Conjugate prior summary

Distribution	Likelihood $p(\mathbf{x}\|\theta)$	Prior $p(\theta)$	Distribution
Bernoulli	$(1-\theta)^{(1-x)}\theta^x$	$\propto (1-\theta)^{(a-1)}\theta^{(b-1)}$	Beta
Binomial	$\propto (1-\theta)^{(N-x)}\theta^x$	$\propto (1-\theta)^{(a-1)}\theta^{(b-1)}$	Beta
Multinomial	$\propto \theta_1^{x_1}\theta_2^{x_2}\theta_3^{x_3}$	$\propto \theta_1^{\alpha_1-1}\theta_2^{\alpha_2-1}\theta_3^{\alpha_3-1}$	Dirichlet
Normal (fixed σ^2)	$\propto \exp\left(-\frac{(x-\theta)^2}{2\sigma^2}\right)$	$\propto \exp\left(-\frac{(\theta-\mu_0)^2}{2\sigma_0^2}\right)$	Normal
Normal (fixed μ)	$\propto \sqrt{\theta}\exp\left(-\frac{\theta(x-\mu)^2}{2}\right)$	$\propto \theta^{a-1}exp(-b\theta)$	Gamma
Poisson	$\propto \theta^x\exp(-\theta)$	$\propto \theta^{a-1}exp(-b\theta)$	Gamma

An example

- Simple economy: m prosumers, n different goods[1]
- Each individual: production $\mathbf{p}_i \in \mathbb{R}_n$, consumption $\mathbf{c}_i \in \mathbb{R}_n$
- Expense of producing "$\mathbf{p}$" for agent $i = e_i(\mathbf{p})$
- Utility (happiness) of consuming "$\mathbf{c}$" units for agent $i = u_i(\mathbf{c})$
- Maximize happiness

$$\max_{\mathbf{p}_i,\mathbf{c}_i} \sum_i (u_i(\mathbf{c}_i) - e_i(\mathbf{p}_i)) \qquad s.t. \qquad \sum_i \mathbf{c}_i = \sum_i \mathbf{p}_i$$

[1] Example borrowed from the first lecture of Prof Gordon's CMU CS 10-725

Walrasian equilibrium

$$\max_{\mathbf{p}_i,\mathbf{c}_i} \sum_i (u_i(\mathbf{c}_i) - e_i(\mathbf{p}_i)) \qquad s.t. \qquad \sum_i \mathbf{c}_i = \sum_i \mathbf{p}_i$$

- Idea: introduce price λ_j to each good j. Let the market decide
  - Price $\lambda_j \uparrow$: consumption of good $j \downarrow$, production of good $j \uparrow$
  - Price $\lambda_j \downarrow$: consumption of good $j \uparrow$, production of good $j \downarrow$
  - Can adjust price until consumption $=$ production for each good

Algorithm: tâtonnement

Assume that the appropriate prices are found, we can ignore the equality constraint, then the problem becomes

$$\max_{\mathbf{p}_i,\mathbf{c}_i} \sum_i (u_i(\mathbf{c}_i) - e_i(\mathbf{p}_i)) \quad \Rightarrow \quad \sum_i \max_{\mathbf{p}_i,\mathbf{c}_i} (u_i(\mathbf{c}_i) - e_i(\mathbf{p}_i))$$

So we can simply optimize production and consumption of each individual independently

Algorithm 1 tâtonnement

```
procedure FindBestPrices
    λ ← [0, 0, ··· , 0]
    for k = 1, 2, ··· do
        Each individual solves for its c_i and p_i for the given λ
        λ ← λ + δ_k Σ_i (c_i − p_i)
```

Lagrange multiplier

Problem

$$\max_{\mathbf{x}} f(\mathbf{x})$$

$$g(\mathbf{x}) = 0$$

Consider $L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda g(\mathbf{x})$ and let $\tilde{f}(\mathbf{x}) = \min_\lambda L(\mathbf{x}, \lambda)$.

Lagrange multiplier

Problem

$$\max_{\mathbf{x}} f(\mathbf{x})$$

$$g(\mathbf{x}) = 0$$

Consider $L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda g(\mathbf{x})$ and let $\tilde{f}(\mathbf{x}) = \min_\lambda L(\mathbf{x}, \lambda)$. Note that

$$\tilde{f}(\mathbf{x}) = \begin{cases} f(\mathbf{x}) \text{ if } g(\mathbf{x}) = 0 \\ -\infty \text{ otherwise} \end{cases}$$

Lagrange multiplier

Problem

$$\max_{\mathbf{x}} f(\mathbf{x})$$

$$g(\mathbf{x}) = 0$$

Consider $L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda g(\mathbf{x})$ and let $\tilde{f}(\mathbf{x}) = \min_\lambda L(\mathbf{x}, \lambda)$. Note that

$$\tilde{f}(\mathbf{x}) = \begin{cases} f(\mathbf{x}) \text{ if } g(\mathbf{x}) = 0 \\ -\infty \text{ otherwise} \end{cases}$$

Therefore, the problem is identical to $\max_{\mathbf{x}} \tilde{f}(\mathbf{x})$ or

$$\max_{\mathbf{x}} \min_{\lambda} (f(\mathbf{x}) - \lambda g(\mathbf{x})),$$

where λ is known to be the Lagrange multiplier.

