Bayesian Inference for ϵ-machines

Christopher C. Strelioff

Complexity Sciences Center & Department of Physics
University of California at Davis

2013.05.14

Overview

- Today
  - Goals of statistical inference
  - Introduction to Bayesian inference
  - **Ex 1:** Biased Coin
  - Unifilar HMMs and ϵ-machines
  - **Ex 2:** EvenOdd Process
    - Infer transition probabilities and start state
- Next Lecture
  - Inferring structure (model topology)
    - Enumeration and model comparison for topological ϵ-machines
  - **Ex 3:** Infer *structure* of EvenOdd process
  - **Ex 4:** Survey of inferring Golden Mean, Even, Simple Nonunifilar Source (SNS)
  - Complications: out-of-class, non-stationary processes

Goals of statistical inference

- First level–single model:
    - Given observed data D, infer parameters θ_i for assumed model M_i
    - Provide point estimate of parameters θ_i
    - Quantify uncertainty in estimate of θ_i given D and M_i
- Second level–many candidate models:
    - Compare many models for data D using candidates from a specified set of models $M_j \in \boldsymbol{\mathcal{M}}$
    - Find best model $M_j \in \boldsymbol{\mathcal{M}}$ for observed data D
    - Quantify uncertainty in model structure M_j given data D and set of models considered $\boldsymbol{\mathcal{M}}$
- Both levels:
    - Estimate functions of model parameters θ_i and quantify uncertainty in estimated value. Ex: C_μ, h_μ, etc.

Bayesian Inference

Basics at the first level

- Given finite amount of observed data D
- Assume a specific model M_i
- The model has a set of unknown parameters θ_i
- Goal is to find the posterior distribution:

$$\mathbb{P}(\theta_i | D, M_i)$$

Describes possible values for θ_i, given assumed model M_i and observed data D

  - Note: form of the posterior is affected by the chosen prior distribution $\mathbb{P}(\theta_i | M_i)$

Bayesian Inference

Elements of the Bayesian method

- **Likelihood:** $\mathbb{P}(D|\theta_i, M_i)$
  - Probability of the data D given the model M_i and its parameters θ_i
- **Prior:** $\mathbb{P}(\theta_i|M_i)$
  - Probability of the parameters θ_i given the model M_i
  - Assumptions, restrictions, 'expert knowledge' ... encoded as prior distribution over model parameters
- **Evidence:** $\mathbb{P}(D|M_i)$
  - Probability of the data D given the model M_i
  - This term is important later - model comparison
- **Posterior:** $\mathbb{P}(\theta_i|D, M_i)$
  - Probability of the model parameters θ_i given data D and the model M_i

Bayes' theorem

- Bayes' theorem connects elements of inference from previous slide

$$\mathbb{P}(\theta_i|D, M_i) = \frac{\mathbb{P}(D|\theta_i, M_i)\mathbb{P}(\theta_i|M_i)}{\mathbb{P}(D|M_i)}$$

- If θ_i is continuous, the evidence is given by

$$\mathbb{P}(D|M_i) = \int d\theta_i \, \mathbb{P}(D|\theta_i, M_i) \, \mathbb{P}(\theta_i|M_i)$$

or, if θ_i is discrete, by

$$\mathbb{P}(D|M_i) = \sum_{\theta_i} \mathbb{P}(D|\theta_i, M_i) \, \mathbb{P}(\theta_i|M_i)$$

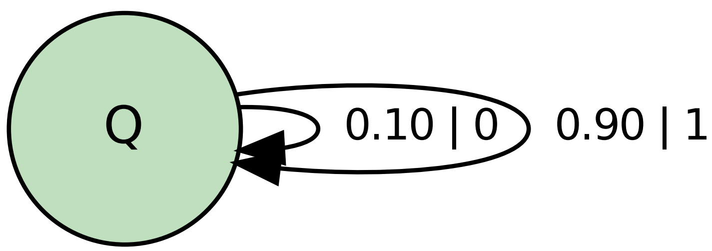
Ex 1: Biased Coin

Biased Coin

Use CMPy to generate data

```
import cmpy
import cmpy.inference.bayesianem as bayesem

# Use string to define biased coin in CMPy
bcoin_str = """Q Q 0 0.1;Q Q 1 0.9"""
bcoin = cmpy.machines.from_string(bcoin_str,
                                  name='biased coin',
                                  style=1)
# draw machine
bcoin.draw(filename='figures/bcoin.pdf', show=False)
```



Likelihood

Biased Coin

- Assume that we have observed data D, but do not know parameters of the model
- Given the Biased Coin model we just defined, the likelihood of observed data is

$$\mathbb{P}(D|\theta_i, M_i) = p(0|Q)^{n(Q0)} \, p(1|Q)^{n(Q1)}$$

where

- We assume $M_i =$ single-state, binary machine
- Unknown parameters are $\theta_i = \{p(0|Q), p(1|Q)\}$
- The data D has $n(Q0)$ zeroes and $n(Q1)$ ones
    - Note: it is useful to think of $n(Qx)$, $x \in \{0, 1\}$ as **edge counts**

Prior

Biased Coin

- The Biased Coin (Binomial/Multinomial) form has a conjugate prior– the Beta/Dirichlet Distribution
- In this case we have

$$\begin{aligned}\mathbb{P}(\theta_i|M_i) &= \frac{\Gamma(\alpha(Q))}{\prod_{x\in\{0,1\}}\Gamma(\alpha(Qx))} \\ &\times \delta\Big(1 - \sum_{x\in\{0,1\}} p(x|Q)\Big) \\ &\times \prod_{x\in\{0,1\}} p(x|Q)^{\alpha(Qx)-1}\end{aligned}$$

- We assign $\alpha(Q0), \alpha(Q1) = 1$, resulting in

$$\alpha(Q) = \sum_{x\in\{0,1\}} \alpha(Qx) = 2$$

Prior

Biased Coin prior (Beta) in CMPy

- In CMPy, get uniform prior by passing machine topology and no data to the InferEM class

```
bcoin_prior = bayesem.InferEM(bcoin)
# use to get summary: print bcoin_prior.summary_string()
```

- This results in the prior expectation (average)

$$\mathbf{E}_{\text{prior}}[p(0|Q)] = \frac{\alpha(Q0)}{\alpha(Q)} = \frac{1}{2}$$

$$\mathbf{E}_{\text{prior}}[p(1|Q)] = \frac{\alpha(Q1)}{\alpha(Q)} = \frac{1}{2}$$

