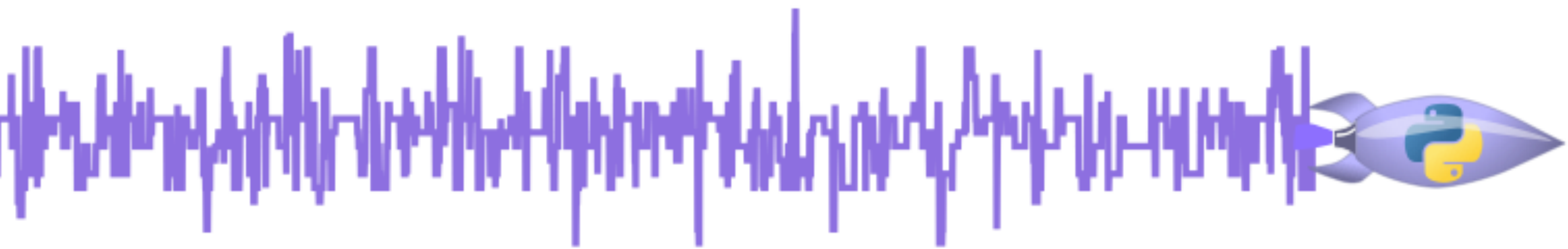
Bayesian Inference in Python



authors: J.W. Richards, J. S. Bloom

Intro to Bayesian Statistics

The goal of Bayesian inference is to produce *posterior probability distributions* on **parameters** of interest

Intro to Bayesian Statistics

The goal of Bayesian inference is to produce *posterior probability distributions* on **parameters** of interest

- If we knew these posterior distributions, estimating the values (and uncertainties on) the parameters would be trivial

Intro to Bayesian Statistics

The goal of Bayesian inference is to produce *posterior probability distributions* on **parameters** of interest

- If we knew these posterior distributions, estimating the values (and uncertainties on) the parameters would be trivial
- Our main tool to find the posterior is **Bayes' Theorem**

$$\underbrace{p(\boldsymbol{\theta}|D,I)}_{\text{'posterior'}} \propto \underbrace{p(D|\boldsymbol{\theta},I)}_{\text{'likelihood'}} \underbrace{p(\boldsymbol{\theta}|I)}_{\text{'prior'}}$$

Intro to Bayesian Statistics

The goal of Bayesian inference is to produce *posterior probability distributions* on **parameters** of interest

- If we knew these posterior distributions, estimating the values (and uncertainties on) the parameters would be trivial
- Our main tool to find the posterior is **Bayes' Theorem**

$$\underbrace{p(\boldsymbol{\theta}|D,I)}_{\text{'posterior'}} \propto \underbrace{p(D|\boldsymbol{\theta},I)}_{\text{'likelihood'}}\,\underbrace{p(\boldsymbol{\theta}|I)}_{\text{'prior'}}$$

- Contrast this to frequentist (classical) statistics, in which the parameters are treated as fixed (and the data as random) and inferences are made via hypothetical repeated experiments

Finding the Posterior Distribution

Conjugate Prior - Simplest case: assume prior distribution that allows a closed-form posterior distribution for the chosen likelihood

Finding the Posterior Distribution

Conjugate Prior - Simplest case: assume prior distribution that allows a closed-form posterior distribution for the chosen likelihood

Example: Normal likelihood with known variance σ^2

$$x_i|\mu \sim N(\mu, \sigma^2) \quad \text{Likelihood}$$

$$\mu \sim N(\mu_0, \sigma_0^2) \quad \text{Prior (conjugate)}$$

$$\mu|x_1, \ldots, x_n \sim N\left(\frac{\frac{\mu_0}{\sigma_0^2} + \frac{\sum_i x_i}{\sigma^2}}{\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}}, \left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}\right)^{-1}\right)$$

Posterior (closed form)

Finding the Posterior Distribution

For more complex models in realistic settings, conjugate priors do not exist

In this setting, exact computation of the posterior is impossible, and ***Markov chain Monte Carlo (MCMC)*** methods are typically used to sample from the posterior

Finding the Posterior Distribution

For more complex models in realistic settings, conjugate priors do not exist

In this setting, exact computation of the posterior is impossible, and ***Markov chain Monte Carlo (MCMC)*** methods are typically used to sample from the posterior

- Gibbs sampling - Alternate between sampling some parameters given all others plus data, until convergence. Requires conjugate sub-models (i.e. closed form for posterior of each subset of parameters given the rest).

