Lecture 22

From Bayesian Workflow to Mixture Models

Last time:

- Deviance, WAIC, Importance sampling, and LOO
- Model Comparison vs ensembling
- Bayesian Model Averaging: pseudo-BMA vs stacking
- Oceanic Tools: ensembling "regularizes" counterfactuals
- But model is over-dispersed: solve by hierarchical

Today:

Deviance

$$D(q) = -\frac{N}{2} E_p[log(q)]$$

Using empirical distribution on sample (deviance is a stochastic quantity)

$$D(q) = -2 \sum_i log(q_i),$$

Bayesian deviance

$D(q) = -\frac{N}{2} E_p[log(pp(y))]$ posterior predictive for points y on the test set or future data

replace joint posterior predictive over new points y by product of marginals (exact if using point estimate):

ELPD: $\sum_i E_p[log(pp(y_i))]$

Since we do not know the true distribution p,

replace elpd: $\sum_i E_p[log(pp(y_i))]$

by the computed "log pointwise predictive density" (lppd) **in-sample**

$$\sum_j log\left\langle p(y_j|\theta)\right\rangle = \sum_j log\left(\frac{1}{S}\sum_s p(y_j|\theta_s)\right)$$

WAIC

$$WAIC = lppd + 2p_W$$

where

$$p_W = 2\sum_i (log(E_{post}[p(y_i|\theta)] - E_{post}[log(p(y_i|\theta))])$$

Once again this can be estimated by

$$\sum_i Var_{post}[log(p(y_i|\theta))]$$

it is tempting to use information criteria to compare models with different likelihood functions. Is a Gaussian or binomial better? Can't we just let WAIC sort it out?
Unfortunately, WAIC (or any other information criterion) cannot sort it out. The problem is that deviance is part normalizing constant. The constant affects the absolute magnitude of the deviance, but it doesn't affect fit to data.

--McElreath

LOOCV

- The idea here is that you fit a model on N-1 data points, and use the Nth point as a validation point. Clearly this can be done in N ways.
- the N-point and N-1 point posteriors are likely to be quite similar, and one can sample one from the other by using importance sampling.

$$E_f[h] = \frac{\sum_s w_s h_s}{\sum_s w_s} \text{ where } w_s = f_s/g_s.$$

Fit the full posterior once. Then we have

$$w_s = \frac{p(\theta_s|y_{-i})}{p(\theta_s|y)} \propto \frac{1}{p(y_i|\theta_s, y_{-i})}$$

- the importance sampling weights can be unstable out in the tails.
- importance weights have a long right tail, pymc (`pm.loo`) fits a generalized pareto to the tail (largest 20% importance ratios) for each held out data point i (a MLE fit). This smooths out any large variations.

What should you use?

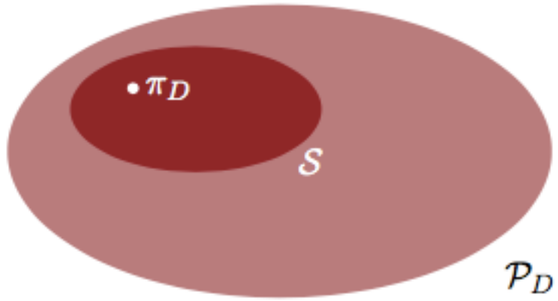
1. LOOCV and WAIC are fine. The former can be used for models not having the same likelihood, the latter can be used with models having the same likelihood.

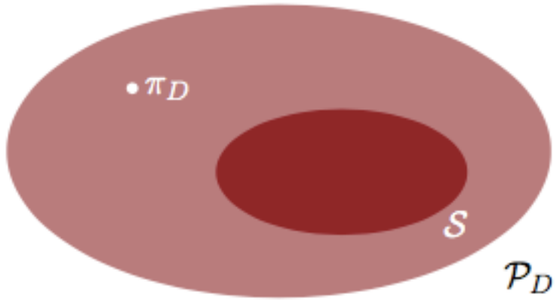
2. WAIC is fast and computationally less intensive, so for same-likelihood models (especially nested models where you are really performing feature selection), it is the first line of attack

3. One does not always have to do model selection. Sometimes just do posterior predictive checks to see how the predictions are,

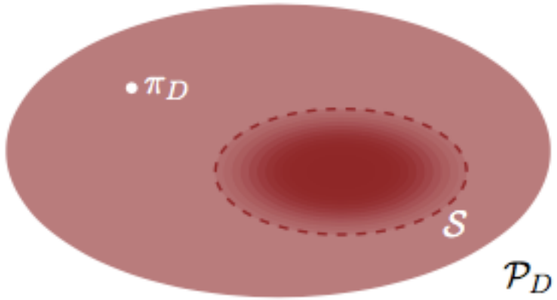
Bayesian Workflow

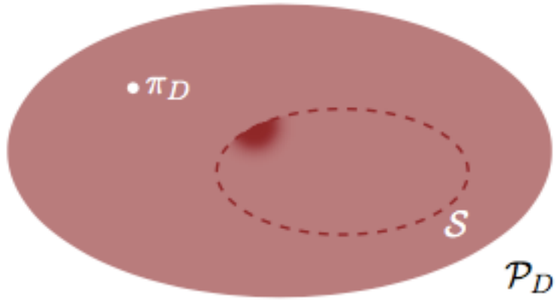
$$p(\theta|y) = \frac{p(\theta, y)}{p(y)}$$



