

# Bayesian Methods

Peter Komar

December 30, 2018

## Contents

<b>1</b>	<b>Foundations</b>	<b>3</b>
1.1	Definitions, identities . . . . .	3
1.2	Bayesian inference . . . . .	4
1.3	Model comparison . . . . .	5
1.4	Prediction . . . . .	6
<b>2</b>	<b>Exact inference and Maximum Likelihood Estimate</b>	<b>7</b>
2.1	Maximum likelihood estimate . . . . .	7
2.2	Exact inference examples . . . . .	9
<b>3</b>	<b>Priors, Regularization, AIC, BIC, LRT</b>	<b>13</b>
3.1	Improper and proper priors . . . . .	13
3.2	Regularization . . . . .	13
3.3	Linear regression . . . . .	13
3.4	Model comparison with asymptotic metrics . . . . .	14
3.5	Example: Linear regression . . . . .	15
<b>4</b>	<b>Graphical models</b>	<b>19</b>
4.1	Elements . . . . .	19
4.2	Real-life examples . . . . .	20
4.3	Plate notation . . . . .	20
4.4	Hierarchical models . . . . .	21
4.5	Example: Beta-Binomial . . . . .	24
<b>5</b>	<b>Hidden variables, EM, Mixture models</b>	<b>26</b>
5.1	Definitions . . . . .	26
5.2	Expectation Maximization . . . . .	26
5.3	Mixture models . . . . .	27
5.4	Gaussian Mixture Model . . . . .	28
<b>6</b>	<b>Curse of Dimensionality, Laplace approximation</b>	<b>31</b>
6.1	High-dimensional example . . . . .	31
6.2	Laplace approximation . . . . .	31
6.3	Example: $(x, y)$ linear regression . . . . .	32
<b>7</b>	<b>Monte Carlo methods</b>	<b>35</b>
7.1	Monte Carlo simulation . . . . .	35
7.2	Markov Chain Monte Carlo method . . . . .	38
7.3	Metropolis Hastings sampling . . . . .	38

<b>8</b>	<b>Gibbs sampling</b>	<b>40</b>
8.1	Gibbs sampling algorithm . . . . .	40
8.2	Example: Triangle distribution in 2D . . . . .	40
8.3	Example: Binomial clustering . . . . .	41
<b>9</b>	<b>Viterbi algorithm, Belief propagation</b>	<b>44</b>
9.1	Hidden Markov Model . . . . .	44
9.2	Viterbi algorithm . . . . .	45
9.3	Belief propagation on chain . . . . .	46

# 1 Foundations

## 1.1 Definitions, identities

Notation

- Upper-case letters ( $A, B, C, X, Y$ ): random variables
- Lower-case letters ( $a, b, c, x, y$ ): real numbers
- $P(X = x) =: P(x)$
- $P(X = x \text{ and } Y = y) =: P(x, y)$
- $P(A = a, \text{ given } B = b) =: P(a | b)$
- $\int_{-\infty}^{+\infty} [\dots] da =: \sum_{a \in \mathbb{R}} [\dots] =: \sum_a [\dots]$

Conditional probability

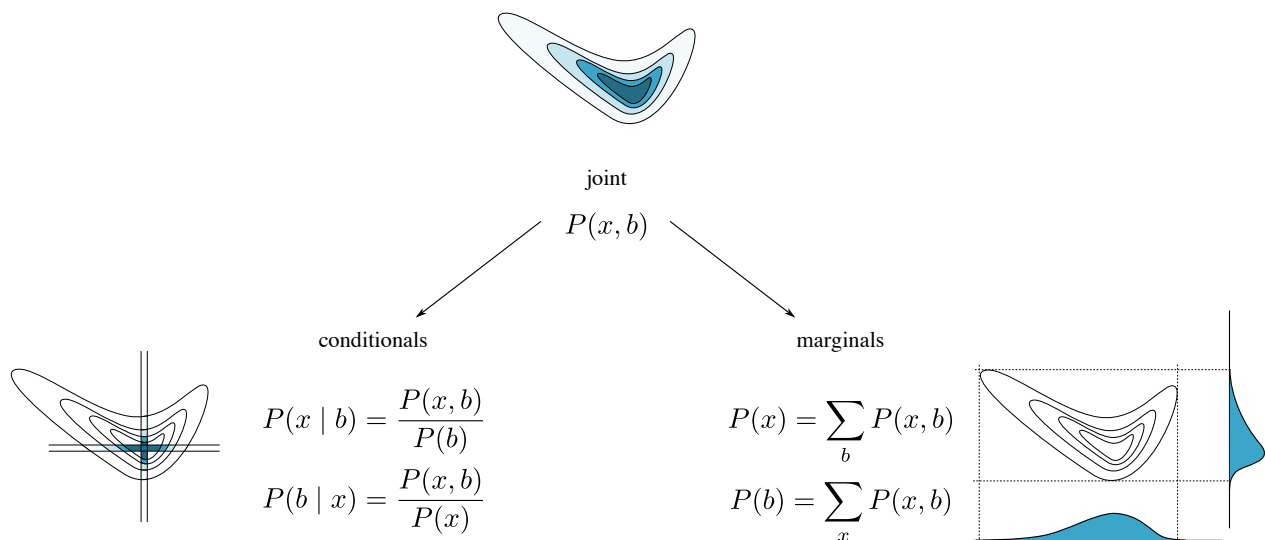
- $P(a | b) = \frac{P(a, b)}{P(b)}$
- $P(a, b) = P(a | b) P(b)$
- $P(a, b | c) = P(a | b, c) P(b | c)$
- $\sum_a P(a | b) = 1$ , but  $\sum_b P(a | b) \neq 1$ , in general
- $\sum_b P(a | b) P(b) = \sum_b P(a, b) = P(a)$

Marginal

- $P(a) = \sum_b P(a | b) P(b)$

Bayes theorem

- $P(b | x) = \frac{1}{P(x)} P(x | b) P(b)$



## 1.2 Bayesian inference

Prior, likelihood, posterior

- Data:  $D = \{x_1, x_2, \dots, x_n\}$ , independent measurements.
- Model:  $M$  with  $\theta$ : parameter(s) to estimate
- Prior:  $P(\theta)$
- Likelihood:  $P(D | \theta) = P(x_1 | \theta) P(x_2 | \theta) \dots P(x_n | \theta) = \prod_{i=1}^n P(x_i | \theta)$
- Unnormalized posterior:  $P^*(\theta | D) = P(D | \theta) P(\theta)$
- Normalization:  $Z = \sum_{\theta} P^*(\theta | D)$
- Posterior:  $P(\theta | D) = \frac{1}{Z} P^*(\theta | D)$

Example

“Three light bulbs of the same make lasted 1, 2 and 5 months of continuous use. Let us estimate the lifetime of this kind of light bulb.”

- $D = \{t_1, t_2, t_3\} = \{1, 2, 5\}$
- $M$ : Light bulbs have average lifetime of  $T$  months.
- $P(T) = \frac{1}{1000}$ , uniform on  $[0, 1000]$ .
- $P(t | T) = \frac{1}{T} \exp\left(-\frac{t}{T}\right)$
- $P(D | T) = \prod_i P(t_i | T) = \prod_{i=1}^3 \frac{1}{T} \exp\left(-\frac{t_i}{T}\right) = \frac{1}{T^3} \exp\left(-\frac{1+2+5}{T}\right)$
- $P^*(T | D) = \frac{1}{T^3} \exp\left(-\frac{8}{T}\right)$
- $Z$  and  $P(T | D)$  can be determined numerically:

```
1 import numpy as np
2
3 T_arr = np.linspace(0.1, 1000, 10_000)
4 Pstar_arr = 1.0/T_arr**3 * np.exp(-8/T_arr)
5 Z = np.sum(Pstar_arr)
6 P_arr = Pstar_arr / Z
```

yielding  $Z = 0.1562$

- $\mathbb{E}(T | D) = \sum_T T P(T | D)$
- $\text{std}(T | D) = \sqrt{\sum_T (T - \mathbb{E}(T))^2 P(T | D)}$

```
1 T_ev = np.sum(T_arr * P_arr)
2 T_std = np.sqrt(np.sum((T_arr - T_ev)**2 * P_arr))
```

yielding  $\mathbb{E}(T | D) = 7.937$ ,  $\text{std}(T | D) = 14.48$ .

### 1.3 Model comparison

New definition: Evidence

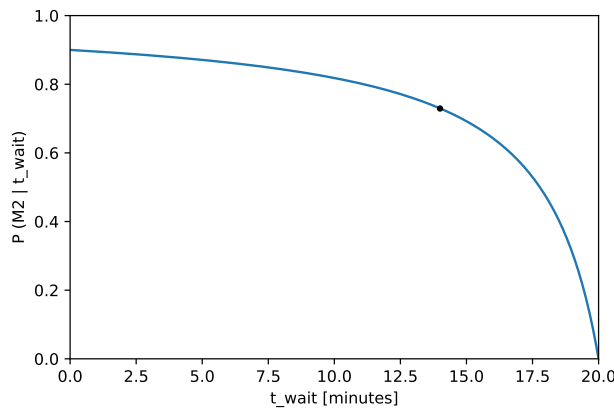
- Data:  $D$
- Model 1:  $M_1$  with parameter  $\theta_1$  and prior  $P(\theta_1 | M_1)$ , and likelihood  $P(D | \theta_1, M_1)$
- Model 2:  $M_2$  with parameter  $\theta_2$  and prior  $P(\theta_2 | M_2)$ , and likelihood  $P(D | \theta_2, M_2)$
- Prior on models:  $P(M_1) = 0.5$ ,  $P(M_2) = 0.5$ .
- Evidence for each model:  $P(D | M_i) = \sum_{\theta} P(D | \theta_i, M_i) P(\theta_i | M_i)$
- Unnormalized posterior:  $P^*(M_1 | D) = P(D | M_1) P(M_1)$ , and  $P^*(M_2 | D) = P(D | M_2) P(M_2)$
- Normalization:  $Z = P^*(M_1 | D) + P^*(M_2 | D)$ .

Example

“Waiting for my baggage at the airport carousel, there are two possibilities: 1) It could miss the plane, and will never come, or 2) It was on the plane and it had a  $1/20$  chance of arriving within any of the 1-minute intervals between 0 and 20 minutes. Now, given what is the posterior probability of model 2 given that 14 minutes has passed and the bag has not arrived?”

- $D = \{\text{Bag has not arrived after } t_{\text{wait}} = 14 \text{ minutes}\}$
- $M_1$ : It never arrives,  $P(D | M_1) = 1$
- $M_2$ : It shows up some time between 0 and 20 minutes,  $P(t_{\text{bag}} | M_2) = 1/20$  for  $t_{\text{bag}} \in [0, 20]$ , and the likelihood is  $P(D | t_{\text{bag}}, M_2) = 1$ , if  $t_{\text{bag}} > t_{\text{wait}}$ , and 0 otherwise.
- $P(M_1) = 0.1$ ,  $P(M_2) = 0.9$
- $P(D | M_1) = 1$
- $P(D | M_2) = \sum_{t_{\text{bag}}} P(D | t_{\text{bag}}, M_2) P(t_{\text{bag}} | M_2) = \sum_{t_{\text{bag}}} [t_{\text{bag}} > 14] \times \frac{1}{20} = \frac{20-14}{20}$
- $P^*(M_1 | D) = 1 \times 0.1$ ,  $P^*(M_2 | D) = \frac{20-14}{20} \times 0.9$
- $Z = 0.1 + \frac{3}{10} \times 0.9 = 0.37$
- $P(M_2 | D) = P^*(M_2 | D)/Z = 0.7297$ .

We can also plot  $P(M_2 | t_{\text{wait}})$  for all waiting times between 0 and 20 minutes.



## 1.4 Prediction

New definition: Predictive distribution

- Data:  $D = \{x_1, x_2, \dots, x_n\}$
- Model:  $M$  with parameter  $\theta$ , prior  $P(\theta)$  and likelihood  $P(x | \theta)$
- Posterior:  $P(\theta | D) = P^*(\theta | D)/Z = \dots$  (see previous sections)
- Predictive distribution:  $P(X_{n+1} = x | D) = \sum_{\theta} P(x | \theta) P(\theta | D)$
- Customized prediction:  $P(f(\theta) | D) = \sum_{\theta} f(\theta) P(\theta | D)$

Example

“Two player, A and B are playing a game of luck, where at the beginning of the game a ball is rolled on a pool table to divide the table in two un-equal halves: A’s side and B’s side. In each subsequent round, a ball is rolled. A point is given to the player on whose side the ball stops. A and B are playing this game until one of them reaches 6 points. The current score is 5 to 3 in favor of A. What is the chance that A will win this game?”

- $D = \{n_A = 5, n_B = 3\}$
- $M$ , first ball:  $P(b) = 1$  in  $[0, 1]$
- $P(\text{A scores} | b) = b$
- $P(D | b) = \text{Binomial}(5 | 5 + 3, b)$
- $P^*(b_0 | D) = \text{Binomial}(5 | 8, b) \times 1$
- $Z = \sum_b \text{Binomial}(5 | 8, b)$  can be calculated numerically

```
1 import numpy as np
2 from scipy.stats import binom
3
4 b_arr = np.linspace(0, 1, 1000)
5 Pstar_arr = binom.pmf(5, 8, b_arr)
6 Z = np.sum(Pstar_arr)
```

- $P(\text{A wins} | b, D) = 1 - P(\text{B wins} | b, D) = 1 - (1 - b)^3 = f(b)$
- $P(\text{A wins} | D) = \sum_b f(b) P^*(b | D) / Z$

```
1 P_arr = Pstar_arr / Z
2 P_Awins = np.sum((1 - (1 - b_arr)**3) * P_arr)
```

yielding  $P(\text{A wins} | D) = 0.909$

## 2 Exact inference and Maximum Likelihood Estimate

### 2.1 Maximum likelihood estimate

MLE-method

- Data:  $D = \{x_1, x_2, \dots, x_N\}$
- Parameter:  $\theta$
- Likelihood:  $P(x_i | \theta)$
- Total log likelihood:  $L(\theta) = \log P(D | \theta) = \sum_{i=1}^N \log P(x_i | \theta)$
- Maximum likelihood estimate  $\theta_{\text{MLE}} = \operatorname{argmax}_{\theta} \log P(D | \theta)$ ,  
Numerically: with gradient descent or EM methods,  
Analytically: equating first derivatives to 0, and solving the system of equations.