Finding the Posterior Distribution

For more complex models in realistic settings

for parameters $\theta_1, ..., \theta_K$ and data $x_1, ..., x_n$
Gibbs' Sampling alternates drawing from

$$p(\theta_1, ..., \theta_p \,|\, \theta_{p+1}, ..., \theta_K, x_1, ..., x_n) \qquad \text{and}$$
$$p(\theta_{p+1}, ..., \theta_K \,|\, \theta_1, ..., \theta_p, x_1, ..., x_n)$$

- Gibbs sampling - Alternate between sampling some parameters given all others plus data, until convergence. Requires conjugate sub-models (i.e. closed form for posterior of each subset of parameters given the rest).

Finding the Posterior Distribution

For more complex models in realistic settings, conjugate priors do not exist

In this setting, exact computation of the posterior is impossible, and ***Markov chain Monte Carlo (MCMC)*** methods are typically used to sample from the posterior

- Gibbs sampling - Alternate between sampling some parameters given all others plus data, until convergence. Requires conjugate sub-models (i.e. closed form for posterior of each subset of parameters given the rest).

Finding the Posterior Distribution

For more complex models in realistic settings, conjugate priors do not exist

In this setting, exact computation of the posterior is impossible, and ***Markov chain Monte Carlo (MCMC)*** methods are typically used to sample from the posterior

- Gibbs sampling - Alternate between sampling some parameters given all others plus data, until convergence. Requires conjugate sub-models (i.e. closed form for posterior of each subset of parameters given the rest).
- Metropolis-Hastings - Generates a random walk using a proposal density and a method for rejecting proposed moves.

MCMC with PyMC

http://pymc-devs.github.com/pymc/

PyMC is the most widely used Markov chain Monte Carlo module in Python

- It allows straightforward coding of probability models and posterior sampling of those models with standard (optimized) MCMC algorithms
- Large and complicated (hierarchical) models can be easily coded in PyMC
- Convergence diagnostics and automatic tuning are provided
- Users can input custom probability distributions and fitting algorithms
- Great documentation

MCMC with PyMC

http://pymc-devs.github.com/pymc/

PyMC is the most widely used Markov chain Monte Carlo module in Python

- It allows straightforward coding of probability models and posterior sampling of those models with standard (optimized) MCMC algorithms
- Large and complicated (hierarchical) models can be easily coded in PyMC
- Convergence diagnostics and automatic tuning are provided
- Users can input custom probability distributions and fitting algorithms
- Great documentation