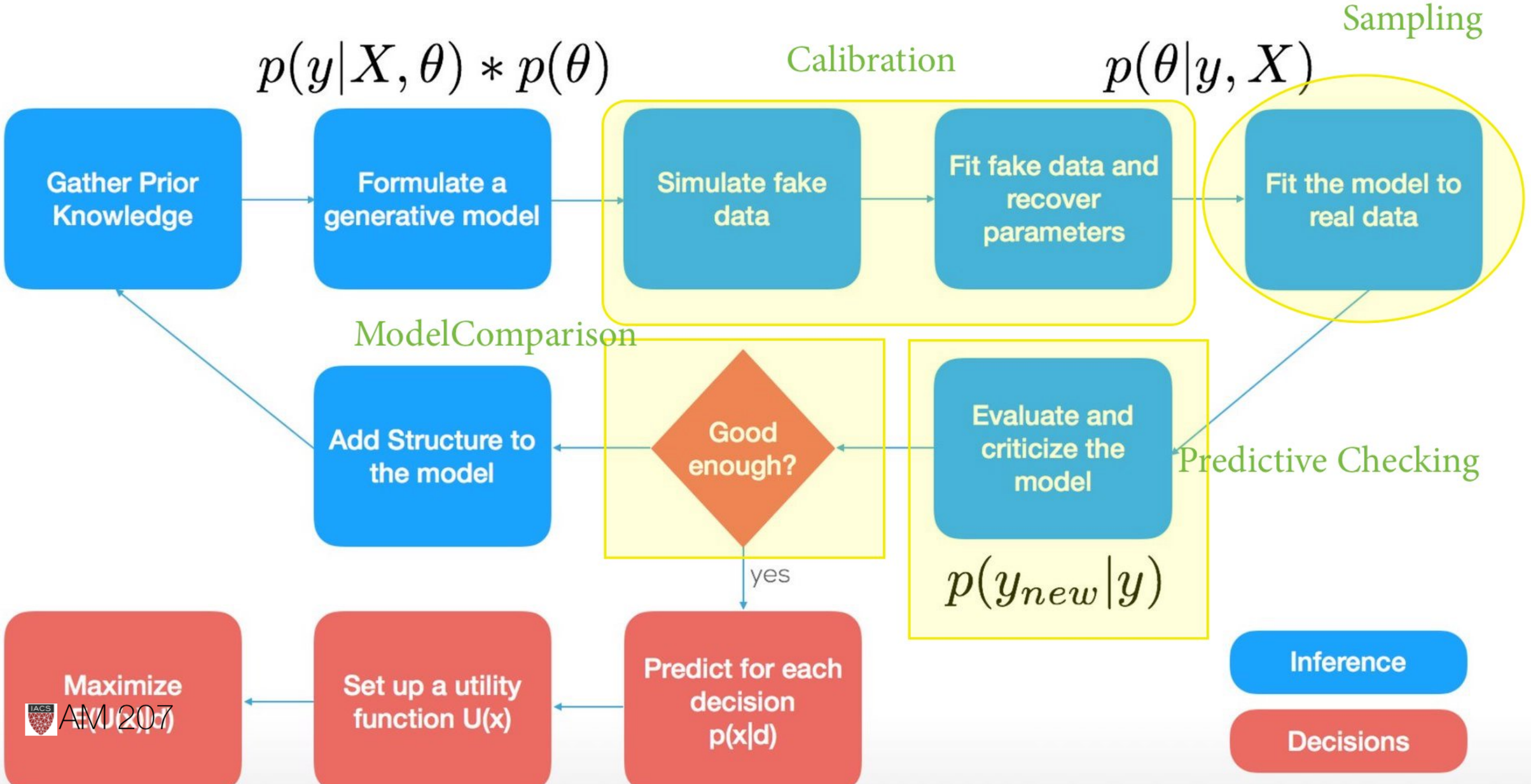


$$= \frac{p(y \mid \theta)p(\theta)}{p(y)}$$

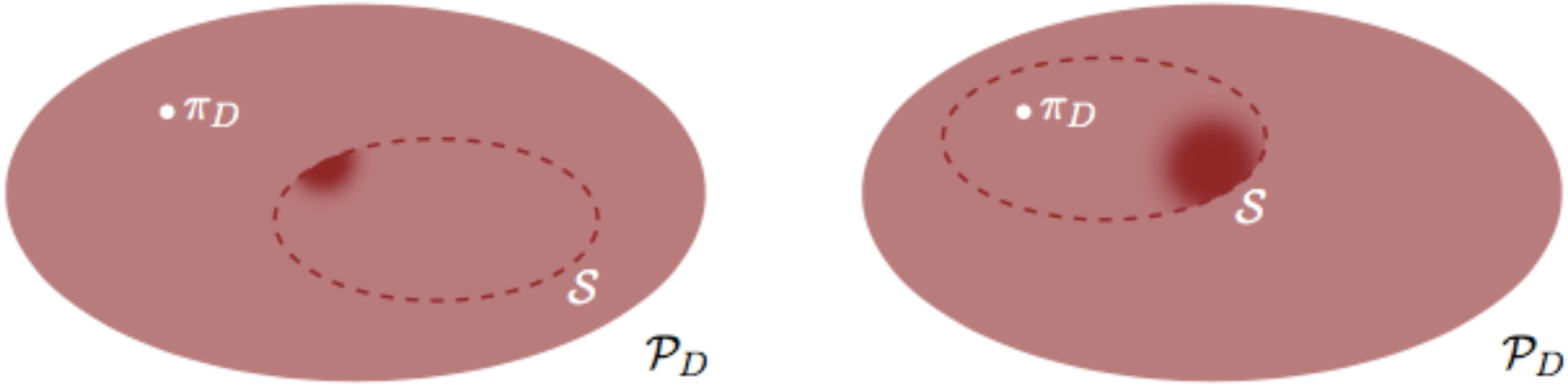




Bayesian Workflow



Think of the prior generatively AND predictively



Bias can come from a prior, but do not construct a prior to allow for overfitting (draws far away from good place). Too many heavy tails can be bad.

Model Calibration

Think about the **Bayesian Joint distribution**.

$$p(\theta, y) = p(y \mid \theta)p(\theta)$$

The prior predictive:

$$p(y) = \int d\theta \, p(\theta, y) = \int d\theta \, p(y \mid \theta)p(\theta)$$

How to choose priors?

- mild regularization
- un-informativity
- sensible parameter space
- should correspond to scales and units of process being modeled
- we should calibrate to them

Drunk Monks: prior selection

- specify $\lambda \sim HalfN(0, 4)$ instead of the crazy $N(0, e^{100})$ we had earlier
- domain knowledge: *A survey of Abbey Heads has told us, that the most a monk could produce, ever, was 10 manuscripts in a day.*
- $max(\lambda + 3\sqrt{\lambda}) < 10$, `5+3*np.sqrt(5)=11.7`
- `halfnorm.ppf(0.99, loc=0,scale=4)=10.3`

Generate Artificial data sets

- from fixed params, but even better, from priors
- $\tilde{\theta} \sim p(\theta)$
- $\tilde{y} \sim p(y \mid \tilde{\theta})$
- callibrate inferences or decisions by analysing this data
- $U(a) = \int d\tilde{\theta} d\tilde{y} p(\tilde{y}, \tilde{\theta}) U(a(\tilde{y}), \tilde{\theta})$

Now fit a posterior to each generated dataset

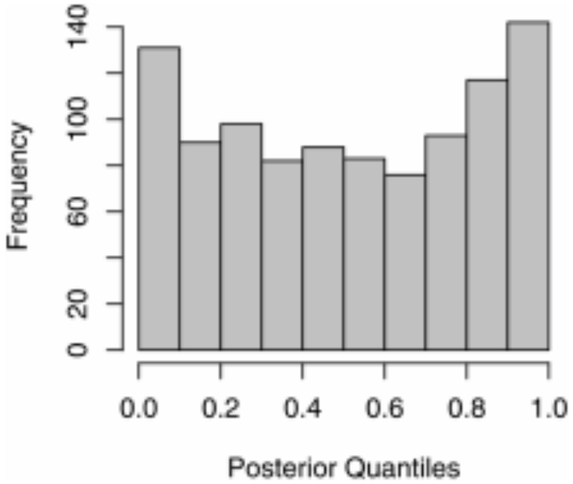
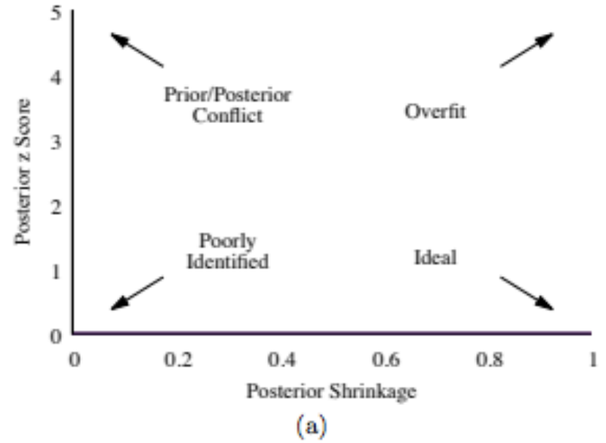


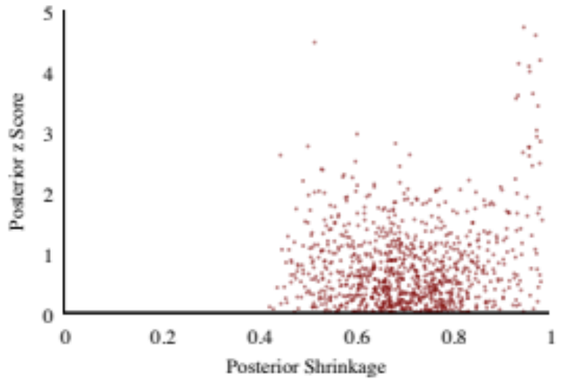
Figure 3. An example of posterior quantiles q from software with error. An effective summary for detecting the error should emphasize quantiles near 0 or 1, such as $h(q) = (\Phi^{-1}(q))^2$.

- see Cook et al
- take each $\tilde{y}$
- get a $\theta \mid \tilde{y}$ posterior
- find the rank of $\tilde{\theta}$ in "its" posterior
- a histogram of ranks should be uniform- this tests our sampling software

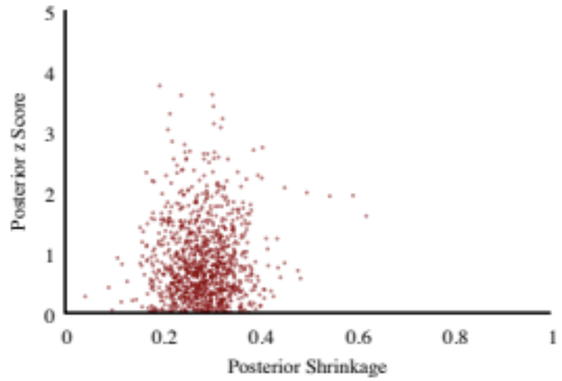
Sensitivity of posterior to range allowed by prior



(a)



(b)



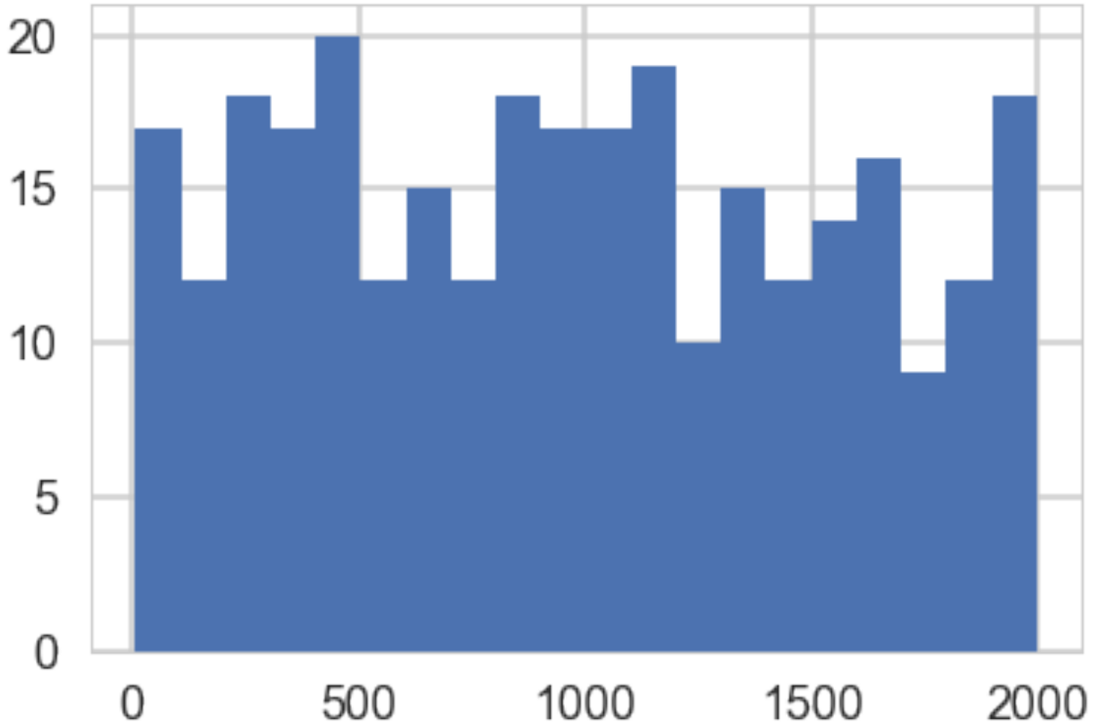
(c)