Prior

Sample from prior distribution– describe uncertainty

```
from numpy import average
from scipy.stats.mstats import mquantiles
num_samples = 2000
bcoin_prior_p0 = []

# generate and store samples
for n in range(num_samples):
    # get sample and extract probability p(0|Q)
    (node,machine) = bcoin_prior.generate_sample()
    p0=machine.probability('0',start=('Q',),logs=False)
    bcoin_prior_p0.append(p0)

# calculate stats on samples -- mean and 95 credible interval
print 'From samples: E[p(0|Q)]=', average(bcoin_prior_p0),
print ' CI (%f,%f)' % tuple(mquantiles(bcoin_prior_p0,
                                       prob=[0.025,1.-0.025],
                                       alphap=1., betap=1.))
```

```
From samples: E[p(0|Q)]= 0.495323053896  CI (0.027745,0.972645)
```

Evidence

Normalization in Bayes' theorem

- The parameters θ_i are continuous, so

$$\mathbb{P}(D|M_i) = \int d\theta_i \, \mathbb{P}(D|\theta_i, M_i) \, \mathbb{P}(\theta_i|M_i)$$

- For the Biased Coin, this results in the form

$$\begin{aligned} \mathbb{P}(D|M_i) &= \frac{\Gamma(\alpha(Q))}{\prod_{x \in \{0,1\}} \Gamma(\alpha(Qx))} \\ &\times \frac{\prod_{x \in \{0,1\}} \Gamma(\alpha(Qx) + n(Qx))}{\Gamma(\alpha(Q) + n(Q))} \end{aligned}$$

where $\Gamma(n) = (n-1)!$

Posterior

Biased Coin

- Using Bayes' theorem

$$\mathbb{P}(\theta_i|D, M_i) = \frac{\mathbb{P}(D|\theta_i, M_i)\mathbb{P}(\theta_i|M_i)}{\mathbb{P}(D|M_i)}$$

  we combine the likelihood, prior and evidence to obtain the posterior

$$\begin{aligned}
\mathbb{P}(\theta_i|D, M_i) &= \frac{\Gamma(\alpha(Q) + n(Q))}{\prod_{x\in\{0,1\}} \Gamma(\alpha(Qx) + n(Qx))} \\
&\times \delta\Big(1 - \sum_{x\in\{0,1\}} p(x|Q)\Big) \\
&\times \prod_{x\in\{0,1\}} p(x|Q)^{n(Qx)+\alpha(Qx)-1}
\end{aligned}$$

- Note that this is also Beta (Dirichlet) form

Posterior

Biased Coin posterior (Beta) in CMPy

- In CMPy, get posterior (with uniform prior) by passing machine topology and data to the InferEM class

```
bcoin_data = bcoin.symbols(200)
bcoin_posterior = bayesem.InferEM(bcoin, bcoin_data)
# use to get summary: print bcoin_posterior.summary_string()
```

- This results in the posterior expectation (average)

$$
\begin{aligned}
\mathbf{E}_{\text{post}}[p(0|Q)] &= \frac{\alpha(Q0) + n(Q0)}{\alpha(Q) + n(Q)} \\
\mathbf{E}_{\text{post}}[p(1|Q)] &= \frac{\alpha(Q1) + n(Q1)}{\alpha(Q) + n(Q)}
\end{aligned}
$$

Posterior

Sample from posterior– describe uncertainty

```
from numpy import average
from scipy.stats.mstats import mquantiles
num_samples = 2000
bcoin_posterior_p0 = []

# generate and store samples
for n in range(num_samples):
    # get sample and extract probability p(0|Q)
    (node,machine) = bcoin_posterior.generate_sample()
    p0=machine.probability('0',start=('Q',),logs=False)
    bcoin_posterior_p0.append(p0)

# calculate stats on samples -- mean and 95 credible interval
print 'From samples: E[p(0|Q)]=', average(bcoin_posterior_p0),
print ' CI (%f,%f)' % tuple(mquantiles(bcoin_posterior_p0,
                                       prob=[0.025,1.-0.025],
                                       alphap=1., betap=1.))
```

```
From samples: E[p(0|Q)]= 0.103942031805  CI (0.066571,0.150929)
```

Prior vs Posterior

Plot samples

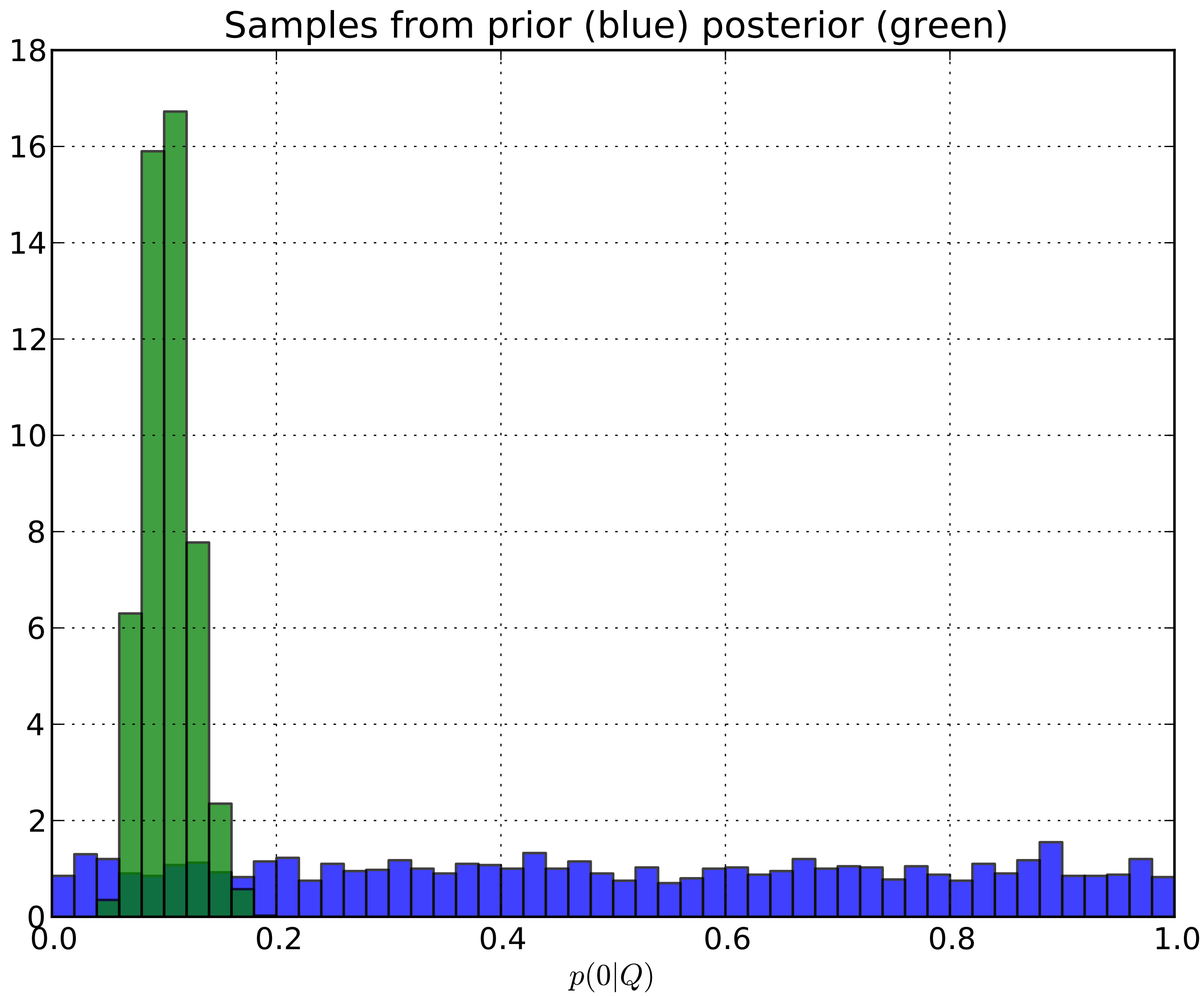
```
import pylab as plt
# prior -- blue
n, bins, patches = plt.hist(bcoin_prior_p0, 50, range=[0.0,1.0],
                            normed=1, facecolor='blue', alpha=0.75,
                            cumulative=False)

# posterior -- green
n, bins, patches = plt.hist(bcoin_posterior_p0, 50, range=[0.0,1.0],
                            normed=1, facecolor='green', alpha=0.75,
                            cumulative=False)

plt.xlabel(r'$p(0 \vert Q)$')
plt.title('Samples from prior (blue) posterior (green)')
plt.grid(True)
plt.savefig('figures/bcoin_p0_hist.pdf')
```

