Bayesian Methods

Peter Komar

December 28, 2018

Contents

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
- $P(X = x) \ =: \ P(x)$
- $P(X = x \text{ and } Y = y) \ =: \ P(x, y)$
- $P(A = a, \text{ given } B = b) \ =: \ P(a \mid b)$
- $\int_{\infty}^{+\infty} [\ldots] da \ =: \ \sum_{a \in \mathbb{R}} [\ldots] \ =: \ \sum_a [\ldots]$

Conditional probability

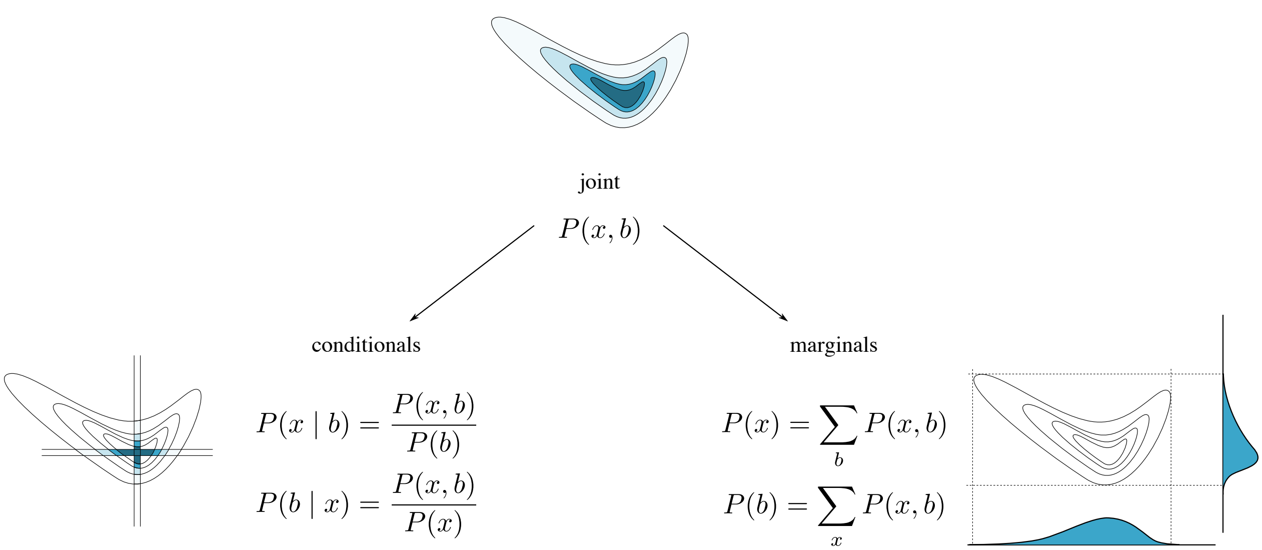
- $P(a \mid b) \ = \ \frac{P(a,b)}{P(b)}$
- $P(a, b) \ = \ P(a \mid b) \ P(b)$
- $P(a, b \mid c) \ = \ P(a \mid b, c) \ P(b \mid c)$
- $\sum_a P(a \mid b) \ = \ 1, \quad$ but $\ \sum_b P(a \mid b) \neq 1$, in general
- $\sum_b P(a \mid b) \ P(b) \ = \ \sum_b P(a, b) \ = \ P(a)$

Marginal

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

Bayes theorem

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



1.2 Bayesian inference

Prior, likelihood, posterior

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

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].
- $P(t \mid T) = \frac{1}{T} \exp\left(-\frac{t}{T}\right)$
- $P(D \mid T) = \prod_i P(t_i \mid T) = \prod_{i=1}^{3} \frac{1}{T} \exp\left(-\frac{t_i}{T}\right) = \frac{1}{T^3} \exp\left(-\frac{1+2+5}{T}\right)$
- $P^*(T \mid D) = \frac{1}{T^3} \exp\left(-\frac{8}{T}\right)$
- Z and $P(T \mid D)$ can be determined numerically:

```python
import numpy as np

T_arr = np.linspace(0.1, 1000, 10_000)
Pstar_arr = 1.0/T_arr**3 * np.exp(-8/T_arr)
Z = np.sum(Pstar_arr)
P_arr = Pstar_arr / Z
```

  yielding $Z = 0.1562$

- $\mathbb{E}(T \mid D) = \sum_T T\, P(T \mid D)$
- $\mathrm{std}(T \mid D) = \sqrt{\sum_T (T - \mathbb{E}(T))^2\, P(T \mid D)}$

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

  yielding $\mathbb{E}(T \mid D) = 7.937, \quad \mathrm{std}(T \mid D) = 14.48.$

1.3 Model comparison

New definition: Evidence

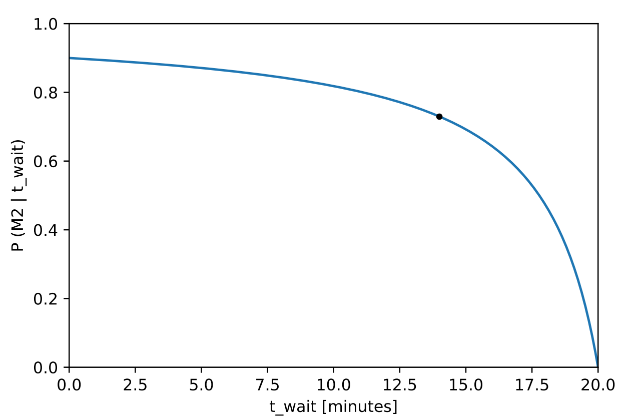
- Data: D
- Model 1: M_1 with parameter θ_1 and prior $P(\theta_1 \mid M_1)$, and likelihood $P(D \mid \theta_1, M_1)$
- Model 2: M_2 with parameter θ_2 and prior $P(\theta_2 \mid M_2)$, and likelihood $P(D \mid \theta_2, M_2)$
- Prior on models: $P(M_1) = 0.5, \quad P(M_2) = 0.5$.
- Evidence for each model: $P(D \mid M_i) = \sum_\theta P(D \mid \theta_i, M_i) P(\theta_i \mid M_i)$
- Unnormalized posterior: $P^*(M_1 \mid D) = P(D \mid M_1)\, P(M_1), \quad$ and $P^*(M_2 \mid D) = P(D \mid M_2)\, P(M_2)$
- Normalization: $Z = P^*(M_1 \mid D) + P^*(M_2 \mid D)$.

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?"

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

We can also plot $P(M_2 \mid t_{\text{wait}})$ 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 θ, prior $P(\theta)$ and likelihood $P(x \mid \theta)$
- Posterior: $P(\theta \mid D) = P^*(\theta \mid D)/Z = \dots$ (see previous sections)
- Predictive distribution: $P(X_{n+1} = x \mid D) = \sum_\theta P(x \mid\theta)\ P(\theta \mid D)$
- Customized prediction: $P(f(\theta) \mid D) = \sum_\theta f(\theta)\ P(\theta \mid D)$

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 $[0, 1]$
- $P(\text{A scores} \mid b) = b$
- $P(D \mid b) = \text{Binomial}(5 \mid 5 + 3,\ b)$
- $P^*(b_0 \mid D) = \text{Binomial}(5 \mid 8, b) \times 1$
- $Z = \sum_b \text{Binomial}(5 \mid 8, b)$ can be calculated numerically

```
import numpy as np
from scipy.stats import binom

b_arr = np.linspace(0, 1, 1000)
Pstar_arr = binom.pmf(5, 8, b_arr)
Z = np.sum(Pstar_arr)
```