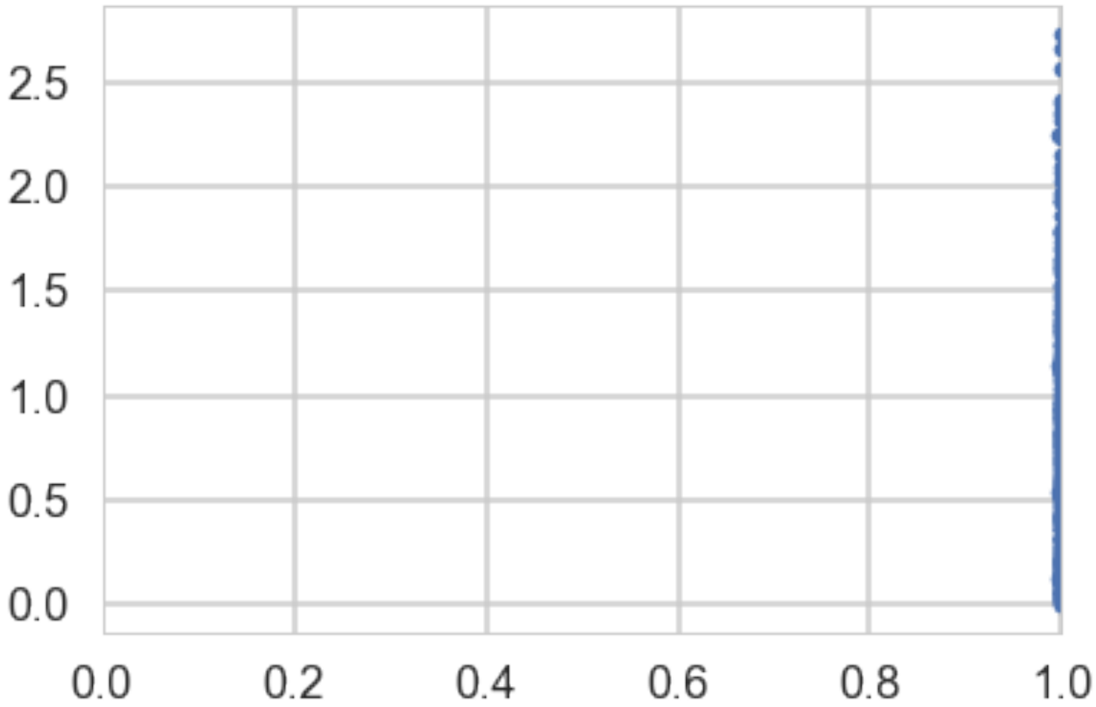
$$z_n = \left| \frac{\mu_n(\theta_n|\tilde{y}) - \tilde{\theta}_n}{\sigma_n(\theta_n|\tilde{y})} \right|$$

$$s_n = 1 - \frac{\sigma_n(\theta_n|\tilde{y})^2}{\tau_n(\tilde{y})^2}$$

where μ and σ are generated-posterior quantities and τ is a prior one, and n indexes the parameters

Drunk Monks pre-obs

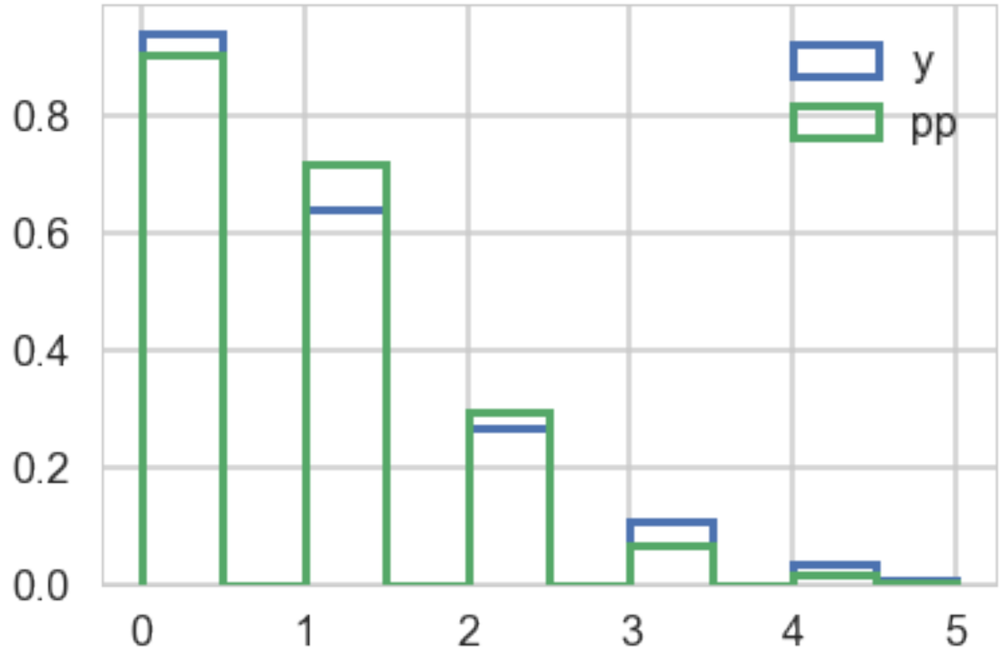




Then move to the REAL DATA posterior

- now we do posterior predictive checks
- the prior checks have specified possible data distributions that can be generated
- the posterior predictive ought to be a subset of these. If not our model is mis-specified
- this may seem strange as we didnt think priors are data generating
- they are not but are defined with respect to the likelihood

Drunk Monks, post-obs



pp check shows need for 0 inflation, so do that, rinse+repeat

The Workflow (from Betancourt, and Savage)

Prior to Observation

1. Define Data and interesting statistics
2. Build Model
3. Analyze the joint, and its data marginal (prior predictive) and its summary statistics
4. fit posteriors to simulated data to calibrate
    - check sampler diagnostics, and correlate with simulated data
    - use rank statistics to evaluate prior-posterior consistency
    - check posterior behaviors and behaviors of decisions

Posterior to Observation

1. Fit the Observed Data and Evaluate the fit
    - check sampler diagnostics, poor performance means generative model not consistent with actual data
2. Analyze the Posterior Predictive Distribution
    - do posterior predictive checks, now comparing actual data with posterior-predictive simulations
    - consider expanding the model
3. Do model comparison
    - usually within a nested model, but you might want to apply a different modeling scheme, in which case use loo
    - you might want to ensemble instead

Latent Variables

- instead of bayesian vs frequentist, think hidden vs not hidden
- key concept: full data likelihood $p(\mathbf{x}, \mathbf{z})$ vs partial data likelihood $p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$
- regression/classification "$\mathbf{z} = y$ or c" and full is supervised with partial being unsupervised
- observed variables $\mathbf{x}$ corresponding to data, and latent variables $\mathbf{z}$

From `edwardlib` docs: $p(\mathbf{x} \mid \mathbf{z})$

describes how any data $\mathbf{x}$ depend on the latent variables $\mathbf{z}$.

- **The likelihood posits a data generating process**, where the data $\mathbf{x}$ are assumed drawn from the likelihood conditioned on a particular hidden pattern described by $\mathbf{z}$.
- The *prior* $p(\mathbf{z})$ is a probability distribution that describes the latent variables present in the data. **The prior posits a generating process of the hidden structure**.

Any bayesian parameters can be considered as $\mathbf{z}$.

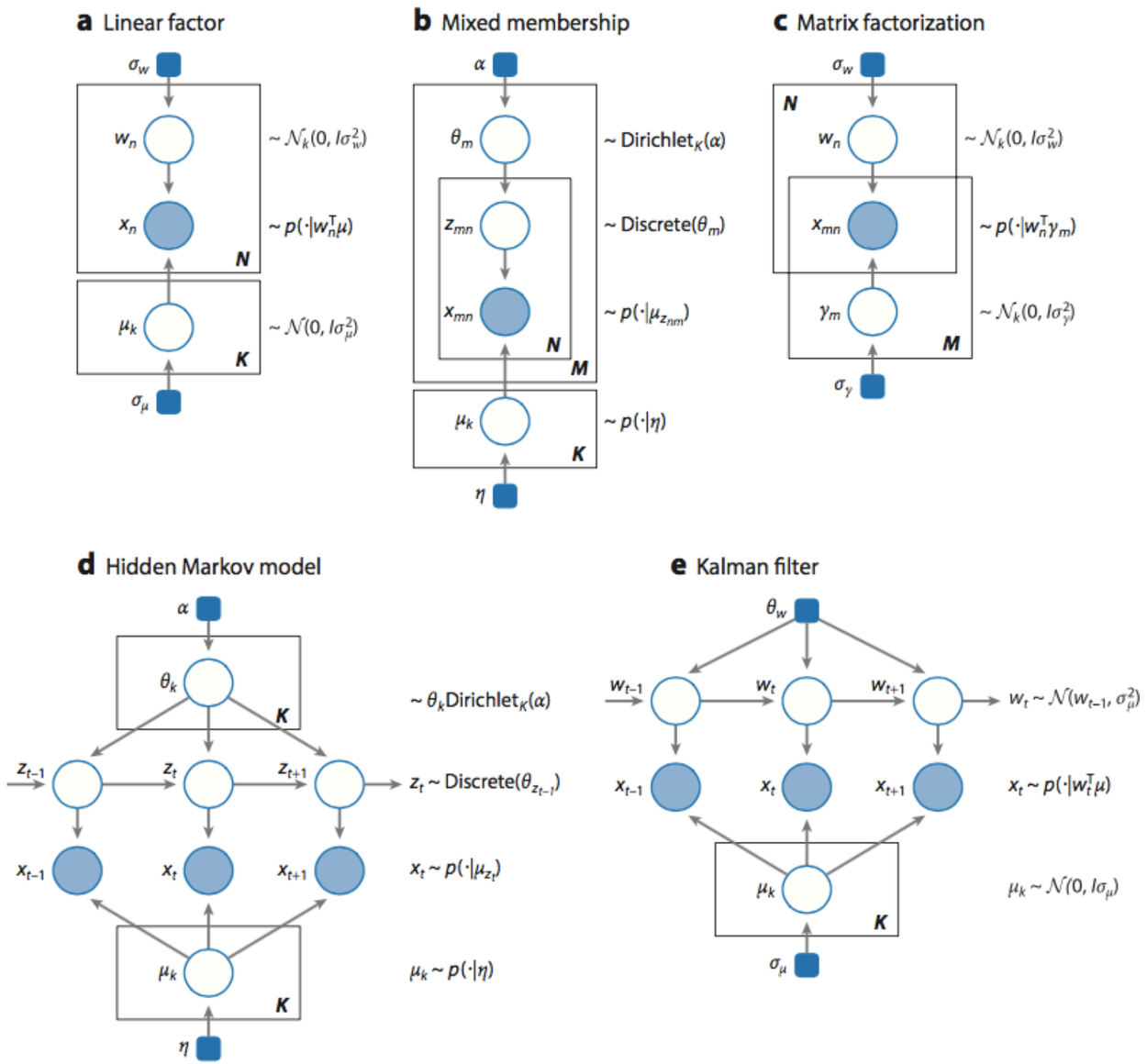
More generally posit hidden structure $\rightarrow$