Prior vs Posterior

Plot samples



Estimate Functions of model parameters

Entropy rate

- As we've seen, the values of model parameters are uncertain given finite data
- As a result, functions of the model parameters are also uncertain
  - Must find mean of function
  - Good idea to quantify uncertainty as well
- Our example function will be h_μ, using sampling from the prior (or posterior)

$$\begin{aligned} \theta_i^* &\sim \mathbb{P}(\theta_i | M_i) \quad \text{sample from prior} \\ h_\mu^* &= h_\mu(\theta_i^*) \quad \text{evaluate function} \end{aligned}$$

Prior and h_μ

Sample from prior– describe uncertainty

```
num_samples = 2000
bcoin_prior_hmu = []

# generate and store samples
for n in range(num_samples):
    # get sample and extract hmu
    (node,machine) = bcoin_prior.generate_sample()
    hmu = machine.entropy_rate()
    bcoin_prior_hmu.append(hmu)

# calculate stats on samples -- mean and 95 credible interval
print 'From samples: E[hmu]=', average(bcoin_prior_hmu),
print ' CI (%f,%f)' % tuple(mquantiles(bcoin_prior_hmu,
                                       prob=[0.025,1.-0.025],
                                       alphap=1., betap=1.))
```

```
From samples: E[hmu]= 0.723794978157  CI (0.078170,0.999480)
```

Posterior and h_μ

Sample from posterior– describe uncertainty

```
num_samples = 2000
bcoin_posterior_hmu = []

# generate and store samples
for n in range(num_samples):
    # get sample and extract hmu
    (node,machine) = bcoin_posterior.generate_sample()
    hmu = machine.entropy_rate()
    bcoin_posterior_hmu.append(hmu)

# calculate stats on samples -- mean and 95 credible interval
print 'From samples: E[hmu]=', average(bcoin_posterior_hmu),
print ' CI (%f,%f)' % tuple(mquantiles(bcoin_posterior_hmu,
                                       prob=[0.025,1.-0.025],
                                       alphap=1., betap=1.))
```

```
From samples: E[hmu]= 0.478269567573  CI (0.341016,0.608264)
```

Prior vs Posterior, h_μ

Plot samples

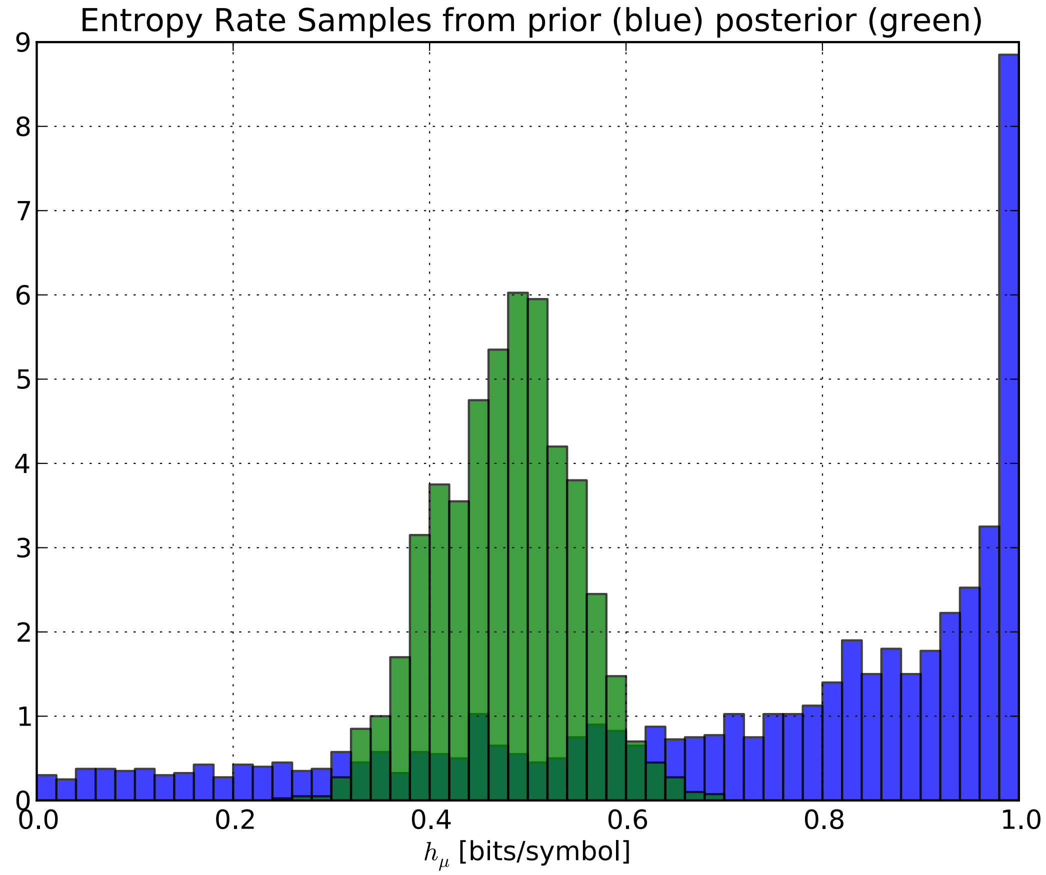
```
plt.clf()
# prior hmu -- blue
n, bins, patches = plt.hist(bcoin_prior_hmu, 50, range=[0.0,1.0],
                            normed=1, facecolor='blue', alpha=0.75,
                            cumulative=False)

# posterior hmu -- green
n, bins, patches = plt.hist(bcoin_posterior_hmu, 50, range=[0.0,1.0],
                            normed=1, facecolor='green', alpha=0.75,
                            cumulative=False)

plt.xlabel(r'$h_{\mu}$ [bits/symbol]')
plt.title('Entropy Rate Samples from prior (blue) posterior (green)')
plt.grid(True)
plt.savefig('figures/bcoin_hmu_hist.pdf')
```