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

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

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

2 Exact inference and Maximum Likelihood Estimate

2.1 Maximum likelihood estimate

MLE-method

- Data: $D = \{x_1, x_2, \dots x_N\}$
- Parameter: θ
- Likelihood: $P(x_i \mid \theta)$
- Total log likelihood: $L(\theta) = \log P(D \mid \theta) = \sum_{i=1}^N \log P(x_i \mid \theta)$
- Maximum likelihood estimate $\theta_{\text{MLE}} = \text{argmax}_\theta \log P(D \mid \theta)$,
  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: $\mu \in \mathbb{R}, \quad \sigma^2 > 0$
- Likelihood: $P(x_i \mid \mu, \sigma^2) = \text{Normal}(x_i \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x_i-\mu)^2}{2\sigma^2}\right]$
- Total log likelihood:

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

- Analytical solution:

$$0 = \left[\frac{\partial L}{\partial \mu}\right]_{\text{MLE}} = \left[\sum_{i=1}^N \frac{\mu - x_i}{\sigma^2}\right]_{\text{MLE}} \quad \Rightarrow \quad \mu_{\text{MLE}} = \frac{1}{N}\sum_{i=1}^N x_i.$$

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

Example 2: Cauchy distribution

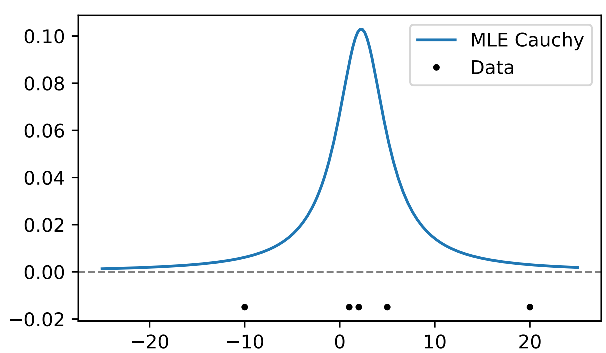
- Data: $D = \{-10, 1, 2, 5, 20\}$
- Parameters: $m \in \mathbb{R}, \quad s > 0.$
- Likelihood: $P(x_i \mid m, s) = \text{Cauchy}(x_i \mid m, s) = \frac{1}{s\pi} \frac{1}{1+[(x_i-m)/s]^2}$
- Total log likelihood:

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

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

```
import numpy as np
from scipy.optimize import minimize

def cauchy_total_log_likelihood(X, m, s):
    X = np.array(X)

    L = 0
    L += -len(X)/2 * np.log(s**2)
    L += -np.sum( np.log(1 + (X - m)**2 / s**2 ) )

    return L

X = [-10, 1, 2, 5, 20]
def func_to_minimize(theta):
    m = theta[0]
    s = theta[1]
    return - cauchy_total_log_likelihood(X, m, s)

m0 = 0
s0 = 10
result = minimize(func_to_minimize, [m0, s0])
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: $D = \{(k_1, n_1), (k_2, n_2), \ldots, (k_N, n_N)\}$, 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: $P(k_i \mid n_i, p) = \text{Binomial}(k_i \mid n_i, p) = \binom{n_i}{k_i}\, p^{k_i}(1-p)^{n_i-k_i}$
- Posterior:

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

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

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

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: $P(\{k_{i,j}\}_{j=1}^M \mid p) = \text{Multinomial}(\{k_{i,j}\}_{j=1}^M \mid p) = k_{i,\text{tot}}! \prod_j \frac{p_j^{k_{i,j}}}{k_{i,j}!}$
- Posterior:

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

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: γ (rate) > 0, flat prior: $P(\gamma) = \text{const.}$
- Likelihood: $P(t_i \mid \gamma) = \text{Exponential}(t_i \mid \gamma) = \gamma e^{-\gamma t_i}$
- Posterior:

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

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: μ (expected value) $\in \mathbb{R}$, σ^2 (variance) > 0, uninformative prior: $P(\mu, \sigma^2) = \text{const.}$
- Likelihood: $P(x_i \mid \mu, \sigma^2) = \text{Normal}(x_i \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big[-\frac{(x_i-\mu)^2}{2\sigma^2}\Big]$
- Posterior:

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

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 μ is

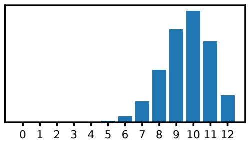
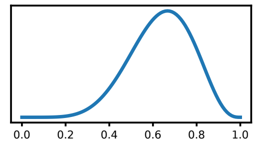
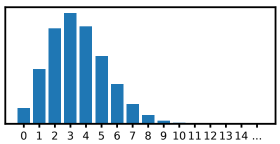
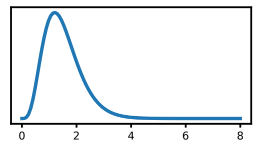
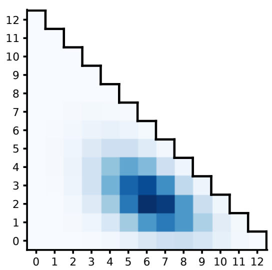
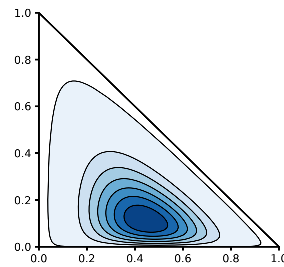
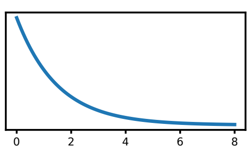
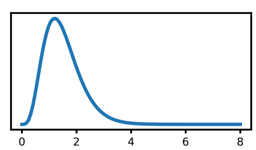
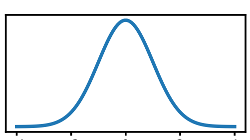
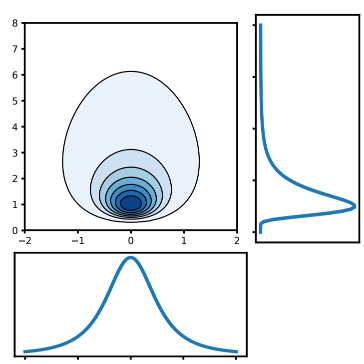
$$\begin{aligned}
P(\mu \mid D) &= \sum_{\sigma^2} P(\mu, \sigma^2 \mid D) = \frac{\Gamma\left(\frac{N-2}{2}\right)}{\Gamma\left(\frac{N-3}{2}\right)}\frac{1}{\sqrt{\pi s^2}}\left[1 + \frac{(\mu-m)^2}{s^2}\right]^{-(N-2)/2} \\
&= \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)}\frac{1}{\sqrt{\pi\nu}}\left[1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

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

where ν is the "degrees of freedom" of the Student's t-distribution. The marginal, and mean, mode and standard deviation of σ^2 is

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

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

Model		Posterior	
Binomial		Beta	
Poisson		Gamma	
Multinomial		Dirichlet	
Exponential		Gamma	
Normal		Normal-Inverse-Gamma	 Inverse-Gamma Student's t

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 θ, likelihood $P(D \mid \theta)$
- Cost function $= -\log P(D \mid \theta) = -L(\theta)$, the log likelihood
- Penalty: penalty(θ), which is high for implausible θ 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:

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

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

  The "hyperparameters" α_1 and α_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 θ_0, and $L_0(\theta_0) = P(D \mid \theta_0, M_0)$, $\theta_{0,\text{MLE}} = \text{argmax}_{\theta_0} L_0(\theta_0)$
- Alternate model: M_1 with parameters θ_1, and $L_1(\theta_1) = P(D \mid \theta_1, M_1)$, $\theta_{1,\text{MLE}} = \text{argmax}_{\theta_1} L_1(\theta_1)$

Akaike Information Criterion (AIC)

- $\text{AIC}(M_i) = -2\Big[L_i(\theta_{i,\text{MLE}}) - \dim(\theta_i)\Big]$ 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)

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

Model evidence

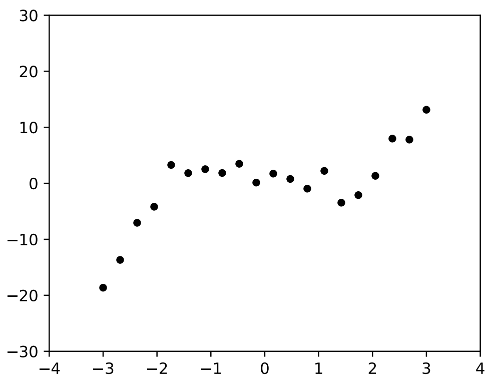
- $P(D \mid M_i) = \sum_{\theta_i} P(D \mid \theta_i, M_i) \approx \exp\left(-\frac{1}{2}\ \text{IC}\right)$,
- where IC can be either AIC or BIC (or WAIC, WBIC)
- Under uniform prior (i.e. $P(M_0) = P(M_1)$), the posterior probability of the alternate model being correct is

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

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 $\mathrm{std}(\varepsilon) = 2$).

