

BayesianEstimationTool (BET) Cheat Sheet

Purpose

`Estimator` organizes hierarchical Bayesian estimation for datasets containing multiple related entries. Models may contain global parameters shared across entries and standardized local Zs that vary by entry. `Train` infers the population/global posterior, training local Zs, and local-Z correlation. `Test` carries the learned population state to new entries and can either sample locals from the trained population or infer them from new observations with a method. `ScoreTrainSamples` and `ScoreTestSamples` evaluate predictions. `Save` and `Load` preserve the learned state and the settings needed to reconstruct how estimation was performed. Inference is supplied through an `Method` interface; built-in methods include NUTS and SVI.

Dependencies: jax, numpyro, numpy

Installation

Install from PyPI with `python -m pip install BayesianEstimationTool`, then import with `import BayesianEstimationTool as bet`.

Conventions

- `import BayesianEstimationTool as bet` — Public import; construct the main object with `bet.Estimator(...)` and supply inference through the `Method` argument.
- `arg{}` — dictionary whose values have no entry dimension; used for one entry's data, local Zs, model output, globals, or other named values.
- `arg{[]}` — dictionary containing arrays spanning multiple entries or posterior samples.
- `arg[]` — 1D sequence.
- `Fn(...) -> value` — function argument or method together with the value it returns.
- `globalDistributions{}` — global-parameter prior dictionary: `{name: NumPyro distribution}`. BET creates the NumPyro sample sites internally; the dictionary keys become the global parameter names.
- `likelihoods{}` — single-entry likelihood dictionary: `{name: (observation, distribution)}`. BET vectorizes it across entries and creates the NumPyro observation sites internally.
- *Italic arguments* — accept `None` as a valid input.

Main Functions

Estimator(data{[]}, ModelFn(globalParams{}, localZs{}, data{}) -> modelOut{}, localZNames[], GlobalPriorFn(dataShapes{}) -> globalDistributions{}, lkjConcentration=2.0, rngSeed) -> Estimator

Creates an Estimator. **localZNames** defines standardized entry-level parameters. With multiple locals, the Estimator learns their correlation with an LKJ Cholesky prior controlled by **lkjConcentration**. **GlobalPriorFn** receives the per-entry shapes of the full data and returns `{name: NumPyro distribution}`. BET creates the named NumPyro sample sites and passes their sampled values to **ModelFn** and likelihood functions as `globalParams{}`. **rngSeed** initializes the Estimator's persistent RNG.

Train(trainDataIdxs[], TrainLikelihoodFn(globalParams{}, localZs{}, data{}, modelOut{}) -> likelihoods{}, Method=method, printSummary=False, rngSeed, saveLocalZs=True, saveModelOut=False) -> methodOutput

Fits **trainDataIdxs** using **Method**. **TrainLikelihoodFn** is evaluated one entry at a time and returns `{name: (observation, NumPyro distribution)}`; BET batches the returned observations/distributions and creates the observation sites internally. A **Method** instance supplies an explicit configuration. A **Method** class reconstructs the saved training **Method** arguments; if its class name differs from the saved **Method** name, BET warns and still attempts reconstruction. The Estimator stores global posterior samples and the learned Cholesky state; **saveLocalZs** stores transformed training-local Z samples and **saveModelOut** stores model-output samples. **printSummary** uses the Estimator's generic posterior summary unless the **Method** replaces or augments it.

Test(testDataIdxs[], TestLikelihoodFn(globalParams{}, localZs{}, data{}, modelOut{}) -> likelihoods{}, Method=method, nGlobalSamples, nLocalZSamples=2000, printSummary=False, rngSeed, saveModelOut=False) -> None | methodOutput[]

Tests on new entries. **TestLikelihoodFn**, when supplied, follows the same single-entry `{name: (observation, distribution)}` contract as training. **nGlobalSamples=None** uses the median trained global/Cholesky state; a positive integer samples that many trained states. With **TestLikelihoodFn=None**, no **Method** is needed and the Estimator draws **nLocalZSamples** locals from the trained population. With a likelihood, a **Method** instance supplies an explicit configuration. A **Method** class reconstructs saved arguments: BET uses the previous **Method**-based test configuration when available, otherwise the training configuration. If the supplied class name differs from the saved **Method** name, BET warns and still attempts reconstruction.

ScoreTrainSamples(ScoreFn(modelOut{}, data{}) -> score{}, nSamples, rngSeed) -> scores{[]}

Scores training predictions. **nSamples=None** uses all available joint training-posterior samples; otherwise the Estimator selects that many. Saved model outputs are reused when available, otherwise they are regenerated from saved parameters.

ScoreTestSamples(ScoreFn(modelOut{}, data{}) -> score{}, nSamples, rngSeed) -> scores{[]}

Scores the latest test state. **nSamples=None** uses all stored local Z samples; otherwise that many are selected for each stored global state. Saved model outputs are reused when available, otherwise they are regenerated.

Save(fileName, saveTrainLocalZs=True, saveTestLocalZs=True, saveTrainModelOut=False, saveTestModelOut=False, saveAllData=False) -> None | record{}

Writes compressed `.npz` state, or returns a nested record when **fileName=None**. Saved metadata includes method class names and constructor arguments so the estimation setup can be reconstructed without serializing method machinery. Functions are not

serialized.

Estimator.Load(fileName, ModelFn, GlobalPriorFn, data) -> Estimator | record{}

Loads saved state. With **ModelFn=None**, returns the saved record without reconstructing an Estimator. Otherwise model functions are supplied explicitly; **data** is required unless the data were saved with `saveAllData=True`.

Nuts(numWarmup=2000, numSamples=2000, numChains=4, acceptProb=0.95, denseMass=False, medianSamples=50) -> Nuts

Creates a NUTS inference Method, exposed as `bet.Nuts`. Pass `bet.Nuts(...)` for an explicit configuration or `bet.Nuts` to reconstruct saved Method arguments. **medianSamples** controls `init_to_median`; **acceptProb** is the target acceptance probability. NUTS retains posterior samples from the MCMC trajectory and prints NumPyro convergence diagnostics including `n_eff` and `r_hat`.

Svi(numSteps=10000, numSamples=2000, stepSize=0.01, numParticles=1, progressBar=True, medianSamples=50, multivariate=False) -> Svi

Creates an SVI inference Method, exposed as `bet.Svi`. Pass `bet.Svi(...)` for an explicit configuration or `bet.Svi` to reconstruct saved Method arguments. Uses `init_to_median`; **numSamples** is the number of posterior samples drawn from the fitted guide for the Estimator. By default the guide is mean-field `AutoNormal`; **multivariate=True** uses `AutoMultivariateNormal`, which can represent posterior covariance but scales less favorably for large latent spaces.