Hidden Markov Models - Observed sequence probability evaluation

December 10th, 2018

1 Hidden Markov Models

*A **Hidden Markov Model** (HMM) is a doubly stochastic process with an underlying stochastic process that is not observable (it is hidden), but can only be observed through another set of stochastic processes that produce the sequence of observed symbols.* (Rabiner, 1989)

The HMM over N states $\mathcal{S} = \{s_i\}_{i<N}$ and a vocabulary $\mathcal{C} = \{c_k\}_{k<C}$ of size C is defined by a tuple $\langle \Pi, A, B \rangle$ where:

- $\Pi \in \mathbb{R}^N$ represents the **initial state distribution** with elements
- $\pi_i = \pi(s_i) = P(S_0 = s_i)$
- $A \in \mathbb{R}^{N \times N}$ represents the **transition probability matrix** with elements $a_{ij} = a(s_i, s_j) = P(S_{t+1} = s_j | S_t = s_i)$
- $B \in \mathbb{R}^{N \times M}$ represents the **emission probability matrix** with elements $b_{ik} = b(s_i, c_k) = P(O_t = c_k | S_t = s_i)$

There are three classical problems related to a HMM:

- **evaluation**: given a HMM and an observed sequence, find the probability that the sequence was generated by that HMM
- **decoding**: given a HMM and an observed sequence, find the most likely sequence of state that might have generated the observed sequece
- **learning**: given a set of observed sequences, find the most likely model that explains those sequences

In what follows you will implement an algorithm for the first problem (`Forward` for *likelihood evaluation*) and apply it on a toy problem.
The second algorithm (*Viterbi*) is applied for decoding the most likely sequence of hidden states that produced the observation sequence.

1.1 The problem: *The Climber Robot*

A robot wanders on a $H \times W$ grid whose cells have different colors and elevations. The robot starts from a random position and moves to a neighbouring cell at each time step observing one of the following four colors: *black*, *red*, *green*, and *blue*.

- There are $N = H \times W$ different states.
- There are $C = 4$ possible values for the observations.

The robot might start from any state.

$$\pi_i = \pi(s_i) = (HW)^{-1} \quad \forall i$$

At each time step the robot moves with probability p_t to one of the neighbours with the highest elevation. With probability $1 - p_t$ he moves at random to a neighbouring cell.

$$a_{ij} = a(s_i, s_j) = \begin{cases} 0 & \text{if } s_j \text{ is not a neighbour of } s_i \\ \frac{p_t}{N_{max}} + \frac{(1-p_t)}{N} & \text{if } s_j \text{ is one of the } N_{max} \text{ neighbours with maximum elevation} \\ \frac{(1-p_t)}{N} & \text{if } s_j \text{ is one of the other } (N - N_{max}) \text{ neighbours of } s_i \end{cases}$$

Whenever the robot reaches a cell, he observes its colour. But its sensors are noisy, hence he will perceive the correct colour with probability p_o, and will perceive a colour at random with probability $1 - p_o$.

$$b_{ik} = b(s_i, c_k) = \begin{cases} p_o + \frac{(1-p_o)}{C} & \text{if } c_k \text{ is the true color of } s_i \\ \frac{(1-p_o)}{C} & \text{otherwise} \end{cases}$$

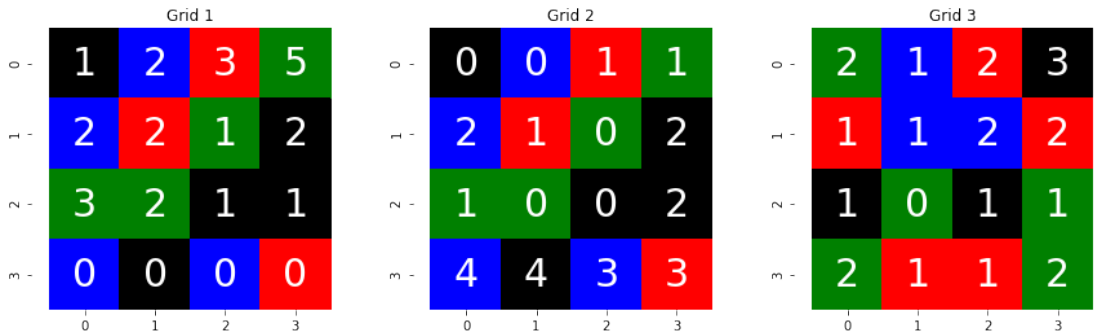
1.1.1 Three toy grids to play with

We'll use the following three grids to test our algorithms.

```
grid1 = Grid("Grid 1",
             [[1, 2, 3, 5], [2, 2, 1, 2], [3, 2, 1, 1], [0, 0, 0, 0]],  # elevation
             [[0, 3, 1, 2], [3, 1, 2, 0], [2, 2, 0, 0], [3, 0, 3, 1]])  # color

grid2 = Grid("Grid 2",
             [[0, 0, 1, 1], [2, 1, 0, 2], [1, 0, 0, 2], [4, 4, 3, 3]],  # elevation
             [[0, 3, 1, 2], [3, 1, 2, 0], [2, 2, 0, 0], [3, 0, 3, 1]])  # color

grid3 = Grid("Grid 3",
             [[2, 1, 2, 3], [1, 1, 2, 2], [1, 0, 1, 1], [2, 1, 1, 2]],  # elevation
             [[2, 3, 1, 0], [1, 3, 3, 1], [0, 2, 0, 2], [2, 1, 1, 2]])  # color

GRIDS = [grid1, grid2, grid3]
```