Example 1: Normal model

- Data:  $D = \{x_i\}_{i=1}^N$
- Parameters:  $\mu \in \mathbb{R}, \sigma^2 > 0$
- Likelihood:  $P(x_i | \mu, \sigma^2) = \text{Normal}(x_i | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x_i - \mu)^2}{2\sigma^2}\right]$
- Total log likelihood:

$$L(\mu, \sigma^2) = \sum_{i=1}^N \log \text{Normal}(x_i | \mu, \sigma^2) = -\frac{N}{2} \log(\sigma^2) - \sum_{i=1}^N \frac{(x_i - \mu)^2}{2\sigma^2} + \text{const.}$$

- Analytical solution:

$$\begin{aligned} 0 &= \left[ \frac{\partial L}{\partial \mu} \right]_{\text{MLE}} = \left[ \sum_{i=1}^N \frac{\mu - x_i}{\sigma^2} \right]_{\text{MLE}} &\Rightarrow \mu_{\text{MLE}} &= \frac{1}{N} \sum_{i=1}^N x_i. \\ 0 &= \left[ \frac{\partial L}{\partial (\sigma^2)} \right]_{\text{MLE}} = \left[ -\frac{N}{2\sigma^2} + \sum_{i=1}^N \frac{(x_i - \mu)^2}{2(\sigma^2)^2} \right]_{\text{MLE}} &\Rightarrow (\sigma^2)_{\text{MLE}} &= \frac{1}{N} \sum_{i=1}^N (x_i - \mu_{\text{MLE}})^2 \end{aligned}$$

Example 2: Cauchy distribution

- Data:  $D = \{-10, 1, 2, 5, 20\}$
- Parameters:  $m \in \mathbb{R}, \quad s > 0.$
- Likelihood:  $P(x_i | m, s) = \text{Cauchy}(x_i | m, s) = \frac{1}{s\pi} \frac{1}{1 + [(x_i - m)/s]^2}$
- Total log likelihood:

$$L(m, s) = \sum_{i=1}^N \log \text{Cauchy}(x_i | m, s) = -N \log(s) - \sum_{i=1}^N \log \left( 1 + \left[ \frac{x_i - m}{s} \right]^2 \right)$$

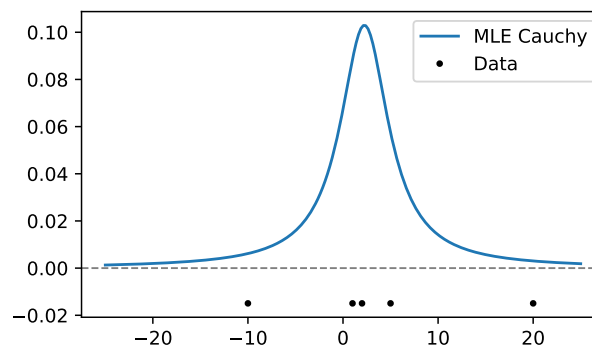
- Numerical maximization (starting from  $m_0 = 0, s_0 = 10$ ):

```

1 import numpy as np
2 from scipy.optimize import minimize
3
4 def cauchy_total_log_likelihood(X, m, s):
5     X = np.array(X)
6
7     L = 0
8     L += -len(X)/2 * np.log(s**2)
9     L += -np.sum( np.log(1 + (X - m)**2 / s**2 ) )
10
11     return L
12
13 X = [-10, 1, 2, 5, 20]
14 def func_to_minimize(theta):
15     m = theta[0]
16     s = theta[1]
17     return - cauchy_total_log_likelihood(X, m, s)
18
19 m0 = 0
20 s0 = 10
21 result = minimize(func_to_minimize, [m0, s0])
22 m_MLE, s_MLE = result.x

```

yielding  $m_{\text{MLE}} = 2.251$ ,  $s_{\text{MLE}} = 3.090$ , the resulting MLE fit is shown below.





## 2.2 Exact inference examples

Binomial model

- Data:  $D = \{(k_1, n_1), (k_2, n_2), \dots, (k_N, n_N)\}$ , where  $k_i$  (successes),  $n_i$  (attempts)  $\in \mathbb{N}$  and  $k_i \leq n_i$
- Parameter:  $p$  (probability of success)  $\in [0, 1]$ , flat prior:  $P(p) = 1$ , on  $[0, 1]$
- Likelihood:  $P(k_i | n_i, p) = \text{Binomial}(k_i | n_i, p) = \binom{n_i}{k_i} p^{k_i} (1-p)^{n_i-k_i}$
- Posterior:

$$\begin{aligned} P(p | D) &= \frac{1}{Z} \prod_{i=1}^N [p^{k_i} (1-p)^{n_i-k_i}] = \frac{1}{Z} p^{k_{\text{tot}}} (1-p)^{n_{\text{tot}}-k_{\text{tot}}} \\ &= \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1} (1-p)^{\beta-1} = \text{Beta}(p | \alpha = k_{\text{tot}} + 1, \beta = n_{\text{tot}} - k_{\text{tot}} + 1), \end{aligned}$$

where  $k_{\text{tot}} = \sum_i k_i$  and  $n_{\text{tot}} = \sum_i n_i$ . Mean, mode and standard deviation are

$$\begin{aligned} \mathbb{E}(p) &= \frac{\alpha}{\alpha + \beta} = \frac{k_{\text{tot}} + 1}{n_{\text{tot}} + 2}, \quad \text{mode}(p) = \frac{\alpha - 1}{\alpha + \beta - 2} = \frac{k_{\text{tot}}}{n_{\text{tot}}}, \\ \text{std}(p) &= \frac{\sqrt{\alpha\beta}}{(\alpha + \beta)\sqrt{\alpha + \beta + 1}} = \frac{\sqrt{(k_{\text{tot}} + 1)(n_{\text{tot}} - k_{\text{tot}} + 1)}}{(n_{\text{tot}} + 2)\sqrt{n_{\text{tot}} + 3}} \end{aligned}$$

Poisson model

- Data:  $D = \{k_1, k_2, \dots, k_N\}$ , where  $k_i$  (number of events)  $\in \mathbb{N}$
- Parameters:  $\lambda$  (expected number of events)  $> 0$ , flat prior:  $P(\lambda) = \text{const.}$
- Likelihood:  $P(k_i | \lambda) = \text{Poisson}(k_i | \lambda) = e^{-\lambda} \frac{\lambda^{k_i}}{k_i!}$
- Posterior:

$$P(\lambda | D) = \frac{1}{Z} \prod_{i=1}^N [e^{-\lambda} \lambda^{k_i}] = \frac{1}{Z} e^{-N\lambda} \lambda^{k_{\text{tot}}} = \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\beta\lambda} = \text{Gamma}(\lambda | \alpha = k_{\text{tot}} + 1, \beta = N),$$

where  $k_{\text{tot}} = \sum_i k_i$ . Mean, mode and standard deviation are

$$\mathbb{E}(\lambda) = \frac{\alpha}{\beta} = \frac{k_{\text{tot}} + 1}{N}, \quad \text{mode}(\lambda) = \frac{\alpha - 1}{\beta} = \frac{k_{\text{tot}}}{N}, \quad \text{std}(\lambda) = \frac{\sqrt{\alpha}}{\beta} = \frac{\sqrt{k_{\text{tot}} + 1}}{N}$$

## Multinomial model

- Data:  $D = \{(k_{1,1}, k_{1,2}, \dots, k_{1,M}), (k_{2,1}, k_{2,2}, \dots, k_{2,M}), \dots, (k_{N,1}, k_{N,2}, \dots, k_{N,M})\}$ , where  $k_{i,j}$  (counts of outcome  $j$ )  $\in \mathbb{N}$ , and  $\sum_j k_{i,j} = 1, \forall i$ .
- Parameters:  $p = (p_1, p_2, \dots, p_M)$ , where  $p_j$  (probability of outcome  $j$ )  $> 0$  and  $\sum_j p_j = 1$ ; flat prior:  $P(p) = \text{const.}$
- Likelihood:  $P(\{k_{i,j}\}_{j=1}^M | p) = \text{Multinomial}(\{k_{i,j}\}_{j=1}^M | p) = k_{i,\text{tot}}! \prod_j \frac{p_j^{k_{i,j}}}{k_{i,j}!}$
- Posterior:

$$P(p | D) = \frac{1}{Z} \prod_{i=1}^N \prod_{j=1}^M (p_j)^{k_{i,j}} = \frac{1}{Z} \prod_{j=1}^M (p_j)^{k_{\text{tot},j}} = \Gamma(\alpha_{\text{tot}}) \prod_{j=1}^M \frac{(p_j)^{\alpha_j - 1}}{\Gamma(\alpha_j)} = \text{Dirichlet}(p | \alpha_j = k_{\text{tot},j} + 1),$$

where  $k_{i,\text{tot}} = \sum_j k_{i,j}$ ,  $k_{\text{tot},j} = \sum_i k_{i,j}$ , and  $\alpha_{\text{tot}} = \sum_j \alpha_j = k_{\text{tot},\text{tot}} + M$ . Mean, mode and marginal standard deviation are

$$\mathbb{E}(p_j) = \frac{\alpha_j}{\alpha_{\text{tot}}} = \frac{k_{\text{tot},j} + 1}{k_{\text{tot},\text{tot}} + M}, \quad \text{mode}(p) : p_j = \frac{\alpha_j - 1}{\alpha_{\text{tot}} - M} = \frac{k_{\text{tot},j}}{k_{\text{tot},\text{tot}}}$$

$$\text{std}(p_j) = \frac{\sqrt{\alpha_j(\alpha_{\text{tot}} - \alpha_j)}}{\alpha_{\text{tot}} \sqrt{\alpha_{\text{tot}} + 1}} = \frac{\sqrt{(k_{\text{tot},j} + 1)(k_{\text{tot},\text{tot}} - k_{\text{tot},j} + M - 1)}}{(k_{\text{tot},\text{tot}} + M) \sqrt{k_{\text{tot},\text{tot}} + M + 1}}$$

## Exponential model

- Data:  $D = \{t_1, t_2, \dots, t_N\}$ , where  $t_i$  (waiting times)  $> 0$
- Parameter:  $\gamma$  (rate)  $> 0$ , flat prior:  $P(\gamma) = \text{const.}$
- Likelihood:  $P(t_i | \gamma) = \text{Exponential}(t_i | \gamma) = \gamma e^{-\gamma t_i}$
- Posterior:

$$P(\gamma | D) = \frac{1}{Z} \prod_{i=1}^N [\gamma e^{-\gamma t_i}] = \frac{1}{Z} \gamma^N e^{-\gamma t_{\text{tot}}} = \frac{\beta^\alpha}{\Gamma(\alpha)} \gamma^{\alpha-1} e^{-\beta \gamma} = \text{Gamma}(\gamma | \alpha = N + 1, \beta = t_{\text{tot}}),$$

where  $t_{\text{tot}} = \sum_i t_i$ . Mean, mode, standard deviation are

$$\mathbb{E}(\gamma) = \frac{\alpha}{\beta} = \frac{N + 1}{t_{\text{tot}}}, \quad \text{mode}(\gamma) = \frac{\alpha - 1}{\beta} = \frac{N}{t_{\text{tot}}}, \quad \text{std}(\gamma) = \frac{\sqrt{\alpha}}{\beta} = \frac{\sqrt{N + 1}}{t_{\text{tot}}}$$

## Normal

- Data:  $D = \{x_1, x_2, \dots, x_N\}$ , where  $x$  (value)  $\in \mathbb{R}$
- Parameters:  $\mu$  (expected value)  $\in \mathbb{R}$ ,  $\sigma^2$  (variance)  $> 0$ , uninformative prior:  $P(\mu, \sigma^2) = \text{const.}$
- Likelihood:  $P(x_i | \mu, \sigma^2) = \text{Normal}(x_i | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x_i - \mu)^2}{2\sigma^2}\right]$
- Posterior:

$$\begin{aligned}
P(\mu, \sigma^2 | D) &= \frac{1}{Z} \prod_{i=1}^N \left\{ \frac{1}{\sqrt{\sigma^2}} \exp\left[-\frac{(x_i - \mu)^2}{2\sigma^2}\right] \right\} = \frac{1}{Z} \left(\frac{1}{\sigma^2}\right)^{N/2} \exp\left[-\frac{Ns^2 + N(\mu - m)^2}{2\sigma^2}\right] \\
&= \frac{\sqrt{\lambda}}{\sqrt{2\pi\sigma^2}} \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\frac{1}{\sigma^2}\right)^{\alpha+1} \exp\left[-\frac{2\beta + \lambda(\mu - \mu_c)^2}{2\sigma^2}\right] \\
&= \text{Normal-Inverse-Gamma}\left(\mu, \sigma^2 \mid \alpha = \frac{N-3}{2}, \beta = \frac{Ns^2}{2}, \mu_c = m, \lambda = N\right),
\end{aligned}$$

where  $m = \frac{1}{N} \sum_i x_i$  is the empirical mean,  $s^2 = \frac{1}{N} \sum_i (x_i - m)^2$  is the empirical variance. The mode is identical to the MLE result

$$\text{mode}(\mu, \sigma^2) = (m, s^2).$$

The marginal, and mean, mode and standard deviation of  $\mu$  is

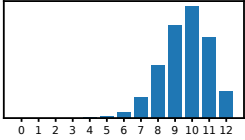
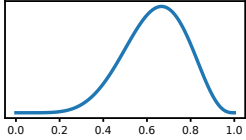
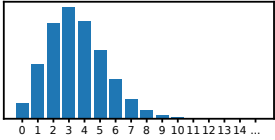
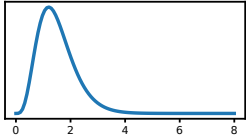
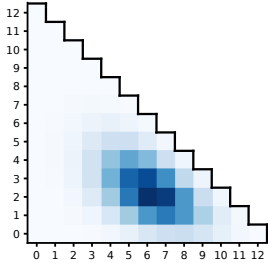
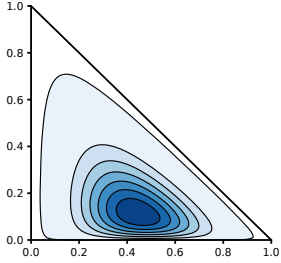
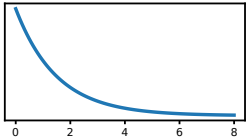
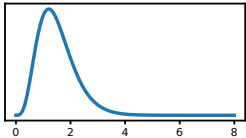
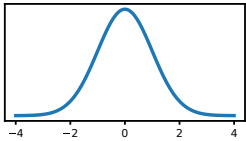
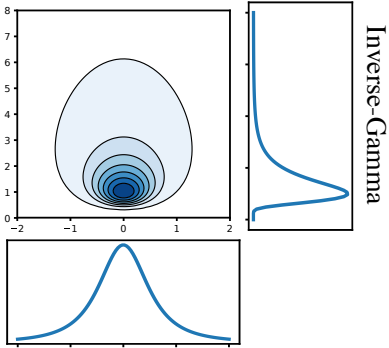
$$\begin{aligned}
P(\mu | D) &= \sum_{\sigma^2} P(\mu, \sigma^2 | D) = \frac{\Gamma\left(\frac{N-2}{2}\right)}{\Gamma\left(\frac{N-3}{2}\right)} \frac{1}{\sqrt{\pi s^2}} \left[1 + \frac{(\mu - m)^2}{s^2}\right]^{-(N-2)/2} \\
&= \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)} \frac{1}{\sqrt{\pi\nu}} \left[1 + \frac{1}{\nu} \left(\frac{\mu - \text{loc}}{\text{scale}}\right)^2\right]^{-(\nu+1)/2} \frac{1}{\text{scale}} \\
&= \text{t-distr}\left(\mu \mid \text{loc} = m, \text{scale} = \frac{s}{\sqrt{N-3}}, \nu = N-3\right),
\end{aligned}$$

$$\mathbb{E}(\mu) = m, \quad \text{mode}(\mu) = m, \quad \text{std}(\mu) = \frac{s}{\sqrt{N-3}} \sqrt{\frac{\nu}{\nu-2}} = \frac{s}{\sqrt{N-5}},$$

where  $\nu$  is the “degrees of freedom” of the Student’s t-distribution. The marginal, and mean, mode and standard deviation of  $\sigma^2$  is

$$\begin{aligned}
P(\sigma^2 | D) &= \sum_{\mu} P(\mu, \sigma^2 | D) = \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\frac{1}{\sigma^2}\right)^{\alpha+1} \exp\left(-\frac{\beta}{\sigma^2}\right) \\
&= \text{Inverse-Gamma}\left(\sigma^2 \mid \alpha = \frac{N-3}{2}, \beta = \frac{Ns^2}{2}\right)
\end{aligned}$$

$$\mathbb{E}(\sigma^2) = \frac{\beta}{\alpha-1} = s^2 \frac{N}{N-5}, \quad \text{mode}(\sigma^2) = \frac{\beta}{\alpha+1} = s^2 \frac{N}{N-1}, \quad \text{std}(\sigma^2) = \frac{\beta}{(\alpha-1)\sqrt{\alpha-2}} = s^2 \frac{\sqrt{2}N}{(N-5)\sqrt{N-7}}$$

Model	Posterior
Binomial 	Beta 
Poisson 	Gamma 
Multinomial 	Dirichlet 
Exponential 	Gamma 
Normal 	Normal-Inverse-Gamma 

### 3 Priors, Regularization, AIC, BIC, LRT

#### 3.1 Improper and proper priors

Improper priors are not normalizable, i.e.  $\sum_{\theta} P(\theta) = \infty$

- Flat prior:  $P(\theta) = \text{const.}$  over any infinite domain.
- Uninformative priors (from transformation invariance or max-entropy principles)
  - Location parameter:  $P(m) = \text{const.}$ , on  $m \in (-\infty, +\infty)$ ,
  - Scale parameter:  $P(s) = \frac{\text{const.}}{s}$ , on  $s \in (0, \infty)$ ,
  - Probability parameter:  $P(p) = \frac{\text{const.}}{p(1-p)}$ , on  $p \in (0, 1)$ .

Proper priors are normalized, i.e.  $\sum_{\theta} P(\theta) = 1$ .