Lagrange multiplier (con't)

Assume the optimum is a saddle point,

$$\max_{\mathbf{x}} \min_{\lambda} (f(\mathbf{x}) - \lambda g(\mathbf{x})) = \min_{\lambda} \max_{\mathbf{x}} (f(\mathbf{x}) - \lambda g(\mathbf{x})),$$

the R.H.S. implies

$$\nabla f(\mathbf{x}) = \lambda \nabla g(\mathbf{x})$$

Inequality constraint

Problem

$$\max_{\mathbf{x}} f(\mathbf{x})$$

$$g(\mathbf{x}) \leq 0$$

Consider $\tilde{f}(\mathbf{x}) = \min_{\lambda \geq 0}(f(\mathbf{x}) - \lambda g(\mathbf{x}))$,

Inequality constraint

Problem

$$\max_{\mathbf{x}} f(\mathbf{x})$$
$$g(\mathbf{x}) \le 0$$

Consider $\tilde{f}(\mathbf{x}) = \min_{\lambda \ge 0}(f(\mathbf{x}) - \lambda g(\mathbf{x}))$, note that

$$\tilde{f}(\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{if } g(\mathbf{x}) \le 0 \\ -\infty & \text{otherwise} \end{cases}$$

Inequality constraint

Problem

$$\max_{\mathbf{x}} f(\mathbf{x})$$
$$g(\mathbf{x}) \leq 0$$

Consider $\tilde{f}(\mathbf{x}) = \min_{\lambda \geq 0}(f(\mathbf{x}) - \lambda g(\mathbf{x}))$, note that

$$\tilde{f}(\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{if } g(\mathbf{x}) \leq 0 \\ -\infty & \text{otherwise} \end{cases}$$

Therefore, we can rewrite the problem as

$$\max_{\mathbf{x}} \min_{\lambda \geq 0} (f(\mathbf{x}) - \lambda g(\mathbf{x}))$$

Inequality constraint (con't)

Assume

$$\max_{\mathbf{x}} \min_{\lambda \geq 0} (f(\mathbf{x}) - \lambda g(\mathbf{x})) = \min_{\lambda \geq 0} \max_{\mathbf{x}} (f(\mathbf{x}) - \lambda g(\mathbf{x}))$$

The R.H.S. implies

$$\nabla f(\mathbf{x}) = \lambda \nabla g(\mathbf{x})$$

Inequality constraint (con't)

Assume

$$\max_{\mathbf{x}} \min_{\lambda \geq 0} (f(\mathbf{x}) - \lambda g(\mathbf{x})) = \min_{\lambda \geq 0} \max_{\mathbf{x}} (f(\mathbf{x}) - \lambda g(\mathbf{x}))$$

The R.H.S. implies

$$\nabla f(\mathbf{x}) = \lambda \nabla g(\mathbf{x})$$

Moreover, at the optimum point $(\mathbf{x}^*, \lambda^*)$, we should have the so-called "complementary slackness" condition

$$\lambda^* g(\mathbf{x}^*) = 0$$

since

$$\max_{\substack{\mathbf{x} \\ g(\mathbf{x}) \leq 0}} f(\mathbf{x}) \equiv \max_{\mathbf{x}} \min_{\lambda \geq 0} (f(\mathbf{x}) - \lambda g(\mathbf{x}))$$

Karush-Kuhn-Tucker conditions

Problem

$$\max_{\mathbf{x}} f(\mathbf{x})$$

$$g(\mathbf{x}) \leq 0, \quad h(\mathbf{x}) = 0$$

Conditions

$$\begin{aligned} \nabla f(\mathbf{x}^*) - \mu^* \nabla g(\mathbf{x}^*) - \lambda^* \nabla h(\mathbf{x}^*) &= 0 \\ g(\mathbf{x}^*) &\leq 0 \\ h(\mathbf{x}^*) &= 0 \\ \mu^* &\geq 0 \\ \mu^* g(\mathbf{x}^*) &= 0 \end{aligned}$$

Overview of source coding

- The objective of "source coding" is to compress some source

Overview of source coding

- The objective of "source coding" is to compress some source
- We can think of compression as "coding". Meaning that we replace each input by a corresponding coded sequence. So encoding is just a mapping/function process

Overview of source coding

- The objective of “source coding” is to compress some source
- We can think of compression as “coding”. Meaning that we replace each input by a corresponding coded sequence. So encoding is just a mapping/function process
- Without loss of generality, we can use binary domain for our coded sequence. So for each input message, it is converted to a sequence of 1s and 0s