Let's Build a Model with PyMC

Example Statistical Model

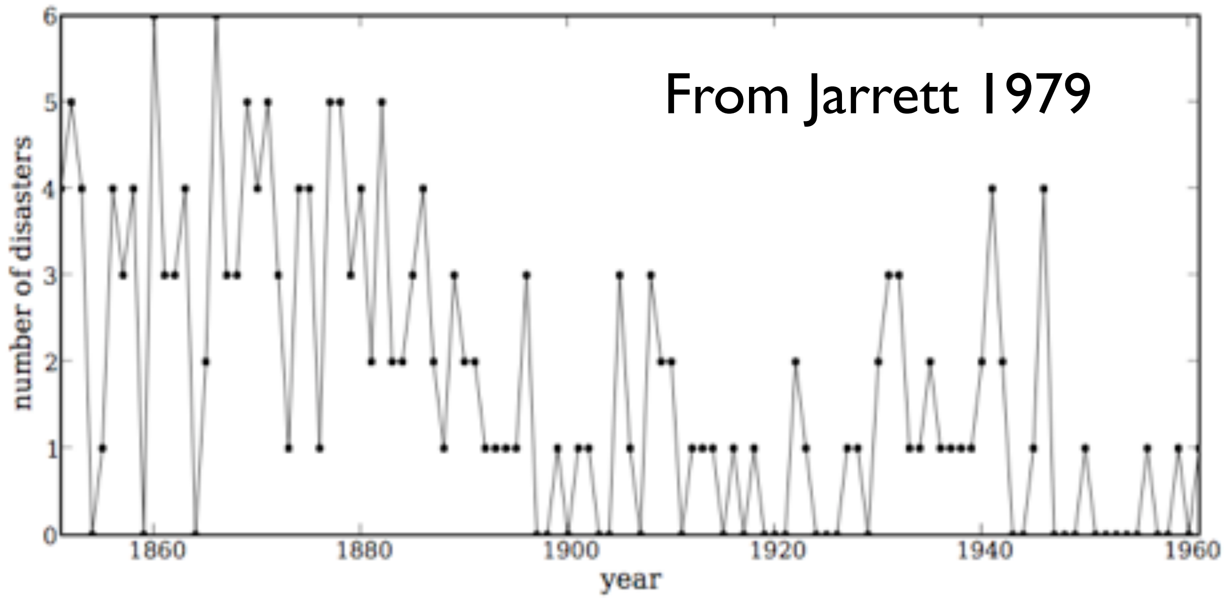


Figure 1: Recorded coal mining disasters in the UK.

Goal - Use data from coal mining disasters to estimate change point in rate of disasters (time and magnitude)

Example Statistical Model

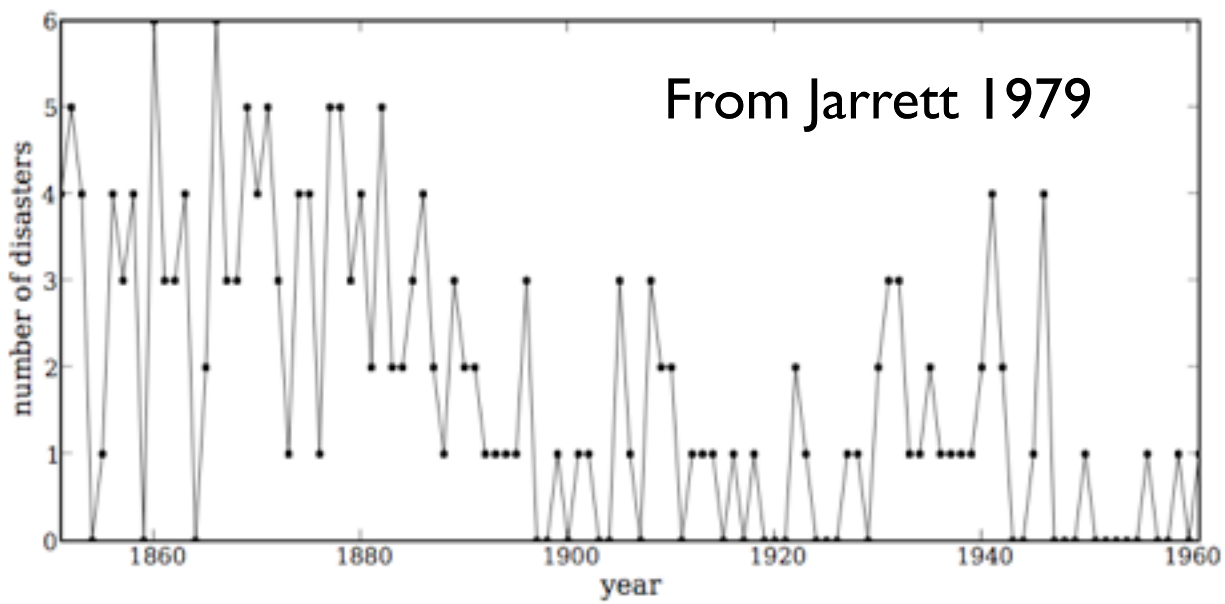


Figure 1: Recorded coal mining disasters in the UK.