```
import numpy as np
from numpy.polynomial.polynomial import polyval
from scipy.stats import norm

c_true = [1, -3, -0.5, 1]
sigma_true = 2
x_data = np.linspace(-3, 3, 20)
y_data = [polyval(x, c_true) + norm.rvs(loc=0, scale=sigma_true)
          for x in x_data]
```



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

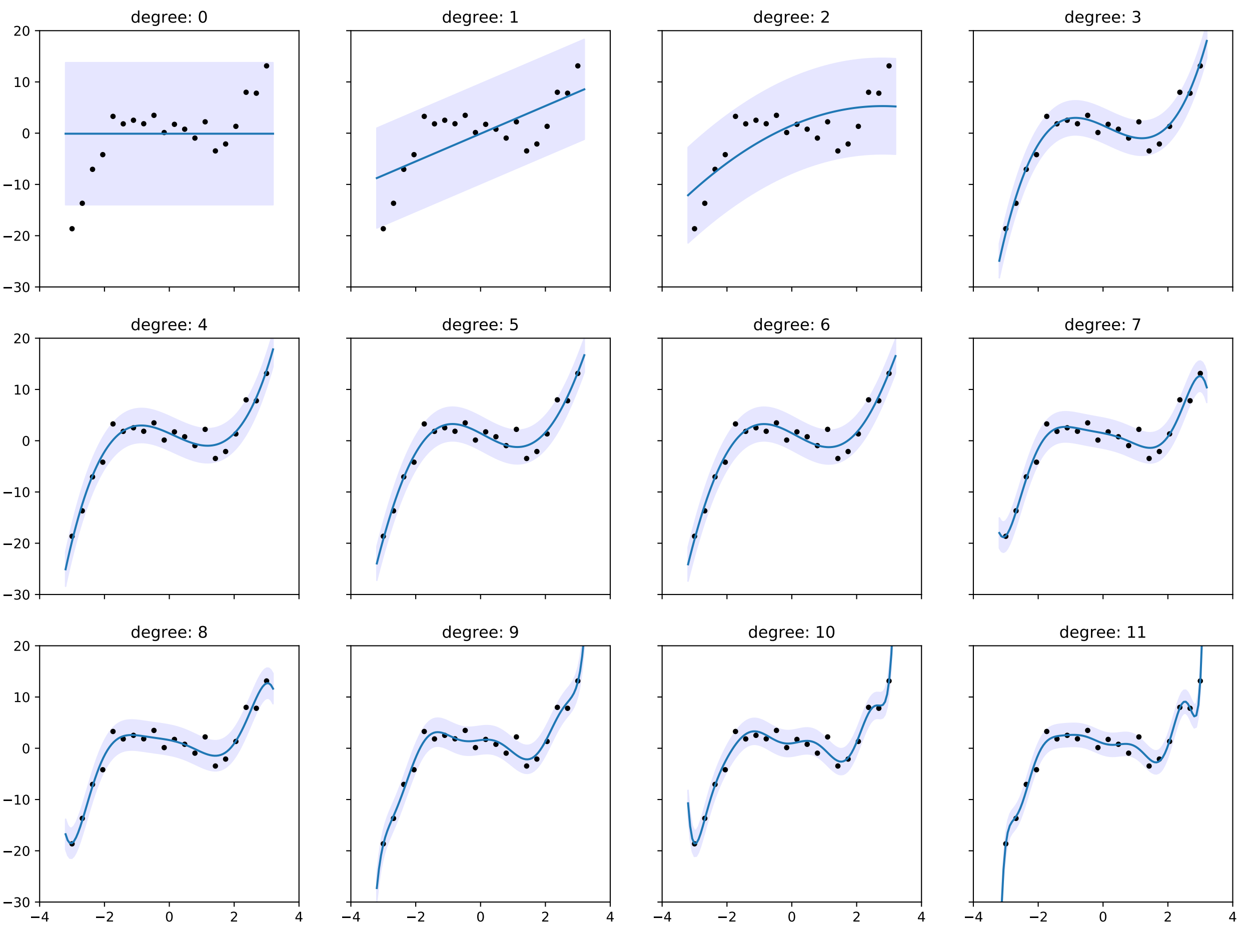
```
def generate_polynomial_features(x_data, degree):
    K = degree
    N = len(x_data)
    X = np.zeros([N, K+1])
    for i, x in enumerate(x_data):
        for k in range(0, K+1, 1):
            X[i,k] = x**k
    return X
```

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

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

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

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



- $\mathrm{AIC}_k = -2[L_k - (k+2)]$

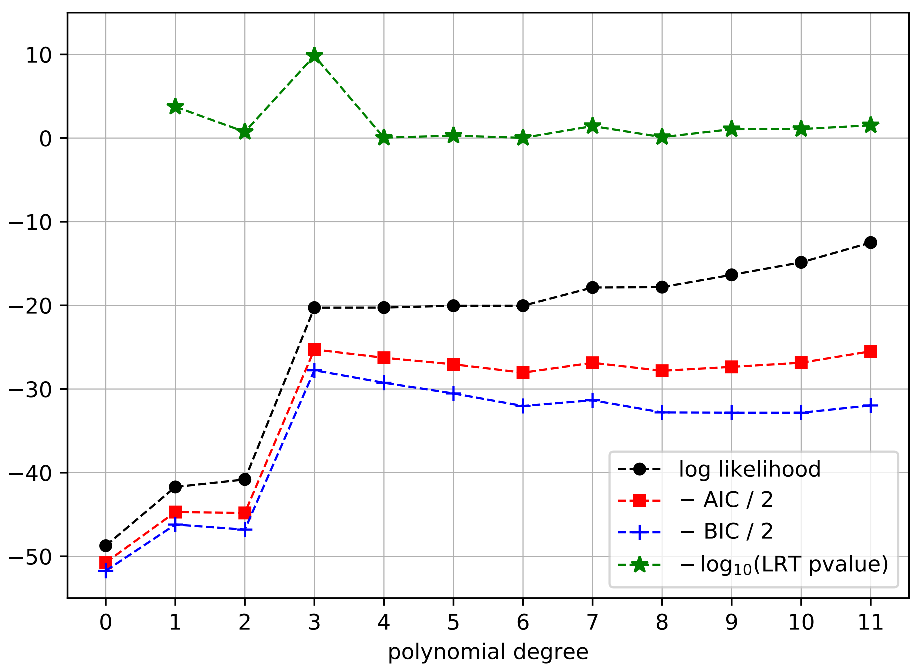
```python
def AIC(X, y, c, sigma2):
        dim = len(c) + 1
        loglike = log_likelihood(X, y, c, sigma2)
        return -2 * (loglike - dim)
```

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

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

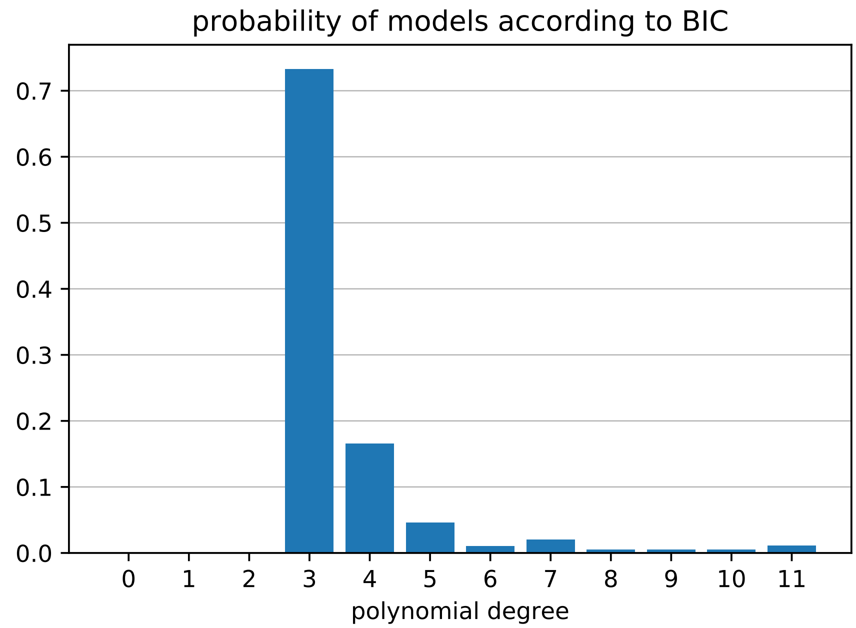
- LRT $\mathrm{pvalue}_k = 1 - \mathrm{cdf}\ \chi^2(2(L_k - L_{k-1})\mid \mathrm{dof} = 1)$
  (calculated against the model with one less degree)