#### 3.2 Regularization

Regularized model training

- Data:  $D$ ,
- Model:  $M$  with parameters  $\theta$ , likelihood  $P(D | \theta)$
- Cost function  $= -\log P(D | \theta) = -L(\theta)$ , the log likelihood
- Penalty:  $\text{penalty}(\theta)$ , which is high for implausible  $\theta$  values.
- Regularized optimum:  $\theta_{\text{reg.opt.}} = \arg \min_{\theta} (-L(\theta) + \text{penalty}(\theta))$

#### 3.3 Linear regression

- Data:  $D = \{ (\{x_{i,k}\}_{k=1}^K, y_i) \}_{i=1}^N$ , where  $x_i$  (feature vector)  $\in \mathbb{R}^K$ ,  $y_i$  (predicted variable)  $\in \mathbb{R}$ .
- Parameters:  $\{b_k\}_{k=1}^K$ , where  $b_k$  (coefficient or weight)  $\in \mathbb{R}$ .
- Model:

$$\begin{aligned}
 y_i &= \sum_{k=1}^K x_{i,k} b_k + \varepsilon_i, \quad \text{with } P(\varepsilon_i) = \text{Normal}(\varepsilon_i | \mu = 0, \sigma^2 = \sigma^2) \\
 y &= Xb + \varepsilon \\
 &\text{or equivalently} \\
 P(y | X, b) &= \prod_{i=1}^N \text{Normal}(y_i | \mu = (Xb)_i, \sigma^2 = \sigma^2)
 \end{aligned}$$

- Log likelihood:  $L(b) = \log P(y | X, b) = -\frac{N}{2} \log(\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^N \left[ y_i - \sum_k x_{i,k} b_k \right]^2$
- Maximum likelihood estimate:  $b_{\text{MLE}} = \arg \max_b L(b) = (X^\top X)^{-1} X^\top y$ ,  $(\sigma^2)_{\text{MLE}} = \frac{1}{N} \|y - Xb\|^2$
- Regularization:
  - “L1 regularization”:  $\text{penalty}(b) = \alpha_1 \sum_k |b_k|$   
 $\Leftrightarrow$  Laplace prior:  $P(b_k) = \text{const.} \times e^{-\alpha_1 |b_k|} = \text{Laplace}(b_k | \text{loc} = 0, \text{scale} = 1/\alpha_1)$
  - “L2 regularization”:  $\text{penalty}(b) = \frac{\alpha_2}{2} \sum_k (b_k)^2$   
 $\Leftrightarrow$  Normal prior:  $P(b_k) = \text{const.} \times e^{-\alpha_2 (b_k)^2/2} = \text{Normal}(b_k | \mu = 0, \sigma^2 = 1/\alpha_2)$
  - “Elastic net regularization”:  $\text{penalty}(b) = \alpha_1 \sum_k |b_k| + \frac{\alpha_2}{2} \sum_k (b_k)^2$

The “hyperparameters”  $\alpha_1$  and  $\alpha_2$  can be optimized using “Leave-one-out” or “M-fold” cross-validation.

### 3.4 Model comparison with asymptotic metrics

Maximum likelihood results from two models

- Data  $D = \{x_i\}_{i=1}^N$
- Null model:  $M_0$  with parameters  $\theta_0$ , and  $L_0(\theta_0) = P(D | \theta_0, M_0)$ ,  $\theta_{0,\text{MLE}} = \text{argmax}_{\theta_0} L_0(\theta_0)$
- Alternate model:  $M_1$  with parameters  $\theta_1$ , and  $L_1(\theta_1) = P(D | \theta_1, M_1)$ ,  $\theta_{1,\text{MLE}} = \text{argmax}_{\theta_1} L_1(\theta_1)$

Akaike Information Criterion (AIC)

- $\text{AIC}(M_i) = -2 \left[ L_i(\theta_{i,\text{MLE}}) - \dim(\theta_i) \right]$  for both  $i = 0, 1$  models.
- If  $\text{AIC}(M_1) < \text{AIC}(M_0)$ , then  $M_1$  is more plausible.

Bayesian Information Criterion (BIC)

- $\text{BIC}(M_i) = -2 \left[ L_i(\theta_{i,\text{MLE}}) - \frac{\ln(N)}{2} \dim(\theta_i) \right]$  for both  $i = 0, 1$  models.
- If  $\text{BIC}(M_1) < \text{BIC}(M_0)$ , then  $M_1$  is more plausible.

Likelihood Ratio Test (LRT)

- $\log \text{LR} = \log \frac{P(D | M_1, \theta_{1,\text{MLE}})}{P(D | M_0, \theta_{0,\text{MLE}})} = L_1(\theta_{1,\text{MLE}}) - L_0(\theta_{0,\text{MLE}})$
- $\text{LRT pvalue} = 1 - \text{cdf } \chi^2 \left( 2 \log \text{LR} \mid \text{dof} = \dim(\theta_1) - \dim(\theta_0) \right)$ , where  $\text{cdf } \chi^2(\dots \mid \text{dof} = d)$  is the cumulative distribution function of the  $\chi^2$  distribution with degrees of freedom  $d$ .

Model evidence

- $P(D | M_i) = \sum_{\theta_i} P(D | \theta_i, M_i) \approx \exp \left( -\frac{1}{2} \text{IC} \right)$ ,
- where IC can be either AIC or BIC (or WAIC, WBIC)
- Under uniform prior (i.e.  $P(M_0) = P(M_1)$ ), the posterior probability of the alternate model being correct is

$$P(M_1 | D) \approx \frac{\exp \left( -\frac{1}{2} \text{IC}_1 \right)}{\exp \left( -\frac{1}{2} \text{IC}_0 \right) + \exp \left( -\frac{1}{2} \text{IC}_1 \right)}$$

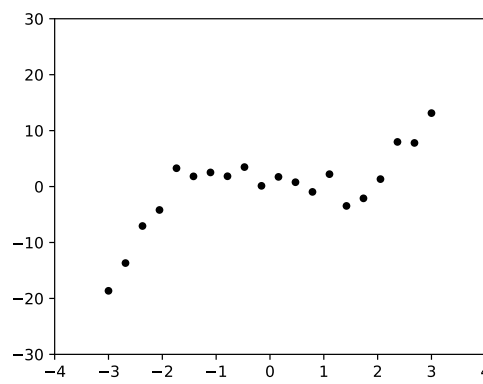
### 3.5 Example: Linear regression

- Data:  $D = \{(x_i, y_i)\}_{i=1}^N$  (generated from  $y = 1 - 3x - x^2/2 + x^3 + \varepsilon$  with  $\text{std}(\varepsilon) = 2$ ).

```

1 import numpy as np
2 from numpy.polynomial.polynomial import polyval
3 from scipy.stats import norm
4
5 c_true = [1, -3, -0.5, 1]
6 sigma_true = 2
7 x_data = np.linspace(-3, 3, 20)
8 y_data = [polyval(x, c_true) + norm.rvs(loc=0, scale=sigma_true)
9           for x in x_data]

```



- Models:  $M_K$ :  $K = 0, 1, 2, \dots$ -degree polynomial, parameters:  $c = (c_0, c_1, \dots, c_K)$ .
- “Linear features”:  $X_i := (1, x_i, (x_i)^2, (x_i)^3, \dots, (x_i)^K)$ .

```

1 def generate_polynomial_features(x_data, degree):
2     K = degree
3     N = len(x_data)
4     X = np.zeros([N, K+1])
5     for i, x in enumerate(x_data):
6         for k in range(0, K+1, 1):
7             X[i, k] = x**k
8     return X

```

- Likelihood:  $y = \sum_{k=0}^K X_{i,k} c_k + \varepsilon$ , with  $P(\varepsilon) = \text{Normal}(\varepsilon \mid 0, \sigma^2)$

```

1 def log_likelihood(X, y, c, sigma2):
2     N = len(y_data)
3     log_like = 0
4     log_like += - N/2.0 * np.log(sigma2)
5     log_like += - 1.0/(2 * sigma2) * vector_norm(y - X.dot(c))**2
6     return log_like

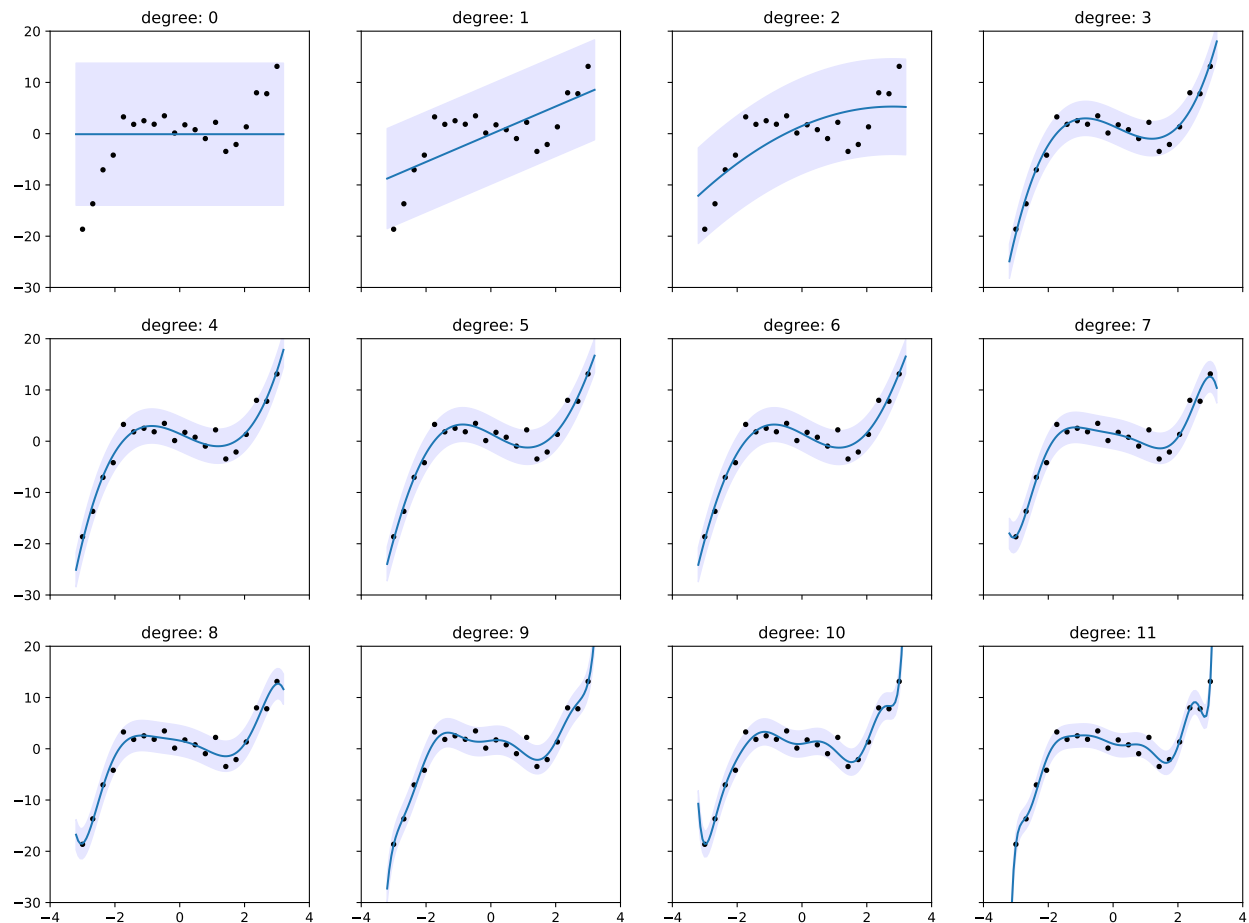
```

- MLE solution:  $c_{\text{MLE}} = (X^\top X)^{-1} X^\top y$ ,  $(\sigma^2)_{\text{MLE}} = \frac{1}{N} \|y - X c_{\text{MLE}}\|^2$

```

1 def fit_MLE_ploynomial(X, y):
2     c_MLE = inv(X.T.dot(X)).dot(X.T).dot(y_data)
3     sigma2_MLE = 1.0/len(y) * vector_norm(y - X.dot(c_MLE))**2
4     return c_MLE, sigma2_MLE

```





- $AIC_k = -2[L_k - (k + 2)]$

```

1 def AIC(X, y, c, sigma2):
2     dim = len(c) + 1
3     loglike = log_likelihood(X, y, c, sigma2)
4     return -2 * (loglike - dim)

```

- $BIC_k = -2[L_k - \frac{\log N}{2}(k + 2)]$

```

1 def BIC(X, y, c, sigma2):
2     N = len(y)
3     dim = len(c) + 1
4     loglike = log_likelihood(X, y, c, sigma2)
5     return -2 * (loglike - np.log(N)/2.0 * dim)

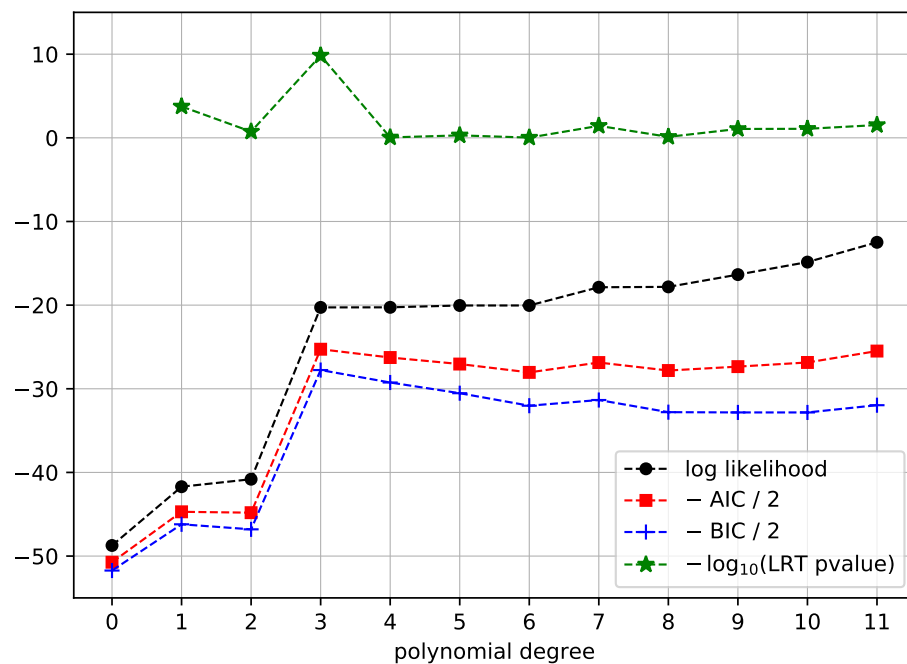
```

- $LRT\ pvalue_k = 1 - cdf\ \chi^2(2(L_k - L_{k-1}) \mid dof = 1)$   
(calculated against the model with one less degree)

```

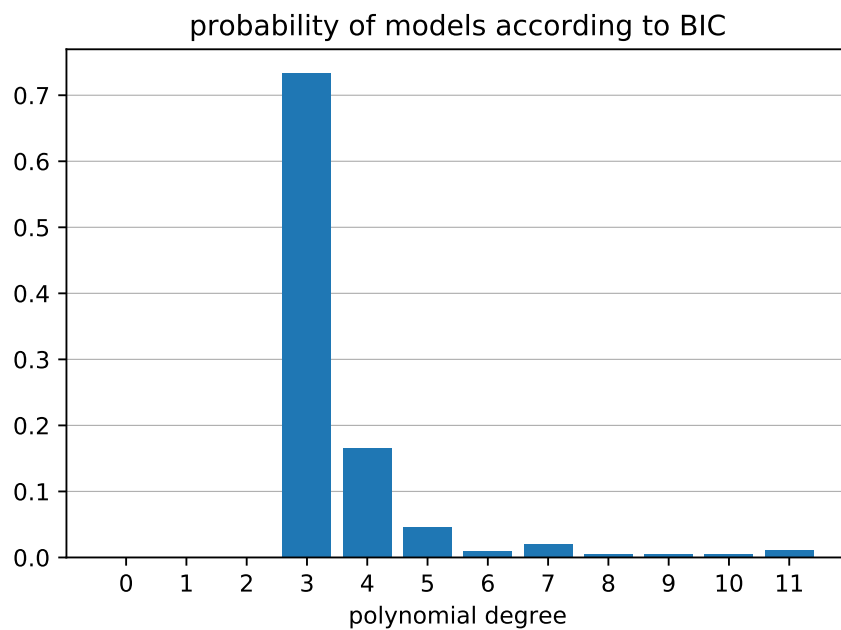
1 from scipy.stats import chi2
2
3 pvalues = [np.nan]
4 for deg in degrees[1:]:
5     L1 = loglikes[deg]
6     L0 = loglikes[deg-1]
7     logLR = L1 - L0
8     dof = 1
9     pvalue = chi2.sf(2*logLR, dof)
10    pvalues.append(pvalue)

```



- BIC weights,  $P(D | M_k) \approx e^{-\text{BIC}_k/2} / \sum_{k'=0}^K e^{-\text{BIC}_{k'}/2}$

```
1 def BIC_weights(BICs):  
2     BICs = np.array(BICs)  
3     w = BICs - np.min(BICs) # for numerical stability  
4     w = np.exp(-0.5*(w))  
5     w /= np.sum(w)  
6     return w
```



## 4 Graphical models

### 4.1 Elements

- $P(x, y) = P(y | x)P(x)$  is represented by

$$x \longrightarrow y$$

- $P(x, y, z) = P(y | z, x)P(z | x)P(x)$ .