Use the following values for p_t and p_o

- $p_t = 0.8$
- $p_o = 0.8$



1.2 Tasks

- **Task 0**: Create a representation for the game *grid*. Implement helper functions such as `get_neighbours(state)` and `get_colors(state)`. The *neighbours* function should return the possible neighbours of any given grid position, together with the probability of transitioning to that neighbour (see rules in Section 1.1). Similarly, the *colors* function should return the list of possible colors together with the probabilities of observing them in that state (see rules in Section 1.1 and the color attributions in to cells in Section 1.1.1).
- **Task 1**: Implement a method to extract the transition probability $a_{ij} = a(s_i, s_j)$ matrix for a given grid.
- **Task 2**: Implement a method to extract the emission probability matrix for a grid $b_{ik} = b(s_i, c_k)$.
- **Task 3**: Implement a method to sample a sequence of observations from a model (grid).
- **Task 4**: Implement the forward algorithm to compute the probability that a sequence of observations was generated by a given model (grid).
- **Task 5**: Implement the Viterbi algorithm to extract the most probable sequence of states that might have generated the observed sequence.

1.3 Extracting the HMM parameters

Given a `Grid` object, build the three matrices of parameters $\langle \Pi, A, B \rangle$.

1.3.1 Initial state distribution

The initial probability Π for each grid cell is computed using a uniform distribution. For $N = H\dot{W}$ grid cells, each cell has initial probability $\frac{1}{N}$.

1.3.2 Task 1: Transition probability matrix

```
# Test transition probabilities

        test_idxs = i: [0, 1, 2, 5, 10, 13, 15], j: [1, 0, 3, 4, 9, 2, 14]
        test_values = ([
            Grid 1: [.5,      .2 / 3, .8 + .2 / 3, .8 / 3 + .05, .85, 0, .1],
            Grid 2: [.1,      .2 / 3, .8 + .2 / 3,          .85, .05, 0, .9],
```

```
            Grid 3: [.5, .4 + .2 / 3, .8 + .2 / 3,          .05, .05, 0, .5]
        ])
```

1.3.3 Task 2: Emission probability matrix

```
# Test emission probabilities

        test_idxs = i: [0, 3, 2, 5, 10, 13, 15], j: [2, 2, 3, 0, 1, 1, 1]
        test_values = ([
            Grid 1: [.05, .85, .05, .05, .05, .05, .85],
            Grid 2: [.05, .85, .05, .05, .05, .05, .85],
            Grid 3: [.85, .05, .05, .05, .05, .85, .05]
        ])
```

1.4 Task 3 - Sampling from the model

Given a model (a `Grid`), implement a function `get_sequence` which returns a sequence of observations and the corresponding states. Pseudocode for `get_sequence`:

Algorithm 1 Sampling a HMM Model

procedure GET_SEQUENCE(grid, length)
  $H, W \leftarrow$ grid shape
  $states, observations \leftarrow [], []$
  for t $\leftarrow 0..(length - 1)$ **do**
    if $t == 0$ **then**
      $state \leftarrow (randint(H), randint(W))$
    else
      $state = sample(grid.get\_neighbours(state))$
    $obs = sample(grid.get\_colors(state))$
    $states \leftarrow states \cup \{state\}$
    $observations \leftarrow states \cup \{obs\}$
  return $states, observations$

Note: For the sampling from a discrete probability distribution use a roulette-style implementation.

1.5 Evaluation

We'll now evaluate the probabiity that a given sequence of observations was generated by a given model. That way we will look at a sequence and see if we can figure out which grid generated it.

1.5.1 Forward values

Given a sequence $\mathbf{o}$ of size T and a model with parameters $\Lambda = \langle \Pi, A, B \rangle$, we need to compute $P(\mathbf{o}|\Lambda)$. Since there are various state sequences that might have generated the observed emissions, we need to sum them up.

$$P(\mathbf{o}|\Lambda) = \sum_{\mathbf{s}\in\mathcal{S}^T} P(\mathbf{s}|\Lambda)\, P(\mathbf{o}|\mathbf{s},\Lambda) = \sum_{\mathbf{s}\in\mathcal{S}^T} P(s_0|\Lambda) \prod_{t=1}^{T-1} P(s_t|s_{t-1},\Lambda) \prod_{t=0}^{T-1} P(o_t|s_t,\Lambda)$$

We can avoid summing over all possible sequences of states by using dynammic programming and exploit the Markov property of the state transitions.