Prior vs Posterior, h_μ

Plot samples




```
print 'true hmu: ', bcoin.entropy_rate(), ' [bits/symbol]'
```

```
true hmu:  0.468995593589  [bits/symbol]
```

Hidden Markov Models

Finite-state, edge-labeled HMMs

Definition

A finite-state, edge-labeled, hidden Markov model (HMM) consists of:

1. *A finite set of hidden states* $\boldsymbol{\mathcal{S}} = \{\sigma_1, \ldots, \sigma_n\}$
2. *A finite output alphabet* $\mathcal{X}$
3. *A set of* $N \times N$ *symbol-labeled transition matrices* $T^{(x)}$*,* $x \in \mathcal{X}$*, where* $T^{(x)}_{i,j}$ *is the probability of transitioning from state* σ_i *to state* σ_j *on symbol* x*. The corresponding overall state-to-state transition matrix is denoted* $T = \sum_{x \in \mathcal{X}} T^{(x)}$*.*

Finite-state ϵ-machine

Definition

A finite-state ϵ-machine is a finite-state, edge-labeled, hidden Markov model with the following properties:

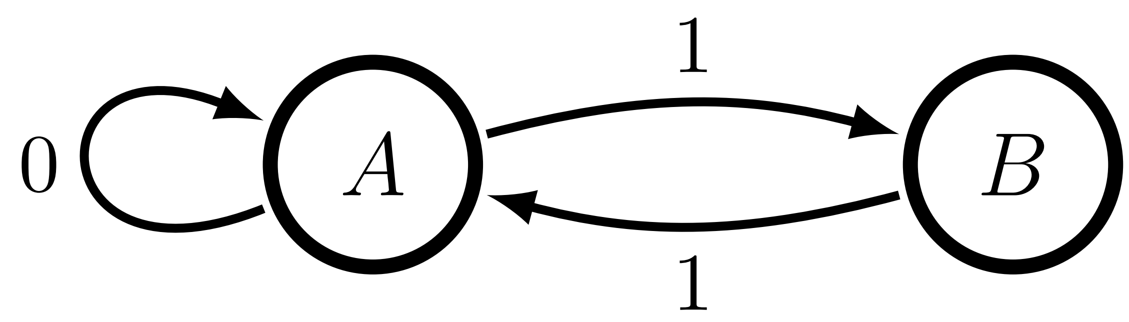
1. *Unifilarity: For each state $\sigma_i \in \boldsymbol{\mathcal{S}}$ and each symbol $x \in \mathcal{X}$ there is at most one outgoing edge from state σ_i that outputs symbol x.*
2. *Probabilistically distinct states: For each pair of distinct states $\sigma_k, \sigma_j \in \boldsymbol{\mathcal{S}}$ there exists some finite word $w = x_0 x_1 \ldots x_{L-1}$ such that:*

$$\mathbb{P}(w|\sigma_0 = \sigma_k) \neq \mathbb{P}(w|\sigma_0 = \sigma_j)$$

Dynamical Models

Moving to (hidden) Markov Models

- We've tackled the single-state 'fair coin' model
- Inference depended on counting edge and state transitions
- How do we handle *hidden* states for HMMs?

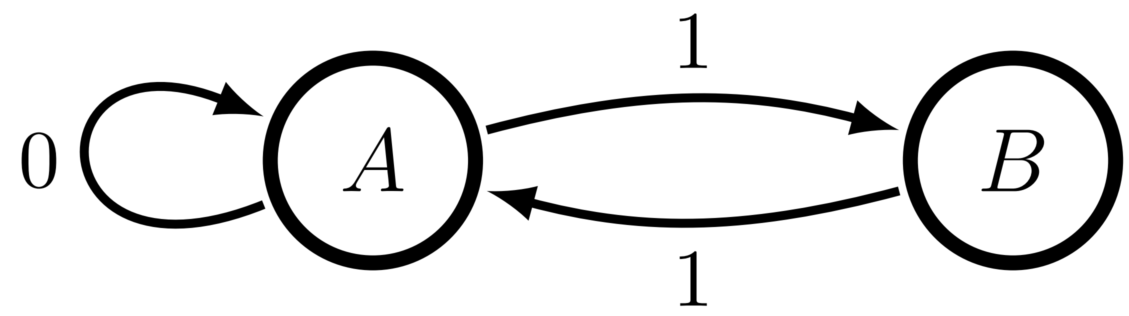


- For example, $D = 1011\ldots$
  - What edges were used? How many times?
  - What states were visited? How many times?

Unifilar HMMs

Unifilarity to the rescue

- Unifilarity: For each state $\sigma_i \in \boldsymbol{\mathcal{S}}$ and each symbol $x \in \mathcal{X}$ there is at most one outgoing edge from state σ_i that outputs symbol x.