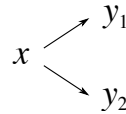
If  $P(y | z, x) = P(y | z)$ , then it is represented by a **chain**

$$x \longrightarrow z \longrightarrow y$$

Note:  $y \perp\!\!\!\perp x | z$ , but  $y \not\perp\!\!\!\perp x | \emptyset$ .

- $P(x, y_1, y_2) = P(y_1, y_2 | x)P(x)$ .

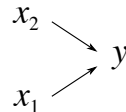
If  $P(y_1, y_2 | x) = P(y_1 | x)P(y_2 | x)$ , then it is represented by a **fork**



Note:  $y_1 \perp\!\!\!\perp y_2 | x$ , but  $y_1 \not\perp\!\!\!\perp y_2 | \emptyset$ .

- $P(x_1, x_2, y) = P(y | x_1, x_2)P(x_1, x_2)$ .

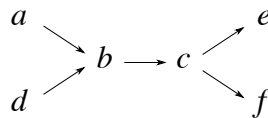
If  $P(x_1, x_2) = P(x_1)P(x_2)$ , then it is represented by a **collider**



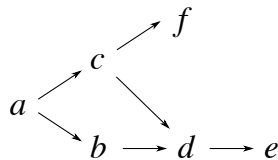
Note:  $x_1 \perp\!\!\!\perp x_2 | \emptyset$ , but  $x_1 \not\perp\!\!\!\perp x_2 | y$  (!).

#### Examples

- $P(a, b, c, d, e) = P(a)P(d)P(b | a, d)P(c | b)P(e | c)P(f | c)$  is represented by



- $P(a, b, c, d, e, f) = P(a)P(c | a)P(b | a)P(f | c)P(d | b, c)P(e | d)$  is represented by

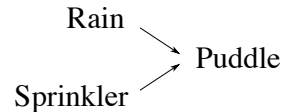


## 4.2 Real-life examples

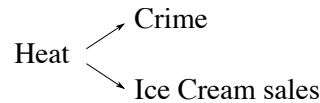
- Fire causes Smoke, Smoke causes Alarm to set off, but given Smoke, there's no correlation between Fire and Alarm, i.e.  $\text{Fire} \perp\!\!\!\perp \text{Alarm} \mid \text{Smoke}$ . This is represented by a chain

$$\text{Fire} \longrightarrow \text{Smoke} \longrightarrow \text{Alarm}$$

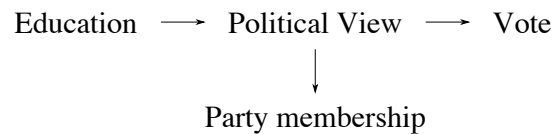
- Both rain and the Sprinkler can cause the formation of a Puddle, they are however independent (until we observe the Puddle), i.e.  $\text{Rain} \perp\!\!\!\perp \text{Sprinkler} \mid \emptyset$ . This is represented by a collider



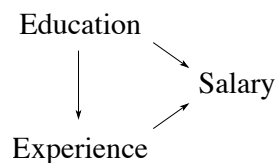
- Heat causes both Ice Cream sales and Crime to increase, but once we know there was a heatwave, they become independent, i.e.  $\text{Crime} \perp\!\!\!\perp \text{Ice Cream} \mid \text{Heat}$ . This is represented by a fork



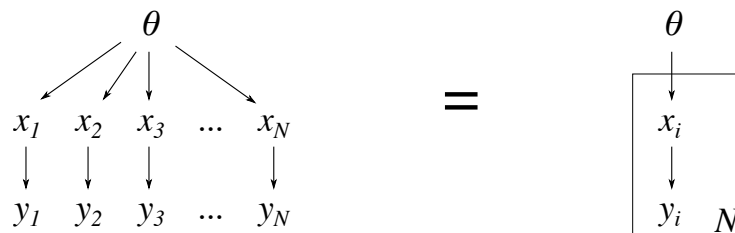
- Education affects Political View, which affects both Party membership and Voting behavior. This can be represented as



- Education and Experience both affect Salary, but Education also affects Experience. This can be represented as



## 4.3 Plate notation

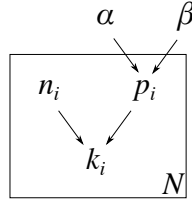


## 4.4 Hierarchical models

### Beta-Binomial model

- Data:  $D = \{(k_i, n_i)\}_{i=1}^N$ , where  $k_i$ (successes)  $\in \{0, 1, \dots, n_i\}$ , and  $n_i$ (attempts)  $\in \mathbb{N}$
- Model:
  - level 1: the parameters,  $p = \{p_i\}_{i=1}^N$ , where  $p_i \in [0, 1]$ , define  $P(k_i | n_i, p_i) = \text{Binomial}(k_i | n_i, p_i)$
  - level 2: the parameters,  $\alpha, \beta$  (both  $> 0$ ), define  $P(p_i | \alpha, \beta) = \text{Beta}(p_i | \alpha, \beta)$ .

This hierarchy can be represented as



- Joint likelihood:

$$P(D, p | \alpha, \beta) = \prod_{i=1}^N \left[ \text{Binom}(k_i | n_i, p_i) \text{Beta}(p_i | \alpha, \beta) \right]$$

- Marginal likelihood:

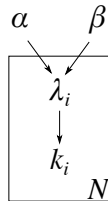
$$P(D | \alpha, \beta) = \prod_{i=1}^N \left[ \int dp_i \text{Binom}(k_i | n_i, p_i) \text{Beta}(p_i | \alpha, \beta) \right] = \prod_{i=1}^N \left[ \text{Beta-Binom}(k_i | n_i, \alpha, \beta) \right]$$

$$\text{where } \text{Beta-Binom}(k | n, \alpha, \beta) = \frac{\Gamma(n+1)\Gamma(\alpha+\beta)}{\Gamma(n+\alpha+\beta)} \frac{\Gamma(k+\alpha)}{\Gamma(k+1)\Gamma(\alpha)} \frac{\Gamma(n-k+\beta)}{\Gamma(n-k+1)\Gamma(\beta)}$$

### Gamma-Poisson model (aka. Negative Binomial model)

- Data:  $D = \{k_i\}_{i=1}^N$ , where  $k_i$ (events)  $\in \mathbb{N}$ .
- Model:
  - level 1: the parameters  $\lambda = \{\lambda_i\}_{i=1}^N$ , where  $\lambda_i > 0$ , define  $P(k_i | \lambda) = \text{Poisson}(k_i | \lambda_i)$
  - level 2: the parameters  $\alpha, \beta$  (both  $> 0$ ), define  $P(\lambda_i | \alpha, \beta) = \text{Gamma}(\lambda_i | \alpha, \beta)$ .

This hierarchy can be represented as



- Joint likelihood:

$$P(D, \lambda | \alpha, \beta) = \prod_{i=1}^N \left[ \text{Poisson}(k_i | \lambda_i) \text{Gamma}(\lambda_i | \alpha, \beta) \right]$$

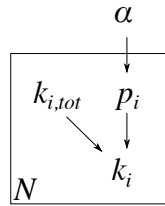
- Marginal likelihood:

$$\begin{aligned}
 P(D \mid \alpha, \beta) &= \prod_{i=1}^N \left[ \int d\lambda_i \text{Poisson}(k_i \mid \lambda_i) \text{Gamma}(\lambda_i \mid \alpha, \beta) \right] = \prod_{i=1}^N \left[ \text{Gamma-Poisson}(k_i \mid \alpha, \beta) \right] \\
 \text{where } \text{Gamma-Poisson}(k \mid \alpha, \beta) &= \frac{\Gamma(k + \alpha)}{\Gamma(k + 1)\Gamma(\alpha)} \left( \frac{1}{\beta + 1} \right)^k \left( \frac{\beta}{\beta + 1} \right)^\alpha = \\
 &= \text{NegativeBinom}(k \mid r, p) = \binom{k + r - 1}{k} p^k (1 - p)^r, \quad \text{with } r = \alpha, p = \frac{1}{\beta + 1}
 \end{aligned}$$

### Dirichlet-Multinomial model

- Data:  $D = \{k_i \in \mathbb{N}^M\}_{i=1}^N = \{(k_{i,1}, k_{i,2}, \dots, k_{i,M})\}_{i=1}^N$ , where  $k_{i,j}$  (number of outcome  $j$ )  $\in \mathbb{N}$
- Model:
  - level 1: the parameters  $p = \{p_i \in \mathbb{R}^M\}_{i=1}^N = \{(p_{i,1}, p_{i,2}, \dots, p_{i,M})\}_{i=1}^N$ , where  $p_{i,j}$  (probability of outcome  $j$  in sample  $i$ )  $> 0$ , and  $\sum_{j=1}^M p_{i,j} = 1, \forall i$ , define  $P(k_i \mid p_i) = \text{Multinomial}(k_i \mid k_{i,\text{tot}}, p_i)$ , where  $k_{i,\text{tot}} = \sum_{j=1}^M k_{i,j}$
  - level 2: the parameters  $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_M)$ , where each  $\alpha_j > 0$ , define  $P(p_i \mid \alpha) = \text{Dirichlet}(p_i \mid \alpha)$

This hierarchy can be represented as



- Joint likelihood:

$$P(D \mid p \mid \alpha) = \prod_{i=1}^N \left[ \text{Multinomial}(k_i \mid k_{i,\text{tot}}, p_i) \text{Dirichlet}(p_i \mid \alpha) \right]$$

- Marginal likelihood:

$$P(D \mid \alpha) = \prod_{i=1}^N \left[ \int dp_i \text{Multinomial}(k_i \mid k_{i,\text{tot}}, p_i) \text{Dirichlet}(p_i \mid \alpha) \right] = \prod_{i=1}^N \left[ \text{Dirichlet-Multinomial}(k_i \mid k_{i,\text{tot}}, \alpha) \right]$$

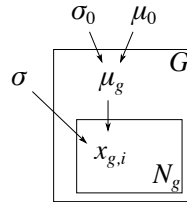
$$\text{where } \text{Dirichlet-Multinomial}(k \mid k_{\text{tot}}, \{\alpha_j\}_{j=1}^M) = \frac{\Gamma(k_{\text{tot}} + 1)\Gamma(\alpha_{\text{tot}})}{\Gamma(k_{\text{tot}} + \alpha_{\text{tot}})} \prod_{j=1}^M \frac{\Gamma(k_j + \alpha_j)}{\Gamma(k_j + 1)\Gamma(\alpha_j)},$$

$$\text{where } \alpha_{\text{tot}} = \sum_{j=1}^M \alpha_j$$

### Random Effect Model

- Data:  $D = \{x_g\}_{g=1}^G = \{(x_{g,1}, x_{g,2}, \dots, x_{g,N_g})\}_{g=1}^G$ , where  $x_{g,i} \in \mathbb{R}$  is measurement  $i$  in group  $g$ , and groups can be of different sizes  $N_g$ .
- Model:
  - level 1: The parameters  $\mu = \{\mu_g\}_{g=1}^G$  and  $\sigma^2$  define  $P(x_{g,i} \mid \mu_g, \sigma) = \text{Normal}(x_{g,i} \mid \mu_g, \sigma^2)$
  - level 2: The parameter  $\sigma_0^2$  define  $P(\mu_g \mid \mu_0, \sigma_0) = \text{Normal}(\mu_g \mid \mu_0, \sigma_0^2)$

This hierarchy can be represented as



- Joint likelihood:

$$P(D, \mu \mid \mu_0, \sigma_0, \sigma) = \prod_{g=1}^G \left[ \text{Normal}(\mu_g \mid \mu_0, \sigma_0^2) \times \prod_{i=1}^{N_g} \text{Normal}(x_{g,i} \mid \mu_g, \sigma^2) \right]$$

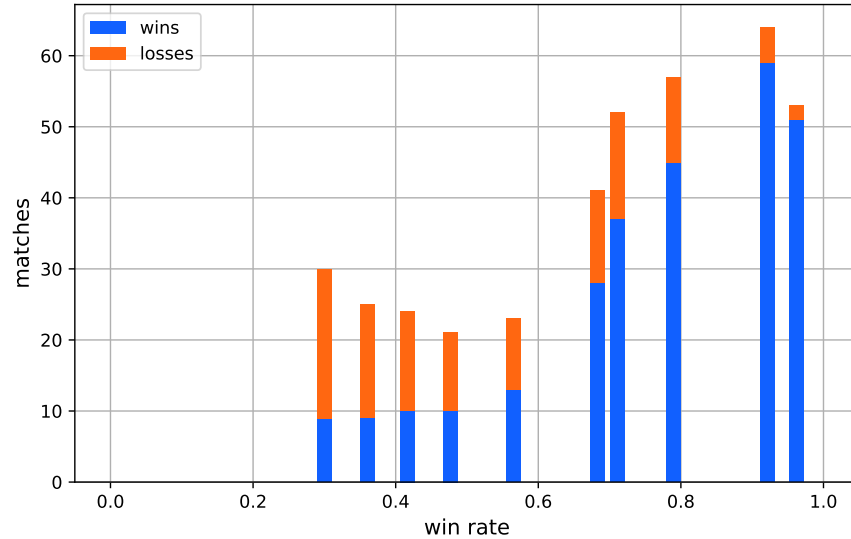
- Marginal likelihood:

$$\begin{aligned} P(D \mid \mu_0, \sigma_0, \sigma) &= \prod_{g=1}^G \int_{-\infty}^{+\infty} d\mu_g \left[ \text{Normal}(\mu_g \mid \mu_0, \sigma_0^2) \times \prod_{i=1}^{N_g} \text{Normal}(x_{g,i} \mid \mu_g, \sigma^2) \right] \\ &= (2\pi\sigma_0^2)^{-\frac{G}{2}} \prod_{g=1}^G \left[ \frac{1}{\sqrt{\xi + N_g}} (2\pi\sigma^2)^{-\frac{N_g-1}{2}} \exp \left( -\frac{N_g}{2\sigma^2} \left[ \frac{\xi}{\xi + N_g} (\mu_g - m_g)^2 + s_g^2 \right] \right) \right] \end{aligned}$$

$$\text{where} \quad \xi = \frac{\sigma^2}{\sigma_0^2}, \quad m_g = \frac{1}{N_g} \sum_{i=1}^{N_g} x_{g,i}, \quad s_g^2 = \frac{1}{N_g} \sum_{i=1}^{N_g} x_{g,i}^2 - m_g^2$$

## 4.5 Example: Beta-Binomial

Life-time performance 10 different boxers are collected. Wins ( $k_i$ ) and losses ( $n_i - k_i$ ) are tallied, and the observed win rate  $p_{i,obs} = k_i/n_i$  is calculated. This is shown below



We would like to determine the win rate of an “typical boxer”, i.e. the distribution of the win rate  $p$ . While the observed values  $p_{obs}$  are good estimates of the individual win rates, 5 boxers played less than 30 matches, while 5 played more than 40, which means the second group provides more information, and their observed win rates need to be taken with more certainty. The Beta-Binomial hierarchical model accounts for this inhomogeneity.