```python
from scipy.stats import chi2

pvalues = [np.nan]
for deg in degrees[1:]:
    L1 = loglikes[deg]
    L0 = loglikes[deg-1]
    logLR = L1 - L0
    dof = 1
    pvalue = chi2.sf(2*logLR, dof)
    pvalues.append(pvalue)
```



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

```
def BIC_weigths(BICs):
        BICs = np.array(BICs)
        w = BICs - np.min(BICs) # for numerical stability
        w = np.exp(-0.5*(w))
        w /= np.sum(w)
        return w
```



4 Graphical models

4.1 Elements

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

$$x \longrightarrow y$$

- $P(x, y, z) = P(y \mid z, x)P(z \mid x)P(x)$.
  If $P(y \mid z, x) = P(y \mid z)$, then it is represented by a **chain**

$$x \longrightarrow z \longrightarrow y$$

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

- $P(x, y_1, y_2) = P(y_1, y_2 \mid x)P(x)$.
  If $P(y_1, y_2 \mid x) = P(y_1 \mid x)P(y_2 \mid x)$, then it is represented by a **fork**

$$\begin{tikzcd} & y_1 \\ x \arrow[ur] \arrow[dr] & \\ & y_2 \end{tikzcd}$$

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

- $P(x_1, x_2, y) = P(y \mid x_1, x_2)P(x_1, x_2)$.
  If $P(x_1, x_2) = P(x_1)P(x_2)$, then it is represented by a **collider**

$$\begin{tikzcd} x_2 \arrow[dr] & \\ & y \\ x_1 \arrow[ur] & \end{tikzcd}$$

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

Examples

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

$$\begin{tikzcd} a \arrow[dr] & & & e \\ & b \arrow[r] & c \arrow[ur] \arrow[dr] & \\ d \arrow[ur] & & & f \end{tikzcd}$$

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

$$\begin{tikzcd} & & f & \\ & c \arrow[ur] \arrow[ddr] & & \\ a \arrow[ur] \arrow[dr] & & & \\ & b \arrow[r] & d \arrow[r] & e \end{tikzcd}$$

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. Fire $\perp\!\!\!\perp$ Alarm | 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. Rain $\perp\!\!\!\perp$ Sprinkler | $\emptyset$. This is represented by a collider

$$\begin{tikzcd}
\text{Rain} \arrow[dr] & \\
& \text{Puddle} \\
\text{Sprinkler} \arrow[ur] &
\end{tikzcd}$$

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

$$\begin{tikzcd}
& \text{Crime} \\
\text{Heat} \arrow[ur] \arrow[dr] & \\
& \text{Ice Cream sales}
\end{tikzcd}$$

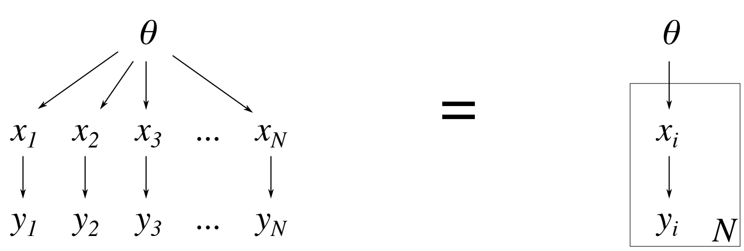
- Education affects Political View, which affects both Party membership and Voting behavior. This can be represented as

$$\begin{tikzcd}
\text{Education} \arrow[r] & \text{Political View} \arrow[r] \arrow[d] & \text{Vote} \\
& \text{Party membership} &
\end{tikzcd}$$

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

$$\begin{tikzcd}
\text{Education} \arrow[dr] \arrow[dd] & \\
& \text{Salary} \\
\text{Experience} \arrow[ur] &
\end{tikzcd}$$

4.3 Plate notation

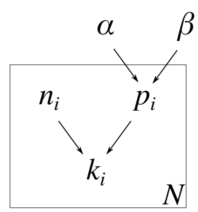


4.4 Hierarchical models

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 $P(k_i \mid n_i, p_i) = \text{Binomial}(k_i \mid n_i, p_i)$
  - level 2: the parameters, α, β (both > 0), define $P(p_i \mid \alpha, \beta) = \text{Beta}(p_i \mid \alpha, \beta)$.

  This hierarchy can be represented as



- Joint likelihood:

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

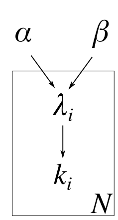
- Marginal likelihood:

$$\begin{aligned} P(D \mid \alpha, \beta) &= \prod_{i=1}^N \left[\int dp_i\, \text{Binom}(k_i \mid n_i, p_i)\ \text{Beta}(p_i \mid \alpha, \beta)\right] = \prod_{i=1}^N \Big[\text{Beta-Binom}(k_i \mid n_i, \alpha, \beta)\Big] \\ \text{where} \qquad & \text{Beta-Binom}(k \mid n, \alpha, \beta) = \frac{\Gamma(n+1)\Gamma(\alpha+\beta)}{\Gamma(n+\alpha+\beta)} \frac{\Gamma(k+\alpha)}{\Gamma(k+1)\Gamma(\alpha)} \frac{\Gamma(n-k+\beta)}{\Gamma(n-k+1)\Gamma(\beta)} \end{aligned}$$

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 $P(k_i \mid \lambda) = \text{Poisson}(k_i \mid \lambda_i)$
  - level 2: the parameters α, β (both > 0), define $P(\lambda_i \mid \alpha, \beta) = \text{Gamma}(\lambda_i \mid \alpha, \beta)$.

  This hierarchy can be represented as



- Joint likelihood:

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

- Marginal likelihood:

$$P(D \mid \alpha, \beta) = \prod_{i=1}^{N} \left[\int d\lambda_i \, \text{Poisson}(k_i \mid \lambda_i) \; \text{Gamma}(\lambda_i \mid \alpha, \beta) \right] = \prod_{i=1}^{N} \Big[\text{Gamma-Poisson}(k_i \mid \alpha, \beta) \Big]$$

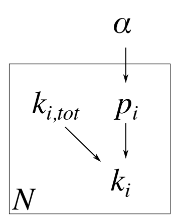
$$\text{where} \qquad \text{Gamma-Poisson}(k \mid \alpha, \beta) = \frac{\Gamma(k+\alpha)}{\Gamma(k+1)\Gamma(\alpha)} \left(\frac{1}{\beta+1} \right)^{k} \left(\frac{\beta}{\beta+1} \right)^{\alpha} =$$

$$= \text{NegativeBinom}(k \mid r, p) = \binom{k+r-1}{k} p^k (1-p)^r, \quad \text{with } r = \alpha, \; p = \frac{1}{\beta+1}$$

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:

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

- Marginal likelihood:

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

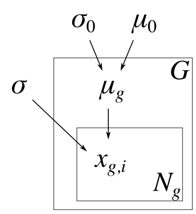
$$\text{where} \qquad \text{Dirichlet-Multinomial}(k \mid k_{\text{tot}}, \{\alpha_j\}_{j=1}^{M}) = \frac{\Gamma(k_{\text{tot}}+1)\Gamma(\alpha_{\text{tot}})}{\Gamma(k_{\text{tot}}+\alpha_{\text{tot}})} \prod_{j=1}^{M} \frac{\Gamma(k_j+\alpha_j)}{\Gamma(k_j+1)\Gamma(\alpha_j)},$$

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

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 σ^2 define $P(x_{g,i} \mid \mu_g, \sigma) = \mathrm{Normal}(x_{g,i} \mid \mu_g, \sigma^2)$
  - level 2: The parameter σ_0^2 define $P(\mu_g \mid \mu_0, \sigma_0) = \mathrm{Normal}(\mu_g \mid \mu_0, \sigma_0^2)$

  This hierarchy can be represented as



- Joint likelihood:

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

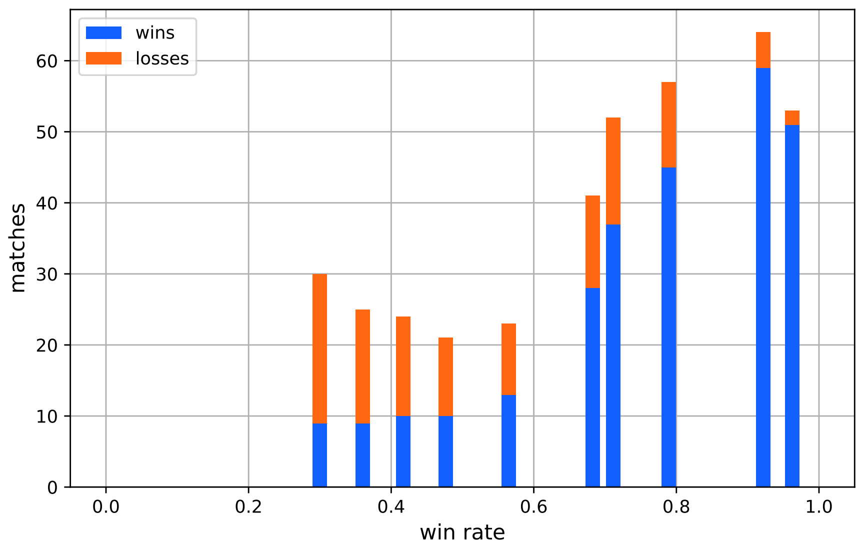
- Marginal likelihood:

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

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

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 $p_{i,\text{obs}} = k_i/n_i$ 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:

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

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

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

which can be implemented as

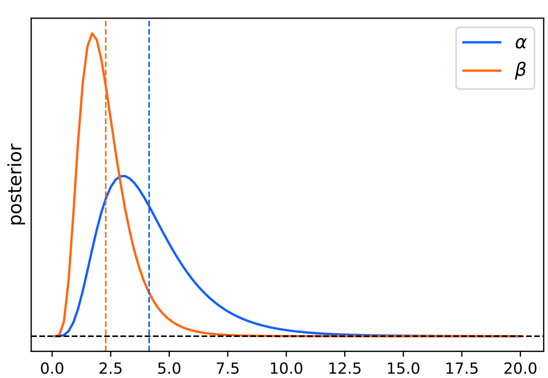
```
from scipy.special import gammaln