e.g. **Ratings Latent Factor Model**:

$$Y_{um} = \mu + \theta_u[0] + \gamma_m[0] + r_{um} + \epsilon_{um}$$

$$r_{um} = \theta_u[1:]^\top \gamma_m[1:]$$

where $\epsilon_{um} \sim N(0, \sigma)$ and γ_m is an item-specific with first element item-specific bias and remaining latent factors for item m; θ_u is ditto for users; μ overall ratings mean and σ is residual variance of ratings.



Mixture Models

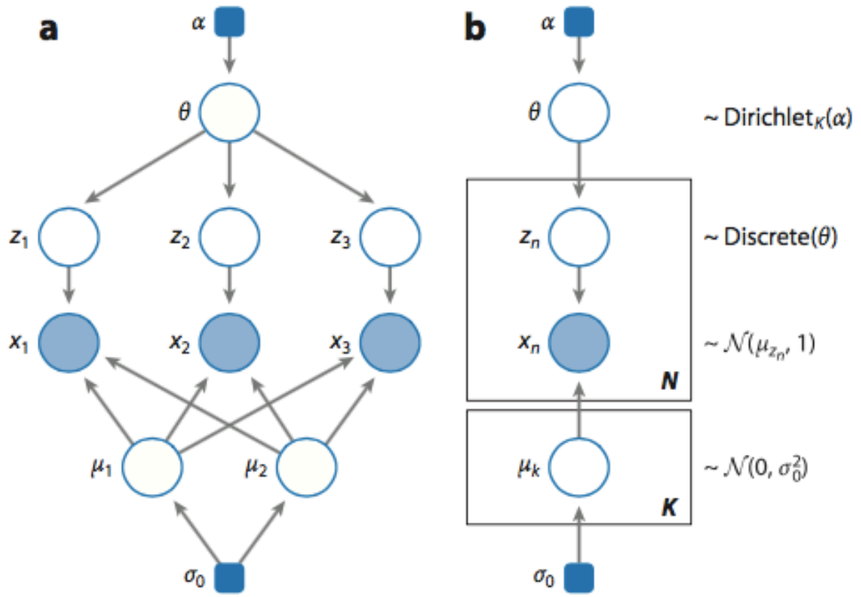


Figure 3

(*a*) A graphical model for a mixture of two Gaussians. There are three data points. The shaded nodes are observed variables, the unshaded nodes are hidden variables, and the blue square boxes are fixed hyperparameters (such as the Dirichlet parameters). (*b*) A graphical model for a mixture of K Gaussians with N data points.

A distribution $p(x|\{\theta_k\})$ is a mixture of K component distributions $p_1, p_2, \ldots p_K$ if:

$$p(x|\{\theta_k\}) = \sum_k \lambda_k p_k(x|\theta_k)$$

with the λ_k being mixing weights, $\lambda_k > 0$, $\sum_k \lambda_k = 1$.

Example: Zero Inflated Poisson

Generative model

$$p(x, z) = p(x|z)p(z)$$

Generative Model: How to simulate from it?

$$Z \sim Categorical(\lambda_1, \lambda_2, \ldots, \lambda_K)$$

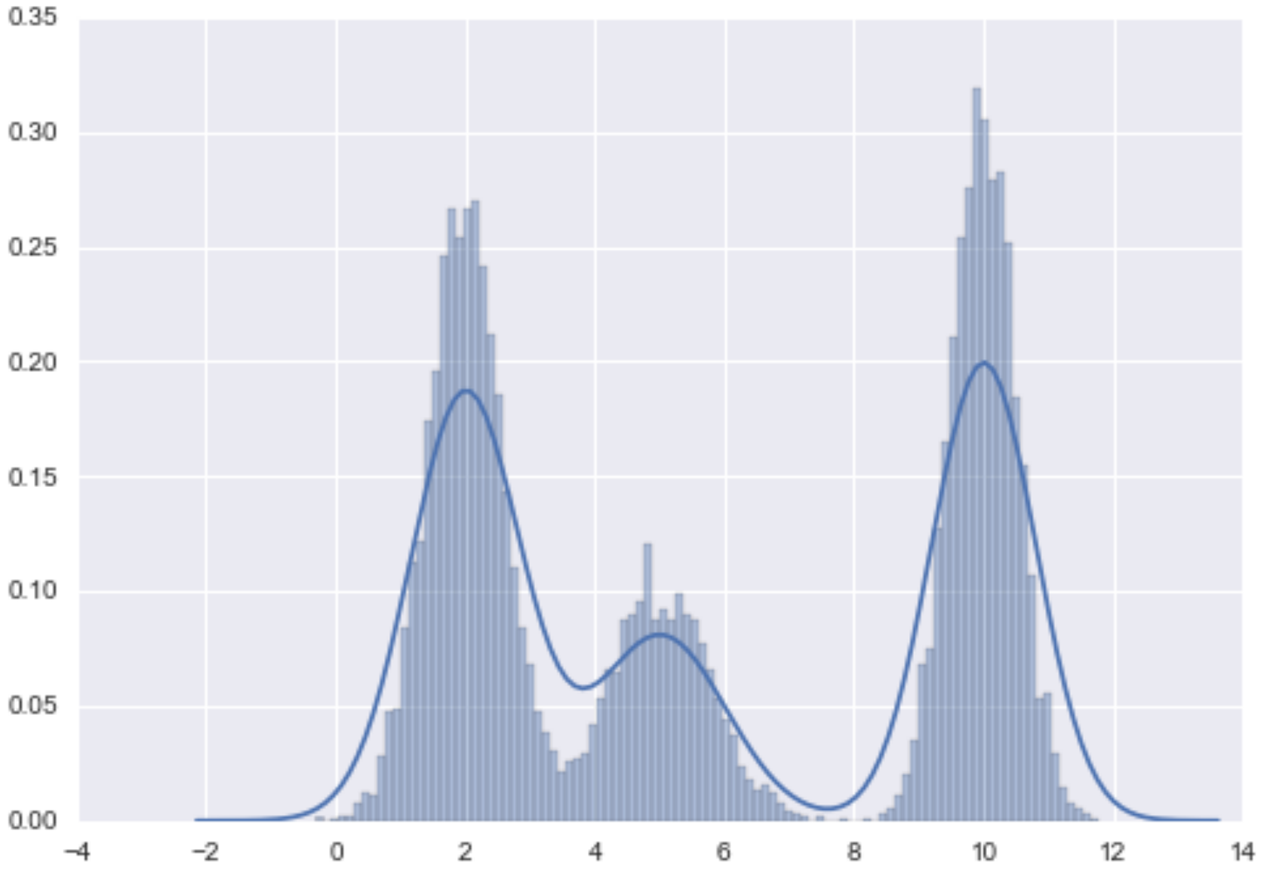
where Z says which component X is drawn from.

Thus λ_j is the probability that the hidden class variable $z = j$.

Then: $X \sim p_z(x|\theta_z)$ and general structure is:

