

STATS 607 Project 04 Report

Dendrogram of Mixing Measures

An Pho

December 2025

a. Motivation

Finite mixture models are a standard tool for model-based clustering, but in practice they are often fit with too many components. This overfitting leads to three related problems:

- many mixture components end up redundant or nearly identical,
- selecting the “right” number of components k is delicate,
- it is hard to understand the hierarchy of sub-populations in the data.

The recent paper by Do et al. (2024) proposes *dendrograms of mixing measures*: instead of clustering data points, they cluster the *atoms* of an overfitted mixing measure and construct a hierarchical tree. This bridges model-based and model-free clustering: the tree is driven by a probabilistic mixture model, but visually resembles an agglomerative dendrogram.

My goal in this project is to:

- implement the dendrogram-of-mixing-measures algorithm as reusable Python code,
- implement the associated Dendrogram Information Criterion (DIC) for model selection,
- reproduce and extend several simulation studies comparing AIC, BIC, and DIC.

Potential beneficiaries include my own research group (for mixture modeling and clustering), as well as anyone interested in interpretable model selection for finite mixtures.

b. Project Description

What I built

The core of the project is a small Python package in `src/dmm` that provides:

- `MixingMeasure`: a class representing a discrete mixing measure $G = \sum_i p_i \delta_{\theta_i}$ (weights, locations, and, for Gaussians, covariances).
- `MixingDendrogram`: an implementation of Algorithms 1–2 in Do et al. (2024), which repeatedly merges the closest pair of atoms under the dissimilarity

$$d(p_i \delta_{\theta_i}, p_j \delta_{\theta_j}) = \frac{\|\theta_i - \theta_j\|^2}{p_i^{-1} + p_j^{-1}}.$$

For Gaussian mixtures I use moment-preserving updates to merge means and covariances.

- **DIC utilities:** functions that, for each cut level κ of the dendrogram, compute the average log-likelihood and the Dendrogram Information Criterion

$$\text{DIC}(\kappa) = -(d_n^{(\kappa)} + \omega_n \ell_n^{(\kappa)}), \quad \omega_n = \log n,$$

and return the selected $\hat{k} = \arg \min_{\kappa} \text{DIC}(\kappa)$.

On top of these primitives, I implemented several experiment scripts under `examples/`:

- `demo_synthetic.py`: end-to-end pipeline on a simple Gaussian mixture.
- `experiment_convergence_strong.py` and `experiment_convergence_weak.py`: convergence rates of the merged mixing measure and the overfitted MLE.
- `experiment_model_selection.py` (serial) and `experiment_model_selection_parallel.py` (`joblib` parallel): AIC/BIC/DIC comparisons in three regimes (well-specified, ε -contamination, skew-normal).
- `profile_model_selection.py`: simple runtime comparison between serial and parallel implementations.

All figures are saved under `out/figures/` and can be reproduced via the provided `Makefile`.

Course concepts and tools

This project touches all three major components of STATS 607:

Frictionless reproducibility. The code is organized as a small Python package (`src/dmm`) plus experiment scripts in `examples/`, with a project-level `Makefile`, `requirements.txt`, and virtual environment (`.venv`) for dependency management. All figures are written to `out/figures/` and can be regenerated with single commands, turning the project into a structured, documented, and reproducible workflow.

Simulation study. Several scripts implement simulation studies to evaluate the dendrogram algorithm and DIC: convergence experiments in strongly and weakly identifiable regimes, and model-selection experiments under well-specified, ε -contaminated, and skew-normal mixtures. These follow an ADEMP-style design (Aim, Data-generating mechanism, Estimand, Methods, Performance measures) and are used to compare AIC, BIC, and DIC empirically.

High performance computing. Because the experiments require many replications and sample sizes, I profiled the code and introduced basic performance improvements: vectorized operations for distance computations and likelihoods, a parallel implementation of the model-selection experiments using `joblib`, and a profiling script that compares serial and parallel runtimes. This reduced wall-clock time by roughly a factor of 3–4 on my machine for the larger experiment grids.

c. Results or Demonstration

Synthetic demo

The script `examples/demo_synthetic.py` simulates data from a 1D Gaussian mixture, fits an overfitted GMM with $k_{\max} = 10$ components, builds the dendrogram of mixing measures, and computes DIC along the tree. The script saves a two-panel figure `demo_synthetic_dendrogram_dic.png` that shows:

- the dendrogram of the k_{\max} estimated components,
- the DIC curve as a function of k with the selected \hat{k} highlighted.

Convergence experiments

Following Do et al. (2024), I implemented two regimes:

- a *strongly identifiable* Gaussian mixture with well-separated means and equal covariances,
- a *weakly identifiable* setting where covariance matrices are unknown and more challenging.

For a grid of sample sizes n , I generate data, fit an overfitted GMM, construct the dendrogram, and measure Wasserstein distances between:

- the true mixing measure G_0 ,
- the overfitted MLE \hat{G}_n ,
- the merged mixing measure \hat{G}_n^m obtained from the dendrogram.

The resulting log–log plots reproduce the qualitative behaviour from the paper: the merged measure converges at least as fast as, and often faster than, the overfitted MLE, especially in weakly identifiable regimes.

Model selection and misspecification

I then compare AIC, BIC, and DIC in three scenarios:

- (i) well-specified Gaussian mixture with true $k_0 = 2$;
- (ii) ε -contamination: a two-component Gaussian mixture plus a small amount of contamination from a Laplace component;
- (iii) skew-normal mixture with two heavily skewed components.

For each sample size n and method, I estimate:

- the proportion of times the correct k_0 is selected;
- the average chosen number of components.

The results are consistent with the paper. I also implemented a parallel version of these experiments using `joblib`. The profiling script shows a speedup of roughly 3–4x on my laptop for moderate replications, with nearly identical empirical selection frequencies.

d. Lessons Learned

The main challenges in this project were algorithmic and computational:

- **Numerical stability:** merging Gaussian components requires careful handling of covariances to preserve moments and avoid degeneracy. I had to debug several cases where ill-conditioned matrices caused EM or merging to fail.
- **Computational cost:** the dendrogram construction is quadratic in the number of atoms, and the Monte Carlo experiments loop over many sample sizes and replications. Adding a parallel implementation and a Makefile-based workflow made it much easier to manage long runs.

From a software-engineering perspective, this project reinforced the value of:

- keeping core logic (`src/dmm`) separate from experiment scripts,
- writing small, composable functions for simulation and plotting,
- using a simple `Makefile` to document and automate the main tasks.

Although my implementation focuses on frequentist overfitted mixtures, the interface is designed so that posterior samples from a Bayesian mixture model could be plugged in with minimal changes, which I hope to explore in future work.