- Data:  $\{n_i\} = [24, 23, 30, 21, 25, 53, 41, 52, 64, 57]$ ,  $\{k_i\} = [10, 13, 9, 10, 9, 51, 28, 37, 59, 45]$ , with  $N = 10$ .
- Parameters:  $\alpha, \beta > 0$
- Model:

$$\log P(D \mid \alpha, \beta) = \sum_{i=1}^N \log \left( \text{Beta-Binom}(k_i \mid n_i, \alpha, \beta) \right)$$

$$\text{where } \log \left( \text{Beta-Binom}(k \mid n, \alpha, \beta) \right) = -f(n, \alpha + \beta) + f(k, \alpha) + f(n - k, \beta),$$

$$\text{where } f(k, \alpha) = \log \Gamma(k + \alpha) - \log \Gamma(k + 1) - \log \Gamma(\alpha)$$

which can be implemented as

```

1 from scipy.special import gammaln
2
3 def log_three_gamma_term(k, a):
4     return gammaln(k+a) - gammaln(k+1) - gammaln(a)
5
6 def log_beta_binom(k, n, a, b):
7     target = 0
8     target += - log_three_gamma_term(n, a+b)
9     target += log_three_gamma_term(k, a)
10    target += log_three_gamma_term(n-k, b)
11    return target

```



```

12
13 def log_likelihood(k, n, a, b):
14     target = 0
15     for ki, ni in zip(k, n):
16         target += log_beta_binom(ki, ni, a, b)
17     return target

```

- While assuming flat priors for  $\alpha$  and  $\beta$ , we can numerically calculate their joint posterior and means.

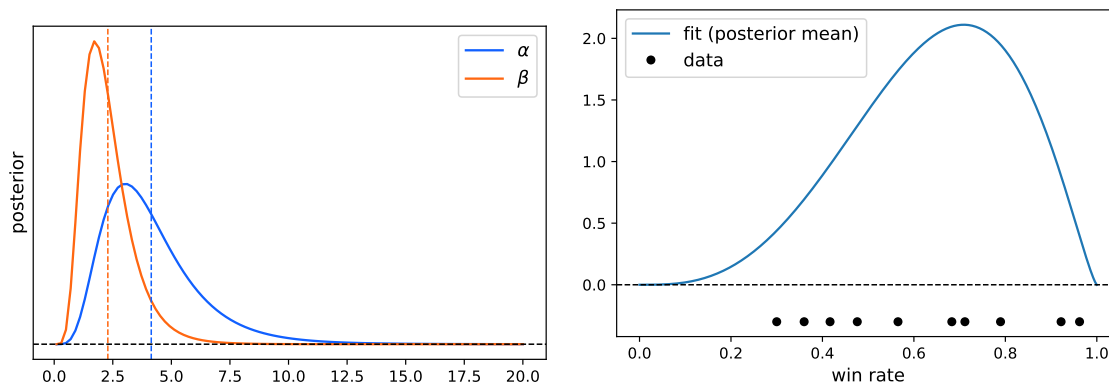
```

1 import numpy as np
2 import pandas as pd
3
4 a_arr, b_arr = np.meshgrid(np.linspace(0.1, 20, 100),
5                             np.linspace(0.1, 20, 100))
6 a_arr = a_arr.flatten()
7 b_arr = b_arr.flatten()
8
9 loglike = []
10 for a, b in zip(a_arr, b_arr):
11     loglike.append(log_likelihood(k, n, a, b))
12
13 df = pd.DataFrame({
14     'a': a_arr,
15     'b': b_arr,
16     'loglike': loglike
17 })
18
19 df['pstar'] = np.exp(df['loglike'] - df['loglike'].max())
20 Z = df['pstar'].sum()
21 df['posterior'] = df['pstar'] / Z
22
23 post_a = df.groupby(by='a')['posterior'].sum().reset_index()
24 post_b = df.groupby(by='b')['posterior'].sum().reset_index()
25
26 a_mean = (post_a['posterior'] * post_a['a']).sum()
27 b_mean = (post_b['posterior'] * post_b['b']).sum()

```

giving  $\mathbb{E}(\alpha | D) = 4.142$ ,  $\mathbb{E}(\beta | D) = 2.289$ .

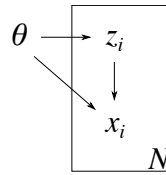
Their (marginal) posteriors and the distribution of the win rate (for the mean  $\alpha$  and  $\beta$  values) are below.



## 5 Hidden variables, EM, Mixture models

### 5.1 Definitions

- Known values:
  - Observations (or data),  $D = \{x_i\}_{i=1}^N$
- Unknown values:
  - Parameters:  $\theta = \{\theta_k\}_{k=1}^K$
  - Hidden variables (or hidden data):  $Z = \{z_i\}_{i=1}^N$  (i.e.  $Z$  is as numerous as  $D$ )



### 5.2 Expectation Maximization

Goal

- Given the likelihood  $P(D | Z, \theta)$ , and prior on hidden variables  $P(Z | \theta)$ ,
- The joint is  $P(D, Z | \theta) = P(D | Z, \theta) P(Z | \theta)$
- The marginal is  $P(D | \theta) = \sum_Z P(D, Z | \theta)$
- We wish to find  $\theta$  that maximizes the marginal, i.e.

$$\theta^{\text{MLE}} = \arg \max_{\theta} \left[ \log P(D | \theta) \right] = \arg \max_{\theta} \left[ \log \left( \sum_Z P(D, Z | \theta) \right) \right]$$

- Direct numerical optimization is usually feasible, but the EM method is often faster.

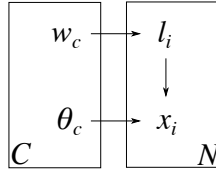
Expectation Maximization (EM) algorithm:

1. Start with realistic  $\theta = \theta^{\text{old}}$
  2. E-step: Calculate  $P(Z | D, \theta^{\text{old}}) = \frac{P(D, Z | \theta^{\text{old}})}{\sum_{Z'} P(D, Z' | \theta^{\text{old}})}$
  3. M-step: Find the optimal  $\theta = \theta^{\text{new}}$  that maximizes  $\sum_Z P(Z | D, \theta^{\text{old}}) \log P(D, Z | \theta)$
  4. Set  $\theta^{\text{old}} \leftarrow \theta^{\text{new}}$ , check convergence, and return to E-step if needed.
- The one-liner iteration formula is based on the joint  $P(D, Z | \theta)$ :

$$\theta^{\text{new}} = \arg \max_{\theta} \left[ \sum_Z \frac{P(D, Z | \theta^{\text{old}})}{\sum_{Z'} P(D, Z' | \theta^{\text{old}})} \log P(D, Z | \theta) \right]$$

### 5.3 Mixture models

- Data:  $D = \{x_i\}_{i=1}^N$
- Parameters:  $\theta, w$ 
  - Components:  $c \in \{1, 2, \dots, C\}$
  - Parameters for each component  $\theta = \{\theta_c\}_{c=1}^C$
  - Mixing proportions:  $w = \{w_c \in [0, 1]\}_{c=1}^C$ , such that  $\sum_c w_c = 1$ .
- Hidden variables: labels  $L = \{l_i\}_{i=1}^N$ , where  $l_i \in \{1, 2, \dots, C\}$
- Model: mixture of distinct distributions
  - Generative distribution for each component:  $P(x_i | l_i = c, \theta) = P(x_i | \theta_c)$
  - Probability of an observation coming from a component:  $P(l_i = c) = w_c$



- Joint

$$P(D, L | \theta, w) = P(D | L, \theta) P(L | w) = \prod_{i=1}^N P(x_i | \theta_{l_i}) w_{l_i}$$

- Marginal

$$P(D | \theta, w) = \sum_L P(D, L | \theta, w) = \prod_{i=1}^N \left[ \sum_{c=1}^C w_c P(x_i | \theta_c) \right]$$

EM algorithm

1. Start with initial values:  $\theta^{\text{old}}, w^{\text{old}}$
2. In E-step, calculate

$$P(l_i = c | x_i, \theta^{\text{old}}) = \frac{w_c^{\text{old}} P(x_i | \theta_c^{\text{old}})}{\sum_{c'} w_{c'}^{\text{old}} P(x_i | \theta_{c'}^{\text{old}})} =: r_{i,c}$$

3. In M-step, calculate

$$\begin{aligned} w_c^{\text{new}} &= \frac{1}{N} \sum_{i=1}^N r_{i,c} \\ \theta_c^{\text{new}} &= \arg \max_{\theta_c} \left[ \sum_{i=1}^N r_{i,c} \log P(x_i | \theta_c) \right] \\ &= \text{MLE of } \theta \text{ with data weights } \{r_{i,c}\}_{i=1}^N \end{aligned}$$

## 5.4 Gaussian Mixture Model

(aka. GMM or “soft K-means clustering”)

- Data:  $\{x_i\}_{i=1}^N$ , where  $x_i \in \mathbb{R}^d$  (point in  $d$  dimension)
- Parameters:
  - Clusters:  $k \in \{1, 2, \dots, K\}$
  - Mixture proportions:  $\{w_k \in [0, 1]\}_{k=1}^K$ , where  $\sum_k w_k = 1$
  - Cluster means:  $\mu = \{\mu_k \in \mathbb{R}^d\}_{k=1}^K$
  - Cluster covariances:  $\Sigma = \{\Sigma_k \in \mathbb{R}^{d \times d}, \text{positive definite}\}_{k=1}^K$ .
- Hidden variables: cluster labels,  $L = \{l_i\}_{i=1}^N$ , where  $l_i \in \{1, 2, \dots, K\}$
- Model

$$P(l_i = k) = w_k$$

$$P(x_i | l_i = k, \mu, \Sigma) = \text{Normal}(x_i | \mu_k, \Sigma_k) = \frac{1}{\sqrt{\det(2\pi\Sigma_k)}} \exp\left(-\frac{1}{2}(x_i - \mu_k)^\top (\Sigma_k)^{-1} (x_i - \mu_k)\right)$$

- Marginal:

$$P(D | \mu, \Sigma, w) = \prod_{i=1}^N \left[ \sum_{k=1}^K w_k \text{Normal}(x_i | \mu_k, \Sigma_k) \right]$$

- EM algorithm

1. Choose realistic  $w^{\text{old}}, \mu^{\text{old}}, \Sigma^{\text{old}}$  initial values.
2. In E-step, calculate

$$r_{i,k} = \frac{w_k^{\text{old}} \text{Normal}(x_i | \mu_k^{\text{old}}, \Sigma_k^{\text{old}})}{\sum_{k'} w_{k'}^{\text{old}} \text{Normal}(x_i | \mu_{k'}^{\text{old}}, \Sigma_{k'}^{\text{old}})}$$

3. In M-step, calculate

$$w_k^{\text{new}} = \frac{1}{N} \sum_{i=1}^N r_{i,k}$$

$$\mu_k^{\text{new}} = \frac{1}{w_k^{\text{new}} N} \sum_{i=1}^N r_{i,k} x_i$$

$$\Sigma_k^{\text{new}} = \frac{1}{w_k^{\text{new}} N} \sum_{i=1}^N r_{i,k} (x_i - \mu_k^{\text{new}})(x_i - \mu_k^{\text{new}})^\top$$

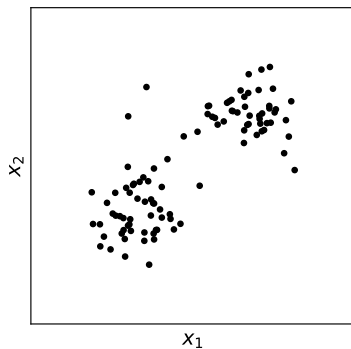
The following python class implements the EM algorithm for fitting GMM.

```

1 class GmmEm:
2     def __init__(self, x):
3         self.x = np.array(x)
4         self.N, self.d = self.x.shape
5         self.K = None
6         self.weights = None
7         self.means = None
8         self.covs = None
9
10    def initialize(self, K):
11        self.K = K
12        m0 = np.mean(x, axis=0)
13        cov0 = np.cov(x.T)
14
15        self.weights = [1.0/K] * K
16        self.means = multivariate_normal.rvs(mean=m0, cov=cov0, size=K)
17        cov_values, _ = np.linalg.eig(cov0)
18        self.covs = np.array([np.eye(self.d) * 0.1 * cov_values.max()
19                               for _ in range(K)])
20
21    def e_step(self):
22        r = []
23        for k in range(K):
24            r.append(self.weights[k] *
25                    multivariate_normal.pdf(self.x,
26                                            mean=self.means[k],
27                                            cov=self.covs[k]))
28
29        r = np.array(r).T
30        r_sum = np.einsum('ik->i', r)
31        r = np.einsum('ik,i->ik', r, 1.0/r_sum)
32        return r
33
34    def m_step(self, r):
35        weights_new = 1.0/N * np.einsum('ik->k', r)
36        means_new = 1.0/N * \
37            np.einsum('k,ik,id->kd',
38                    1.0/weights_new,
39                    r,
40                    self.x)
41        deviations = np.array([self.x - means_new[k] for k in range(self.K)])
42        covs_new = 1.0/N * \
43            np.einsum('k,ik,kid,kiD->kdD',
44                    1.0/weights_new,
45                    r,
46                    deviations,
47                    deviations)
48
49        self.weights = weights_new
50        self.means = means_new
51        self.covs = covs_new

```

Example: GMM in 2D with  $K = 2$

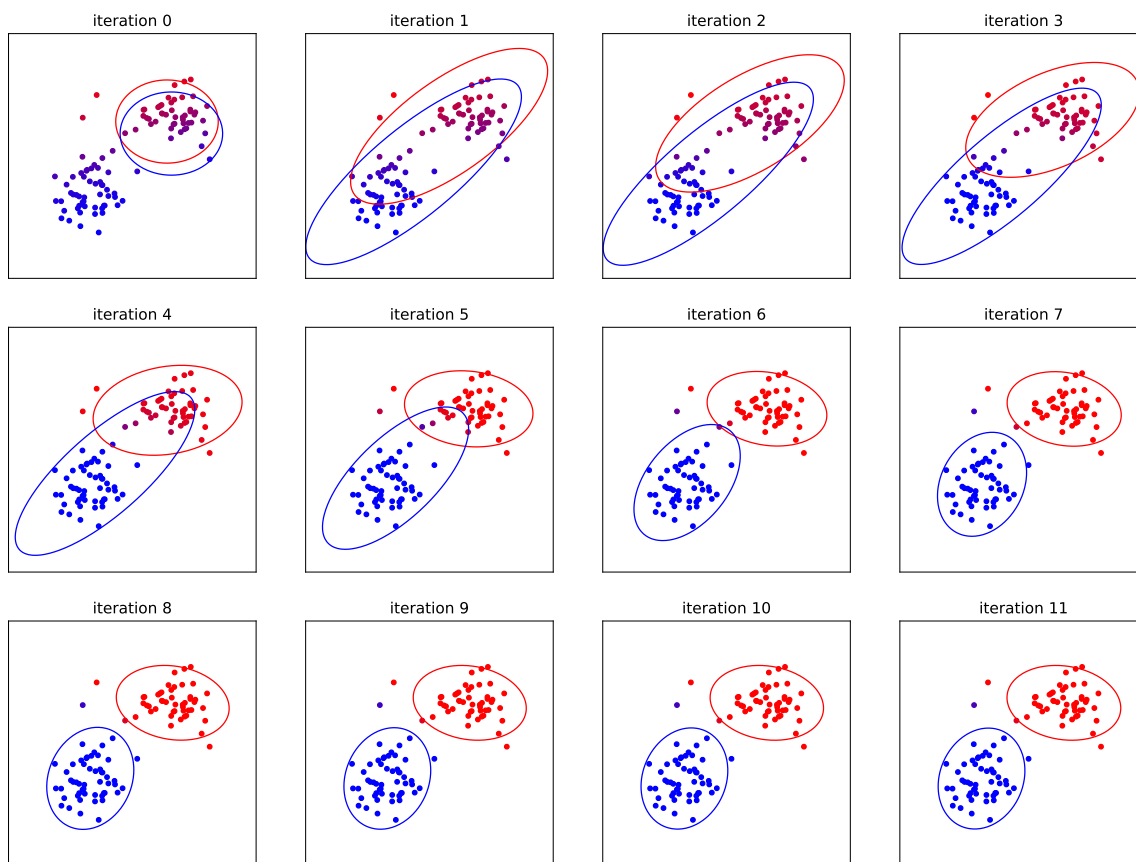


Iterating the E- and M-steps a couple of times, we arrive to the final set of  $r$  values.

```

1 gmm = GmmEm(x)
2
3 K = 2
4 gmm.initialize(K)
5 for it in range(12):
6     r = gmm.e_step()
7     gmm.m_step(r)

```



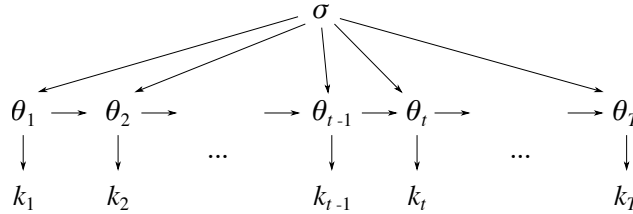
## 6 Curse of Dimensionality, Laplace approximation

### 6.1 High-dimensional example

- Data:  $D = \{k_t\}_{t=1}^T$ , where  $k_t \in \mathbb{N}$  is the number of influenza cases at small clinic on each day of the year ( $T = 365$ ).
- Parameters:
  - $\theta = \{\theta_t\}_{t=1}^T$ , with  $\theta_t = \log(\lambda_t)$  where  $\lambda_t > 0$  is the intensity of influenza on a given day  $t$ .
  - $\sigma > 0$ , the typical change  $\theta_t - \theta_{t-1}$ .
- Model:
  - Prior:  $P(\theta_t | \theta_{t-1}) = \text{Normal}(\theta_t | \theta_{t-1}, \sigma^2)$ , and  $P(\sigma) = \text{const.}$
  - Data generation process:  $P(k_t | \theta_t) = \text{Poisson}(k_t | \lambda = \exp(\theta_t))$
- Posterior:

$$P(\theta | D) = \frac{1}{Z} P^*(\theta | D) = \frac{1}{Z} \prod_{t=1}^T [P(\theta_t | \theta_{t-1}) P(k_t | \theta_t)]$$

with the understanding that “ $P(\theta_1 | \theta_0)$ ” = 1. Here  $Z$  is the normalization constant.



