CP5: Dynamic Programming for Hidden Markov Models

Last modified: 2023-04-22 22:54

Status: **RELEASED.**

What to turn in:

- ZIP file of source code: https://www.gradescope.com/courses/496674/assignments/2844503/
- PDF report file: https://www.gradescope.com/courses/496674/assignments/2844507/

Your ZIP file should include

- All starter code .py files (with your edits) (in the top-level directory)
- These will be *auto-graded*. Be sure to check gradescope test results for immediate feedback.

Your PDF should include (in order):

- Your full name
- Collaboration statement
- About how many hours did you spend (coding, asking for help, writing it up)?
- Code for the forward algorithm on page 1
- Code for the viterbi algorithm on page 2

Questions?: Post to the cp5 topic on the discussion forums.

Jump to: Problem 1 Problem 2 Starter Code

Overview and Background

In this coding practical, we will implement 2 possible algorithms for Hidden Markov models:

- the Forward algorithm (Problem 1)
- - Useful for computing forward marginals over states: $\log p(z_t = k | x_{1:t}, \theta)$
- - Useful for computing incomplete log likelihoods: $\log p(x_{1:T} | \theta)$
- the Viterbi algorithm (Problem 2)
- - Useful for inferring "optimal" hidden state sequences: $\hat{z_{1:T}} = \arg\max_{z_{1:T}} \log p(z_{1:T} | x_{1:T}, \theta)$

The problems will try to address several key practical questions:

- How can we use dynamic programming to accomplish these computations?
- How can I turn written mathematics into working NumPy code?
- How can we be sure to do all computations without underflow/overflow/NaNs?

Probabilistic Model: Key Assumptions

We are given T observed feature vectors: $\{x_t\}_{t=1}^T$, with $x_t \in \mathbb{R}^D$.

We assume a hidden Markov model with K hidden states, and parameters $\theta = \pi, A, \varphi$, where:

$$p(z_{1:T} | \theta) = \text{CatPMF}(z_1 | \pi) \cdot \prod_{t=2}^{T} \text{CatPMF}(z_t | A_{z_{t-1}})$$

$$p(x_{1:T} | z_{1:T}, \theta) = \prod_{t=1}^{T} \prod_{k=1}^{K} L(x_t | \varphi_k)^{\delta(z_t, k)}$$

where we are keeping the "data likelihood" or "emission" distribution (also called the data-given-state distribution) general with pdf L, so that we could handle binary data (using $L = \text{BernPMF}$) or real-valued data (e.g. $L = \text{MVNormPDF}$), or even categorical or something else.

We have three kinds of parameters:

- Initial state probabilities: $\pi = [\pi_1 \ \pi_2 \ \pi_3 \ \dots \ \pi_K]$, $\pi \in \Delta^K$
- Transition probabilities: $A = \{A_j\}_{j=1}^K$, $A_j \in \Delta^K$
- Per-state emission distribution parameters: $\varphi = \{\varphi_k\}_{k=1}^K$

Background on the Forward Algorithm (for problem 1)

For helpful background, review the lecture notes from day 20 as well as Bishop's PRML textbook Sec. 13.2.2 (plus also 13.2.4 for specific notes on numerical stability).

Inputs:

- π : 1D array, (K,)
- initial state probabilities (or equivalently, their logs)
- A : 2D array, (K, K)
- transition probabilities (or equivalently, their logs)
- $\log L$: 2D array, (T, K)
- emission log pdf evaluated at each timestep t and each state k
- $\log L_{tk} = \log p(x_t | z_t = k, \varphi)$

Definition of a "forward message" or a "forward marginal probability (over states)"

$$\alpha_{tk} \triangleq p(z_t = k | x_1, \dots x_t, \theta)$$

Base case update for α at $t = 1$:

$$\alpha_{1k} = \frac{\pi_k e^{\log L_{1k}}}{\sum_{k=1}^K \pi_k e^{\log L_{1k}}} \qquad = \frac{p(z_1 = k, x_1 | \theta)}{p(x_1 | \theta)}$$