$$p(x|\{\theta_z\}) = \sum_z p(x,z) = \sum_z p(z)p(x|z,\theta_z)\,.$$

Gaussian Mixture Model



$$p(x|\{\theta_k\}) = \sum_k \lambda_k N(x|\mu_k, \Sigma_k)$$

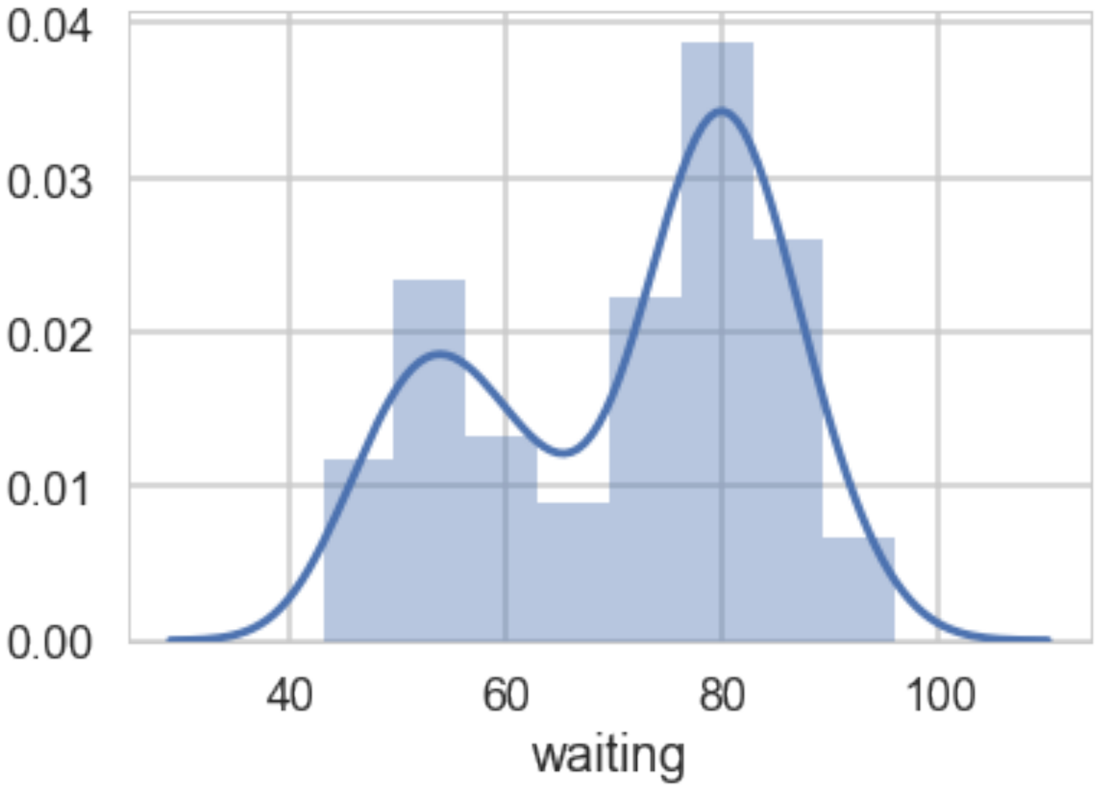
Generative:

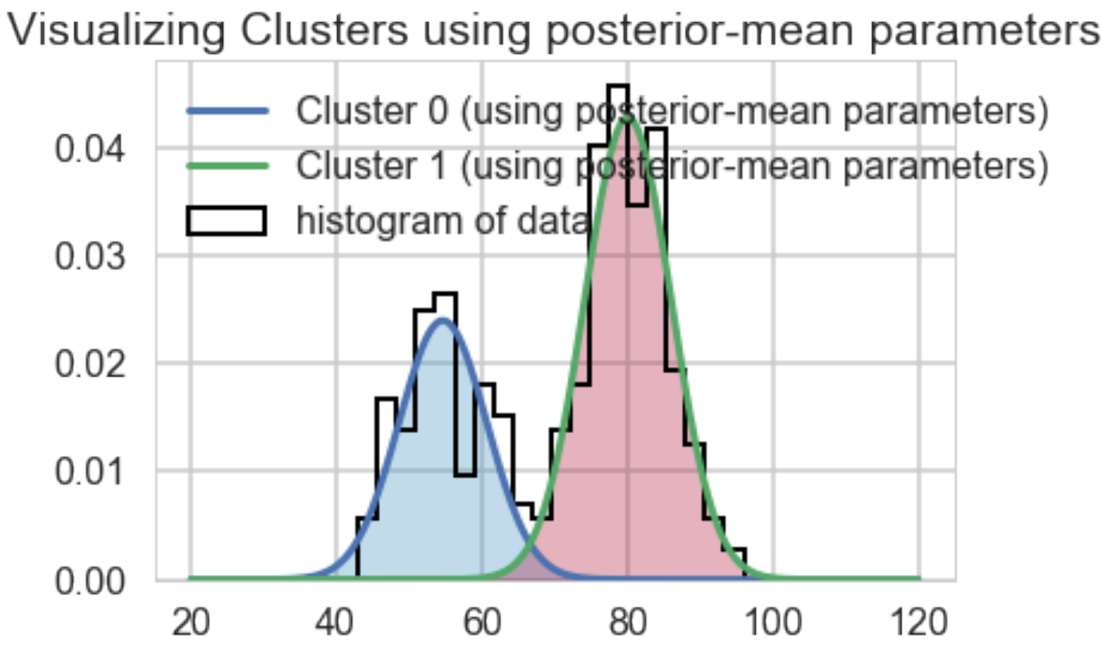
```
mu_true = np.array([2, 5, 10])
sigma_true = np.array([0.6, 0.8, 0.5])
lambda_true = np.array([.4, .2, .4])
n = 10000

# Simulate from each distribution according to mixing proportion psi
z = multinomial.rvs(1, lambda_true, size=n) #categorical
x=np.array([np.random.normal(mu_true[i.astype('bool')][0],\
    sigma_true[i.astype('bool')][0]) for i in z])
```

```
multinomial.rvs(1,[0.6,0.1, 0.3], size=10)
array([[1, 0, 0],[0, 0, 1],...[1, 0, 0],[1, 0, 0]])
```

Old faithful Geyser

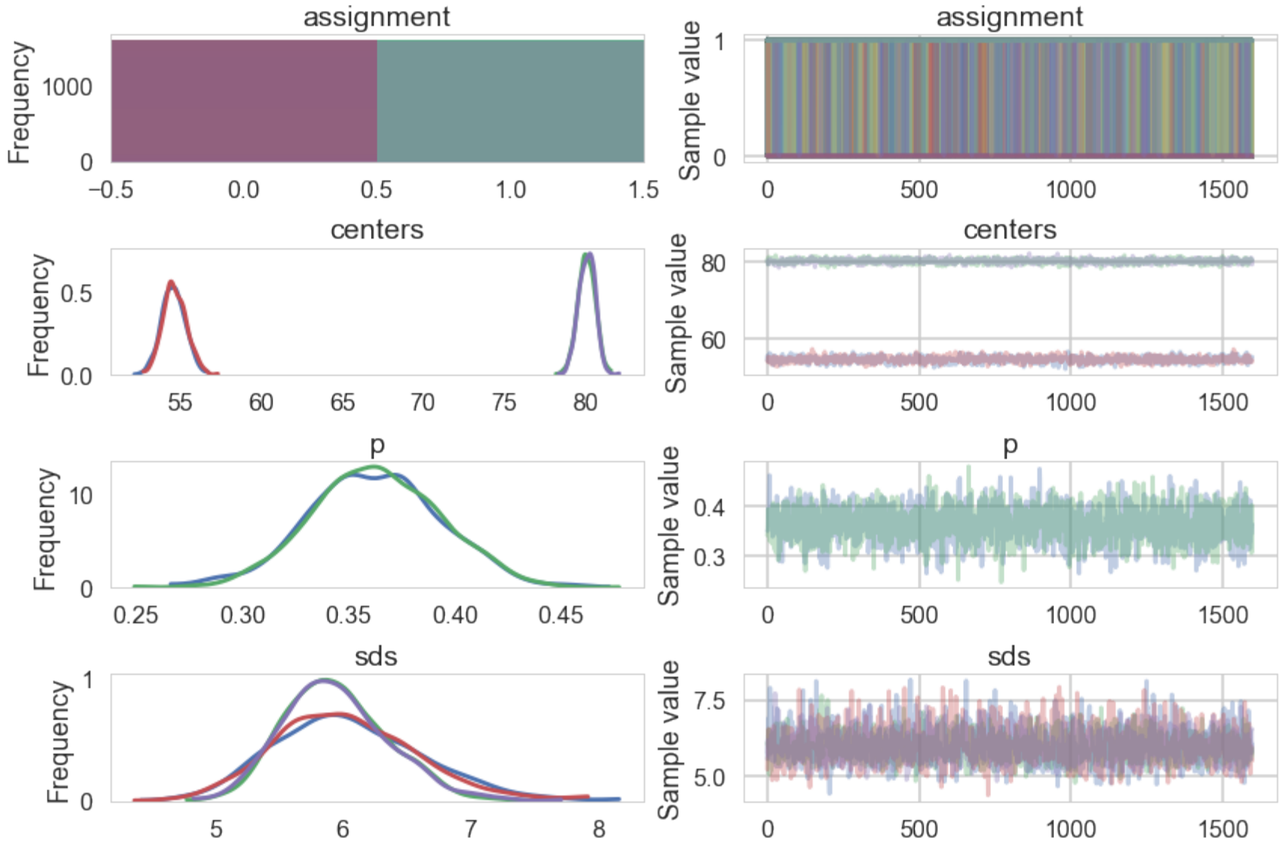




Sampling mixture models: 2 Gaussians

```
with pm.Model() as ofmodel:
    p1 = pm.Uniform('p', 0, 1)
    p2 = 1 - p1
    p = tt.stack([p1, p2])
    assignment = pm.Categorical("assignment", p,
                        shape=ofdata.shape[0])
    sds = pm.Uniform("sds", 0, 40, shape=2)
    centers = pm.Normal("centers",
            mu=np.array([50, 80]),
            sd=np.array([20, 20]),
            shape=2)

    observations = pm.Normal("obs",
        mu=centers[assignment],
        sd=sds[assignment],
        observed=ofdata.waiting)
```



Sampling mixture models: 3 Gaussians

```
with pm.Model() as mof:
    p = pm.Dirichlet('p', a=np.array([1., 1., 1.]), shape=3)
    # cluster centers
    means = pm.Normal('means', mu=[0,20,40], sd=5, shape=3)
    sds = pm.Uniform('sds', lower=0, upper=20, shape=3)
    # latent cluster of each observation
    category = pm.Categorical('category',p=p,
                              shape=data.shape[0])
    # likelihood for each observed value
    points = pm.Normal('obs', mu=means[category],
                       sd=sds[category], observed=data)
```

Generative Classifier

For a feature vector x, we use Bayes rule to express the posterior of the class-conditional as:

