STAT 570 Probabilistic Machine Learning Assignment 2

Problem 1: Mobile health application

In this assignment you will develop a hidden Markov Model (HMM) to model daily step counts recorded by a mobile device. The setup is as follows:

- Observations: Y_t denotes the number of steps measured by the mobile device for day t $(t = 1, ..., T)$.
- Hidden states: X_t denotes a latent state representing the subject's activity level. The state space is

$$\mathcal{X} = \{\text{low}, \text{medium}, \text{high}\}.$$

- State dynamics: the hidden states evolve as a Markov chain with a 3x3 transition matrix P. Specifically,

$$P(X_t = x' | X_{t-1} = x) = P_{x,x'} \quad x, x' \in \mathcal{X}.$$

- Emission model: The observation on day t is modeled by a Poisson distribution:

$$Y_t | X_t = x \sim \text{Poisson}(\lambda_x), \quad x \in \mathcal{X}.$$

We denote the full parameter set by

$$\theta = (\mu, P, \lambda),$$

where μ denotes the initial distribution, P denotes the transition matrix, and $\lambda = (\lambda_{low}, \lambda_{med}, \lambda_{high})$ denotes the Poisson rates corresponding to each state.

Q1. Forward-Backward recursion (20 points)

Implement the forward-backward recursion and compute the marginal likelihood $p(y_{1:T}|\theta)$.

Instructions

Write function `forward(y, theta)` in `problem1.py`. Your function should return a Numpy array of dimension $K \times T$, where each column stores $\log \alpha_t(x_t) = \log p(x_t, y_{1:t})$ at time t. Implement function `backward(y, theta)` in `problem2.py`. This function should return a Numpy array containing $\log \beta_t(x_t) = \log p(y_{t+1:T}|x_t)$. Then, implement `marginal_log_likelihood(y, theta)`, which returns a single numeric value (a float).

Q2. Baum-Welch (20 points)

Implement the EM algorithm to estimate the unknown parameters μ and λ_x, while keeping P fixed. Load the data file `steps.csv`. This file contains $N \times T$ matrix of sequence of steps from $N = 30$ subjects over $T = 365$ days. Plot the point estimates for μ and λ^n, for $n = 1, ..., N$ and briefly comment on the results.

Instructions

Step 1: Express the complete data log-likelihood $\log p(x_{1:T}, y_{1:T}|\theta)$.

For each iteration $i = 1, ..., I$ of EM algorithm, we perform the following steps.

1. E-step. Write out the Q-function:

$$Q(\theta|\theta^i) = \mathbb{E}_{X_{1:T} \sim p(\cdot|y_{1:T}, \theta^i)}[\log p(X_{1:T}, y_{1:T}|\theta)].$$

Identify the expression for a quantities that depends only on the parameters θ^i.

2. M-step. $\theta^{i+1} = \text{argmax}_\theta Q(\theta|\theta^i)$.

Derive the update equation for μ_x and λ_x for $x \in \mathcal{X}$.

Write a function `em(Y, iterations, mu_init, lambda_init, P)`.

- `Y` is a Numpy array of dimension $N \times T$ storing the sequences of observed steps from multiple subjects.
- `iterations` is the number of EM iterations.
- `mu_init` is the initial distribution.
- `lambda_init` is the numpy array of length 3.
- `P` is the transition matrix (we will keep this fixed).

The function should return a tuple of two Numpy arrays μ of dimension 3 and λ of dimension $N \times 3$.

Q3: Modeling (10 points)

Propose ideas for extending the HMM. We want to capture the distribution of observed steps over a larger population and incorporate additional information to better model the steps.

Instructions

In a written answer, describe how you would extend the model to incorporate:

- individual specific covariates (e.g„ age, gender),
- contextual information such as weather, weekday vs weekend to account for missing data or low step counts on active days,
- how would you utilize the information across N subjects to improve the estimation for λ_x parameters?

Explain your modeling strategy (for example, how you might let the Poisson rate depend on covariates via a log-linear model, or how you might let transition probabilities depend on context via logistic regression). No code implementation is necessary.

Problem 2: Phylogenetic tree likelihood

In this problem we will work with phylogenetic trees. We are given a fixed tree topology, observed character data at the leaves, and a substitution model.

- A fixed topology $T = (V, E, b)$ where,
  - V is the set of nodes
  - E is the set of edges,
  - $b : E \to (0, \infty)$ is a branch length for each $e \in E$.
- A set of leaf nodes $L \subset V$ for which molecular sequences are observed.
- A substitution model that governs the evolution of characters along branches. This is specified by a tuple (π, Q) where Q denotes the rate matrix of continuous time Markov chain and π denotes the stationary distribution.
- At each node $v \in V$, we have a random molecular sequence denoted by Y_v
  - The character at loci s is denoted $Y_{v,s}$ for $s = 1, ..., S$.
  - The value taken by $Y_{v,s}$ is denoted by $\Sigma = \{A, C, G, T\}$.
  - The characters evolve independently across site s.