def log_three_gamma_term(k, a):
    return gammaln(k+a) - gammaln(k+1) - gammaln(a)

def log_beta_binom(k, n, a, b):
    target = 0
    target += - log_three_gamma_term(n, a+b)
    target += log_three_gamma_term(k, a)
    target += log_three_gamma_term(n-k, b)
    return target
```

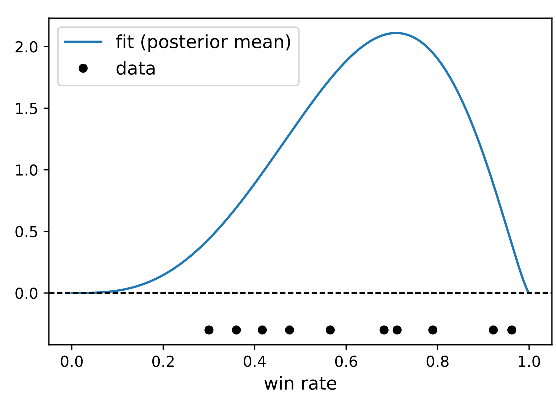
```
def log_likelihood(k, n, a, b):
    target = 0
    for ki, ni in zip(k, n):
        target += log_beta_binom(ki, ni, a, b)
    return target
```

- While assuming flat priors for α and β, we can numerically calculate their joint posterior and means.

```
import numpy as np
import pandas as pd

a_arr, b_arr = np.meshgrid(np.linspace(0.1, 20, 100),
                           np.linspace(0.1, 20, 100))
a_arr = a_arr.flatten()
b_arr = b_arr.flatten()

loglike = []
for a, b in zip(a_arr, b_arr):
    loglike.append(log_likelihood(k, n, a, b))

df = pd.DataFrame({
    'a': a_arr,
    'b': b_arr,
    'loglike': loglike
})

df['pstar'] = np.exp(df['loglike'] - df['loglike'].max())
Z = df['pstar'].sum()
df['posterior'] = df['pstar'] / Z

post_a = df.groupby(by='a')['posterior'].sum().reset_index()
post_b = df.groupby(by='b')['posterior'].sum().reset_index()

