Bayesian analysis for Data Science

Externado para la vida

Mathematics Department

Outline

What is PyMC?

- PyMC is a probabilistic programming library for Bayesian statistical modeling and inference.
- It uses Theano / Aesara backend for automatic differentiation.
- Enables MCMC, variational inference, and model diagnostics.
- Models are defined using Python syntax.

Installing PyMC

Recommended with conda (Windows/Mac/Linux):

- `conda create -n pymc-env python=3.10`
- `conda activate pymc-env`
- `conda install -c conda-forge pymc`

With pip (less stable):

- `pip install pymc`

Optional: install Jupyter

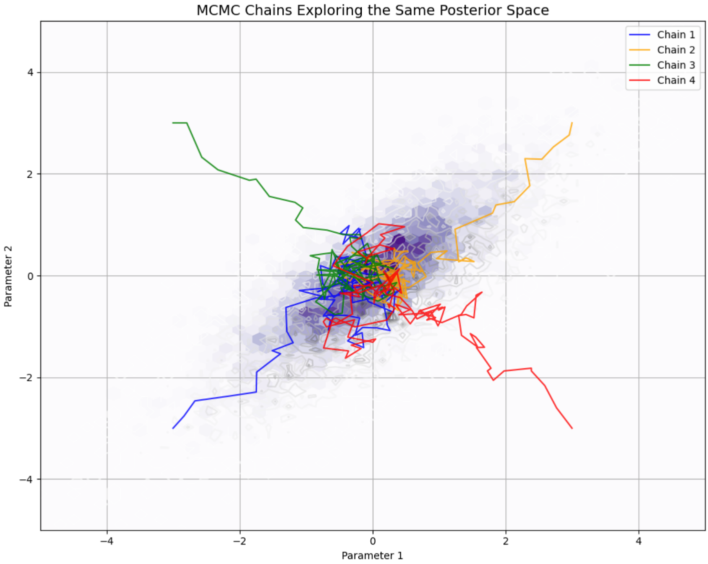
- `pip install notebook jupyterlab`

Parallel Execution on Mac/Linux/Windows

- PyMC uses `multiprocessing` by default for parallel chains.
- Number of cores can be controlled via `pm.sample()`'s `cores` argument:
- `pm.sample(draws=1000, chains=4, cores=4)`
- Windows needs special care due to multiprocessing spawning.
- Wrap your model inside a function and guard execution:

```
with model(): trace = pm.sample(draws=1000, chains=4,
cores=4)
```

Chains



MCMC Chains Exploring the Same Posterior Space
Chain 1
Chain 2
Chain 3
Chain 4
Parameter 2
Parameter 1
4
2
0
−2
−4
−4
−2
0
2
4

Beta-Binomial Example

```
with pm.Model() as beta_binom:
    alpha = pm.HalfNormal("alpha", 10)
    beta = pm.HalfNormal("beta", 10)
    theta = pm.Beta("theta", alpha=alpha, beta=beta)
    y_obs = pm.Binomial("y_obs", n=10, p=theta, observed=7)
    trace = pm.sample(1000)
    pm.model_to_graphviz(beta_binom)
```

Bayesian Linear Regression

```
with pm.Model() as model:
    alpha = pm.Normal("alpha", mu=0, sigma=10)
    beta = pm.Normal("beta", mu=0, sigma=10)
    sigma = pm.HalfNormal("sigma", sigma=1)
    mu = alpha + beta * x
    y_obs = pm.Normal("y_obs", mu=mu, sigma=sigma, observed=y)
    trace = pm.sample()
```

Hierarchical Intercept Model

```
with pm.Model() as model:
    mu_a = pm.Normal("mu_alpha", 0, 5)
    sigma_a = pm.HalfNormal("sigma_alpha", 1)
    alpha = pm.Normal("alpha", mu=mu_a, sigma=sigma_a, shape=n
    beta = pm.Normal("beta", mu=0, sigma=1)
    mu = alpha[group_idx] + beta * x
    y_obs = pm.Normal("y_obs", mu=mu, sigma=1, observed=y)
    trace = pm.sample()
```

Posterior Diagnostics

Use ArviZ to check convergence and summarize:

- `az.plot`$_{t}race(trace)$
- `az.summary(trace)`
- `az.plot`$_{f}orest(trace)$
- `az.plot`$_{p}osterior(trace)$

Tips for Modeling

- Use weakly informative priors unless you have strong prior knowledge.
- Standardize predictors if needed.
- Start with simple models, then add complexity.
- Check $\hat{R}$ and effective sample size.

- Use `idata = pm.sample(return`¡$nferencedata = True$)

Checklist for Bayesian Models with MCMC

Data Preparation

- Clean and explore the data.
- Simulate data with known parameters.

Model Definition

- Define structure and choose priors.
- Check prior assumptions visually.

Model Construction

- Implement likelihood.

Sampling

- Choose MCMC algorithm, set parameters, monitor warnings.

Checklist Continued

Evaluation

- Trace plots, R-hat, ESS, divergence analysis.

Model Comparison

- WAIC, LOO, Bayes Factor, posterior predictive checks.

Tuning and Validation

- Adjust priors, validate on test set.

Documentation

- Assumptions, interpretation, reproducibility.

Conclusion

- PyMC enables flexible, interpretable Bayesian inference.
- It integrates well with modern data science workflows.
- Combine it with ArviZ for diagnostics and visualization.
- Great for hierarchical models, decision making, and uncertainty quantification.