstract class (partially implemented class) `HMM`.

Task 1: In `hmm.py`, study and try to understand the code of class `HMM`, i.e. which functions each HMM subclass will provide and what their purpose is.

Hints:

- Make sure you know what the inputs and outputs of individual methods are.
- Concentrate on methods `pt()`, `pe()`, and `simulate()`.

2.2 Weather-Umbrella domain

Now we know what the general interface of an HMM shall be. To make a practically usable implementation of a particular domain, we will derive a new class from `HMM`. *Do not change anything in `hmm.py`!*

The Weather-Umbrella domain known from the lectures is implemented in `weather.py` as a class `WeatherHMM` derived from class `HMM`.

Task 2: In `weather.py`, study the code of class `WeatherHMM`. Concentrate on what particular changes/additions were made to the base `HMM`.

2.3 HMM as a data generator

Having a fully specified HMM, e.g. `WeatherHMM`, we can use it to generate data (states and observations) from the modeled domain.

Task 3: In `PM02.py`, understand and execute function `run_simulation()` repeatedly, observe the outputs. Consider the variety of possible outputs although the underlying model is the same!

Task 4: In `PM02.py`, modify function `run_simulation()` such that you initialize the `WeatherHMM` object with a different transition and/or emission models. Run repeatedly and observe the outputs.

Hints:

- You can copy and modify `default_T` and `default_E` data structures from `weather.py` and provide one or both of them to the `WeatherHMM()` "constuctor".

3 Inference for HMM

In this part, you shall implement some inference algorithms for HMM.

3.1 Update beliefs by time step

You shall implement the following update equation:

$$B(X) \leftarrow \sum_{x'} P(X|x') \cdot x',$$

where $B(X)$ is a probability distribution over states (a `Counter` object), X represents the set of possible current states, and x' are the previous states.

Task 5: In `hmm_inference.py`, fill in the function `update_belief_by_time_step()` such that it implements the above update equation.

Hints:

- You should return a new `Counter` object, not just the modified input argument.
- The function de facto realizes a matrix-vector multiplication (transition matrix $\times$ beliefs). There are more ways how to implement this, but most likely, you should use 2 nested `for`-loops.
- For the default `WeatherHMM` and $B(+rain) = 0.1, B(-rain) = 0.9$, a correct implementation shall work like this:

```
>>> from weather import WeatherHMM
>>> from hmm_inference import update_belief_by_time_step
>>> from collections import Counter
>>> wtr = WeatherHMM()
>>> B = Counter({'+rain': 0.1, '-rain': 0.9})
>>> update_belief_by_time_step(B, wtr)
Counter({'+rain': 0.34, '-rain': 0.66})
```

3.2 Prediction

We will use function `update_belief_by_time_step()` repeatedly to predict the belief state in more distant future. This is just the prediction in Markov Chain, i.e. the beliefs shall converge to the *stationary distribution* if it exists.

Task 6: In `hmm_inference.py`, fill in the function `predict()` such that it updates the belief state by the given number of time steps.

Task 7: In `PM02.py`, execute function `run_prediction()`. In the unmodified form, the function runs prediction for the default `WeatherHMM` with the uniform prior distribution over the states. What is the output? Can you explain it?

Task 8: Experiment with function `run_prediction()`. Try to provide different prior beliefs and/or different transition matrix. Does the sequence converge to the same distribution even if you change the prior beliefs? Observe for which priors and transition matrices the beliefs become less/more uncertain.

Hints:

- For the initial belief and transition matrix set like this:

```
prior = Counter({'+rain': 1})
T = {
    '-rain':
        {'-rain': 0.9,
         '+rain': 0.1},
    '+rain':
        {'-rain': 0.2,
         '+rain': 0.8}
}
wtr = WeatherHMM(trans_model=T)
```

the output of a correct implementation shall be:

```
Prediction from initial state Counter({'+rain': 1})
Counter({'+rain': 0.8, '-rain': 0.2})
Counter({'+rain': 0.6600000000000001, '-rain': 0.3400000000000001})
Counter({'+rain': 0.5620000000000002, '-rain': 0.43800000000000017})
Counter({'-rain': 0.5066000000000002, '+rain': 0.49340000000000017})
Counter({'-rain': 0.5546200000000002, '+rain': 0.4453800000000002})
Counter({'-rain': 0.5882340000000003, '+rain': 0.4117660000000002})
Counter({'-rain': 0.6117638000000003, '+rain': 0.3882362000000002})
Counter({'-rain': 0.6282346600000003, '+rain': 0.3717653400000002})
Counter({'-rain': 0.6397642620000004, '+rain': 0.36023573800000025})
Counter({'-rain': 0.6478349834000005, '+rain': 0.35216501660000027})
```

3.3 Update beliefs by evidence

Now we shall implement the following update equation:

$$B(X) \leftarrow \alpha P(e|X) \cdot B(X),$$

where $B(X)$ is a belief distribution over the states, as before, X is the set of possible current states, e is the single observation we use for the update, and α is a normalization constant.

Task 9: In `hmm_inference.py`, fill in the function `update_belief_by_evidence()` such that it implements the above update equation.