Q1. Felsenstein pruning algorithm – bottom-up pass (20 points)

Compute the likelihood $p(Y_L)$ by marginalization of the sequences at the internal nodes $v \in V \setminus L$. To that end, you will implement a dynamic programming algorithm, making a bottom-up pass over the given tree T.

Instructions

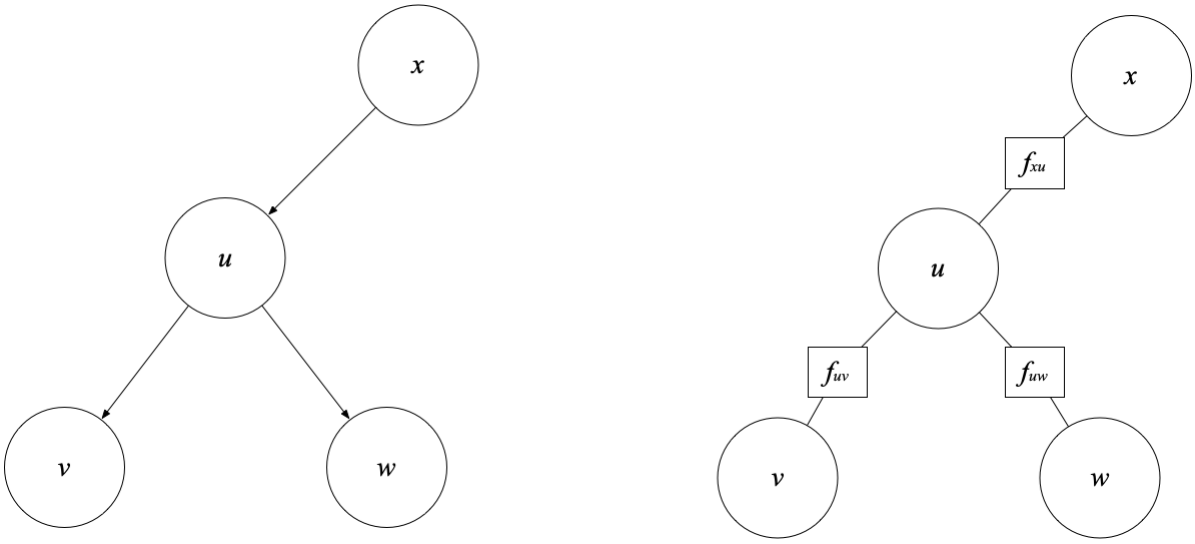
Implement `bottom_up_pass(obs, root, ctmc_model)` in `problem2.py`. Your function should return a python dictionary, where key is given by the nodes $v \in V$ and the value is a numpy array of dimension $|\Sigma| \times S$.

Implement `marginal_likelihood(bottom_up_table, pi)` that returns a single float, the marginal likelihood $p(y_L)$. The argument `bottom_up_table` represents the dynamic programming tables for each node v, returned from a call to `bottom_up_pass` and pi is the stationary distribution of CTMC.

- Familiarize yourself with the implementation of tree in `node.py` and `jukes_cantor.py`.
- The dynamic programming table for each node stores $P(Y_{\lfloor v \rfloor} | Y_{v,s} = y)$, where $Y_{\lfloor v \rfloor}$ denotes the leaf nodes at the subtree of T rooted at v and $y \in \Sigma$.
- Begin by examining how the dynamic programming tables for the leaf nodes are constructed (`construct_dp_table`).

Q2. Two pass algorithm (25 points)

Following bottom-up pass, we would like to impute the sequences at the internal nodes of the tree by incorporating all of the observations. To that end, we will derive the message passing algorithm to facilitate the top-down pass. First, consider the following tree and its factor graph representation.



- Derive the bottom-up messages coming from v, w to node u via f_{uv} and f_{uw}.
- Derive top-down messages from node u to factor node f_{uv}.
- Generalize this to complete the top-down pass.

Instructions

Implement `two_pass_algorithm(obs, root, ctmc_model)` in `problem2.py`. Your function should return a dictionary with key given by the nodes $v \in V$ and the value given by a numpy array of dimension $|\Sigma| \times S$, storing the probability mass function over each possible value for $Y_{v,s} \in \Sigma$ at site s. The entry of the table stores $P(Y_{v,s} = y, Y_{\lceil v \rceil})$ for $y \in \Sigma$ for $s = 1, ..., S$ and $Y_{\lceil v \rceil}$ denote the observations $Y_L \setminus Y_{\lfloor v \rfloor}$.

Q3. Ancestral sequence reconstruction (5 points)

For a given internal node $v \in V \setminus L$, return the most likely sequence y_v.

Instructions

Implement `reconstruction(v, bottom_up_dt, top_down_dt)` in `problem2.py`, where `bottom_up_dt`, `top_down_dt` denote bottom-up and top-down dynamic programming tables constructed via calls to `two_pass_algorithm`. `v` is the query node. Return $Y_{v,s}$ a string of length S.