Overview of source coding

- The objective of “source coding” is to compress some source
- We can think of compression as “coding”. Meaning that we replace each input by a corresponding coded sequence. So encoding is just a mapping/function process
- Without loss of generality, we can use binary domain for our coded sequence. So for each input message, it is converted to a sequence of 1s and 0s
- Consider encoding (compressing) a sequence $x_1, x_2, \cdots$ one symbol at a time, resulting $c(x_1), c(x_2), \cdots$

Overview of source coding

- The objective of "source coding" is to compress some source
- We can think of compression as "coding". Meaning that we replace each input by a corresponding coded sequence. So encoding is just a mapping/function process
- Without loss of generality, we can use binary domain for our coded sequence. So for each input message, it is converted to a sequence of 1s and 0s
- Consider encoding (compressing) a sequence $x_1, x_2, \cdots$ one symbol at a time, resulting $c(x_1), c(x_2), \cdots$
- Denote the lengths of $x_1, x_2, \cdots$ as $l(x_1), l(x_2), \cdots$, one of the major goal is to have $E[l(X)]$ to be as small as possible

Overview of source coding

- The objective of "source coding" is to compress some source
- We can think of compression as "coding". Meaning that we replace each input by a corresponding coded sequence. So encoding is just a mapping/function process
- Without loss of generality, we can use binary domain for our coded sequence. So for each input message, it is converted to a sequence of 1s and 0s
- Consider encoding (compressing) a sequence $x_1, x_2, \cdots$ one symbol at a time, resulting $c(x_1), c(x_2), \cdots$
- Denote the lengths of $x_1, x_2, \cdots$ as $l(x_1), l(x_2), \cdots$, one of the major goal is to have $E[l(X)]$ to be as small as possible
- However, we want to make sure that we can losslessly decode the message also!

Uniquely decodable code

- To ensure that we can recover message without loss, we must make sure that no message share the same codeword

Uniquely decodable code

- To ensure that we can recover message without loss, we must make sure that no message share the same codeword
- We say a code is "singular" (broken) if $c(x_1) = c(x_2)$ for some different x_1 and x_2

Uniquely decodable code

- To ensure that we can recover message without loss, we must make sure that no message share the same codeword
- We say a code is "singular" (broken) if $c(x_1) = c(x_2)$ for some different x_1 and x_2
- Even when a code is not "singular", we still cannot guarantee that we can always recover the original message losslessly, consider 4 different possible input symbols a, b, c, d and an encoding map $c(\cdot)$:
  - $a \mapsto 0, b \mapsto 1, c \mapsto 10, d \mapsto 11$
  - What should be the message for 1110?

Uniquely decodable code

- To ensure that we can recover message without loss, we must make sure that no message share the same codeword
- We say a code is "singular" (broken) if $c(x_1) = c(x_2)$ for some different x_1 and x_2
- Even when a code is not "singular", we still cannot guarantee that we can always recover the original message losslessly, consider 4 different possible input symbols a, b, c, d and an encoding map $c(\cdot)$:
  - $a \mapsto 0, b \mapsto 1, c \mapsto 10, d \mapsto 11$
  - What should be the message for 1110?
    - *dba*? Or *bbba*?

Uniquely decodable code

- To ensure that we can recover message without loss, we must make sure that no message share the same codeword
- We say a code is "singular" (broken) if $c(x_1) = c(x_2)$ for some different x_1 and x_2
- Even when a code is not "singular", we still cannot guarantee that we can always recover the original message losslessly, consider 4 different possible input symbols a, b, c, d and an encoding map $c(\cdot)$:
    - $a \mapsto 0, b \mapsto 1, c \mapsto 10, d \mapsto 11$
    - What should be the message for 1110?
        - *dba*? Or *bbba*?
- So it is not sufficient to just have $c(\cdot)$ to map to different output for each input. Let's overload the notation $c(\cdot)$ a little bit and for any message sequence $\mathbf{x} = x_1, x_2, \cdots, x_n$, encode sequence $x_1, x_2, \cdots, x_n$ to $c(\mathbf{x}) = c(x_1, x_2, \cdots, x_n) = c(x_1)c(x_2)\cdots c(x_n)$

Uniquely decodable code

- To ensure that we can recover message without loss, we must make sure that no message share the same codeword
- We say a code is "singular" (broken) if $c(x_1) = c(x_2)$ for some different x_1 and x_2
- Even when a code is not "singular", we still cannot guarantee that we can always recover the original message losslessly, consider 4 different possible input symbols a, b, c, d and an encoding map $c(\cdot)$:
  - $a \mapsto 0, b \mapsto 1, c \mapsto 10, d \mapsto 11$
  - What should be the message for 1110?
    - *dba*? Or *bbba*?
- So it is not sufficient to just have $c(\cdot)$ to map to different output for each input. Let's overload the notation $c(\cdot)$ a little bit and for any message sequence $\mathbf{x} = x_1, x_2, \cdots, x_n$, encode sequence $x_1, x_2, \cdots, x_n$ to $c(\mathbf{x}) = c(x_1, x_2, \cdots, x_n) = c(x_1)c(x_2)\cdots c(x_n)$
  - We say $c(\mathbf{x})$ is uniquely decodable if all input sequences map to different outputs

