Information Theory

Chapter II: Source coding

What does coding do?

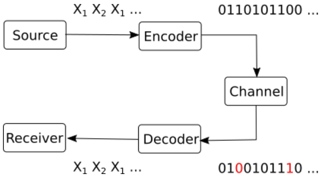


Figure 1: Communication system

- Why coding?

1. Source coding
   - Convert source messages to channel symbols (for example 0,1)
   - Minimize number of symbols needed
   - (Adapt probabilities of symbols to maximize mutual information)
2. Error control
   - Protection against channel errors / Adds new (redundant) symbols

Source-channel separation theorem

Source-channel separation theorem (informal):

- It is possible to obtain the best reliable communication by performing the two tasks separately:
  1. Source coding: to minimize number of symbols needed
  2. Error control coding (channel coding): to provide protection against noise

Source coding

- Assume we code for transmission over ideal channels with no noise
- Transmitted symbols are perfectly recovered at the receiver
- Main concerns:
  - minimize the number of symbols needed to represent the messages
  - make sure we can decode the messages
- Advantages:
  - Efficiency
  - Short communication times
  - Can decode easily

Definitions

- Let $S = \{s_1, s_2, ...s_N\}$ = an input discrete memoryless source
- Let $X = \{x_1, x_2, ...x_M\}$ = the alphabet of the code
  - Example: binary: {0,1}
- A **code** is a mapping from S to the set of all codewords:

$$C = \{c_1, c_2, ...c_N\}$$

Message	Codeword
s_1	$c_1 = x_1 x_2 x_1 ...$
s_2	$c_2 = x_1 x_2 x_2 ...$
...	....
s_N	$c_N = x_2 x_2 x_2 ...$

- Codeword length l_i = the number of symbols in c_i

Encoding and decoding

- **Encoding**: given a sequence of messages, replace each message with its codeword
- **Decoding**: given a sequence of symbols, deduce the original sequence of messages
- Example: at blackboard

Example: ASCII code

Letter	ASCII Code	Binary	Letter	ASCII Code	Binary
a	097	01100001	A	065	01000001
b	098	01100010	B	066	01000010
c	099	01100011	C	067	01000011
d	100	01100100	D	068	01000100
e	101	01100101	E	069	01000101
f	102	01100110	F	070	01000110
g	103	01100111	G	071	01000111
h	104	01101000	H	072	01001000
i	105	01101001	I	073	01001001
j	106	01101010	J	074	01001010
k	107	01101011	K	075	01001011
l	108	01101100	L	076	01001100
m	109	01101101	M	077	01001101
n	110	01101110	N	078	01001110
o	111	01101111	O	079	01001111
p	112	01110000	P	080	01010000
q	113	01110001	Q	081	01010001
r	114	01110010	R	082	01010010
s	115	01110011	S	083	01010011
t	116	01110100	T	084	01010100
u	117	01110101	U	085	01010101
v	118	01110110	V	086	01010110
w	119	01110111	W	087	01010111
x	120	01111000	X	088	01011000
y	121	01111001	Y	089	01011001
z	122	01111010	Z	090	01011010

Figure 2: ASCII code (partial)

Average code length

- How to measure representation efficiency of a code?
- **Average code length** = average of the codeword lengths:

$$\bar{l} = \sum_i p(s_i) l_i$$

- The probability of a codeword = the probability of the corresponding message
- Smaller average length: code more efficient (better)
- How small can the average length be?

Definitions

A code can be:

- **non-singular**: all codewords are different
- **uniquely decodable**: for any received sequence of symbols, there is only one corresponding sequence of messages
  - i.e. no sequence of messages produces the same sequence of symbols
  - i.e. there is never a confusion at decoding
- **instantaneous** (also known as **prefix-free**): no codeword is prefix to another code
  - A *prefix* = a codeword which is the beginning of another codeword

Examples: at the blackboard

The graph of a code

Example at blackboard

Instantaneous codes are uniquely decodable

- Theorem:
  - An instantaneous code is uniquely decodable
- Proof:
  - There is exactly one codeword matching the beginning of the sequence
    - Suppose the true initial codeword is **c**
    - There can't be a shorter codeword **c'**, since it would be prefix to **c**
    - There can't be a longer codeword **c''**, since **c** would be prefix to it
  - Remove first codeword from sequence
  - By the same argument, there is exactly one codeword matching the new beginning, and so on ...
- Note: the converse is not necessary true; there exist uniquely decodable codes which are not instantaneous