- Again, $D = 1011\ldots$
  - Assume $\sigma_{i,0} = A$: requires states are AB – not possible!
  - Assume $\sigma_{i,0} = B$: requires states are $BAABA\ldots$

Inferring unifilar HMMs

- Now, given an assumed model structure M_i **and** an assumed start state $\sigma_{i,0}$, we can infer transition probabilities θ_i
- We have to do this for each possible start state $\sigma_{i,0} \in \boldsymbol{\mathcal{S}}_i$ in the machine!
- Subtle issues:
  - Not all transition probabilities have to be inferred– some are probability 1.0 by definition of the model topology
  - Define $\boldsymbol{\mathcal{S}}_i^* \subseteq \boldsymbol{\mathcal{S}}_i$ to be the set of states with more than one out-going edge
  - Infer transition probabilities for edges from $\boldsymbol{\mathcal{S}}_i^*$

$$\theta_i = \{p(x|\sigma_i)|\sigma_i \in \boldsymbol{\mathcal{S}}_i^*\}$$

Likelihood

Unifilar HMMs

$$\mathbb{P}(\mathbf{D}|\theta_i, \sigma_{i,0}, M_i) = \begin{cases} \prod_{\sigma_i \in \boldsymbol{\mathcal{S}}_i} \prod_{x \in \mathcal{X}} p(x|\sigma_i)^{n(\sigma_i x|\sigma_{i,0})} \\ \\ 0 \end{cases}$$

- $n(\sigma_i x|\sigma_{i,0})$ are edge counts given assumed start state $\sigma_{i,0}$
  - State counts are

$$n(\sigma_i \bullet |\sigma_{i,0}) = \sum_{x \in \mathcal{X}} n(\sigma_i x|\sigma_{i,0})$$

- Likelihood can be zero for any model, start state combination

Prior

Unifilar HMMs

- Prior is a product of Beta (Dirichlet) Distributions– one for each state in $\boldsymbol{\mathcal{S}}_i^*$

$$
\begin{aligned}
\mathbb{P}(\theta_i|\sigma_{i,0}, M_i) &= \prod_{\sigma_i \in \boldsymbol{\mathcal{S}}_i^*} \Bigg\{ \frac{\Gamma(\alpha(\sigma_i \bullet |\sigma_{i,0}))}{\prod_{x\in\mathcal{X}} \Gamma(\alpha(\sigma_i x|\sigma_{i,0}))} \\
&\times \quad \delta\left(1 - \sum_{x\in\mathcal{X}} p(x|\sigma_i)\right) \\
&\times \quad \prod_{x\in\mathcal{X}} p(x|\sigma_i)^{\alpha(\sigma_i x|\sigma_{i,0})-1} \Bigg\}
\end{aligned}
$$

- where

$$
\alpha(\sigma_i \bullet |\sigma_{i,0}) = \sum_{x\in\mathcal{X}} \alpha(\sigma_i x|\sigma_{i,0})
$$

Prior

Unifilar HMMs

- Typically use $\alpha(\sigma_i x | \sigma_{i,0}) = 1$ for all edges in $\boldsymbol{\mathcal{S}}_i^*$
  - Choose $\alpha(\sigma_i x | \sigma_{i,0})$ to be the same for all start states
  - Again, this results in a uniform density on the simplex
- The prior expectations (averages) are

$$\mathbf{E}_{\text{prior}}\left[p(x|\sigma_i)\right] = \begin{cases} \dfrac{\alpha(\sigma_i x | \sigma_{i,0})}{\alpha(\sigma_i \bullet | \sigma_{i,0})} & \sigma_i \in \boldsymbol{\mathcal{S}}_i^* \\ 0 \text{ or } 1 & \text{else} \end{cases}$$

Evidence

Unifilar HMMs

- The evidence is the same type of normalization term–important later

$$
\begin{aligned}
\mathbb{P}(\mathbf{D}|\sigma_{i,0}, M_i) &= \int d\theta_i \, \mathbb{P}(\mathbf{D}|\theta_i, \sigma_{i,0}, M_i)\mathbb{P}(\theta_i|\sigma_{i,0}, M_i) \\
&= \prod_{\sigma_i \in \boldsymbol{\mathcal{S}}_i^*} \left\{ \frac{\Gamma(\alpha(\sigma_i \bullet |\sigma_{i,0}))}{\prod_{x\in\mathcal{X}} \Gamma(\alpha(\sigma_i x|\sigma_{i,0}))} \right. \\
&\times \left. \frac{\prod_{x\in\mathcal{X}} \Gamma(\alpha(\sigma_i x|\sigma_{i,0}) + n(\sigma_i x|\sigma_{i,0}))}{\Gamma(\alpha(\sigma_i \bullet |\sigma_{i,0}) + n(\sigma_i \bullet |\sigma_{i,0}))} \right\}
\end{aligned}
$$

Posterior

Unifilar HMMs

- Using Bayes' theorem and terms from the previous slide we obtain the posterior

$$
\begin{aligned}
P(\theta_i | \mathbf{D}, \sigma_{i,0}, M_i) \quad &= \quad \prod_{\sigma_i \in \boldsymbol{\mathcal{S}}_i^*} \left\{ \frac{\Gamma(\alpha(\sigma_i \bullet | \sigma_{i,0}) + n(\sigma_i \bullet | \sigma_{i,0}))}{\prod_{x \in \mathcal{X}} \Gamma(\alpha(\sigma_i x | \sigma_{i,0}) + n(\sigma_i x | \sigma_{i,0}))} \right. \\
&\times \quad \delta\left(1 - \sum_{x \in \mathcal{X}} p(x|\sigma_i)\right) \\
&\times \quad \left. \prod_{x \in \mathcal{X}} p(x|\sigma_i)^{\alpha(\sigma_i x | \sigma_{i,0}) + n(\sigma_i x | \sigma_{i,0}) - 1} \right\}
\end{aligned}
$$

Posterior

Unifilar HMMs

- The posterior expectations (averages) are

$$\mathbf{E}_{\text{posterior}}\left[p(x|\sigma_i)\right] = \begin{cases} \dfrac{\alpha(\sigma_i x|\sigma_{i,0}) + n(\sigma_i x|\sigma_{i,0})}{\alpha(\sigma_i \bullet |\sigma_{i,0}) + n(\sigma_i \bullet |\sigma_{i,0})} & \sigma_i \in \boldsymbol{\mathcal{S}}_i^* \\ 0 \text{ or } 1 & \text{else} \end{cases}$$

- Of course, there has to be a viable path for the observed data given the assume model and start state
    - This statement is true for all terms that require counts from the data: likelihood, evidence and posterior!

What about that (annoying) start state?