- Numerical solution would require evaluating  $P^*$  on a grid of different  $\theta$  values. Even, at the very extreme, when we consider only 2 values for each  $\theta_t$ , the number of evaluations becomes

$$2^{365} \approx 10^{109} > 10^{86} \text{ (the number of protons in the observable part of the universe),}$$

which makes it impossible to pursue this strategy.

### 6.2 Laplace approximation

- Goal: Determine posterior mean and variance of each parameter  $\theta_t$
- Challenge: The dimension of  $\theta = \{\theta_t\}_{t=1}^T$ , i.e.  $T$  is too high for direct numerical evaluation.
- Method: Approximate  $P^*(\theta | D)$  near its maximum with a multi-variate normal distribution.

$$P^*(\theta | D) \approx \text{Normal}(\theta | \mu, \Sigma) = \frac{1}{\sqrt{\det(2\pi\Sigma)}} \exp \left[ -\frac{1}{2}(\theta - \mu)^\top \Sigma^{-1}(\theta - \mu) \right]$$

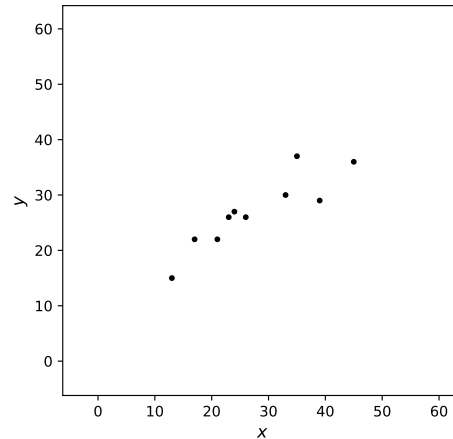
$$\text{where } \mu = \arg \max_{\theta} [\log P^*] \in \mathbb{R}^T$$

$$\Sigma = \left[ -\frac{d}{d\theta} \frac{d}{d\theta} \log P^* \right]_{\theta=\mu}^{-1} \in \mathbb{R}^{T \times T}$$

where  $\mu$  can be found with direct numerical or analytical minimization or an expectation maximization algorithm, and  $\Sigma$  can be evaluated analytically or approximated numerically.

### 6.3 Example: $(x, y)$ linear regression

- Data:  $D = \{D_x, D_y\}$ ,
  - where  $D_x = \{x_i\}_{i=1}^N = [21, 24, 17, 39, 23, 45, 33, 26, 13, 35]$ ,
  - and  $D_y = \{y_i\}_{i=1}^N = [22, 27, 22, 29, 26, 36, 30, 26, 15, 37]$



- Parameters:  $a$  (slope),  $b$  (intercept),  $\sigma^2$  (strength of  $y$ -noise), using flat priors, i.e.  $P(a, b, \sigma^2) = \text{const.}$
- Model:

$$P(D_y \mid D_x, a, b, \sigma^2) = \prod_{i=1}^N \text{Normal}(y_i \mid \mu(x_i), \sigma^2), \quad \text{where } \mu(x_i) = ax_i + b$$

- Unnormalized posterior:

$$\log P^*(a, b, \sigma^2 \mid D) = -\frac{N}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^N \left[ y_i - (ax_i + b) \right]^2$$

- MLE estimate:

$$\left. \begin{aligned} 0 &= \frac{\partial}{\partial a} \log P^* = \frac{1}{\sigma^2} \left[ \sum_i y_i x_i - a \sum_i x_i^2 - b \sum_i x_i \right] \\ 0 &= \frac{\partial}{\partial b} \log P^* = \frac{1}{\sigma^2} \left[ \sum_i y_i - a \sum_i x_i - bN \right] \\ 0 &= \frac{\partial}{\partial (\sigma^2)} \log P^* = -\frac{N}{2\sigma^2} + \frac{1}{2(\sigma^2)^2} \sum_i \left[ y_i - (ax_i + b) \right]^2 \end{aligned} \right\} \Rightarrow \begin{cases} a_{\text{MLE}} = (\overline{yx} - \bar{y}\bar{x}) / (\overline{x^2} - \bar{x}^2) \\ b_{\text{MLE}} = \bar{y} - a_{\text{MLE}} \bar{x} \\ (\sigma^2)_{\text{MLE}} = \frac{1}{N} \sum_i \left[ y_i - (a_{\text{MLE}} x_i + b_{\text{MLE}}) \right]^2 \end{cases}$$

where

$$\bar{x} = \frac{1}{N} \sum_i x_i = 27.6, \quad \bar{y} = \frac{1}{N} \sum_i y_i = 27.0, \quad \overline{x^2} = \frac{1}{N} \sum_i x_i^2 = 854.0, \quad \overline{yx} = \frac{1}{N} \sum_i y_i x_i = 798.9,$$

```

1 import numpy as np
2
3 def xy_linear_regression_MLE(x, y):
4     N = len(x)
5     ev_x = np.mean(x)
6     ev_y = np.mean(y)
7     ev_xx = np.mean(x * x)
8     ev_yx = np.mean(y * x)
9     ev_yy = np.mean(y * y)

```



```

10
11     a_MLE = (ev_yx - ev_y * ev_x) / (ev_xx - ev_x**2)
12     b_MLE = ev_y - a_MLE * ev_x
13     sigma2_MLE = 1.0/N * np.sum((y - (a_MLE * x + b_MLE))**2)
14
15     return a_MLE, b_MLE, sigma2_MLE

```

giving  $\mu = (a_{\text{MLE}}, b_{\text{MLE}}, (\sigma^2)_{\text{MLE}})$ , with  $a_{\text{MLE}} = 0.5822$ ,  $b_{\text{MLE}} = 10.93$ ,  $(\sigma^2)_{\text{MLE}} = 7.737$ .

- Laplace approximation:

First, we calculate all second order derivatives at the MLE point:

$$\begin{aligned}
 \frac{\partial}{\partial a} \frac{\partial}{\partial a} \log P^* &= -\frac{N}{\sigma^2} \overline{x^2} \\
 \frac{\partial}{\partial b} \frac{\partial}{\partial a} \log P^* &= \frac{\partial}{\partial a} \frac{\partial}{\partial b} \log P^* = -\frac{N}{\sigma^2} \bar{x} \\
 \frac{\partial}{\partial b} \frac{\partial}{\partial b} \log P^* &= -\frac{N}{\sigma^2} \\
 \frac{\partial}{\partial(\sigma^2)} \frac{\partial}{\partial a} \log P^* &= \frac{\partial}{\partial a} \frac{\partial}{\partial(\sigma^2)} \log P^* = 0 \\
 \frac{\partial}{\partial(\sigma^2)} \frac{\partial}{\partial b} \log P^* &= \frac{\partial}{\partial b} \frac{\partial}{\partial(\sigma^2)} \log P^* = 0 \\
 \frac{\partial}{\partial(\sigma^2)} \frac{\partial}{\partial(\sigma^2)} \log P^* &= -\frac{N}{2(\sigma^2)^2}
 \end{aligned}$$

from which we construct the second derivative at the MLE point:

$$-\nabla \nabla \log P^*|_{\text{MLE}} = \frac{N}{(\sigma^2)_{\text{MLE}}} \begin{bmatrix} \overline{x^2} & \bar{x} & 0 \\ \bar{x} & 1 & 0 \\ 0 & 0 & \frac{1}{2(\sigma^2)_{\text{MLE}}} \end{bmatrix}$$

```

1 minus_dd_logPstar = N / sigma2_MLE * \
2 np.array([
3     [ev_xx, ev_x, 0],
4     [ev_x, 1, 0],
5     [0, 0, 1/(2*sigma2_MLE)]]
6 ])
7 Sigma = - np.linalg.inv(minus_dd_logPstar)

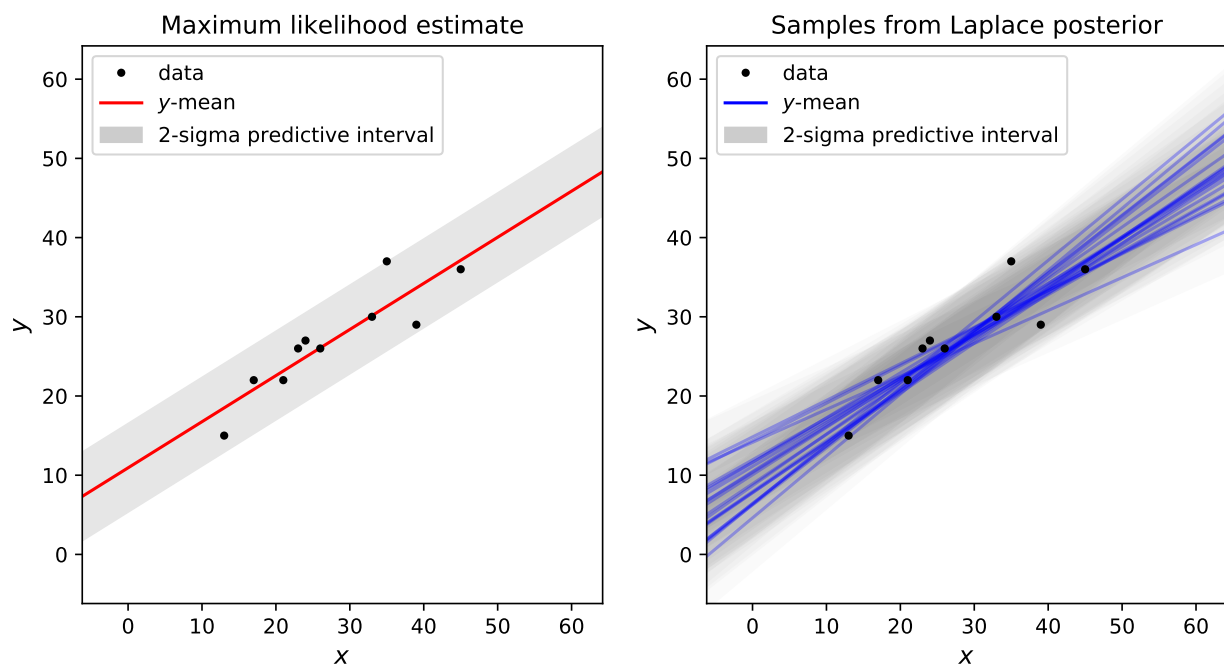
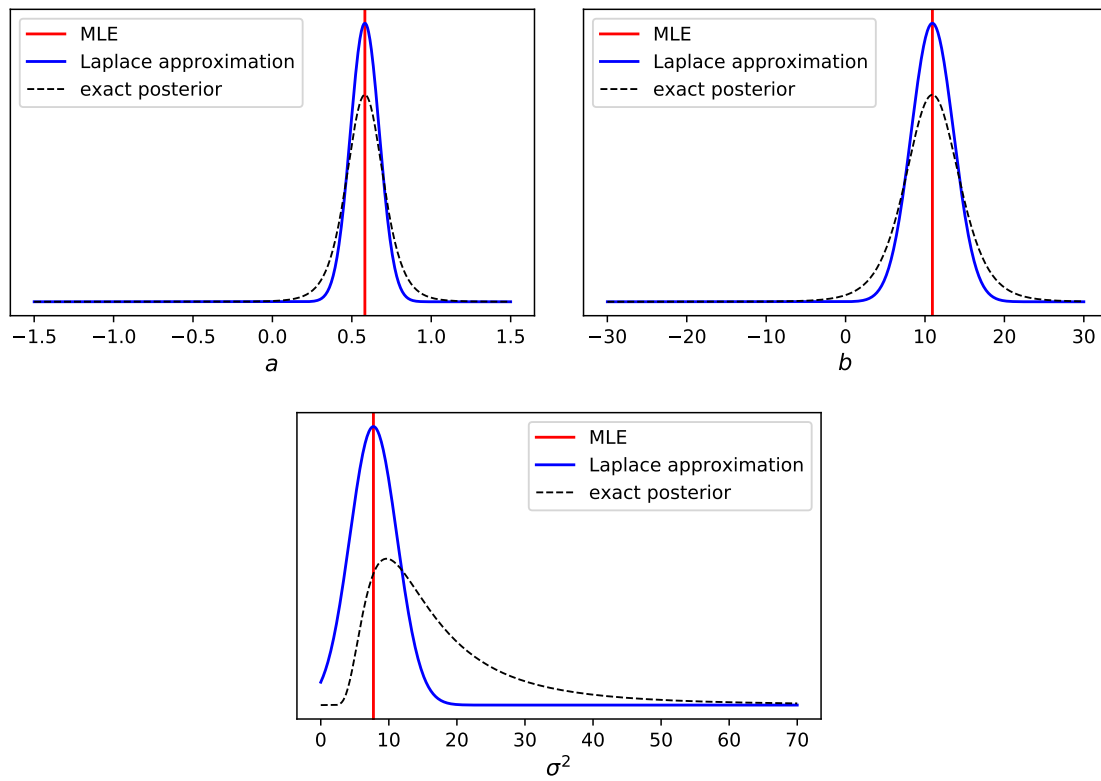
```

giving

$$\Sigma = \left[ -\nabla \nabla \log P^*|_{\text{MLE}} \right]^{-1} = \begin{bmatrix} 0.0839 & -0.2315 & 0.0 \\ -0.2315 & 7.1634 & 0.0 \\ 0.0 & 0.0 & 11.197 \end{bmatrix}$$

and

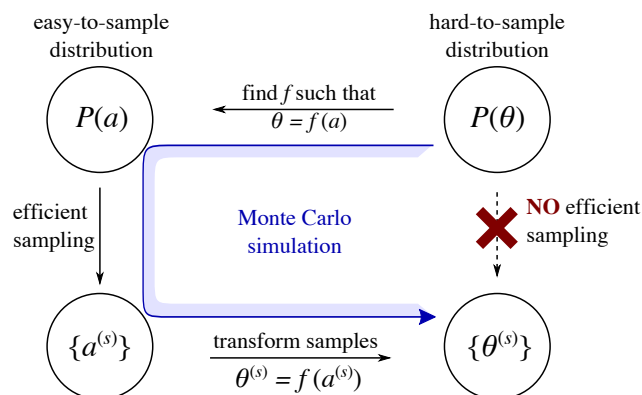
$$\begin{aligned}
 \text{Var}(a \mid D) &\approx \Sigma_{1,1} = 0.0839, \\
 \text{Var}(b \mid D) &\approx \Sigma_{2,2} = 7.1634, \\
 \text{Var}(\sigma^2 \mid D) &\approx \Sigma_{3,3} = 11.197
 \end{aligned}$$



## 7 Monte Carlo methods

### 7.1 Monte Carlo simulation

- **Goal:** Analyze complicated distributions by drawing samples from them:  $P(\theta) \mapsto \{\theta^{(s)}\}$ .
- **Challenge:** From the vast majority of distributions, we don't know how to draw samples efficiently.
- **Method:**
  1. Draw samples from an easy-to-sample distribution.
  2. Transform the drawn values so they become samples from the distribution of question.