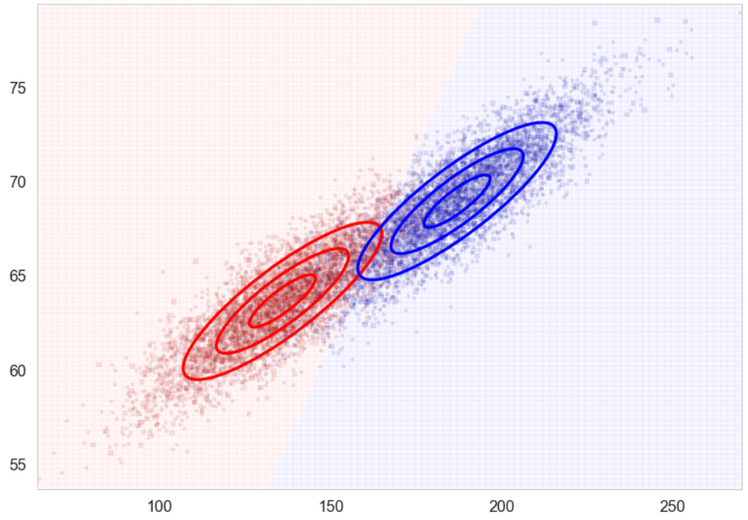
$$p(z = c|x, \theta) = \frac{p(z = c|\theta)p(x|z = c, \theta)}{\sum_{c'} p(z = c'|\theta)p(x|z = c', \theta)}$$

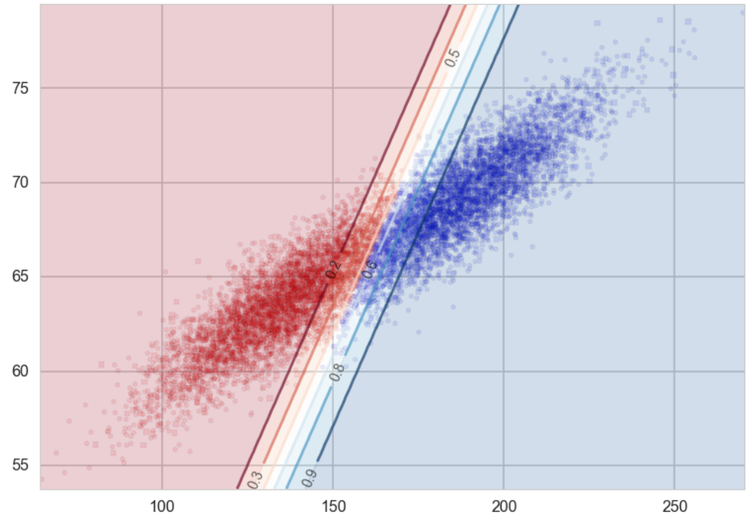
This is a **generative classifier**, since it specifies how to generate the data using the class-conditional density $p(x|z = c, \theta)$ and the class prior $p(z = c|\theta)$.

Discriminative classifier

Directly fit the class posterior, $p(z = c|x, \theta)$.

For example, a Gaussian Mixture model vs logistic regression.





Generative vs Discriminative classifiers

- LDA vs logistic respectively.
- Both have "generative" bayesian models: $p(c|x,\theta)$ or $p(y|x,\theta)$. Here think of $\mathbf{z} = \theta$
- LDA is generative as it models $p(x|c)$ while logistic models $p(c|x)$ directly. Here think of $\mathbf{z} = c$
- we do know c on the training set, so think of the unsupervised learning counterparts of these models where you dont know c

Supervised vs Unsupervised Learning

In **Supervised Learning**, Latent Variables $\mathbf{z}$ are observed.

In other words, we can write the full-data likelihood $p(\mathbf{x}, \mathbf{z})$

In **Unsupervised Learning**, Latent Variables $\mathbf{z}$ are hidden.

We can only write the observed data likelihood:

$$p(\mathbf{x}) = \sum_z p(\mathbf{x}, \mathbf{z}) = \sum_z p(\mathbf{z}) p(\mathbf{x}|\mathbf{z})$$

GMM supervised formulation

$$Z \sim \text{Bernoulli}(\lambda)$$
$$X|z=0 \sim \mathcal{N}(\mu_0, \Sigma_0),\ X|z=1 \sim \mathcal{N}(\mu_1, \Sigma_1)$$

Full-data loglike: $l(x, z|\lambda, \mu_0, \mu_1, \Sigma) = -\sum_{i=1}^{m} \log((2\pi)^{n/2}|\Sigma|^{1/2})$

$$-\frac{1}{2}\sum_{i=1}^{m}(x-\mu_{z_i})^T \Sigma^{-1}(x-\mu_{z_i}) + \sum_{i=1}^{m}[z_i \log\lambda + (1-z_i)\log(1-\lambda)]$$

Solution to MLE

$$\lambda = \frac{1}{m}\sum_{i=1}^{m} \delta_{z_i,1}$$

$$\mu_0 = \frac{\sum_{i=1}^{m} \delta_{z_i,0}\, x_i}{\sum_{i=1}^{m} \delta_{z_i,0}}$$

$$\mu_1 = \frac{\sum_{i=1}^{m} \delta_{z_i,1}\, x_i}{\sum_{i=1}^{m} \delta_{z_i,1}}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{z_i})(x_i - \mu_{z_i})^T$$

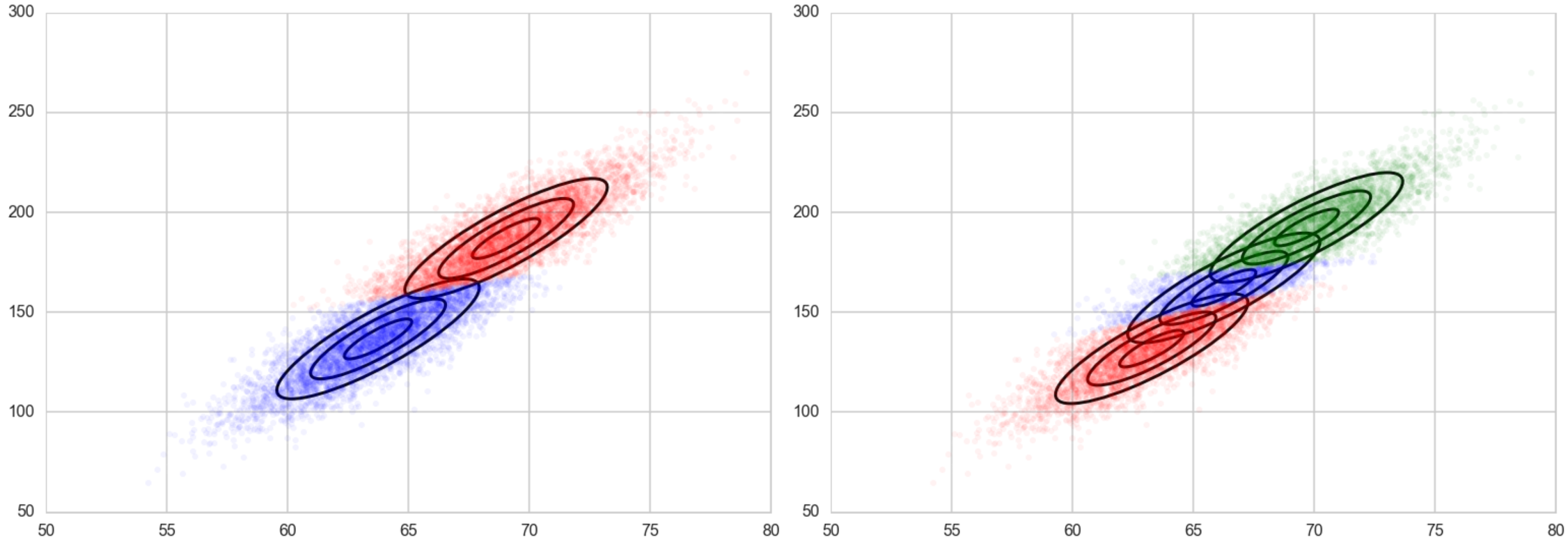
Classification

We can use the log likelihood at a given x as a classifier: assign class depending upon which probability $p(x_j|\lambda, z, \Sigma)$ is larger. (JUST x likelihood, as we want to compare probabilities at fixed zs).

$$log\, p(x_j|\lambda, z, \Sigma) = -\sum_{i=1}^{m} \log((2\pi)^{n/2}|\Sigma|^{1/2}) - \frac{1}{2}\sum_{i=1}^{m}(x-\mu_{z_i})^T\,\Sigma^{-1}(x-\mu_{z_i})$$

The first term of the likelihood does not matter since it is independent of z.

Unsupervised: How many clusters $\mathbf{z}$?



Unsupervised: Soft k-means

responsibility of cluster k for point i, and can be computed as before using Bayes rule as follows:

$$p(z_k = c|x_i, \theta) = \frac{p(z_k = c|\theta)p(x_i|z_k = c, \theta)}{\sum_{c'} p(z_k = c'|\theta)p(x_i|z_k = c', \theta)}$$

Here we never observe z_k for any samples, whereas before with the generative GDA classifier, we did observe z_k on the training set.

Concrete Formulation of unsupervised learning

Estimate Parameters by $\mathbf{x}$-MLE:

$$
\begin{aligned}
l(x|\lambda, \mu, \Sigma) &= \sum_{i=1}^{m} \log p(x_i|\lambda, \mu, \Sigma) \\
&= \sum_{i=1}^{m} \log \sum_{z} p(x_i|z_i, \mu, \Sigma)\, p(z_i|\lambda)
\end{aligned}
$$

Not Solvable analytically! EM and Variational. Or do MCMC.

Semi-supervised learning

We have some labels, but typically very few labels: not enough to form a good training set. Likelihood a combination.

$$l(\{x_i\}, \{x_j\}, \{z_i\}|\theta, \lambda) = \sum_i log\, p(x_i, z_i|\lambda, \theta) + \sum_j log\, p(x_j|\lambda, \theta)$$

$$= \sum_i log\, p(z_i|\lambda) p(x_i|z,\theta) + \sum_j log \sum_z p(z_j|\lambda) p(x_j|z_j, \theta)$$

Here i ranges over the data points where we have labels, and j over the data points where we dont.