Prefix-free code

- For practical purpose, we would like to be able to decode a symbol "once it is available". Consider a code with map
    - $a \mapsto 10, b \mapsto 00, c \mapsto 11, d \mapsto 110$

Prefix-free code

- For practical purpose, we would like to be able to decode a symbol "once it is available". Consider a code with map
  - $a \mapsto 10, b \mapsto 00, c \mapsto 11, d \mapsto 110$
  - One can show that it is uniquely decodable. However, consider an input sequence $cbbb \mapsto 11000000$

Prefix-free code

- For practical purpose, we would like to be able to decode a symbol "once it is available". Consider a code with map
  - $a \mapsto 10, b \mapsto 00, c \mapsto 11, d \mapsto 110$
  - One can show that it is uniquely decodable. However, consider an input sequence $cbbb \mapsto 11000000$
  - When the decoder read the first 3 bits, it is not able to determine if the first input symbol is c or d

Prefix-free code

- For practical purpose, we would like to be able to decode a symbol "once it is available". Consider a code with map
  - $a \mapsto 10, b \mapsto 00, c \mapsto 11, d \mapsto 110$
  - One can show that it is uniquely decodable. However, consider an input sequence $cbbb \mapsto 11000000$
  - When the decoder read the first 3 bits, it is not able to determine if the first input symbol is c or d
  - Actually, it will be until the decoder read the last bit that it will be able to confirm that the first input symbol is c. It is definitely not something very desirable

Prefix-free code

- For practical purpose, we would like to be able to decode a symbol "once it is available". Consider a code with map
  - $a \mapsto 10, b \mapsto 00, c \mapsto 11, d \mapsto 110$
  - One can show that it is uniquely decodable. However, consider an input sequence $cbbb \mapsto 11000000$
  - When the decoder read the first 3 bits, it is not able to determine if the first input symbol is c or d
  - Actually, it will be until the decoder read the last bit that it will be able to confirm that the first input symbol is c. It is definitely not something very desirable
- Instead, for a mapping $a \mapsto 1, b \mapsto 01, c \mapsto 001, d \mapsto 0001$, I will argue that we can always decode a symbol "once it is available"

Prefix-free code

- For practical purpose, we would like to be able to decode a symbol "once it is available". Consider a code with map
  - $a \mapsto 10, b \mapsto 00, c \mapsto 11, d \mapsto 110$
  - One can show that it is uniquely decodable. However, consider an input sequence $cbbb \mapsto 11000000$
  - When the decoder read the first 3 bits, it is not able to determine if the first input symbol is c or d
  - Actually, it will be until the decoder read the last bit that it will be able to confirm that the first input symbol is c. It is definitely not something very desirable
- Instead, for a mapping $a \mapsto 1, b \mapsto 01, c \mapsto 001, d \mapsto 0001$, I will argue that we can always decode a symbol "once it is available"
  - Note that the catch is that there is no codeword being the "prefix" of another codeword
  - We call such code a prefix-free code or an instantaneous code

Kraft's Inequality

Let $l_1, l_2, \cdots, l_K$ satisfy $\sum_{k=1}^{K} 2^{-l_k} \leq 1$. Then, there exists a uniquely decodable code for symbols $x_1, x_2, \cdots, x_K$ such that $l(x_1) = l_1$, $l(x_2) = l_2, \cdots, l(x_K) = l_K$.

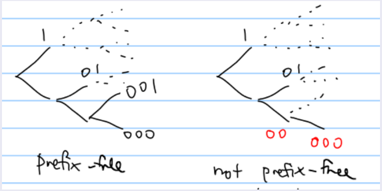
Kraft's Inequality

Let $l_1, l_2, \cdots, l_K$ satisfy $\sum_{k=1}^{K} 2^{-l_k} \leq 1$. Then, there exists a uniquely decodable code for symbols $x_1, x_2, \cdots, x_K$ such that $l(x_1) = l_1$, $l(x_2) = l_2, \cdots, l(x_K) = l_K$.

Intuition

Consider # "descendants" of each codeword at the "l_{max}"-level, then for prefix-free code, we have

$$\sum_{k=1}^{K} 2^{l_{max}-l} \leq 2^{l_{max}}$$



Kraft's Inequality

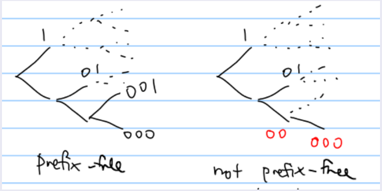
Let $l_1, l_2, \cdots, l_K$ satisfy $\sum_{k=1}^{K} 2^{-l_k} \le 1$. Then, there exists a uniquely decodable code for symbols $x_1, x_2, \cdots, x_K$ such that $l(x_1) = l_1$, $l(x_2) = l_2, \cdots, l(x_K) = l_K$.