Uniquely decodable codes are non-singular

- Theorem:
  - An uniquely decodable code is non-singular
- Proof:
  - If the code is singular, some codewords are not unique (different messages, same codeword)
  - Don't know which of those messages was there => not uniquely decodable
  - So if the code is uniquely-decodable, it must also be non-singular ($A \rightarrow B \Leftrightarrow \overline{B} \rightarrow \overline{A}$)
- Relation between code types:
  - Instantaneous $\subset$ uniquely decodable $\subset$ non-singular

Graph-based decoding of instantaneous codes

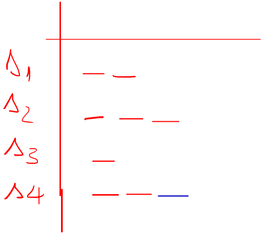
- How to decode an instantaneous code: graph-based decoding
- Advantage on instantaneous code over uniquely decodable: simple decoding
- Why the name *instantaneous*?
    - The codeword can be decoded as soon as it is fully received
    - Counter-example: Uniquely decodable, non-instantaneous, delay 6: $\{0, 01, 011, 1110\}$

Existence of instantaneous codes

- When can an instantaneous code exist?
- Kraft inequality theorem:
  - There exists an instantaneous code with D symbols and codeword lengths $l_1, l_2, \ldots l_n$ if and only if the lengths satisfy the following inequality:

$$\sum_i D^{-l_i} \leq 1.$$

$2 = \{0, 1\}$



$2^{-2} + 2^{-3} + 2^{-1} + 2^{-3} \leq 1$?

0.25 0.125 0.5 ~~0.25~~ > 1 => nu ex. cod instantaneu

0.125 = 1

- Proof: At blackboard
- Comments:
  - If lengths do not satisfy this, no instantaneous code exists
  - If the lengths of a code satisfy this, that code can be instantaneous or not (there exists an instantaneous code, but not necessarily that one)
  - Kraft inequality means that the codewords lengths cannot be all very small

Instantaneous codes with equality in Kraft

- From the proof $=>$ we have equality in the relation

$$\sum_i D^{-l_i} = 1$$

  only if the lowest level is fully covered $<=>$ no unused branches
- For an instantaneous code which satisfies Kraft with equality, all the graph branches terminate with codewords (there are no unused branches)
  - This is most economical: codewords are as short as they can be

Kraft inequality for uniquely decodable codes

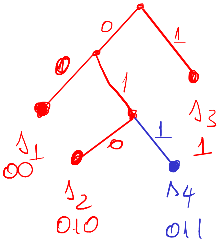
- Instantaneous codes must obey Kraft inequality
- How about uniquely decodable codes?
- McMillan theorem (no proof given):
  - Any uniquely decodable code **also** satisfies the Kraft inequality:

$$\sum_i D^{-l_i} \leq 1.$$

- Consequence:
  - For every uniquely decodable code, there exists in instantaneous code with the same lengths!
  - Even though the class of uniquely decodable codes is larger than that of instantaneous codes, it brings no benefit in codeword length
  - We can always use just instantaneous codes.

Finding an instantaneous code for given lengths

- How to find an instantaneous code with code lengths $\{l_i\}$
  1. Check that lengths satisfy Kraft relation
  2. Draw graph
  3. Assign nodes in a certain order (e.g. descending probability)
- Easy, standard procedure
- Example: at blackboard



Optimal codes

- We want to **minimize the average length** of a code:

$$\underset{\text{minimize}}{\bar{l}} = \sum_i p(s_i) l_i \qquad l_i \geq 1$$

P		Code A	Code B	
0.4	s_1	00	0	0
0.3	s_2	01	10	0
0.2	s_3	11	110	1
0.1	s_4	10	111	1

11010

$\bar{l}_A = 2b$

$\bar{l}_B = 0.4 \cdot 1 + 0.3 \cdot 2 + 0.2 \cdot 3 + 0.1 \cdot 3 = 1.9b$

- But the lengths must obey the Kraft inequality (for uniquely decodable)
- So we reach the following **constrained optimization problem**:

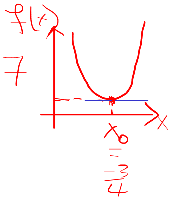
$$\boxed{\begin{aligned} &\textbf{minimize} \quad \sum_i p(s_i) l_i \\ &\text{subject to} \quad \sum_i D^{-l_i} \leq 1 \end{aligned}} = \text{cost function}$$

$\boxed{l_i = \text{integers}}$

$f(x) = 2x^2 + 3x + 7$

$\frac{\partial f}{\partial x} = 0 \qquad 4x + 3 = 0$

$x = -\frac{3}{4}$



The method of Lagrange multipliers

- Method of Lagrange multipliers: standard mathematical tool
- To solve the following constrained optimization problem

$$\begin{aligned} &\textbf{minimize } f(x) \\ &\text{subject to } g(x) = 0 \end{aligned}$$

one must build a new function $L(x, \lambda)$ (the **Lagrangean function**):

1) $$L(x, \lambda) = f(x) - \lambda g(x)$$

and the solution x is among the solutions of the system:

2) $$\begin{cases} \dfrac{\partial L(x, \lambda)}{\partial x} = 0 \\ \dfrac{\partial L(x, \lambda)}{\partial \lambda} = 0 \end{cases}$$

- If there are multiple variables x_i, derivation is done for each one

Solving for minimum average length of code

$(e^x)' = e^x$ $\quad 2^x = 2^x \ln(2)$ $\quad 2^{-x} = 2^{-x}\ln(2)\cdot(-1)$

- In our case:
  - The unknown x are l_i
  - The function is $f(x) = \bar{l} = \sum_i p(s_i) l_i$
  - The constraint is $g(x) = \sum_i D^{-l_i} - 1$

minimize $f(x) = \sum_i p(s_i)\cdot l_i$

such that $g(x) = \underbrace{\sum_i D_2^{-l_i} - 1}_{g(x)} = 0$

$p(s_1) l_1 + \dots + p(s_N) l_N$ $\qquad 2^{-l_1} + 2^{-l_2} + \dots + 2^{-l_N}$

$$L(x,\lambda) = \sum_i p(s_i)\cdot l_i - \lambda\cdot\left(\sum_i 2^{-l_i} - 1\right)$$

$$\begin{cases} \frac{\partial L}{\partial l_1} = p(s_1) + \lambda\cdot 2^{-l_1}\cdot\ln 2 = 0 \\ \frac{\partial L}{\partial l_2} = p(s_2) + \lambda\cdot 2^{-l_2}\cdot\ln 2 = 0 \\ \vdots \\ \frac{\partial L}{\partial l_N} = p(s_N) + \lambda\cdot 2^{-l_N}\cdot\ln 2 = 0 \\ \frac{\partial L}{\partial \lambda} = -\left(\sum 2^{-l_i} - 1\right) = 0 \end{cases}$$

$\iff 1 = \sum_i 2^{-l_i}$

- (Solve at blackboard)
- The optimal values are:

$$\boxed{l_i = -\log_2(p(s_i))}$$

- Intuition: using $l_i = -\log(p(s_i))$ satisfies Kraft with equality, so the lengths cannot be any shorter, in general

$\Rightarrow$ $2^{-l_1} = \frac{-p(s_1)}{\lambda\cdot\ln 2}$

$l_1 = -\log_2\left(\frac{-p(s_1)}{\lambda\cdot\ln 2}\right)$ $\quad \lambda\cdot\ln 2 = -1$

$1 + \lambda\cdot\ln 2\cdot\sum 2^{-l_i} = 0$

$\Rightarrow \lambda\cdot\ln 2 = -1/\left(\sum 2^{-l_i}\right) = -1$

$l_1 = -\log_2(p(s_1))$

$l_2 = -\log_2(p(s_2))$

Optimal lengths

- The optimal codeword lengths are:

$$\boxed{l_i = -\log(p(s_i))}$$

s_1 : $p(s_1)$ = mare => l_i = scurt

- Higher probability => smaller codeword
  - more efficient
  - language examples: "da", "nu", "the", "le" ...
- Smaller probability => longer codeword
  - it appears rarely => no problem
- Overall, we obtain the minimum average length