- Remember, an assumed start state also implies the complete path through hidden states for the observed data and assumed model due to unifilarity
- Often, only one start state is possible for a given model topology and observed data set
- To be more systematic, we apply Bayes' Theorem at the level of start state
    - Remember, we already know $\mathbb{P}(D|\sigma_{i,0}, M_i)$

Start State

Unifilar HMMs

- Application of Bayes' theorem gets us

$$\mathbb{P}(\sigma_{i,0}|D, M_i) = \frac{\mathbb{P}(D|\sigma_{i,0}, M_i)\mathbb{P}(\sigma_{i,0}|M_i)}{\mathbb{P}(D|M_i)}$$

where

$$\mathbb{P}(D|M_i) = \sum_{\sigma_{i,0} \in \boldsymbol{\mathcal{S}}_i} \mathbb{P}(D|\sigma_{i,0}, M_i)\mathbb{P}(\sigma_{i,0}|M_i)$$

- CMPy code uses the default prior that all start states are equally likely

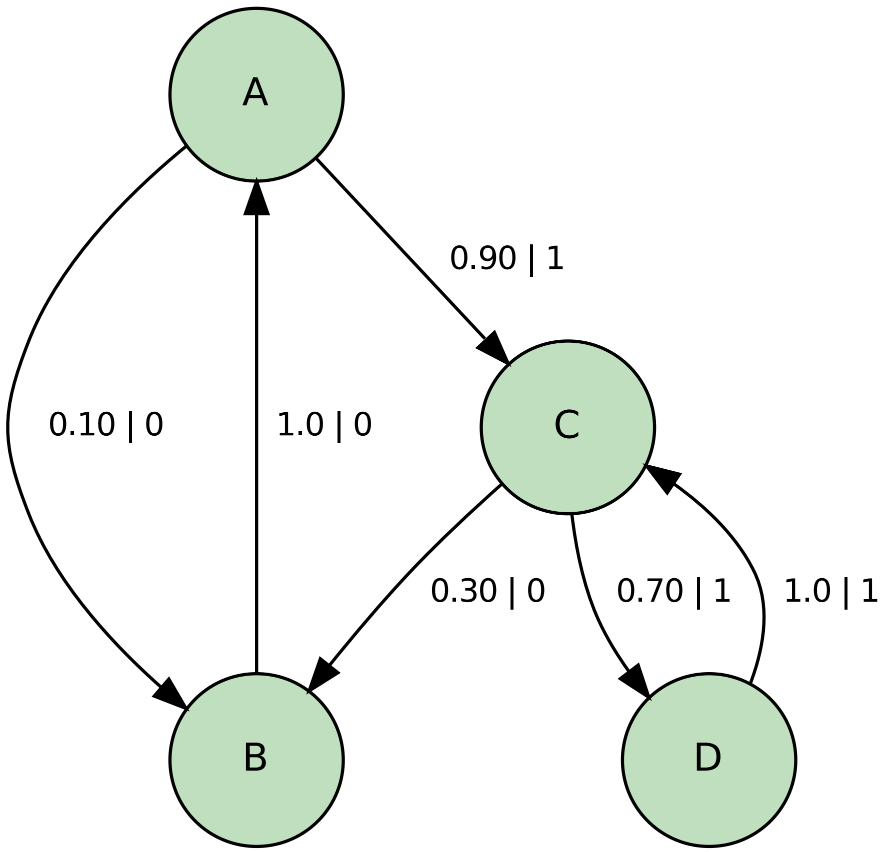
$$\mathbb{P}(\sigma_{i,0}|M_i) = \frac{1}{|\boldsymbol{\mathcal{S}}_i|}$$

Ex 2: EvenOdd Process

EvenOdd Process

Use CMPy to generate data

```
eo_str = """A B 0 0.1;A C 1 0.9;B A 0 1.0;
            C B 0 0.3;C D 1 0.7;D C 1 1.0"""
eomachine = cmpy.machines.from_string(eo_str,name='biased EvenOdd',
                                      style=1)
# draw machine
eomachine.draw(filename='figures/evenodd.pdf', show=False)
```



$$\boldsymbol{\mathcal{S}}_i = \{A, B, C, D\}$$

$$\boldsymbol{\mathcal{S}}_i^* = \{A, C\}$$

Prior and Posterior

EvenOdd Process in CMPy

```
## instantiate prior
eo_prior = bayesem.InferEM(eomachine)

# set machine node to A
eomachine.set_current_node('A')
# generate data, using symbols_iter()
eo_data = []
for d in eomachine.symbols_iter(200):
    eo_data.append(d)

## instantiate posterior
eo_posterior = bayesem.InferEM(eomachine, eo_data)

# what is the machine state after generating data?
print 'last state:', eomachine.get_current_node()
```

```
last state: A
```

Prior for EvenOdd Process

Hidden state dynamics

```
# start node and last node?
for sN in eo_prior.get_possible_start_nodes():
    pr = eo_prior.probability_start_node(sN)
    print 'Pr(sN=%s):%f' % (sN, pr),
    print ' -> last state:', eo_prior.get_last_node(sN)
    print 'state path: ', eo_prior.get_state_path(sN)[:10]
```

```
Pr(sN=A):0.250000  -> last state: None
state path:  []
Pr(sN=C):0.250000  -> last state: None
state path:  []
Pr(sN=B):0.250000  -> last state: None
state path:  []
Pr(sN=D):0.250000  -> last state: None
state path:  []
```

Posterior for EvenOdd Process

Hidden state dynamics

```
# start node and last node?
for sN in eo_posterior.get_possible_start_nodes():
    pr = eo_posterior.probability_start_node(sN)
    print 'Pr(sN=%s):%f' % (sN, pr),
    print ' -> last state:', eo_posterior.get_last_node(sN)
    print 'state path: ', eo_posterior.get_state_path(sN)[:10]
```

```
Pr(sN=A):0.458333  -> last state: A
state path:  ['A', 'C', 'B', 'A', 'C', 'D', 'C', 'B', 'A', 'C']
Pr(sN=D):0.541667  -> last state: A
state path:  ['D', 'C', 'B', 'A', 'C', 'D', 'C', 'B', 'A', 'C']
```

Prior, Posterior and C_μ, h_μ

Sample from prior & posterior

```
num_samples = 2000
eo_prior_hmu = []; eo_prior_Cmu = [];
eo_posterior_hmu = []; eo_posterior_Cmu = [];

# generate and store samples
for n in range(num_samples):
    # prior
    (node,machine) = eo_prior.generate_sample()
    hmu = machine.entropy_rate()
    Cmu = machine.statistical_complexity()
    eo_prior_hmu.append(hmu); eo_prior_Cmu.append(Cmu)

    # posterior
    (node,machine) = eo_posterior.generate_sample()
    hmu = machine.entropy_rate()
    Cmu = machine.statistical_complexity()
    eo_posterior_hmu.append(hmu); eo_posterior_Cmu.append(Cmu)
```