Intuition

Consider # "descendants" of each codeword at the "l_{max}"-level, then for prefix-free code, we have

$$\sum_{k=1}^{K} 2^{l_{max}-l} \le 2^{l_{max}}$$

$$\Rightarrow \sum_{k=1}^{K} 2^{-l_k} \le 1$$



Forward Proof

Given $l_1, l_2, \cdots, l_K$ satisfy $\sum_{k=1}^{K} 2^{-l_k} \leq 1$, we can assign nodes on a tree as previous slides. More precisely,

- Assign i-th node as a node at level l_i, then cross out all its descendants
- Repeat the procedure for i from 1 to K
- We know that there are sufficient tree nodes to be assigned since the Kraft's inequaltiy is satisfied

The corresponding code is apparently prefix-free and thus is uniquely decodable

Converse Proof

Consider message from coding k symbols $\mathbf{x} = x_1, x_2, \cdots, x_k$

$$\left(\sum_{x\in\mathcal{X}} 2^{-l(x)}\right)^k = \left(\sum_{x_1\in\mathcal{X}} 2^{-l(x_1)}\right)\left(\sum_{x_2\in\mathcal{X}} 2^{-l(x_2)}\right)\cdots\left(\sum_{x_k\in\mathcal{X}} 2^{-l(x_k)}\right)$$

$$= \sum_{x_1,x_2,\cdots,x_k\in\mathcal{X}^k} 2^{-(l(x_1)+l(x_2)+\cdots+l(x_k))}$$

$$= \sum_{\mathbf{x}\in\mathcal{X}^k} 2^{-l(\mathbf{x})}$$

Converse Proof

Consider message from coding k symbols $\mathbf{x} = x_1, x_2, \cdots, x_k$

$$\left(\sum_{x\in\mathcal{X}} 2^{-l(x)}\right)^k = \left(\sum_{x_1\in\mathcal{X}} 2^{-l(x_1)}\right)\left(\sum_{x_2\in\mathcal{X}} 2^{-l(x_2)}\right)\cdots\left(\sum_{x_k\in\mathcal{X}} 2^{-l(x_k)}\right)$$
$$= \sum_{x_1,x_2,\cdots,x_k\in\mathcal{X}^k} 2^{-(l(x_1)+l(x_2)+\cdots+l(x_k))}$$
$$= \sum_{\mathbf{x}\in\mathcal{X}^k} 2^{-l(\mathbf{x})} = \sum_{m=1}^{kl_{max}} a(m)2^{-m},$$

where $a(m)$ is the number of codeword with length m. However, for the code to be uniquely decodable, $a(m) \leq 2^m$, where 2^m is the number of available codewords with length m.

Converse Proof

Consider message from coding k symbols $\mathbf{x} = x_1, x_2, \cdots, x_k$

$$\left(\sum_{x\in\mathcal{X}} 2^{-l(x)}\right)^k = \left(\sum_{x_1\in\mathcal{X}} 2^{-l(x_1)}\right)\left(\sum_{x_2\in\mathcal{X}} 2^{-l(x_2)}\right)\cdots\left(\sum_{x_k\in\mathcal{X}} 2^{-l(x_k)}\right)$$
$$= \sum_{x_1,x_2,\cdots,x_k\in\mathcal{X}^k} 2^{-(l(x_1)+l(x_2)+\cdots+l(x_k))}$$
$$= \sum_{\mathbf{x}\in\mathcal{X}^k} 2^{-l(\mathbf{x})} = \sum_{m=1}^{kl_{max}} a(m)2^{-m},$$

where $a(m)$ is the number of codeword with length m. However, for the code to be uniquely decodable, $a(m) \le 2^m$, where 2^m is the number of available codewords with length m. Therefore,

$$\sum_{x\in\mathcal{X}} 2^{-l(x)} \le (kl_{max})^{1/k}$$

Converse Proof

Consider message from coding k symbols $\mathbf{x} = x_1, x_2, \cdots, x_k$

$$\left(\sum_{x\in\mathcal{X}} 2^{-l(x)}\right)^k = \left(\sum_{x_1\in\mathcal{X}} 2^{-l(x_1)}\right)\left(\sum_{x_2\in\mathcal{X}} 2^{-l(x_2)}\right)\cdots\left(\sum_{x_k\in\mathcal{X}} 2^{-l(x_k)}\right)$$
$$= \sum_{x_1,x_2,\cdots,x_k\in\mathcal{X}^k} 2^{-(l(x_1)+l(x_2)+\cdots+l(x_k))}$$
$$= \sum_{\mathbf{x}\in\mathcal{X}^k} 2^{-l(\mathbf{x})} = \sum_{m=1}^{kl_{max}} a(m)2^{-m},$$

where $a(m)$ is the number of codeword with length m. However, for the code to be uniquely decodable, $a(m) \leq 2^m$, where 2^m is the number of available codewords with length m. Therefore,

$$\sum_{x\in\mathcal{X}} 2^{-l(x)} \leq (kl_{max})^{1/k} \approx 1 \text{ as } k \to \infty$$