a_mean = (post_a['posterior'] * post_a['a']).sum()
b_mean = (post_b['posterior'] * post_b['b']).sum()
```

giving $\mathbb{E}(\alpha \mid D) = 4.142$, $\mathbb{E}(\beta \mid D) = 2.289$.
Their (marginal) posteriors and the distribution of the win rate (for the mean α and β values) are below.

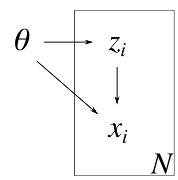




5 Hidden variables, EM, Mixture models

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

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

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

Expectation Maximization (EM) algorithm:

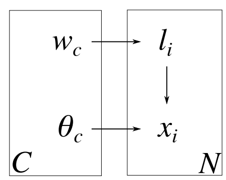
1. Start with realistic $\theta = \theta^{\text{old}}$
2. E-step: Calculate $P(Z \mid D, \theta^{\text{old}}) = \frac{P(D, Z \mid \theta^{\text{old}})}{\sum_{Z'} P(D, Z' \mid \theta^{\text{old}})}$
3. M-step: Find the optimal $\theta = \theta^{\text{new}}$ that maximizes $\sum_Z P(Z \mid D, \theta^{\text{old}}) \log P(D, Z \mid \theta)$
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 $P(D, Z \mid \theta)$:

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

5.3 Mixture models

- Data: $D = \{x_i\}_{i=1}^N$
- Parameters: θ, 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: $P(x_i \mid l_i = c, \theta) = P(x_i \mid \theta_c)$
  - Probability of an observation coming from a component: $P(l_i = c) = w_c$



- Joint

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

- Marginal

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

EM algorithm

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

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

3. In M-step, calculate

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

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

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

- Marginal:

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

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

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

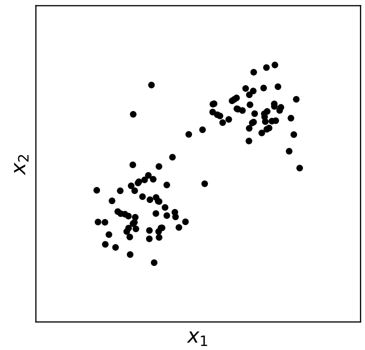
  3. In M-step, calculate

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

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

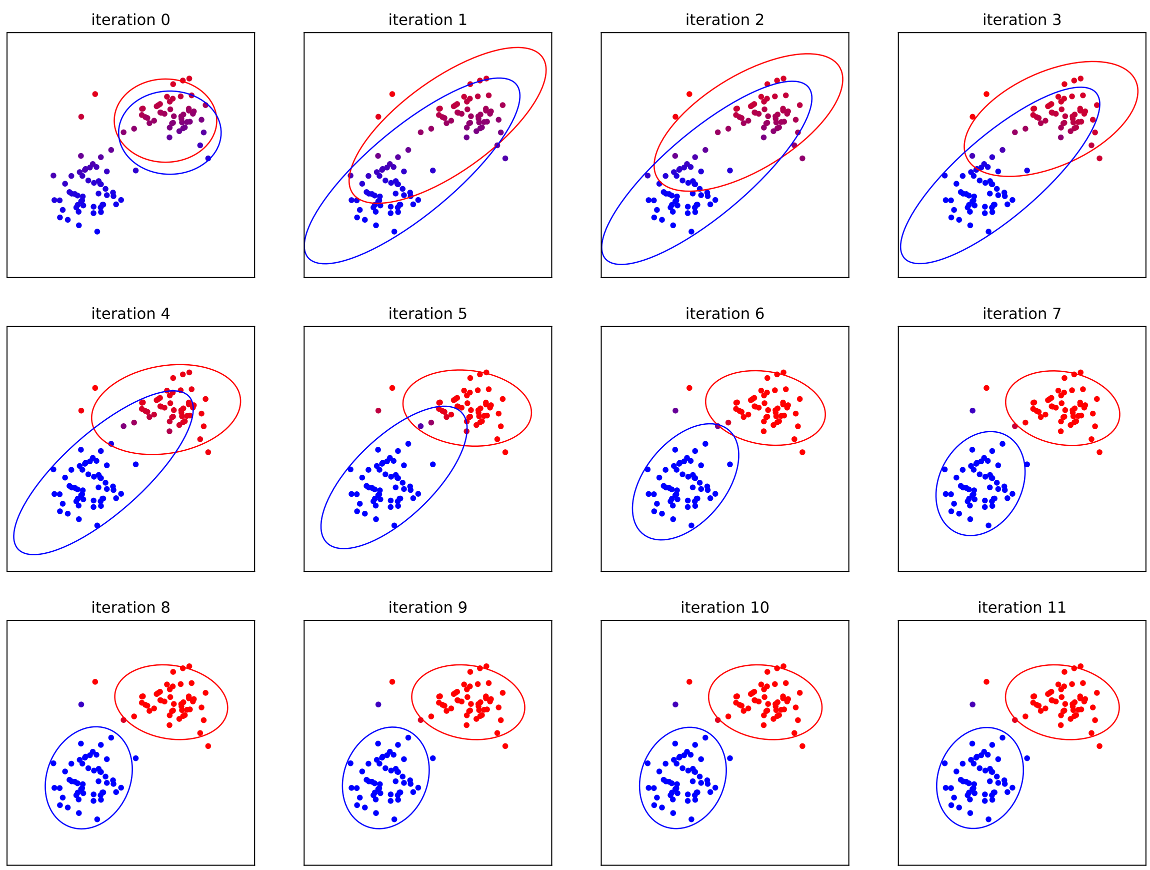
```python
class GmmEm:
    def __init__(self, x):
        self.x = np.array(x)
        self.N, self.d = self.x.shape
        self.K = None
        self.weights = None
        self.means = None
        self.covs = None

    def initialize(self, K):
        self.K = K
        m0 = np.mean(x, axis=0)
        cov0 = np.cov(x.T)

        self.weights = [1.0/K] * K
        self.means = multivariate_normal.rvs(mean=m0, cov=cov0, size=K)
        cov_values, _ = np.linalg.eig(cov0)
        self.covs = np.array([np.eye(self.d) * 0.1 *cov_values.max()
                              for _ in range(K)])

    def e_step(self):
        r = []
        for k in range(K):
            r.append(self.weights[k] *
                     multivariate_normal.pdf(self.x,
                                             mean=self.means[k],
                                             cov=self.covs[k]))
        r = np.array(r).T
        r_sum = np.einsum('ik->i', r)
        r = np.einsum('ik,i->ik', r, 1.0/r_sum)
        return r

    def m_step(self, r):
        weights_new = 1.0/N * np.einsum('ik->k', r)
        means_new = 1.0/N * \
            np.einsum('k, ik,id->kd',
                      1.0/weights_new,
                      r,
                      self.x)
        deviations = np.array([self.x - means_new[k] for k in range(self.K)])
        covs_new = 1.0/N * \
            np.einsum('k,ik,kid,kiD->kdD',
                      1.0/weights_new,
                      r,
                      deviations,
                      deviations)

        self.weights = weights_new
        self.means = means_new
        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.

```
gmm = GmmEm(x)

K = 2
gmm.initialize(K)
for it in range(12):
    r = gmm.e_step()
    gmm.m_step(r)
```



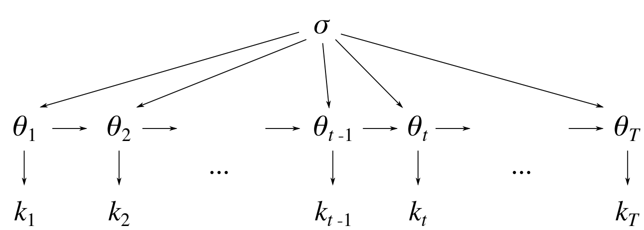
6 Curse of Dimensionality, Laplace approximation

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: $P(\theta_t \mid \theta_{t-1}) = \text{Normal}(\theta_t \mid \theta_{t-1}, \sigma^2)$, and $P(\sigma) = \text{const.}$
  - Data generation process: $P(k_t \mid \theta_t) = \text{Poisson}(k_t \mid \lambda = \exp(\theta_t))$
- Posterior:

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

with the understanding that "$P(\theta_1 \mid \theta_0)'' = 1$. Here Z is the normalization constant.



- Numerical solution would require evaluating P^* on a grid of different θ values. Even, at the very extreme, when we consider only 2 values for each θ_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 θ_t
- Challenge: The dimension of $\theta = \{\theta_t\}_{t=1}^T$, i.e. T is too high for direct numerical evaluation.
- Method: Approximate $P^*(\theta \mid D)$ near its maximum with a multi-variate normal distribution.