Entropy = minimal codeword average length

The minimum value of $\overline{\ell} \approx \overline{\ell}_{min} = \sum_i p(s_i) \cdot \ell_i = \boxed{-\sum_i p(s_i) \cdot \log p(s_i)}$

$H(S)$

- If the optimal values are:

$$\boxed{l_i = -\log(p(s_i))}$$

- Then the minimal average length is:

$$\min \bar{l} = \sum_i p(s_i) l_i = -\sum_i p(s_i) \log(p(s_i)) = H(S)$$

- The **entropy** of a source = the **minimum average length** necessary to encode the messages
  - e.g. the minimum number of bits required to represent the data in binary form

Meaning of entropy

$H(S) = 2.3\ b/msg$

- This tells us something about entropy
  - This is what entropy means in practice
  - Small entropy => can be written (encoded) with few bits
  - Large entropy => requires more bits for encoding
- This tells us something about the average length of codes
  - The average length of an uniquely decodable code must be at least as large as the source entropy

$$H(S) \leq \bar{l}$$

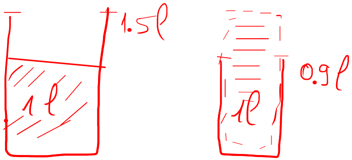
- One can never represent messages, on average, with a code having average length less than the entropy

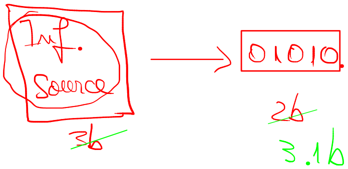
Analogy of entropy and codes

- Analogy: 1 liter of water
  - 1 liter of water = the quantity of water that can fit in any bottle of size $\geq$ 1 liter, but not in any bottle $<$ 1 liter

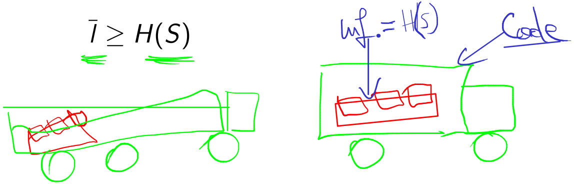
$$Bottle \geq water$$

  - Information of the source = the water
  - The code used for representing the messages = the bottle that carries the water





$$\bar{l} \geq H(S)$$



Efficiency and redundancy of a code

- **Efficiency** of a code ($M = $ size of code alphabet): $=2$ $\{0,1\}$

$$\eta = \frac{H(S)}{\bar{l}\log M}$$

$2 = 2$ $= 1$

$$\eta = \frac{H(S)}{\bar{\ell}} \leq 1$$

$\eta = 95\%$

  - usually $M = 2$ so $\eta = \frac{H(S)}{\bar{l}}$
  - but if $M > 2$ a factor of $\log M$ is needed because $H(S)$ in bits (binary) but $\bar{l}$ not in bits (M symbols)

- **Redundancy** of a code:

$$\rho = 1 - \eta$$

$\rho = 1 - \eta$

$\rho = 5\%$

- These measures indicate how close is the average length to the optimal value
- When $\eta = 1$: **optimal code**

$\bar{\ell} = H(S)$

Optimal codes

nr natural $\quad 2^{-\text{ceva}}$

- Problem: $\boxed{l_i = -\log(p(s_i))}$ might not be an integer number
  - but the codeword lengths must be natural numbers
- An **optimal code** = a code that attains the minimum average length $\bar{l} = H(S)$
- An optimal code can always be found for a source where all $p(s_i)$ are powers of 2
  - e.g. $1/2$, $1/4$, $1/2^n$, known as *dyadic distribution*
  - the lengths $l_i = -\log(p(s_i))$ are all natural numbers => can be attained
  - the code with lengths l_i can be found with the graph-based procedure