Prior vs Posterior, h_μ

Plot h_μ samples

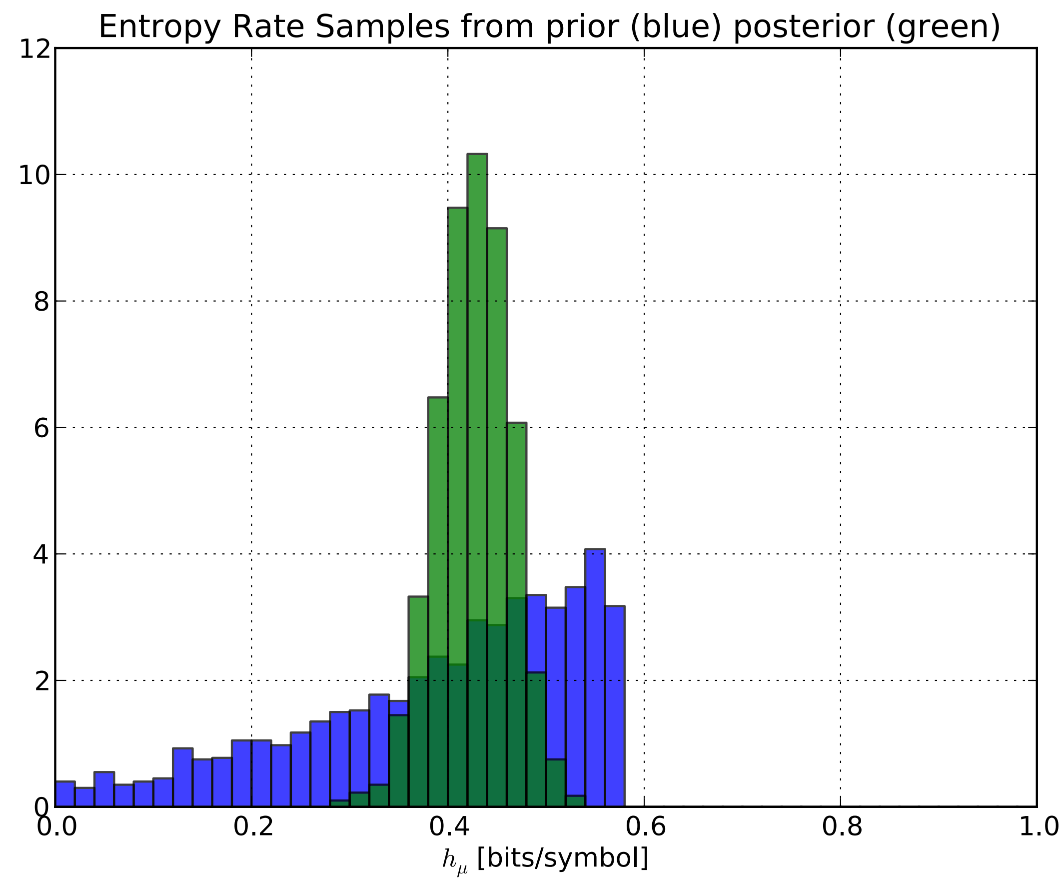
```
plt.clf()
# prior hmu -- blue
n, bins, patches = plt.hist(eo_prior_hmu, 50, range=[0.0,1.0],
                            normed=1, facecolor='blue', alpha=0.75,
                            cumulative=False)

# posterior hmu -- green
n, bins, patches = plt.hist(eo_posterior_hmu, 50, range=[0.0,1.0],
                            normed=1, facecolor='green', alpha=0.75,
                            cumulative=False)

plt.xlabel(r'$h_{\mu}$ [bits/symbol]')
plt.title('Entropy Rate Samples from prior (blue) posterior (green)')
plt.grid(True)
plt.savefig('figures/eo_hmu_hist.pdf')
```

Prior vs Posterior, h_μ

Plot h_μ samples




```
true hmu:  0.438432153702  [bits/symbol]
```

Prior vs Posterior, C_μ

Plot C_μ samples

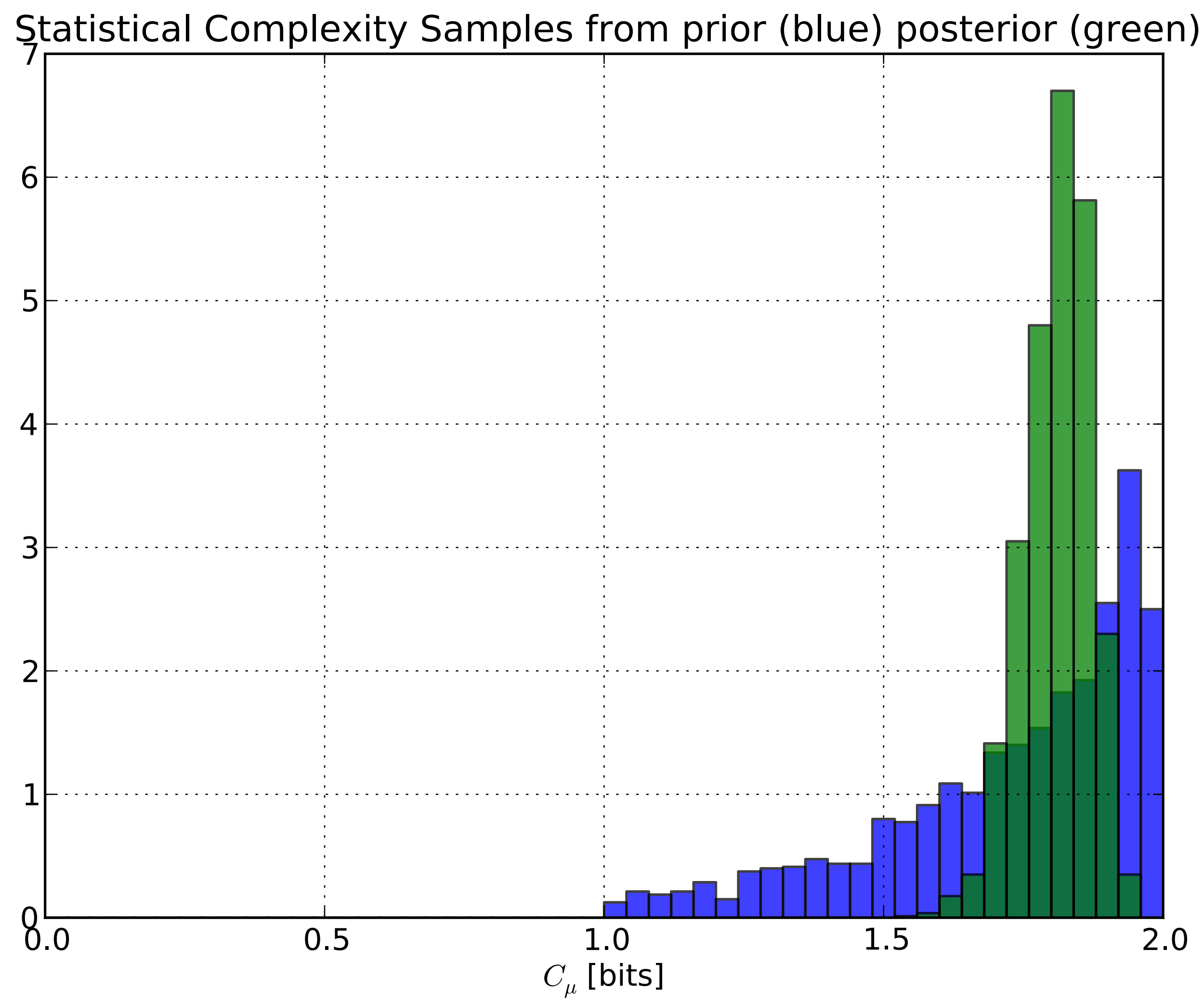
```
plt.clf()
# prior Cmu -- blue
n, bins, patches = plt.hist(eo_prior_Cmu, 50, range=[0.0,2.0],
                            normed=1, facecolor='blue', alpha=0.75,
                            cumulative=False)

# posterior Cmu -- green
n, bins, patches = plt.hist(eo_posterior_Cmu, 50, range=[0.0,2.0],
                            normed=1, facecolor='green', alpha=0.75,
                            cumulative=False)

plt.xlabel(r'$C_{\mu}$ [bits]')
plt.title('Statistical Complexity Samples from prior (blue) posterior
n)')
plt.grid(True)
plt.savefig('figures/eo_Cmu_hist.pdf')
```