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

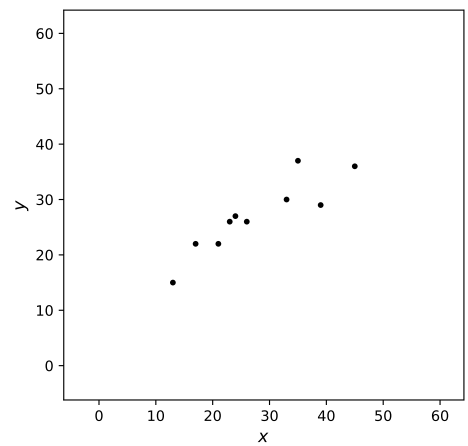
$$\text{where} \qquad \mu = \arg\max_\theta \big[\log P^* \big] \qquad \in \mathbb{R}^T$$

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

where μ can be found with direct numerical or analytical minimization or an expectation maximization algorithm, and Σ 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), σ^2 (strength of y-noise), using flat priors, i.e. $P(a, b, \sigma^2) = \text{const.}$
- Model:

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

- Unnormalized posterior:

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

- MLE estimate:

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

where

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

```
import numpy as np

def xy_linear_regression_MLE(x, y):
    N = len(x)
    ev_x = np.mean(x)
    ev_y = np.mean(y)
    ev_xx = np.mean(x * x)
    ev_yx = np.mean(y * x)
    ev_yy = np.mean(y * y)
```

```

    a_MLE = (ev_yx - ev_y * ev_x) / (ev_xx - ev_x**2)
    b_MLE = ev_y - a_MLE * ev_x
    sigma2_MLE = 1.0/N * np.sum((y - (a_MLE * x + b_MLE))**2)

    return a_MLE, b_MLE, sigma2_MLE
```

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

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

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

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

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

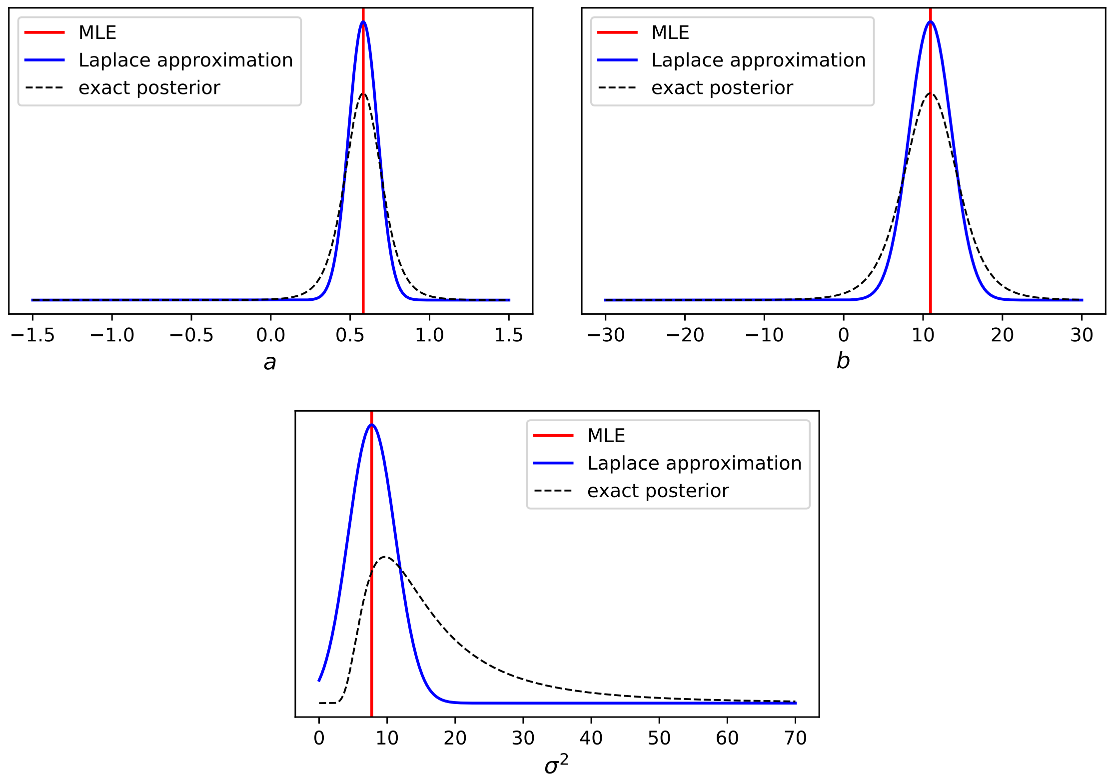
```
minus_dd_logPstar = N / sigma2_MLE * \
np.array([
    [ev_xx, ev_x, 0],
    [ev_x, 1, 0],
    [0, 0, 1/(2*sigma2_MLE)]
])
Sigma = - np.linalg.inv(minus_dd_logPstar)
```

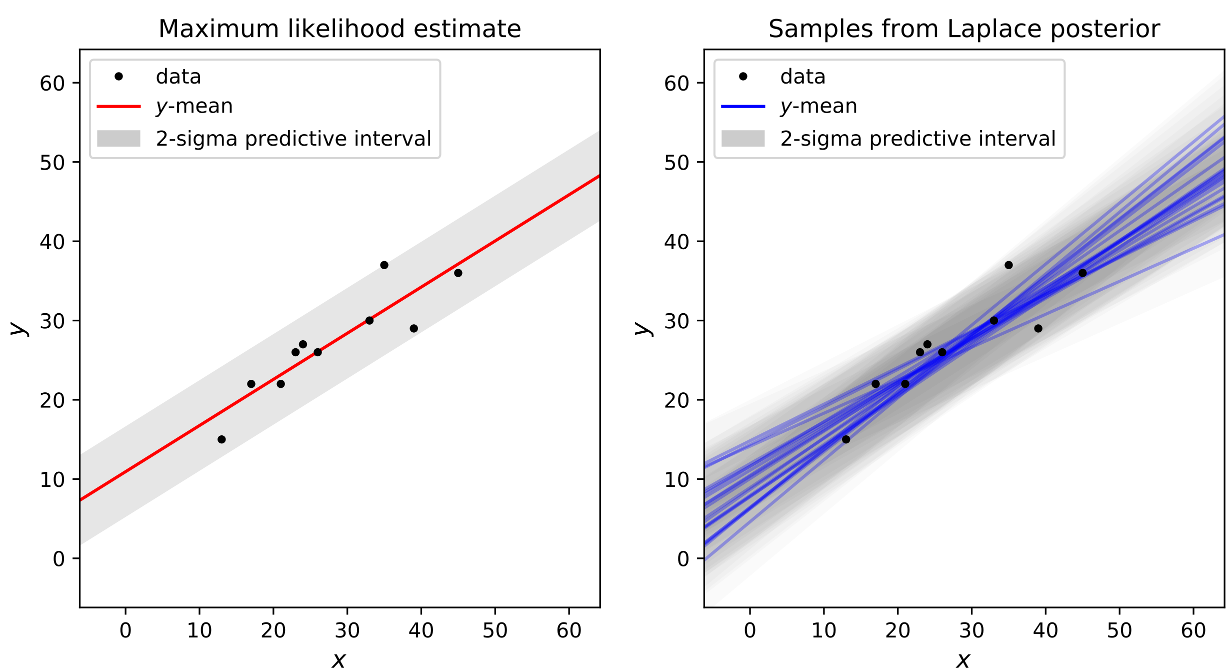
giving

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

and

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

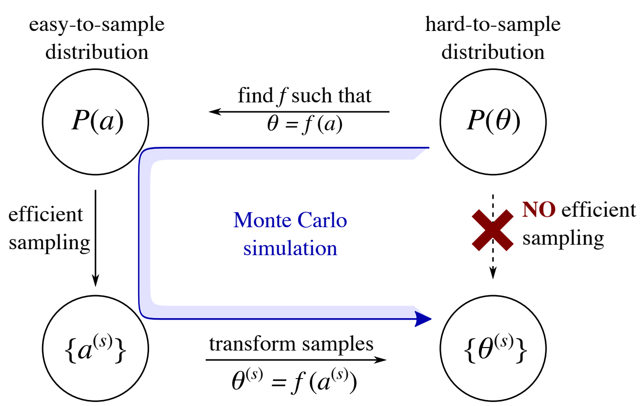

MLE
Laplace approximation
exact posterior
−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5
a
MLE
Laplace approximation
exact posterior
−30 −20 −10 0 10 20 30
b
MLE
Laplace approximation
exact posterior
0 10 20 30 40 50 60 70
σ2