Hints:

- You should return a new `Counter` object, not just the modified input argument.
- We shall use this function in different contexts, and sometimes we want the output to be normalized, sometimes not. If the parameter `normalize=True` is given, you shall normalize the updated beliefs by using function `utils.normalized()`.
- The function de facto realizes a vector-scalar multiplication (beliefs $\times$ likelihood of the evidence). You should probably use a single `for`-loop.
- If implemented correctly, you should be able to run the following code and observe the results:

```
>>> from hmm_inference import update_belief_by_evidence
>>> from weather import WeatherHMM
>>> from collections import Counter
>>> wtr = WeatherHMM()
>>> B = Counter({'+rain': 0.5, '-rain': 0.5})
>>> update_belief_by_evidence(B, '-umb', wtr, normalize=True)
Counter({'+rain': 0.1111111111111112, '-rain': 0.888888888888889})
```

3.4 Repetitive updates by evidence

When using HMM, we usually assume that we have only a single piece of evidence for each time slice. But we may actually update the beliefs by several pieces of evidence *in a single time slice*. In the Weather-Umbrella domain, this would mean that we can observe not only our boss bringing an umbrella or not, but several such bosses, each of which is independent, but follows the same probabilistic rules (i.e. they bring an umbrella according to the same emission model).

Task 10: In `PM02.py`, understand and execute function `run_evidence_upates()`, and observe outputs. Does the order of observations matter? Is it the expected result?

3.5 Filtering: Forward algorithm

We shell implement an algorithm which solves the filtering task, i.e. the estimation of

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

We know from the lectures that it can be solved by using forward algorithm, which propagates the beliefs estimates through the model:

$$f_t = \alpha P(e_t|X_t) \sum_{x_{t-1}} P(X_t|x_{t-1}) f_{t-1},$$

where $f_t = P(X_t|e_1^t)$ is the belief distribution (`Counter`) in time t, f_{t-1} is the previous belief distribution over states, X_t is the set of states in time t, x_{t-1} represents previous states, and e_t is the observation at time t.

We shall implement the whole algorithm in 2 steps:

- `forward1()` which shall perform a single update step (note the `'1'` character at the end of the function name), and
- `forward()` which will propagate the beliefs through the whole HMM and will return the belief states for individual steps.

Task 11: In `hmm_inference.py`, fill in function `forward1()`, such that it will implement the update equation for f_t above.

Hints:

- The function directly combines the update for time step (which gives 1-step prediction of beliefs) and update by evidence.

- Again, the function can be used in several contexts, sometimes with the normalization of each f_t, sometimes without it. Pass the `normalize` parameter to the `update_beliefs_by_evidence()`.
- If implemented correctly, you should be able to run and observe:

```
>>> from collections import Counter
>>> from weather import WeatherHMM
>>> from hmm_inference import forward1
>>> wtr = WeatherHMM()
>>> f = Counter({'+rain': 0.5, '-rain': 0.5})
>>> forward1(f, '+umb', wtr, normalize=True)
Counter({'+rain': 0.8181818181818181, '-rain': 0.18181818181818182})
```

Task 12: In `hmm_inference.py`, fill in the function `forward()` which shall implement the whole forward pass along the HMM given an evidence sequence.

Hints:

- Here we want proper probability distributions over individual states, so normalization shall be turned on.

Task 13: In `PM02.py`, understand and execute function `run_filtering()`, and observe outputs.

Hints:

- It uses uniform ínitial 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:

```
Filtering
Initial distribution: Counter({'+rain': 0.5, '-rain': 0.5})
+umb Counter({'+rain': 0.8181818181818181, '-rain': 0.18181818181818182})
+umb Counter({'+rain': 0.883357041251778, '-rain': 0.1166429587482219})
-umb Counter({'-rain': 0.8093320602764746, '+rain': 0.19066793972352525})
+umb Counter({'+rain': 0.7307940045849821, '-rain': 0.2692059954150179})
+umb Counter({'+rain': 0.8673388895754848, '-rain': 0.13266111042451526})
```

As can be seen from the filtered beliefs above, the single different observation on day 3 is sufficient to make the algorithm think that it did not rain on day 3. With different HMM parameters, it may actually happen that the algorithm would estimate $R_3 = +rain$ as more probable than $R_3 = -rain$.

Task 14: Think about the possibilities how you could force the filtering algorithm to estimate $R_3 = +rain$ as more probable than $R_3 = -rain$ by changing either the transition model, or the emission model. After you will have your guess, evaluate it using `PM02.py` by modifying function `run_filtering()` accordingly and observing the results.

4 Homework

If you would like to do a bit more, you can try to implement function `hmm_inference.likelihood()` which would compute the likelihood that the sequence of observations came from a particular HMM.

The function is very similar to `forward()` function, it will have the same parameters, but it will return only a single number. See the lecture slides for a short description. If implemented properly, and you execute `run_likelihood()` which compares a `WeatherHMM` with default and modified emission models, you shall observe:

```
Likelihood of HMM1: 0.0343037005
Likelihood of HMM2: 0.03832461249999999
```

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

5 Have fun!