Minimum rate required to compress a source

$$min_{l_1,l_2,\cdots,l_K} \sum_{k=1}^{K} p_k l_k \text{ subject to } \sum_{k=1}^{K} 2^{-l_k} \leq 1 \text{ and } l_1,\cdots,l_K \geq 0$$

$$\equiv max_{l_1,l_2,\cdots,l_K} -\sum_{k=1}^{K} p_k l_k \text{ subject to } \sum_{k=1}^{K} 2^{-l_k} - 1 \leq 0 \text{ and } -l_1,\cdots,-l_K \leq 0$$

KKT conditions

$$-\nabla\left(\sum_{k=1}^{K} p_k l_k\right) - \mu_0 \nabla\left(\sum_{k=1}^{K} 2^{-l_k} - 1\right) + \sum_{k=1}^{K} \mu_k \nabla l_k = 0$$

Minimum rate required to compress a source

$$min_{l_1,l_2,\cdots,l_K} \sum_{k=1}^{K} p_k l_k \text{ subject to } \sum_{k=1}^{K} 2^{-l_k} \leq 1 \text{ and } l_1,\cdots,l_K \geq 0$$

$$\equiv max_{l_1,l_2,\cdots,l_K} -\sum_{k=1}^{K} p_k l_k \text{ subject to } \sum_{k=1}^{K} 2^{-l_k} - 1 \leq 0 \text{ and } -l_1,\cdots,-l_K \leq 0$$

KKT conditions

$$-\nabla\left(\sum_{k=1}^{K} p_k l_k\right) - \mu_0 \nabla\left(\sum_{k=1}^{K} 2^{-l_k} - 1\right) + \sum_{k=1}^{K} \mu_k \nabla l_k = 0$$

$$\sum_{k=1}^{K} 2^{-l_k} - 1 \leq 0, \quad l_1,\cdots,l_K \geq 0, \quad \mu_0,\mu_1,\cdots,\mu_K \geq 0$$

Minimum rate required to compress a source

$$min_{l_1,l_2,\cdots,l_K} \sum_{k=1}^{K} p_k l_k \text{ subject to } \sum_{k=1}^{K} 2^{-l_k} \leq 1 \text{ and } l_1,\cdots,l_K \geq 0$$

$$\equiv max_{l_1,l_2,\cdots,l_K} -\sum_{k=1}^{K} p_k l_k \text{ subject to } \sum_{k=1}^{K} 2^{-l_k} - 1 \leq 0 \text{ and } -l_1,\cdots,-l_K \leq 0$$

KKT conditions

$$-\nabla\left(\sum_{k=1}^{K} p_k l_k\right) - \mu_0 \nabla\left(\sum_{k=1}^{K} 2^{-l_k} - 1\right) + \sum_{k=1}^{K} \mu_k \nabla l_k = 0$$

$$\sum_{k=1}^{K} 2^{-l_k} - 1 \leq 0, \quad l_1,\cdots,l_K \geq 0, \quad \mu_0,\mu_1,\cdots,\mu_K \geq 0$$

$$\mu_0\left(\sum_{k=1}^{K} 2^{-l_k} - 1\right) = 0, \quad \mu_k l_k = 0$$

Minimum rate required to compress a source

Since we expect $l_k > 0$, $\mu_k = 0$.

Minimum rate required to compress a source

Since we expect $l_k > 0$, $\mu_k = 0$. Expand the first equation, we get

$$-p_j + \mu_0 2^{-l_j} \log 2 = 0 \Rightarrow 2^{-l_j} = \frac{p_j}{\mu_0 \log 2}$$

Minimum rate required to compress a source

Since we expect $l_k > 0$, $\mu_k = 0$. Expand the first equation, we get

$$-p_j + \mu_0 2^{-l_j} \log 2 = 0 \Rightarrow 2^{-l_j} = \frac{p_j}{\mu_0 \log 2}$$

And by $\sum_{k=1}^K 2^{-l_k} \leq 1$, we have

$$\sum_{k=1}^K \frac{p_j}{\mu_0 \log 2} = \frac{1}{\mu_0 \log 2} \leq 1 \Rightarrow \mu_0 \geq \frac{1}{\log 2}$$

Minimum rate required to compress a source

Since we expect $l_k > 0$, $\mu_k = 0$. Expand the first equation, we get

$$-p_j + \mu_0 2^{-l_j} \log 2 = 0 \Rightarrow 2^{-l_j} = \frac{p_j}{\mu_0 \log 2}$$

And by $\sum_{k=1}^{K} 2^{-l_k} \leq 1$, we have

$$\sum_{k=1}^{K} \frac{p_j}{\mu_0 \log 2} = \frac{1}{\mu_0 \log 2} \leq 1 \Rightarrow \mu_0 \geq \frac{1}{\log 2}$$

Note that as $\mu_0 \downarrow$, $\frac{p_j}{\mu_0 \log 2} \uparrow$ and $l_j \downarrow$.