Maximum likelihood estimate
data
y-mean
2-sigma predictive interval
0 10 20 30 40 50 60
x
y
Samples from Laplace posterior
data
y-mean
2-sigma predictive interval
0 10 20 30 40 50 60
x
y

7 Monte Carlo methods

7.1 Monte Carlo simulation

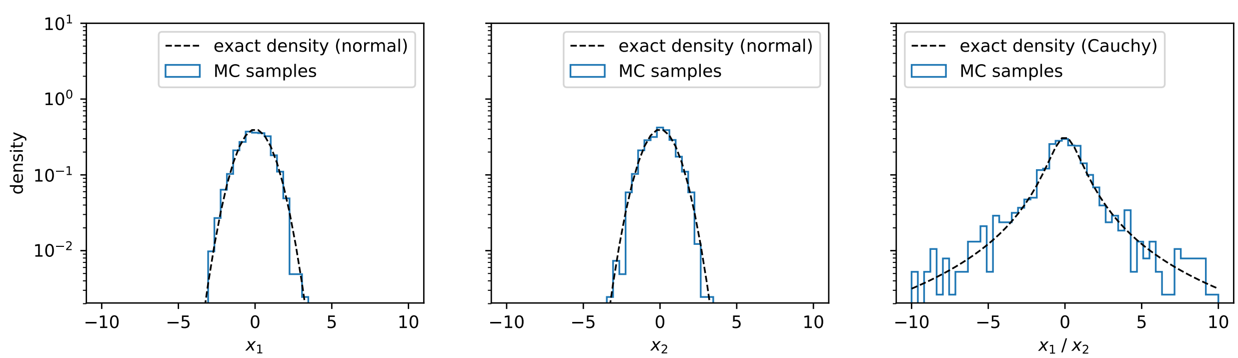
- **Goal:** Analyze complicated distributions by drawing samples from them: $P(\theta) \mapsto \{\theta^{(s)}\}$.
- **Challenge:** From the vast majority of distributions, we don't know how to draw samples efficiently.
- **Method:**
  1. Draw samples from an easy-to-sample distribution.
  2. Transform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

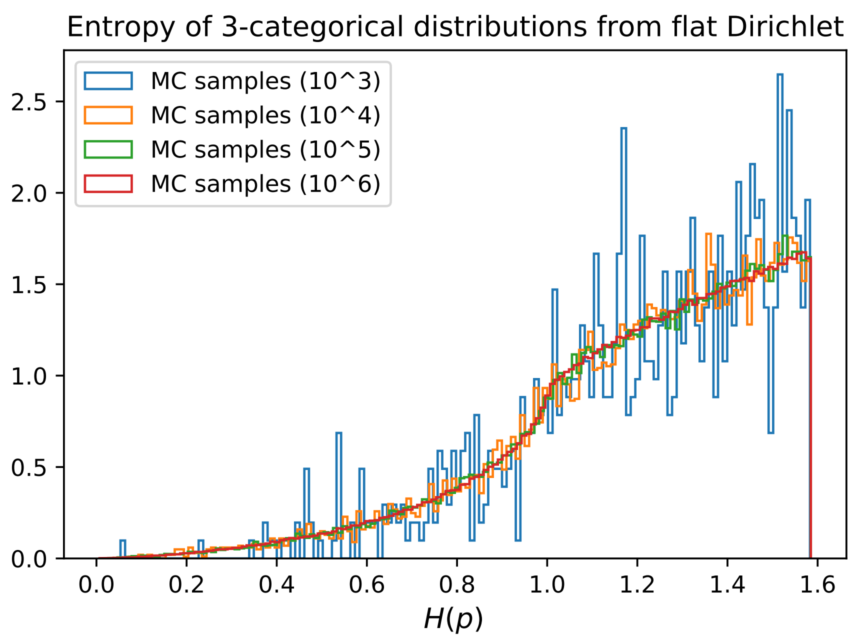
```
from scipy.stats import norm

samples = 1000
X1 = norm.rvs(loc=0, scale=1, size=samples)
X2 = norm.rvs(loc=0, scale=1, size=samples)
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 $P(p) = \mathrm{Dirichlet}(p \mid \alpha = (1,1,1))$
- Output: $h := H(p) = -\sum_k p_k \log_2 p_k$, $\;P(h) = ?$
- MC method:

```
import numpy as np
from scipy.stats import dirichlet

def entropy(p):
    h = 0
    for pk in p:
        if pk > 0:
            h += - pk * np.log2(pk)
    return h

alpha = (1,1,1)
sample_size = 10_000
p_samples = dirichlet.rvs(alpha, size=sample_size)
h_samples = []
for p in p_samples:
    h_samples.append(entropy(p))
```



Example: Monty Hall problem

- Input:
  - In a game show, there a 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), pick 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.

$$\begin{aligned} P(\text{win} \mid \text{switch}) = P(r = p_2 \mid p_2 \neq p_1) &= ? \\ P(\text{win} \mid \text{don't switch}) = P(r = p_2 \mid p_2 = p_1) &= ? \end{aligned}$$

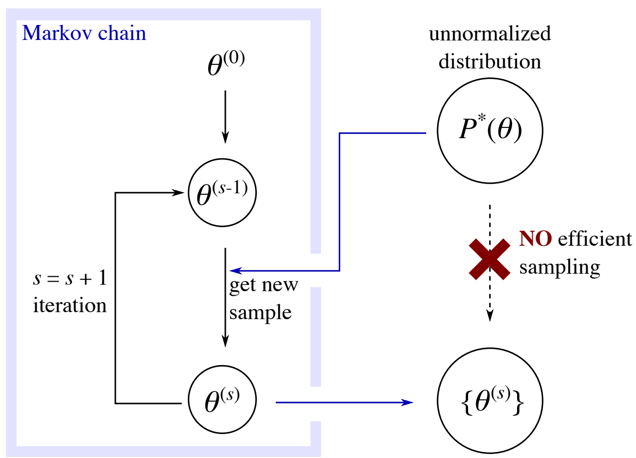
- MC method:

```python
from numpy.random import choice

doors = {1, 2, 3}

games = 1000
wins_with_switch = 0
wins_with_no_switch = 0
for g in range(games):
    reward = choice(list(doors))
    pick_1 = choice(list(doors))

    can_open = doors - set([pick_1]).union(set([reward]))
    openned = choice(list(can_open))
    remaining = doors - set([openned])

    pick_2 = list(remaining - set([pick_1]))[0]
    if pick_2 == reward:
        wins_with_switch += 1

    pick_2 = pick_1
    if pick_2 == reward:
        wins_with_no_switch += 1

P_win_with_switch = wins_with_switch / float(games)
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 $P^*()$ 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 ε 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 $\Delta L >= 0$, 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: $\log P^*(x) = x/2 - \left(1 - x^2\right)^2$, where $x \in \mathbb{R}$.
- 1-dimensional Metropolis-Hastings sampler:

```
def propose_MH(x, stepsize):
    epsilon = norm.rvs(loc=0, scale=stepsize)
    return x + epsilon

def new_sample_MH(x_current, x_proposed, log_Pstar):
    delta_L = log_Pstar(x_proposed) - log_Pstar(x_current)
    if delta_L >= 0:
        return x_proposed
    if np.random.random() < np.exp(delta_L):
        return x_proposed
    else:
        return x_current
```

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

```
def log_Pstar_camel(x):
    return 0.5 * x -(1 - x**2)**2

x0 = 0
stepsize = 0.1
iterations = 20000
x_samples = []

x_curr = x0
for it in range(iterations):
    x_proposed = propose_MH(x_curr, stepsize)
    x = new_sample_MH(x_curr, x_proposed, log_Pstar_camel)
    x_samples.append(x)
    x_curr = x
```