$\bar{l} = H(S)$

$-\log_2 l$

$p(s_i) = \boxed{\frac{1}{4}} = 2^{-2}$

$l_i = -\log_2\left(\frac{1}{4}\right) = 2$

Non-optimal codes

- What if $-\log(p(s_i))$ is not a natural number? i.e. $p(s_i)$ is not a power of 2
- Shannon's solution: round to next largest natural number

$$l_i = \lceil -\log(p(s_i)) \rceil$$

i.e. $-\log(p(s_i)) = 2.15 => \underline{l_i = 3}$

$l_1 = 2.15 \simeq 2$

$$2^{-l_1} + 2^{-l_2} + \ldots + 2^{-l_N} = 1$$

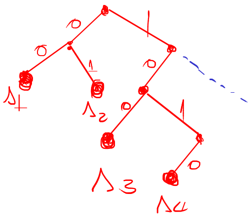
Shannon coding

$\lfloor 2.15 \rfloor = 2$

$\lceil 2.15 \rceil = 3$

	$p(s_i)$	$\lceil -\log_2(p(s_i)) \rceil$	Codewords
s_1	0.4	2	00
s_2	0.3	2	01
s_3	0.2	3	100
s_4	0.1	4	1010

- Shannon coding:
  1. Arrange probabilities in descending order
  2. Use codeword lengths $l_i = \lceil -\log(p(s_i)) \rceil$
  3. Find any instantaneous code for these lengths *
     - * Note: simplified version
     - Shannon actually prescribed the way to compute the codewords
- The code obtained = a "*Shannon code*"
- Simple scheme, better algorithms are available
  - Example: compute lengths for $S : (0.9, 0.1)$
- But still enough to prove fundamental results



Average length of Shannon code

Theorem:

- The average length of a Shannon code satisfies

$$\boxed{\underbrace{H(S)} \leq \bar{l} < \underbrace{H(S) + 1}}$$

$8 \leq 8.2 < 9$

8.7

8.99

Average length of Shannon code

Proof:

1. The first inequality is because H(S) is minimum length
2. The second inequality:
   a. Use Shannon code:

$$l_i = \lceil -\log(p(s_i)) \rceil = -\log(p(s_i)) + \epsilon_i$$

   where $0 \le \epsilon_i < 1$

   b. Compute average length:

$$\bar{l} = \sum_i p(s_i) l_i = H(S) + \underbrace{\sum_i p(s_i)\epsilon_i}_{<1}$$

   c. Since $\epsilon_i < 1 => \sum_i p(s_i)\epsilon_i < \sum_i p(s_i) = 1$

$\lceil 2.3 \rceil = 3$

$2.3 + \underbrace{0.7}_{\varepsilon}$

$H(s) \le \bar{\ell}$

$\boxed{\bar{\ell} \le H(s) + 1}$

$\bar{\ell} = \sum_i p(s_i) \cdot \underbrace{\lceil -\log_2(p(s_i)) \rceil}_{(-\log_2(p(s_i)) + \underbrace{\varepsilon_i}_{<1})}$

$= \underbrace{\sum_i p(s_i)(-\log_2 p(s_i))}_{H(s)} + \underbrace{\sum_i p(s_i) \cdot \underbrace{\varepsilon_i}_{<1}}_{<\sum p(s_i) = 1}$

$\bar{\ell} = H(s) + \text{ceva} < 1$

$\Rightarrow \bar{\ell} < H(s) + 1$

Average length of Shannon code

$$H(S) = 70$$

$$\overline{\ell} \in [70, 71)$$

$$\eta = \frac{H(S)}{\overline{\ell}} > \frac{70}{71} = 98.5\%$$

- Average length of Shannon code is **at most 1 bit longer** than the minimum possible value
  - That's quite efficient
  - There exist even better codes, in general
- Q: Can we get even closer to the minimum length?
- A: Yes, as close as we want!
  - In theory, at least ... :)
  - See next slide.

Shannon's first theorem

Shannon's first theorem (coding theorem for noiseless channels):

- It is possible to encode an infinitely long sequences of messages from a source S with an average length as close as desired to H(S), but never below H(S)

$$\boxed{\bar{\ell} \longrightarrow H(S)}$$

$$\cancel{\bar{\ell} < H(S)}$$

Key points:

- we can always obtain $\bar{l} \to H(S)$
- for an infinitely long sequence

Shannon's first theorem

Proof:

- Average length can never go below H(S) because this is minimum
- How can it get very close to H(S) (from above)?
  1. Use ***n*-th order extension S^n** of S
  2. Use Shannon coding for S^n, so it satisfies

$$\underbrace{H(S^n)}_{n \cdot H(S)} \leq \overline{l_{S^n}} < \underbrace{H(S^n)}_{n \cdot H(S)} + 1 \quad \Big| : n$$