Recursive case for α at $t \geq 2$:

$$\alpha_{tk} = \frac{\sum_{j=1}^K \alpha_{t-1,j} A_{jk} e^{\log L_{t,k}}}{\sum_{k=1}^K \sum_{j=1}^K \alpha_{t-1,j} A_{jk} e^{\log L_{t,k}}} \qquad = \frac{p(z_t = k, x_t | x_{1:t-1}, \theta)}{p(x_t | x_{1:t-1}, \theta)}$$

You can see how the denominator in this update already computes $p(x_t | x_{1:t-1}, \theta)$ as a "side effect". If we are careful to track this denominator term as we compute all the α terms, we can compute the incomplete log likelihood of the whole sequence:

$$p(x_{1:T} | \theta) = p(x_1 | \theta) \prod_{t=2}^T p(x_t | x_{1:t-1}, \theta)$$

This is just a statement of the product rule. Note: you'd want to do this computation using sums of logarithms to avoid underflow!

Background on Viterbi Algorithm (for problem 2)

For helpful background, review the lecture notes from day 21 as well as Bishop's PRML textbook Sec. 13.2.5 on the Viterbi algorithm.

For a concise write-up of the Viterbi algorithm as a dynamic programming algorithm, see Page 8 here.

Note that here in this CP5 spec, we've written $\log L$ to remind us that this value will be computed in log space, but in the lecture notes the same quantity (in log space) is just noted as L. Make sure you know which is which!

Hints for Implementation

Scalability : Avoid for loops as much as possible. Anything you can do with predefined numpy functions that are vectorized (e.g. using `my_sum = np.sum(vec_K)` instead of `for v in vec_K: my_sum += v`), please do so. In our solutions, we have for each algorithm exactly one for loop over timesteps t, and zero for loops over clusters/states k. Recommended reading on vectorization: https://realpython.com/numpy-array-programming/.

Numerical stability : For the Forward Algorithm, you should either try to use the logsumexp trick, or otherwise smartly deal with the fact that the log likelihood probabilities provided by the emission distribution G may be quite small and vulnerable to underflow.

Starter Code

You can find the starter code and datasets in the course Github repository here:

https://github.com/tufts-ml-courses/cs136-23s-assignments/tree/main/unit5_CP

Note: also includes **cp5_template.tex** : a template for your solution PDF

Problem 1: Forward Algorithm Implementation

In problem 1, you'll do the following coding tasks

Tasks for Code Implementation

(All code steps will be evaluated primarily by Autograder, but also read thru by Instructor/TA).

WRITING CODE 1 Implement the `run_forward_algorithm` method of starter code `forward_alg.py`. Make sure it passes all test cases. (Including some that may be 'hidden' from view until after grading occurs.)

Tasks for PDF report

Include your final solution to `run_forward_algorithm` as nicely-typeset code in your PDF (so we can give feedback or spot errors)

Please only include the method itself. Omit the long docstring we provide in starter code. See cp5_template.tex.

Please make the code *readable* and include relevant comments.

Problem 2: Viterbi Algorithm Implementation

In problem 2, you'll do the following coding tasks

Tasks for Code Implementation

(All code steps will be evaluated primarily by Autograder, but also read thru by Instructor/TA).

WRITING CODE 2 Implement the `run_viterbi_algorithm` method of starter code `viterbi_alg.py`. Make sure it passes all test cases. (Including some that may be 'hidden' from view until after grading occurs.)

Tasks for PDF report

Include your final solution to `run_viterbi_algorithm` as nicely-typeset code in your PDF (so we can give feedback or spot errors)

Please only include the method itself. Omit the long docstring we provide in starter code. See cp5_template.tex.

Please make the code *readable* and include relevant comments.