Minimum rate required to compress a source

Since we expect $l_k > 0$, $\mu_k = 0$. Expand the first equation, we get

$$-p_j + \mu_0 2^{-l_j} \log 2 = 0 \Rightarrow 2^{-l_j} = \frac{p_j}{\mu_0 \log 2}$$

And by $\sum_{k=1}^K 2^{-l_k} \le 1$, we have

$$\sum_{k=1}^K \frac{p_j}{\mu_0 \log 2} = \frac{1}{\mu_0 \log 2} \le 1 \Rightarrow \mu_0 \ge \frac{1}{\log 2}$$

Note that as $\mu_0 \downarrow$, $\frac{p_j}{\mu_0 \log 2} \uparrow$ and $l_j \downarrow$. Therefore, if we want to decrease code rate, we should reduce μ_0 as much as possible. Thus, take $\mu_0 = \frac{1}{\log 2}$. Then $2^{-l_j} = p_j \Rightarrow l_j = -\log_2 p_j$. Thus, the minimum rate becomes

$$\sum_{k=1}^K p_k l_k = -\sum_{k=1}^K p_k \log_2 p_k \triangleq H(p_1, \cdots, p_K)$$

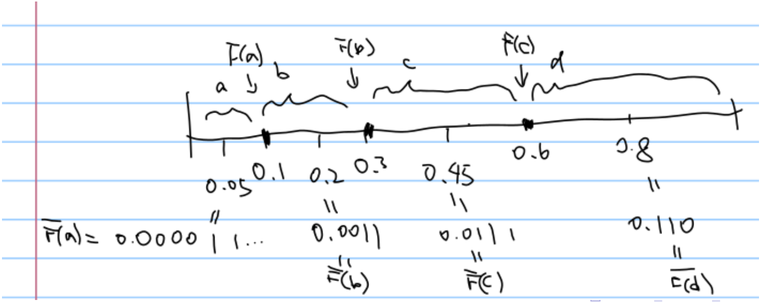
Shannon-Fano-Elias code

Key idea

Each codeword corresponds to an intervel of $[0, 1]$

Example

110 corresponds to $[0.110, 0.110\dot{1}] = [0.11, 0.111) = [0.75, 0.875)$



Shannon-Fano-Elias code

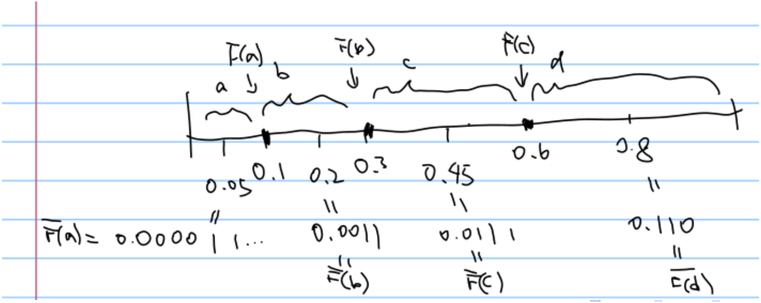
Key idea

Each codeword corresponds to an intervel of $[0, 1]$

Example

110 corresponds to $[0.110, 0.110\dot{1}] = [0.11, 0.111) = [0.75, 0.875)$

011 corresponds to $[0.011, 0.011\dot{1}] = [0.011, 0.1) = [0.375, 0.5)$



Example

Consider a source that
$p(x_1) = 0.25, p(x_2) = 0.25, p(x_3) = 0.2, p(x_4) = 0.15, p(x_5) = 0.15$

x	$p(x)$	$F(x)$	$\overline{F}(x)$	$\overline{F}(x)$ in Binary	$l(x) = \left\lceil \log \frac{1}{p(x)} \right\rceil + 1$	Codeword
1	0.25	0.25	0.125	0.001	3	001
2	0.25	0.5	0.375	0.011	3	011
3	0.2	0.7	0.6	$0.1\overline{0011}$	4	1001
4	0.15	0.85	0.775	$0.110\overline{0011}$	4	1100
5	0.15	1.0	0.925	$0.111\overline{0110}$	4	1110

Property

- The length of the codeword of x is $\lceil \log_2 \frac{1}{p(x)} \rceil + 1$. This ensures that the "code interval" of each codeword does not overlap

Property

- The length of the codeword of x is $\lceil \log_2 \frac{1}{p(x)} \rceil + 1$. This ensures that the "code interval" of each codeword does not overlap
- SFE code is prefix-free $\rightarrow$ uniquely decodable

Property

- The length of the codeword of x is $\lceil \log_2 \frac{1}{p(x)} \rceil + 1$. This ensures that the "code interval" of each codeword does not overlap
- SFE code is prefix-free $\rightarrow$ uniquely decodable
  - If a codeword is prefix of another (say 10 and 1010), the corresponding intervals must overlap each other (consider $[0.10, 0.11)$ and $[0.101, 0.11)$)

Property

- The length of the codeword of x is $\lceil \log_2 \frac{1}{p(x)} \rceil + 1$. This ensures that the "code interval" of each codeword does not overlap
- SFE code is prefix-free $\rightarrow$ uniquely decodable
  - If a codeword is prefix of another (say 10 and 1010), the corresponding intervals must overlap each other (consider $[0.10, 0.11)$ and $[0.101, 0.11)$)
  - Since no codeword can overlap in SFE, no code word can be prefix of another