Prior vs Posterior, C_μ

Plot C_μ samples




```
true Cmu:  1.84152253296  [bits]
```

Prior vs Posterior, C_μ, h_μ

Plot C_μ and h_μ samples

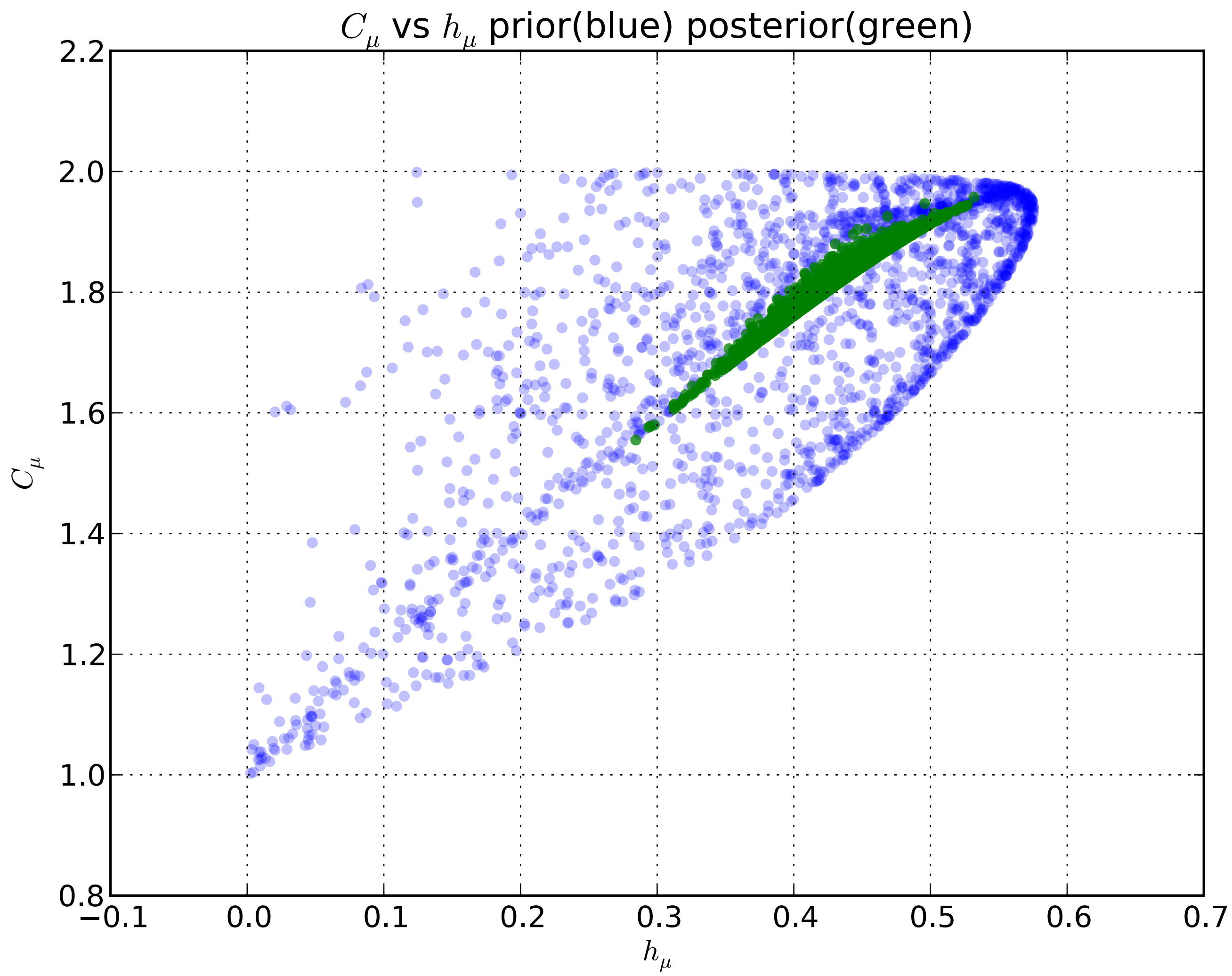
```
# prior - blue
plt.clf()
plt.scatter(eo_prior_hmu, eo_prior_Cmu,s=20,
facecolor='blue', edgecolor='none',alpha=0.25)

# posterior - green
plt.scatter(eo_posterior_hmu, eo_posterior_Cmu,s=20,
            facecolor='green', edgecolor='none',alpha=0.75)

plt.ylabel(r'$C_{\mu}$')
plt.xlabel(r'$h_{\mu}$')
plt.title(r'$C_{\mu}$ vs $h_{\mu}$ prior(blue) posterior(green)')
plt.grid(True)
plt.savefig('figures/eo_Cmuhmu.pdf')
```

Prior vs Posterior, C_μ, h_μ

Plot C_μ and h_μ samples




```
true Cmu:  1.84152253296  [bits]
true hmu:  0.438432153702  [bits/symbol]
```