Goal - Use data from coal mining disasters to estimate change point in rate of disasters (time and magnitude)

$$(D_t|s,e,l) \sim \text{Poisson}(r_t), \quad r_t = \begin{cases} e & \text{if} \quad t < s \\ l & \text{if} \quad t \geq s \end{cases}, \quad t \in [t_l, t_h]$$

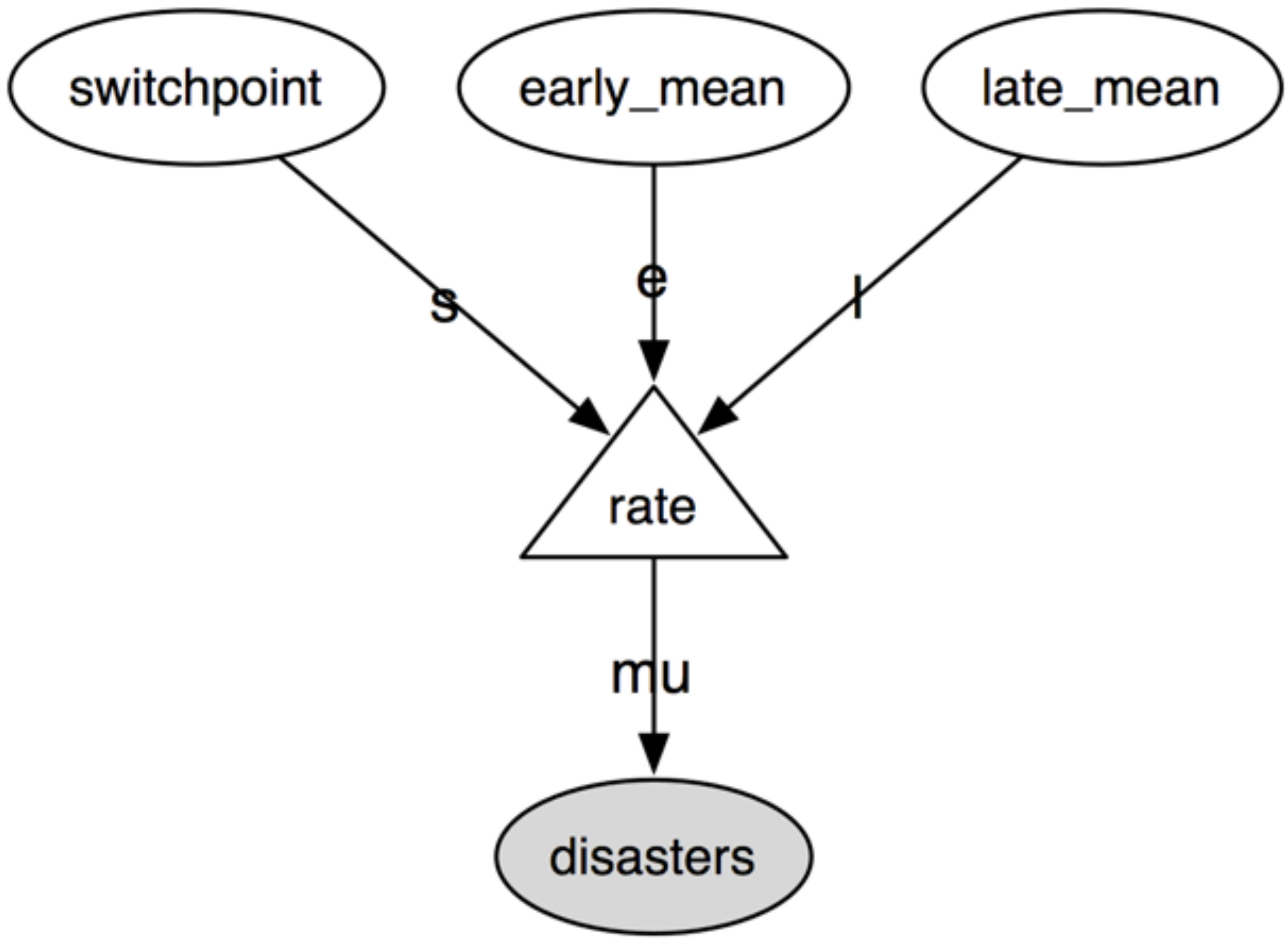
$$s \sim \text{Discrete Uniform}(t_l, t_h)$$
$$e \sim \text{Exponential}(r_e)$$
$$l \sim \text{Exponential}(r_l)$$