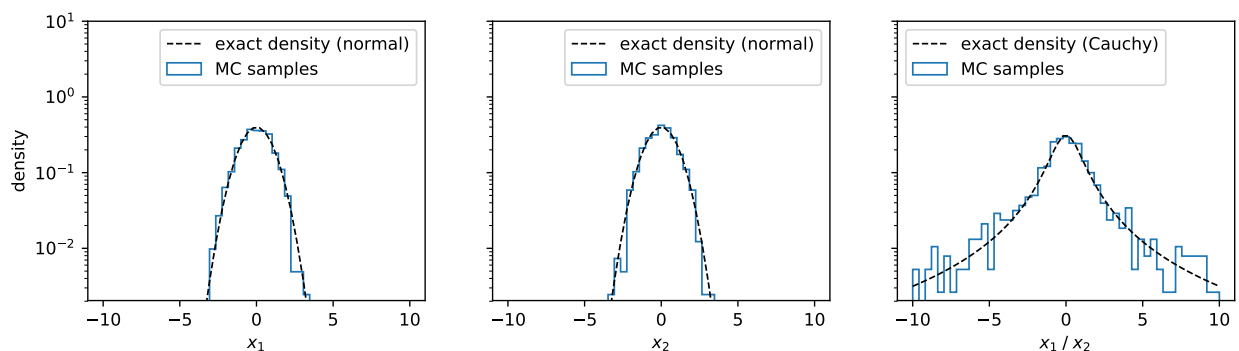
**Example:** Ratio of two normal variables

- Input:  $P(x_1) = \text{Normal}(x_1 \mid 0, 1)$ ,  $P(x_2) = \text{Normal}(x_2 \mid 0, 1)$
- Output:  $y := x_1/x_2$ ,  $P(y) = ?$
- MC method:

```

1 from scipy.stats import norm
2
3 samples = 1000
4 X1 = norm.rvs(loc=0, scale=1, size=samples)
5 X2 = norm.rvs(loc=0, scale=1, size=samples)
6 Y = X1 / X2

```



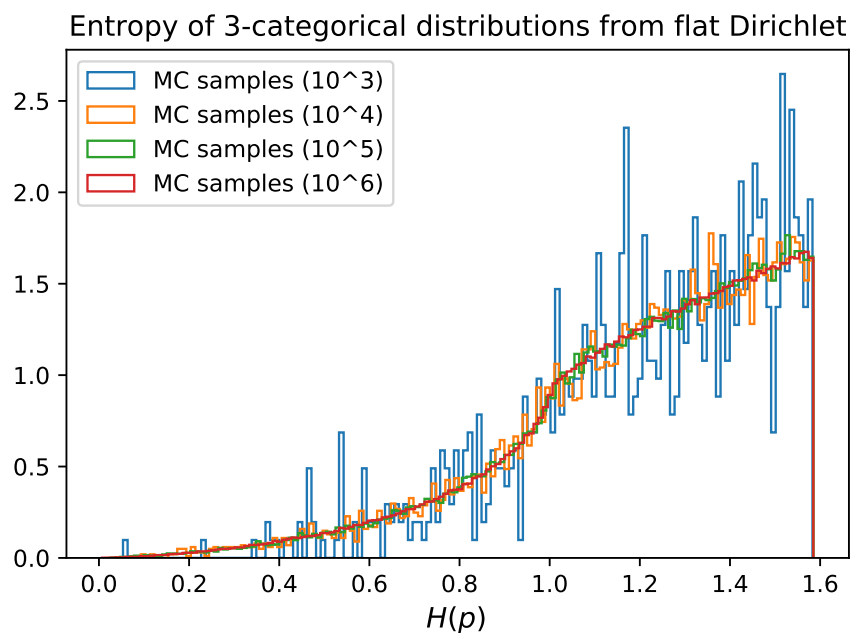
**Example:** Entropy of distributions from flat Dirichlet

- Input:  $p = (p_1, p_2, p_3) \in [0, 1]^{\times 3}$ , such that  $\sum_{k=1}^3 p_k = 1$ , where  $P(p) = \text{Dirichlet}(p \mid \alpha = (1, 1, 1))$
- Output:  $h := H(p) = -\sum_k p_k \log_2 p_k$ ,  $P(h) = ?$
- MC method:

```

1 import numpy as np
2 from scipy.stats import dirichlet
3
4 def entropy(p):
5     h = 0
6     for pk in p:
7         if pk > 0:
8             h += - pk * np.log2(pk)
9     return h
10
11 alpha = (1,1,1)
12 sample_size = 10_000
13 p_samples = dirichlet.rvs(alpha, size=sample_size)
14 h_samples = []
15 for p in p_samples:
16     h_samples.append(entropy(p))

```



**Example:** Monty Hall problem

- Input:
  - In a game show, there are three doors  $D = \{1, 2, 3\}$
  - A reward is placed behind door  $r \in D$  uniformly randomly, i.e.  $P(r) = \text{uniform}$ .
  - The player picks a door  $p_1 \in D$  uniformly randomly, i.e.  $P(p_1) = \text{uniform}$  (He doesn't know where the reward is.)
  - The game show master (who knows where the reward is), picks a door from the remaining two that does not have the reward, and opens it,  $o \in D_{\text{can open}}$ , where  $D_{\text{can open}} = D \setminus (\{r\} \cup \{p_1\})$  randomly, i.e.  $P(o) = \text{uniform}$ .
  - The game show master offers the player another chance to pick one of the remaining two doors  $D_{\text{remaining}} = D \setminus \{o\}$ . He can stick to his first choice, i.e.  $p_2 = p_1$ , or switch to the other door, i.e.  $p_2 \in D_{\text{remaining}} \setminus \{p_1\}$
  - The game show master opens door  $p_2$ , and the player wins if  $p_2 = r$ , and loses otherwise.
- Output: What's the probability of winning with the two strategies, i.e.

$$P(\text{win} \mid \text{switch}) = P(r = p_2 \mid p_2 \neq p_1) = ?$$

$$P(\text{win} \mid \text{don't switch}) = P(r = p_2 \mid p_2 = p_1) = ?$$

- MC method:

```

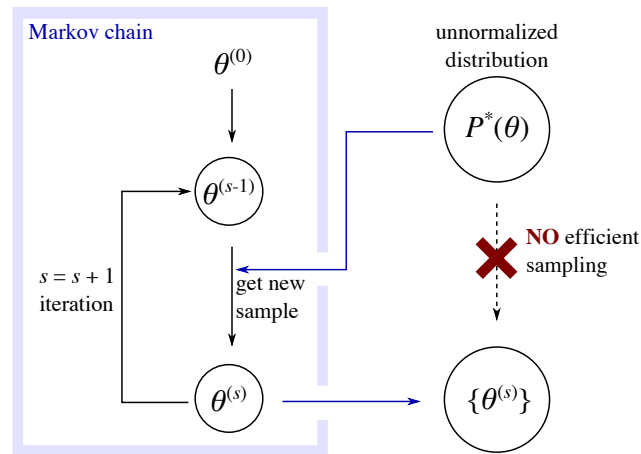
1 from numpy.random import choice
2
3 doors = {1, 2, 3}
4
5 games = 1000
6 wins_with_switch = 0
7 wins_with_no_switch = 0
8 for g in range(games):
9     reward = choice(list(doors))
10    pick_1 = choice(list(doors))
11
12    can_open = doors - set([pick_1]).union(set([reward]))
13    opened = choice(list(can_open))
14    remaining = doors - set([opened])
15
16    pick_2 = list(remaining - set([pick_1]))[0]
17    if pick_2 == reward:
18        wins_with_switch += 1
19
20    pick_2 = pick_1
21    if pick_2 == reward:
22        wins_with_no_switch += 1
23
24 P_win_with_switch = wins_with_switch / float(games)
25 P_win_with_no_switch = wins_with_no_switch / float(games)

```

Yielding something similar to  $P(\text{win} \mid \text{switch}) \approx 0.653$ , and  $P(\text{win} \mid \text{don't switch}) \approx 0.347$ .

## 7.2 Markov Chain Monte Carlo method

- **Goal:** Draw samples from an unnormalized posterior  $P^*(\theta) \mapsto \{\theta^{(s)}\}$
- **Challenge:** We don't know how to do this directly.
- **Method:**
  1. Initialize  $\theta^{(0)}$ .
  2. Obtain a new  $\theta^{(s+1)}$  value using the current value  $\theta^{(s)}$  and the  $P^*(\cdot)$  function.
  3. Add the new value to the list of samples  $\{\theta^{(s)}\}$ . Return to step 2 with  $s \leftarrow s + 1$ .



Various Markov chain-based methods exist: Metropolis-Hastings sampling, Gibbs sampling, Hamiltonian sampling.

## 7.3 Metropolis Hastings sampling

1. Start with  $\theta^{(0)}$ .
2. Propose a new value:  $\theta^{\text{new}} = \theta^{(s)} + \varepsilon$ , where  $\varepsilon$  is drawn from  $P(\varepsilon) = \text{Normal}(\varepsilon \mid 0, s^2)$ , where  $s$  is a fixed “step size”.
3. Evaluate  $\Delta L = \log P^*(\theta^{\text{new}}) - \log P^*(\theta^{(s)})$ , and depending on its value, we obtain  $\theta^{(s+1)}$ :

(a) If  $\Delta L \geq 0$ , then

$$\theta^{(s+1)} = \theta^{\text{new}},$$

(b) if  $\Delta L < 0$ , then

$$\theta^{(s+1)} = \begin{cases} \theta^{\text{new}} & \text{with probability } \exp(\Delta L) \\ \theta^{(s)} & \text{with probability } 1 - \exp(\Delta L) \end{cases}$$

**Example:** Bimodal distribution

- Unnormalized distribution:  $\log P^*(x) = x/2 - (1 - x^2)^2$ , where  $x \in \mathbb{R}$ .
- 1-dimensional Metropolis-Hastings sampler:

```

1 def propose_MH(x, stepsize):
2     epsilon = norm.rvs(loc=0, scale=stepsize)
3     return x + epsilon
4
5 def new_sample_MH(x_current, x_proposed, log_Pstar):
6     delta_L = log_Pstar(x_proposed) - log_Pstar(x_current)
7     if delta_L >= 0:
8         return x_proposed
9     if np.random.random() < np.exp(delta_L):
10        return x_proposed
11    else:
12        return x_current

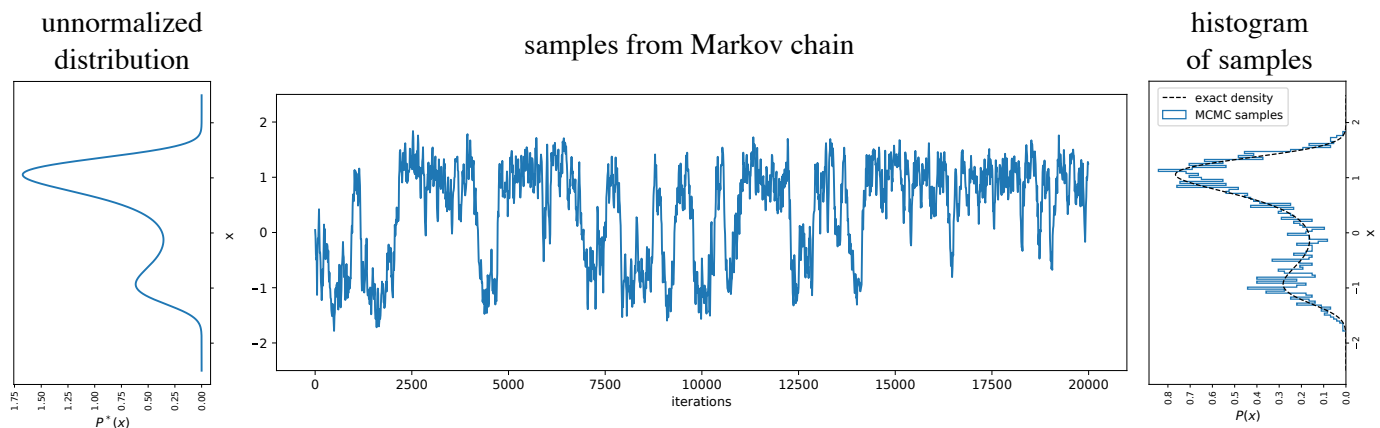
```

- Applying it to the  $\log P^*$  in question:

```

1 def log_Pstar_camel(x):
2     return 0.5 * x - (1 - x**2)**2
3
4 x0 = 0
5 stepsize = 0.1
6 iterations = 20000
7 x_samples = []
8
9 x_curr = x0
10 for it in range(iterations):
11     x_proposed = propose_MH(x_curr, stepsize)
12     x = new_sample_MH(x_curr, x_proposed, log_Pstar_camel)
13     x_samples.append(x)
14     x_curr = x

```



## 8 Gibbs sampling

**Requirement:**  $P^*(\theta)$  is difficult to sample, but after partitioning  $\theta$  into two (or more) sets of parameters,  $\theta_1, \theta_2, \dots$ , their conditionals, i.e.  $P(\theta_1 | \theta_2, \dots)$  and  $P(\theta_2 | \theta_1, \dots)$ , are easy to sample.

### 8.1 Gibbs sampling algorithm

1. Derive the conditional distributions  $P(\theta_1 | \theta_2)$ ,  $P(\theta_2 | \theta_1)$
2. Initialize  $\theta_1^{(0)}$
3. Draw  $\theta_2^{(s+1)}$  from  $P(\theta_2 | \theta_1^{(s)})$
4. Draw  $\theta_1^{(s+1)}$  from  $P(\theta_1 | \theta_2^{(s+1)})$
5. Add  $\theta^{(s+1)} = (\theta_1^{(s+1)}, \theta_2^{(s+1)})$  to the collection of samples  $\{\theta^{(s)}\}$ . Return to step 3.