$\alpha_t(s)$ represents the probability of seeing the first t observations and reaching state s.

$$\alpha_t(s) = \begin{cases} \pi(s)\, b(s, o_0) & \text{if } t = 0 \\ \sum_{s'} \alpha_{t-1}(s')\, a(s', s)\, b(s, o_t) & \text{if } t > 0 \end{cases}$$

The above equation gives an interative method to compute the α values in sequence.
In the end: $P(\mathbf{o}|\Lambda) = \sum_s \alpha_{T-1}(s)$

Algorithm 2 Compute probability of a sequence given a model

```
1:  procedure FORWARD(grid, observations)
2:      N = grid.nr_states
3:      T = card(observations)
4:      alpha = zeros(T, N)
5:
6:      //Task4 − computepandalphavalues
7:      //Hint : use the functions implemented
8:      //in the previous tasks (transition and emission probabilities)
9:
10:     return p, alpha
```

To determine the most likely model (grid), run the `forward` algorithm against each of the 3 defined grids and return the one with the highest sequence observation probability.

Additionally, to observer how sequence length influences model discrimination capability, make several runs in which you select a model at random, generate a sequence from it and then run the forward algorithm to determine the most likely source. Compute the number of times the algorithm correctly determines the original model based on the observed sequence. Analyse how the percentage of correct guessing changes as the *length* of the sequence increases.

1.6 Task 5 - Decoding

For decoding we'll use the Viterbi algorithm.

$\delta_t(s)$ represents the probability of the most probable path ending in state s at time step t.

$$\delta_t(s) = \max_{\mathbf{s}\in\mathcal{S}^t} P(S_{0:t-1} = \mathbf{s}, S_t = s, O_{0:t} = \mathbf{o}_{0:t}|\Lambda)$$

$$\delta_t(s_j) = \begin{cases} \pi(s_j)\, b(s_j, o_0) & \text{if } t = 0 \\ \max_{s_i} \delta_{t-1}(s_i)\, a(s_i, s_j)\, b(s_j, o_t) & \text{if } t > 0 \end{cases}$$

The state that ends the most probable sequence of states is the one that corresponds to the largest δ_{T-1}.

$s_t = \text{argmax}_s\, \delta_{T-1}(s)$

In order to be able to reconstruct the most probable path, back links should be stored while computing $\delta_1, \ldots \delta_{T-1}$:

$$back_t\left(s_j\right) = \underset{s_i}{\operatorname{argmax}}\, \delta_{t-1}\left(s_i\right) a\left(s_i, s_j\right)$$

The task is to write a function `viterbi(grid, observations)`, which takes as parameters a given model (grid) and a sequence of observed colors and returns the list of most probable states that generated those colors, as well as δ (under matrix form: $\delta[t, s]$, where $t = 0..T-1$ and $s \in grid.states$

To test your Viterbi implementation evaluate your output (list of states and δ matrix) against the following example call $(states, delta) = viterbi(grid2, [0, 0, 1, 3])$, where $[0, 0, 1, 3]$ are the observed colors.

```
# Testing Viterbi

test_states = [(1, 3), (2, 3), (3, 3), (3, 2)]
test_values = [ [5.31250000e-02, 3.12500000e-03, 3.12500000e-03, 3.12500000e-03,
                 3.12500000e-03, 3.12500000e-03, 3.12500000e-03, 5.31250000e-02,
                 3.12500000e-03, 3.12500000e-03, 5.31250000e-02, 5.31250000e-02,
                 3.12500000e-03, 5.31250000e-02, 3.12500000e-03, 3.12500000e-03],
                [1.77083333e-04, 2.65625000e-04, 7.29166667e-05, 1.77083333e-04,
                 2.39062500e-03, 7.29166667e-05, 1.77083333e-04, 3.01041667e-03,
                 7.29166667e-05, 1.77083333e-04, 3.01041667e-03, 3.91354167e-02,
                 2.30208333e-03, 2.39062500e-03, 2.25781250e-03, 2.30208333e-03],
                [7.96875000e-06, 8.85416667e-07, 1.05364583e-04, 1.00347222e-05,
                 7.96875000e-06, 9.48281250e-04, 1.00347222e-05, 1.30451389e-04,
                 5.57812500e-05, 7.96875000e-06, 1.30451389e-04, 1.30451389e-04,
                 1.03593750e-04, 1.03593750e-04, 1.27942708e-04, 2.88297569e-02],
                [2.65625000e-08, 4.03019531e-05, 5.01736111e-08, 4.56579861e-06,
                 6.85133203e-04, 1.85937500e-07, 2.37070312e-06, 4.51562500e-07,
                 5.17968750e-07, 2.37070312e-06, 4.34837963e-07, 1.44148785e-04,
                 7.63140625e-05, 5.54418403e-06, 2.20547641e-02, 5.65289352e-06]]
```