Property

- The length of the codeword of x is $\lceil \log_2 \frac{1}{p(x)} \rceil + 1$. This ensures that the "code interval" of each codeword does not overlap
- SFE code is prefix-free $\rightarrow$ uniquely decodable
  - If a codeword is prefix of another (say 10 and 1010), the corresponding intervals must overlap each other (consider $[0.10, 0.11)$ and $[0.101, 0.11)$)
  - Since no codeword can overlap in SFE, no code word can be prefix of another
- Average code rate is upper bounded by $H(X) + 2$

$$\sum_{x \in \mathcal{X}} p(x) l(x) = \sum_{x \in \mathcal{X}} p(x) \left(\left\lceil \log_2 \frac{1}{p(x)} \right\rceil + 1 \right)$$
$$\leq \sum_{x \in \mathcal{X}} p(x) \left(\log_2 \frac{1}{p(x)} + 2 \right) = H(X) + 2$$

"Symbol grouping" trick

- Let's consider two symbols as a super-symbol and compress the pair at each time with SFE code
- The code rate is bounded by $H(X_S) + 2$, where

"Symbol grouping" trick

- Let's consider two symbols as a super-symbol and compress the pair at each time with SFE code
- The code rate is bounded by $H(X_S) + 2$, where

$$H(X_S) = -\sum_{x_1, x_2 \in \mathcal{X}^2} p(x_1, x_2) \log_2 p(x_1, x_2)$$

"Symbol grouping" trick

- Let's consider two symbols as a super-symbol and compress the pair at each time with SFE code
- The code rate is bounded by $H(X_S) + 2$, where

$$\begin{aligned} H(X_S) &= -\sum_{x_1,x_2\in\mathcal{X}^2} p(x_1,x_2)\log_2 p(x_1,x_2) \\ &= -\sum_{x_1,x_2\in\mathcal{X}^2} p(x_1,x_2)\log_2(p(x_1)p(x_2)) \end{aligned}$$

"Symbol grouping" trick

- Let's consider two symbols as a super-symbol and compress the pair at each time with SFE code
- The code rate is bounded by $H(X_S) + 2$, where

$$\begin{aligned} H(X_S) &= -\sum_{x_1,x_2 \in \mathcal{X}^2} p(x_1,x_2)\log_2 p(x_1,x_2) \\ &= -\sum_{x_1,x_2 \in \mathcal{X}^2} p(x_1,x_2)\log_2 (p(x_1)p(x_2)) \\ &= -\sum_{x_1,x_2 \in \mathcal{X}^2} p(x_1,x_2)\log_2 p(x_1) - \sum_{x_1,x_2 \in \mathcal{X}^2} p(x_1,x_2)\log_2 p(x_2) \end{aligned}$$

"Symbol grouping" trick

- Let's consider two symbols as a super-symbol and compress the pair at each time with SFE code
- The code rate is bounded by $H(X_S)+2$, where

$$\begin{aligned}
H(X_S) &= -\sum_{x_1,x_2\in\mathcal{X}^2} p(x_1,x_2)\log_2 p(x_1,x_2)\\
&= -\sum_{x_1,x_2\in\mathcal{X}^2} p(x_1,x_2)\log_2(p(x_1)p(x_2))\\
&= -\sum_{x_1,x_2\in\mathcal{X}^2} p(x_1,x_2)\log_2 p(x_1) - \sum_{x_1,x_2\in\mathcal{X}^2} p(x_1,x_2)\log_2 p(x_2)\\
&= -\sum_{x_1\in\mathcal{X}} p(x_1)\log_2 p(x_1) - \sum_{x_2\in\mathcal{X}} p(x_2)\log_2 p(x_2)\\
&= 2H(X)
\end{aligned}$$

Therefore, the code rate per original symbol is upper bounded by

$$\frac{1}{2}(H(X_S)+2) = H(X)+1$$

Forward proof of Source Coding Theorem

In theory, we can group as many symbol as we want (we want do it in practice, why?), say we group N symbols at a time and compress it using SFE code.

Forward proof of Source Coding Theorem

In theory, we can group as many symbol as we want (we want do it in practice, why?), say we group N symbols at a time and compress it using SFE code. The code rate per original symbol is upper bounded by

$$\frac{1}{N}(H(X_S)+2)=\frac{1}{N}(NH(X)+2)=H(X)+\frac{2}{N}$$

Forward proof of Source Coding Theorem

In theory, we can group as many symbol as we want (we want do it in practice, why?), say we group N symbols at a time and compress it using SFE code. The code rate per original symbol is upper bounded by

$$\frac{1}{N}(H(X_S)+2) = \frac{1}{N}(NH(X)+2) = H(X) + \frac{2}{N}$$

Therefore as long as a given rate $R > H(X)$, we can always find a large enough N such that the code rate using the "grouping trick" and SFE code is below R. This concludes the forward proof