### 8.2 Example: Triangle distribution in 2D

- Unnormalized distribution  $P^*(\theta_1, \theta_2) = \max(0, 1 - |\theta_1| - |\theta_2|)$
- 2-dimensional Gibbs sampler: The conditionals are

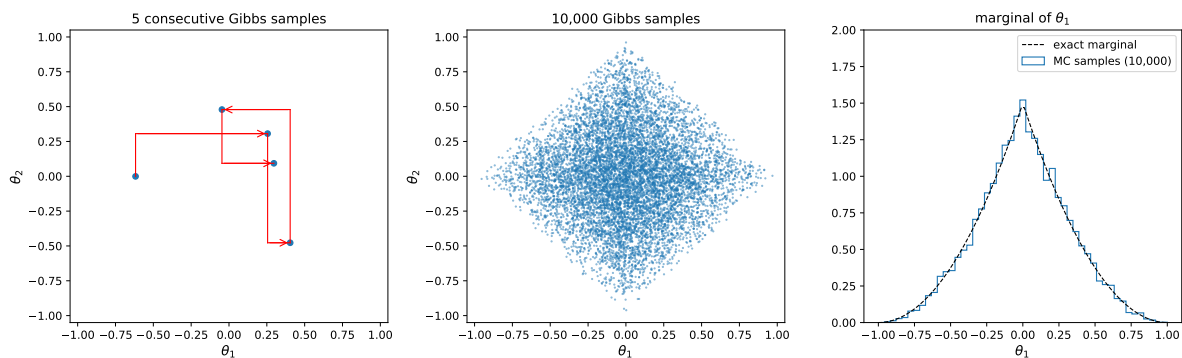
$$\begin{aligned} P(\theta_1 | \theta_2) &= \text{Triangle}(\theta_1 | \text{loc} = 0, \text{scale} = 1 - |\theta_2|) \\ P(\theta_2 | \theta_1) &= \text{Triangle}(\theta_2 | \text{loc} = 0, \text{scale} = 1 - |\theta_1|), \end{aligned}$$

where  $\text{Triangle}(x | \text{loc}, \text{scale}) = \frac{1}{2} \max(0, \text{scale} - |x - \text{loc}|)$  is the symmetric 1-d triangle distribution.

```

1 from scipy.stats import triang
2 def triangle(loc=0, scale=1):
3     return triang(c=0.5, loc=-scale + loc, scale=2*scale)
4
5 iters = 10_000
6 theta1_samples = []
7 theta2_samples = []
8 theta_1 = 0
9 for it in range(iters):
10     theta_2 = triangle(scale=1-abs(theta_1)).rvs()
11     theta_1 = triangle(scale=1-abs(theta_2)).rvs()
12     theta1_samples.append(theta_1)
13     theta2_samples.append(theta_2)

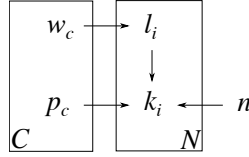
```





### 8.3 Example: Binomial clustering

- Data:  $D = \{k_i\}_{i=1}^N$ , where  $k_i$ (successes)  $\in \{0, 1, 2, \dots, n\}$ , out of  $n$  trials.
- Parameters:
  - Cluster weights:  $w = \{w_c\}_{c=1}^C$ , where  $w_c \in [0, 1]$  such that  $\sum_c w_c = 1$ , with flat prior, i.e.  $P(w) = \text{const.}$
  - Probability of success in each cluster:  $p = \{p_c\}_{c=1}^C$ , where  $p_c \in [0, 1]$ , with a weak prior  $P(p_c) = \text{Beta}(p_c \mid \alpha_c^{(0)}, \beta_c^{(0)})$ , where  $\alpha_c^{(0)}, \beta_c^{(0)}$  are small positive numbers.
  - Hidden labels:  $L = \{l_i\}_{i=1}^N$ , where  $l_i \in \{1, 2, \dots, C\}$ .
- Model:
  - Level 1:  $P(l_i = c \mid w) = w_c$
  - Level 2:  $P(k_i \mid l_i = c, p) = \text{Binomial}(k_i \mid n, p_c)$



- Unnormalized posterior:

$$P^*(L, w, p \mid D) \propto \left[ \prod_{i=1}^N P(k_i \mid p_{l_i}) P(l_i \mid w) \right] P(w) P(p) \propto \prod_{i=1}^N \text{Binomial}(k_i \mid n, p_{l_i}) w_{l_i}$$

- Partitioning:

$$\theta = (w, p, L) \quad \rightarrow \quad \theta_1 = (w, p), \quad \theta_2 = L$$

- $P(\theta_1 \mid \theta_2, D)$  conditionals:

$$P(w, p \mid L, D) = P(w \mid L) P(p \mid L, D)$$

where

$$\begin{aligned} P(w \mid L) &\propto P(L \mid w) P(w) \propto \prod_{i=1}^N w_{l_i} = \prod_{c=1}^C (w_c)^{N_c} \\ &= \text{Dirichlet}(w \mid \alpha_c = 1 + N_c) \\ P(p \mid L, D) &\propto P(D \mid L, \mu, \sigma) P(p) \propto \left[ \prod_{i=1}^N \text{Binomial}(k_i \mid n, p_{l_i}) \right] \times \left[ \prod_{c=1}^C \text{Beta}(p_c \mid \alpha_c^{(0)}, \beta_c^{(0)}) \right] \\ &= \prod_{c=1}^C \text{Beta}(p_c \mid \alpha = \alpha_c^{(0)} + K_c, \beta = \beta_c^{(0)} + nN_c - K_c), \end{aligned}$$

where  $N_c = \sum_i \delta_{l_i, c}$ ,  $K_c = \sum_i \delta_{l_i, c} k_i$ , where  $\delta_{l_i, c} = 1$  if  $l_i = c$  and 0 otherwise.

- $P(\theta_2 \mid \theta_1, D)$  conditionals:

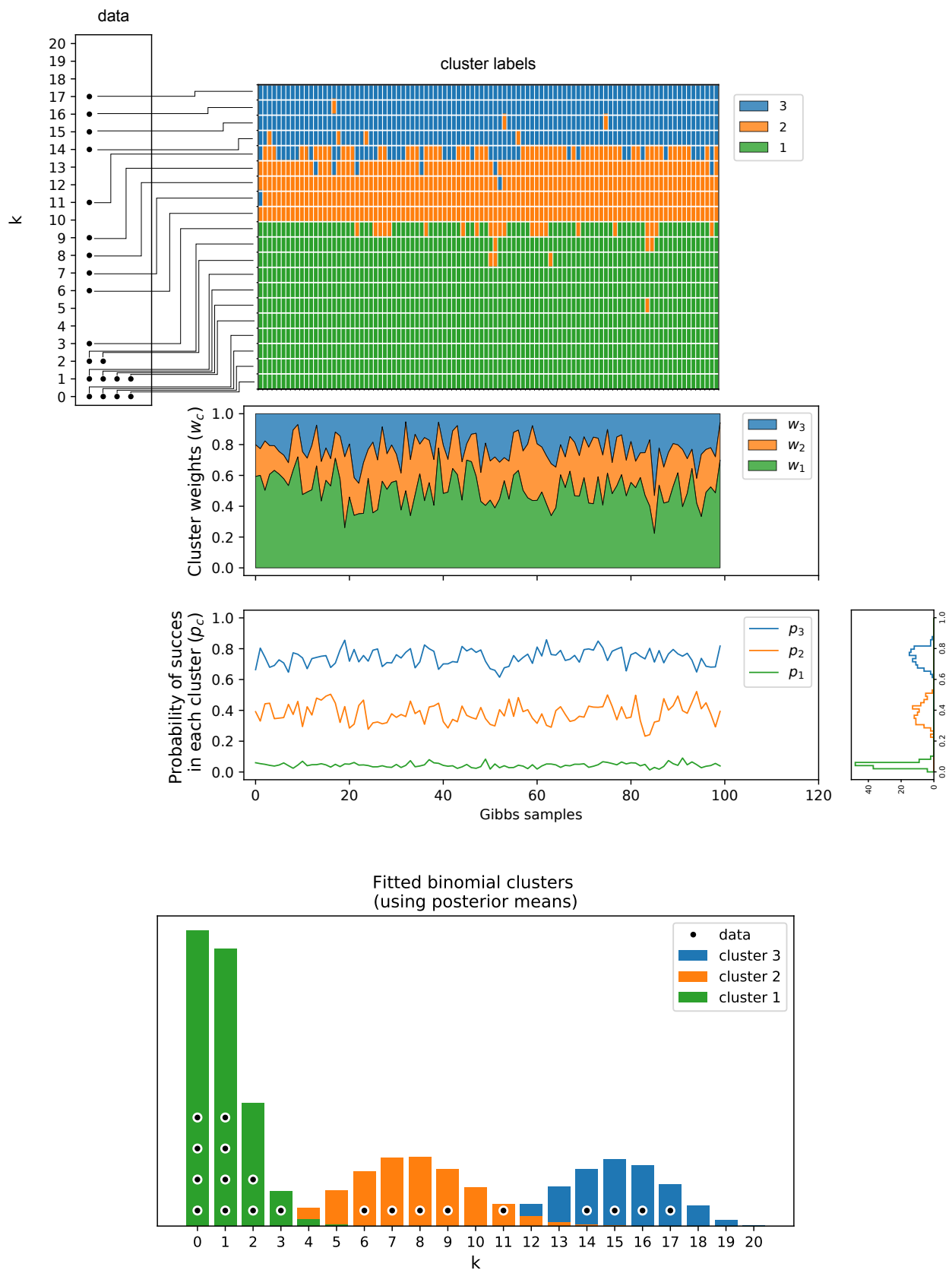
$$\begin{aligned} P(L \mid w, p, D) &\propto P(D \mid L, p) P(L \mid w) = \prod_{i=1}^N \text{Binomial}(k_i \mid p_{l_i}) w_{l_i} \\ &= \prod_{i=1}^N \text{Categorical}(l_i \mid f_c = \frac{w_c \text{Binomial}(k_i \mid n, p_c)}{\sum_{c'} w_{c'} \text{Binomial}(k_i \mid n, p_{c'})}) \end{aligned}$$

- Gibbs sampling:

```

1 import numpy as np
2 from numpy.random import choice
3 from scipy.stats import binom, beta, dirichlet
4
5 def sample_labels(k_data, n, w, p):
6     labels = []
7     for ki in k_data:
8         f = w * binom.pmf(ki, n, p)
9         f /= np.sum(f)
10        labels.append(choice(clusters, p=f))
11    return np.array(labels)
12
13 def sample_w(labels, clusters):
14     alpha = []
15     for c in clusters:
16         alpha.append(1 + np.sum(labels == c))
17    return dirichlet.rvs(alpha)[0]
18
19 def sample_p(labels, k_data, n, clusters, alpha0, beta0):
20     N = len(k_data)
21     p = []
22     for c in clusters:
23         Kc = np.sum(k_data[labels == c])
24         Nc = np.sum(labels == c)
25         a = alpha0[c] + Kc
26         b = beta0[c] + n*Nc - Kc
27         p.append(beta.rvs(a, b))
28    return np.array(p)
29
30 clusters = list(range(3))
31 alpha0 = [0, 0.5, 1]
32 beta0 = [1, 0.5, 0]
33
34 w = np.array([1./3] * 3)
35 p = np.array([0.01, 0.5, 0.6])
36
37 labels_samples = []
38 w_samples = []
39 p_samples = []
40 iters = 100
41 for it in range(iters):
42     labels = sample_labels(k_data, n, w, p)
43     w = sample_w(labels, clusters)
44     p = sample_p(labels, k_data, n, clusters, alpha0, beta0)
45     labels_samples.append(labels)
46     w_samples.append(w)
47     p_samples.append(p)

```



## 9 Viterbi algorithm, Belief propagation

### 9.1 Hidden Markov Model

- Data:  $D = \{x_t\}_{t=1}^T$
- Parameters
  - Hidden states:  $Z = \{z_t\}_{t=1}^T$ , where  $z_t \in S = \{1, 2, \dots, s_{\max}\}$
  - Prior probability:  $w = \{w_s\}_{s \in S}$ , where  $w_s \in [0, 1]$ , such that  $\sum_s w_s = 1$ .
  - Transition probabilities:  $\{A_{s,r}\}_{s,r \in S}$ , where  $A_{s,r} \in [0, 1]$ , such that  $\sum_r A_{s,r} = 1$ , for all  $s$ .
  - Emission probabilities:  $\{B_s(x)\}_{s \in S}$ , where  $B_s(x)$  is a probability density of  $x$ , i.e.  $x \mapsto \mathbb{R}$ , and  $\sum_x B_s(x) = 1$ , for all  $s$ .

- Model:

- level 1:  $P(x_t | z_t, B) = B_{z_t}(x_t)$
- level 2:  $P(Z | w, A) = P(z_1 | w) \prod_{t=2}^T P(z_t | z_{t-1}, A) = w_{z_1} \prod_{t=2}^T A_{z_{t-1}, z_t}$

- Joint

$$P(D, Z | w, A, B) = P(D | Z, B)P(Z | w, A) = \left[ \prod_{t=1}^T B_{z_t}(x_t) \right] \times \left[ w_{z_1} \prod_{t=2}^T A_{z_{t-1}, z_t} \right]$$

- Most likely sequence of states:

$$Z_{\text{MLE}} = \arg \max_Z P(D, Z | w, A, B) = (\text{see “Viterbi algorithm” below})$$

- Marginal likelihood, marginals of  $z_t$  and joint marginal of  $(z_{t-1}, z_t)$ :

$$\left. \begin{aligned} P(D | w, A, B) &= \sum_Z P(D, Z | w, A, B) \\ P(z_t | D, w, A, B) &= \sum_{Z \setminus \{z_t\}} P(D, Z | w, A, B) \\ P(z_{t-1}, z_t | D, w, A, B) &= \sum_{Z \setminus \{z_{t-1}, z_t\}} P(D, Z | w, A, B) \end{aligned} \right\} = (\text{see “Belief propagation” below})$$

- Maximum-likelihood estimate of level 2 parameters

$$(w, A, B)_{\text{MLE}} = \arg \max_{w, A, B} P(D | w, A, B) = (\text{see “Baum-Welch” algorithm below})$$

## 9.2 Viterbi algorithm

Inputs:

- $f_1(s) := P(x_1, z_1 = s \mid w, B) = w_s B_s(x_1)$ , for  $s \in S$
- $f_t(r, s) := P(x_t, z_t = s \mid z_{t-1} = r, A, B) = A_{r,s} B_s(x_t)$ , for  $t = 2, 3, \dots, T$ , and  $r, s \in S$
- $F(Z) := f_1(z_1) \prod_{t=2}^T f_t(z_{t-1}, z_t)$

Outputs:

- $F_{\max} := \max_Z P(D, Z \mid w, A, B) = \max_Z F(Z)$
- $(z_1^*, z_2^*, \dots, z_T^*) := Z_{\text{MLE}} = \arg \max_Z F(Z)$

Algorithm:

1. **“Discover”**

For  $s \in S$ :

$$\phi_1(s) := f_1(s)$$

For  $t \in [2, 3, \dots, T]$ :

For  $s \in S$ :

$$\begin{aligned} \phi_t(s) &:= \max_{r \in S} \left( \phi_{t-1}(r) f_t(r, s) \right) \\ a_t(s) &:= \arg \max_{r \in S} \left( \phi_{t-1}(r) f_t(r, s) \right) \end{aligned}$$

2. **“Retrace”**

$$\begin{aligned} F_{\max} &= \max_{s \in S} \left( \phi_T(s) \right) \\ z_T^* &= \arg \max_{s \in S} \left( \phi_T(s) \right) \end{aligned}$$

For  $t \in [T-1, T-2, \dots, 2, 1]$ :

$$z_t^* = a_{t+1}(z_{t+1}^*)$$

### 9.3 Belief propagation on chain

Inputs:

- $f_1(s) := P(x_1, z_1 = s \mid w, B) = w_s B_s(x_1)$ , for  $s \in S$
- $f_t(r, s) := P(x_t, z_t = s \mid z_{t-1} = r, A, B) = A_{r,s} B_s(x_t)$ , for  $t = 2, 3, \dots, T$ , and  $r, s \in S$
- $F(Z) := f_1(z_1) \prod_{t=2}^T f_t(z_{t-1}, z_t)$

Outputs:

- $N := \sum_Z F(Z)$
- $M_t(s) := \sum_{Z \setminus \{z_t\}} F(z_1, z_2, \dots, z_{t-1}, s, z_{t+1}, \dots, z_T)$ ,  $t \in \{1, 2, \dots, T\}$ ,  $s \in S$ ,
- $Q_t(r, s) := \sum_{Z \setminus \{z_{t-1}, z_t\}} F(z_1, z_2, \dots, z_{t-2}, r, s, z_{t+1}, \dots, z_T)$ ,  $t \in \{2, \dots, T\}$ ,  $s \in S$ ,

Algorithm:

1. **“Forward”**

For  $s \in S$ :

$$L_1(s) := f_1(s)$$

For  $t \in [2, 3, \dots, T]$ :

For  $s \in S$ :

$$L_t(s) := \sum_{r \in S} L_{t-1}(r) f_t(r, s)$$

2. **“Backward”**

For  $r \in S$ :

$$R_T(r) := 1$$

For  $t \in [T-1, T-2, \dots, 2, 1]$ :

For  $s \in S$ :

$$R_t(r) := \sum_{s \in S} f_{t+1}(r, s) R_{t+1}(s)$$

3. **“Combine”**

$$N = \sum_{s \in S} L_T(s)$$

For  $t \in [1, 2, \dots, T]$ :

For  $s \in S$ :

$$M_t(s) = L_t(s) R_t(s)$$

For  $t \in [2, \dots, T]$ :

For  $(s, r) \in S \times S$ :

$$Q_t(r, s) = L_{t-1}(r) f_t(r, s) R_t(s)$$