D_t = # of disasters in year t

r_t = disaster rate in year t

s = year of changepoint

e, l = old, new rate


switchpoint
early_mean
late_mean
s
e
l
rate
mu
disasters

Coding up a model in PyMC

file: DisasterModel.py

Coding up a model in PyMC

file: DisasterModel.py

```
import pymc
```

Coding up a model in PyMC

file: DisasterModel.py

```
import pymc

# priors on s, e, l
s = pymc.DiscreteUniform('s', lower=0, upper=110, doc='Switchpoint[year]')
e = pymc.Exponential('e', beta=1)
l = pymc.Exponential('l', beta=1)
```

Coding up a model in PyMC

file: DisasterModel.py

```
import pymc

# priors on s, e, l
s = pymc.DiscreteUniform('s', lower=0, upper=110, doc='Switchpoint[year]')
e = pymc.Exponential('e', beta=1)
l = pymc.Exponential('l', beta=1)

# r as a function of s, e, l
@pymc.deterministic(plot=False)
def r(s=s, e=e, l=l):
    """Concatenate Poisson means"""
    out = np.empty(len(disasters_array))
    out[:s] = e
    out[s:] = l
    return out
```

Coding up a model in PyMC

file: DisasterModel.py

```
import pymc

# priors on s, e, l
s = pymc.DiscreteUniform('s', lower=0, upper=110, doc='Switchpoint[year]')
e = pymc.Exponential('e', beta=1)
l = pymc.Exponential('l', beta=1)

# r as a function of s, e, l
@pymc.deterministic(plot=False)
def r(s=s, e=e, l=l):
    """Concatenate Poisson means"""
    out = np.empty(len(disasters_array))
    out[:s] = e
    out[s:] = l
    return out

# likelihood
D = pymc.Poisson('D', mu=r, value=disasters_array, observed=True)
```

Coding up a model in PyMC

file: DisasterModel.py

```
import pymc

# priors on s, e, l
s = pymc.DiscreteUniform('s', lower=0, upper=110, doc='Switchpoint[year]')
e = pymc.Exponential('e', beta=1)
l = pymc.Exponential('l', beta=1)

# r as a function of s, e, l
@pymc.deterministic(plot=False)
def r(s=s, e=e, l=l):
    """Concatenate Poisson means"""
    out = np.empty(len(disasters_array))
    out[:s] = e
    out[s:] = l
    return out

# likelihood
D = pymc.Poisson('D', mu=r, value=disasters_array, observed=True)
```

To 'fit' the model with PyMC:

```
import DisasterModel
from pymc import MCMC
M = MCMC(DisasterModel)
```

To the Notebook

Checking Convergence

Determining whether an MCMC has converged can be difficult, especially in high-dimensional parameter spaces

A number of **diagnostics** (both formal and informal) exist, and many of these are available in PyMC

First check - start multiple chains from different starting values and see that they converge to the same place
More formal methods - Raftery-Lewis, Geweke, autocorrelation, etc.
Goodness of fit - Posterior Predictive Checks which simulate data from your fitted model and compare to the observed data (checks convergence AND the suitability of the chosen model)

Other MCMC Code on the Market

WinBUGS, OpenBUGS - Bayesian inference Using Gibbs Sampling

in R -

mcmc, rbugs, BRugs, MCMCpack, adaptMCMC, etc.
See rpy2 (http://rpy.sourceforge.net/rpy2.html)

in Python -

bayesian-inference (http://code.google.com/p/bayesian-inference/)
emcee (http://dan.iel.fm/emcee/)