  3. But $H(S^n) = nH(S)$, and average length **per message of** S is

$$\overline{l_S} = \frac{\overline{l_{S^n}}}{n}$$

     because messages of S^n are just n messages of S glued together
  4. So, dividing by n:

$$\boxed{H(S) \leq \overline{l_S} < H(S) + \frac{1}{n}}$$

$n \longrightarrow \infty$

  5. If extension order $n \to \infty$, then

$$\overline{l_S} \to H(S)$$

$S : \cancel{\begin{pmatrix} s_1 & s_2 & \dots & s_n \\ \cdot & \cdot & & \cdot \end{pmatrix}}$

$S^2 : \begin{pmatrix} s_1 s_1 & s_1 s_2 & & s_n s_n \\ \cdot & \cdot & & \cdot \end{pmatrix}$

$S^n : \begin{pmatrix} s_1 s_1 \dots s_2 & \dots \end{pmatrix}$

$H(S^2) = 2 \cdot H(S)$

$$\boxed{H(S^n) = n \cdot H(S)}$$

Shannon's first theorem

- Analogy: how to buy things online without paying for delivery :)
  - FanCourier taxes 15 lei per delivery
    - not efficient to buy something worth a few lei
  - How to improve efficiency? Buy n things bundled together!
  - The delivery cost **per unit** is now $\frac{15}{n}$
  - As $n \to \infty$, the delivery cost per unit $\to 0$
    - What's 15 lei when you pay ∞ lei...

K mesaje

S^n : K^n mesaje $\quad O(K^n)$

$S : (s_1\ s_2\ s_3\ s_4\ s_5)$

S^{10} : (how many?)

$\underbrace{s_1 s_1 \dots s_1}_{10} \dots\dots \underbrace{s_5 s_5 \dots s_5}_{10}$

$\underset{5}{\_}\ \underset{5}{\_}\ \underset{5}{\_} \dots \underset{5}{\_} \quad 5^{10}$

Comments:

- Shannon's first theorem shows that we can approach H(S) to any desired accuracy using extensions of large order of the source
  - This is not practical: the size of S^n gets too large for large n
  - Other (better) algorithms than Shannon coding are used in practice to approach $H(S)$

S : (256)

S^{10} : (256^{10}) $= 2^{80} \approx (1000)^8$ = gigantic!

Coding with the wrong code

- Consider a source with probabilities $\boxed{p(s_i)}$
- We use a code designed for a different source: $\boxed{l_i = -\log(q(s_i))}$!
- The message probabilities are $p(s_i)$ but the code is designed for $q(s_i)$
- Examples:
  - design a code based on a sample data file (like in lab)
  - but we use it to encode various other files => probabilities might differ slightly
  - e.g. design a code based a Romanian text, but encode a text in English
- What happens?

a	b	c
0.7	0.2	0.1
0.69	0.25	0.06

Coding with the wrong code

$p(s_i)$

Ideal

$l_i = -\log_2 p(s_i)$

Real:

$l_i = -\log_2 q(s_i)$

- We lose some efficiency:
  - Codeword lengths $\overline{l_i}$ are not optimal for our source => increased $\overline{l}$
- If code were optimal, best average length = entropy $H(S)$:

$$\overline{l_{optimal}} = -\sum p(s_i) \underbrace{\log p(s_i)}_{l_i}$$

- But the actual average length we obtain is:

$$\overline{l_{actual}} = \sum p(s_i) l_i = -\sum p(s_i) \log q(s_i)$$

$\overline{l_{actual}} - \overline{l_{optimal}}$

The Kullback–Leibler distance

- Difference between average lengths is:

$$\overline{l_{actual}} - \overline{l_{optimal}} = \boxed{\sum_i p(s_i) \log(\frac{p(s_i)}{q(s_i)})} = D_{KL}(p||q)$$