Semi-supervised learning

Basic Idea: there is structure in $p(x)$ which might help us divine the conditionals, thus combine full-data and $\mathbf{x}$-likelihood.

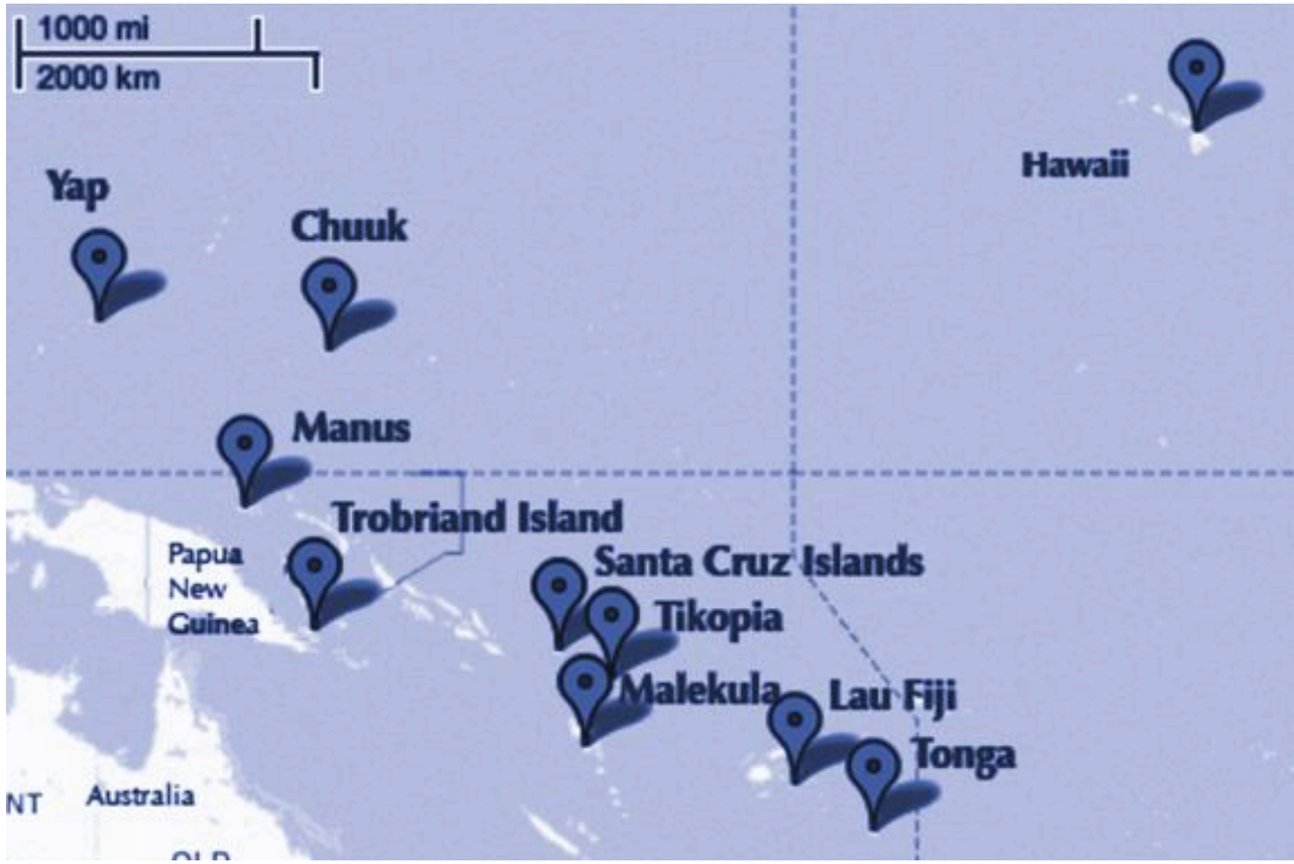
Include x on the validation set in the likelihood, and x and z on the training set in the likelihood.

Has been very useful for Naive Bayes.

Oceanic Tools

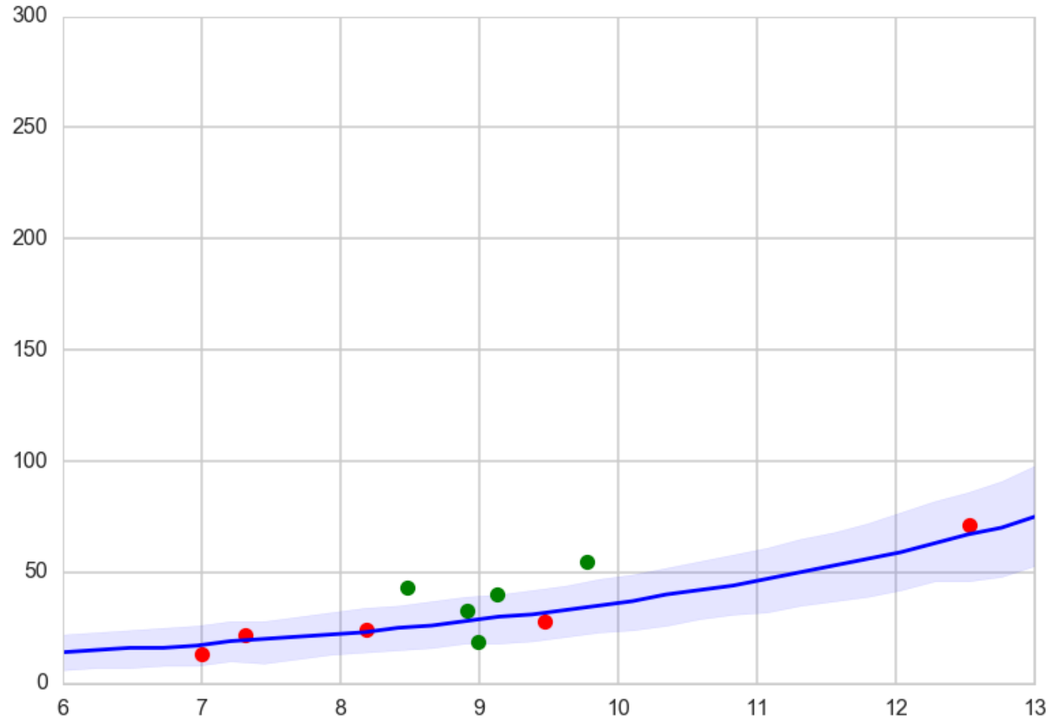
From Mcelreath:

The island societies of Oceania provide a natural experiment in technological evolution. Different historical island populations possessed tool kits of different size. These kits include fish hooks, axes, boats, hand plows, and many other types of tools. A number of theories predict that larger populations will both develop and sustain more complex tool kits. So the natural variation in population size induced by natural variation in island size in Oceania provides a natural



Overdispersion for only p

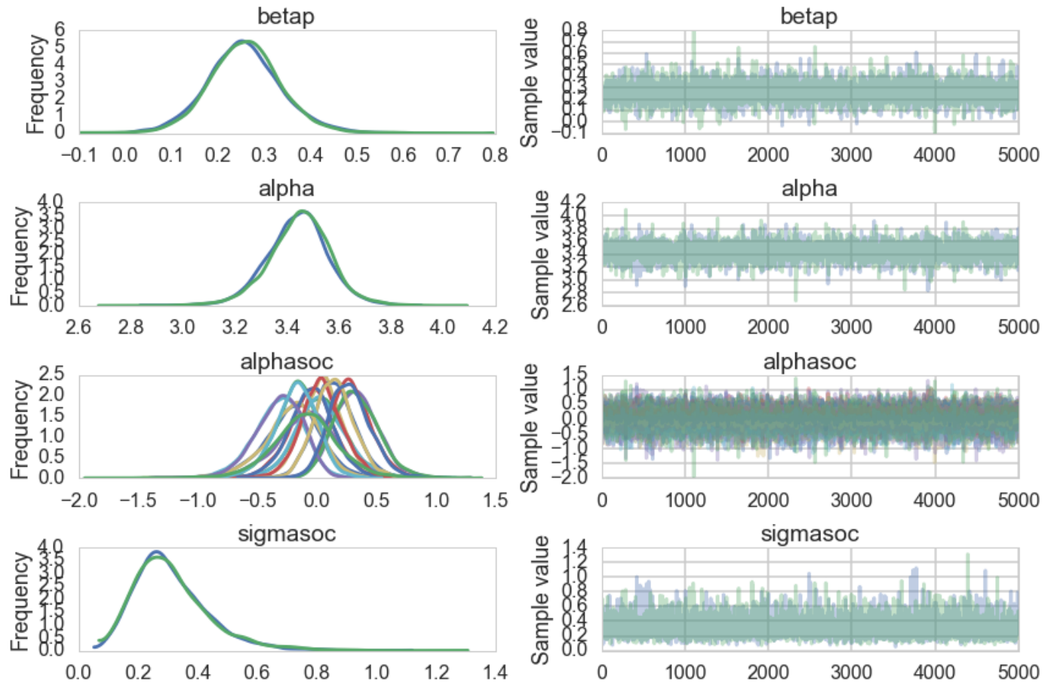
```
m2c_onlyp: loglam = alpha + betap*df.logpop_c
```

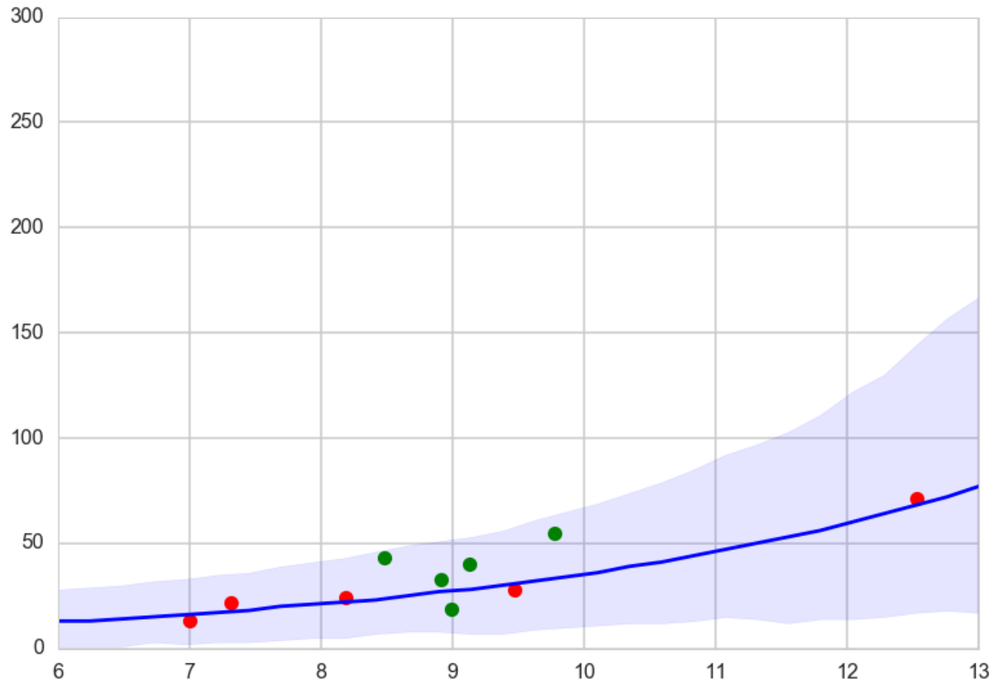


Varying hierarchical intercepts model

```
with pm.Model() as m3c:
    betap = pm.Normal("betap", 0, 1)
    alpha = pm.Normal("alpha", 0, 100)
    sigmasoc = pm.HalfCauchy("sigmasoc", 1)
    alphasoc = pm.Normal("alphasoc", 0, sigmasoc, shape=df.shape[0])
    loglam = alpha + alphasoc + betap*df.logpop_c
    y = pm.Poisson("ntools", mu=t.exp(loglam), observed=df.total_tools)
```

Hierarchical Model Posterior predictive





much wider, includes data areas

What if we model the correlation between societies based on the distance between them?

How?

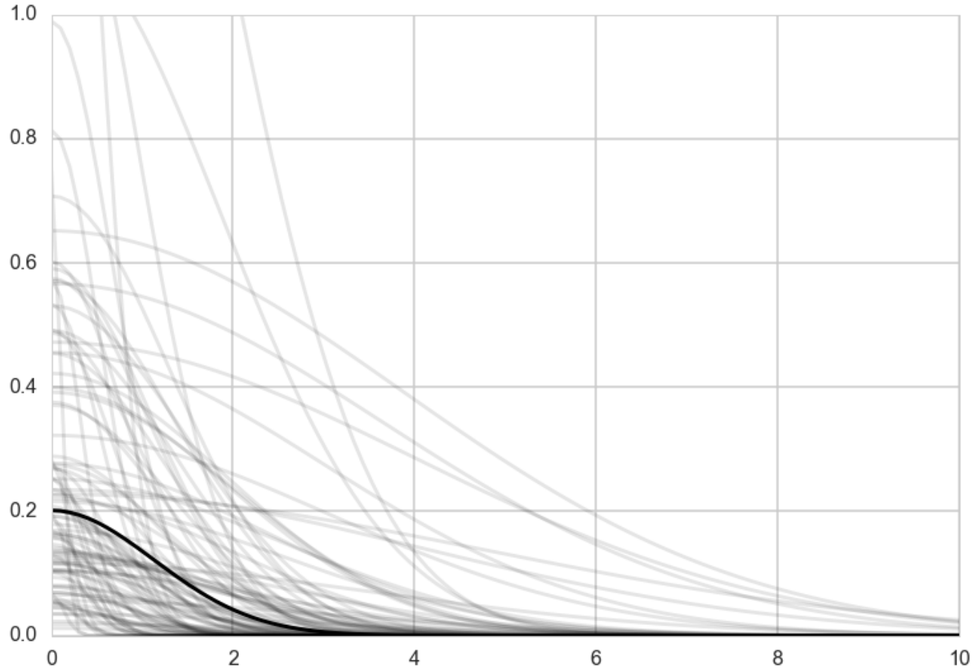
Replace independent intercepts by correlated ones.

Draw from a Multivariate Normal with a modeled covariance matrix.

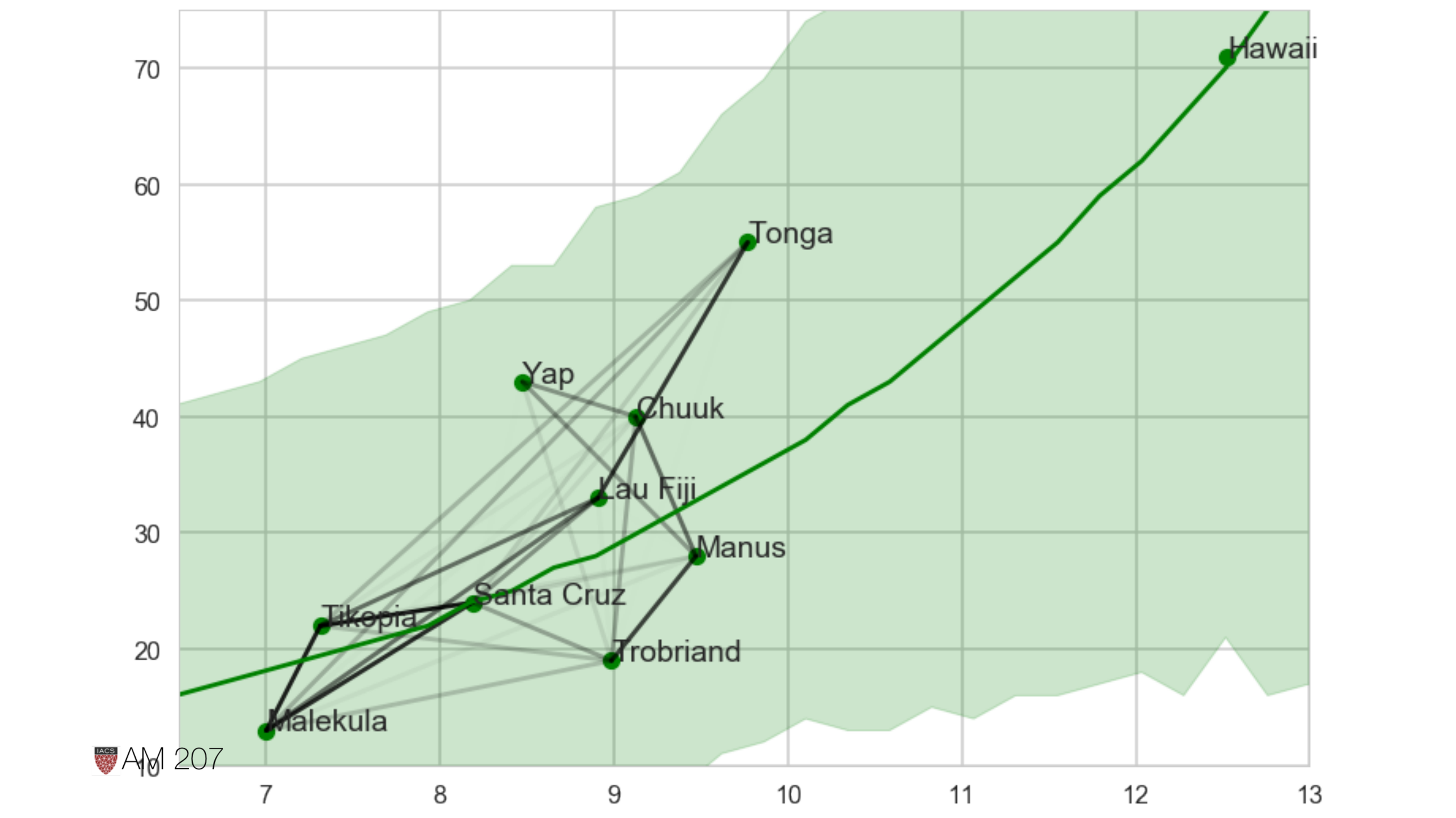
The idea sounds familiar!

We can model society specific intercepts for oceanic tools as draws from a 0 mean MVN.

Covariance posteriors:



$$
\begin{aligned}
T_i &\sim \text{Poisson}(\lambda_i) \\
\log \lambda_i &= \alpha + \gamma_{\text{SOCIETY}[i]} + \beta_P \log P_i \\
\gamma &\sim \text{MVNormal}\big((0, \ldots, 0), \mathbf{K}\big) \\
\mathbf{K}_{ij} &= \eta^2 \exp(-\rho^2 D_{ij}^2) + \delta_{ij}(0.01) \\
\alpha &\sim \text{Normal}(0, 10) \\
\beta_P &\sim \text{Normal}(0, 1) \\
\eta^2 &\sim \text{HalfCauchy}(0, 1) \\
\rho^2 &\sim \text{HalfCauchy}(0, 1)
\end{aligned}
$$


Hawaii
Tonga
Yap
Chuuk
Lau Fiji
Manus
Santa Cruz
Tikopia
Trobriand
Malekula
70
60
50
40
30
20
10
7
8
9
10
11
12
13