- The difference = **the Kullback-Leibler distance** between the two distributions
  - is always ≥ 0 => improper code means increased $\bar{l}$ (bad)
  - distributions more different => larger average length (worse)
- The KL distance between the distributions = the number of extra bits used because of a code optimized for a different distribution $q(s_i)$ than the true distribution of our data $p(s_i)$

a b c

0.7 0.2 0.1

D_{KL}

0.69 0.15 0.16

The Kullback–Leibler distance

Reminder: where is the Kullback–Leibler distance used

- Here: Using a code optimized for a different distribution:
  - Average length is increased with $D_{KL}(p||q)$
- In chapter IV (Channels): Definition of mutual information:
  - Distance between $p(x_i \cap y_j)$ and the distribution of two independent variables $p(x_i) \cdot p(y_j)$

$$I(X, Y) = \sum_{i,j} p(x_i \cap y_j) \log\left(\frac{p(x_i \cap y_j)}{p(x_i)p(y_j)}\right)$$

Shannon-Fano coding (binary)

Shannon-Fano (binary) coding procedure:

1. Sort the message probabilities in descending order
2. Split into two subgroups as nearly equal as possible
3. Assign first bit 0 to first group, first bit 1 to second group
4. Repeat on each subgroup
5. When reaching one single message => that is the codeword

Example: blackboard

Comments:

- Shannon-Fano coding does not always produce the shortest code lengths
- Connection: yes-no answers (example from first chapter)

$$S : \begin{pmatrix} s_1 & s_2 & s_3 & s_4 \\ 0.4 & 0.3 & 0.2 & 0.1 \end{pmatrix}$$

s_1	0.4	0		
s_2	0.3	1	0	
s_3	0.2	1	1	0
s_4	0.1	1	1	1

0.32

0.25

0.24

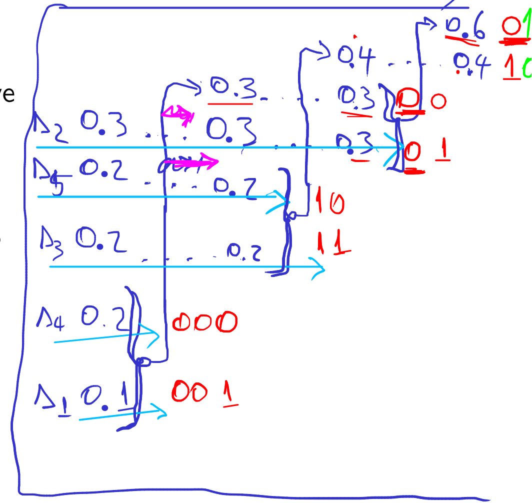
0.05

Huffman coding (binary)

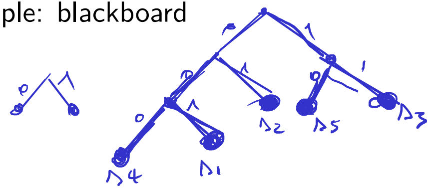
$$S: \begin{pmatrix} s_1 & s_2 & s_3 & s_4 & s_5 \\ 0.1 & 0.3 & 0.2 & 0.2 & 0.2 \end{pmatrix}$$

Huffman coding procedure (binary):

1. Sort the message probabilities in descending order
2. Join the last two probabilities, insert result into existing list, preserve descending order
3. Repeat until only two messages are remaining
4. Assign first bit 0 and 1 to the final two messages
5. Go back step by step: every time we had a sum, append 0 and 1 to the end of existing codeword



Example: blackboard



s_1	001
s_2	~~000~~ 01
s_3	11
s_4	000
s_5	10

Properties of Huffman coding

Properties of Huffman coding:

- Produces a code with the **smallest average length** (better than Shannon-Fano)
- Assigning 0 and 1 can be done in any order => different codes, same lengths
- When inserting a sum into an existing list, may be equal to another value => options
  - we can insert above, below or in-between equal values
  - leads to codes with different *individual* lengths, but same *average* length
- Some better algorithms exist which do not assign a codeword to every single message (they code a while sequence at once, not every message)

Huffman coding (M symbols)

General Huffman coding procedure for codes with M symbols:

- Have M symbols $\{x_1, x_2, ...x_M\}$
- Add together the last M symbols
- When assigning symbols, assign all M symbols
- **Important**: at the final step must have M remaining values
  - May be necessary to add *virtual* messages with probability 0 at the end of the initial list, to end up with exactly M messages in the last step
- Example : blackboard

Example: compare Huffman and Shannon-Fano

Example: compare binary Huffman and Shannon-Fano for:

$$p(s_i) = \{0.35, 0.17, 0.17, 0.16, 0.15\}$$

Probability of symbols

- For every symbol x_i we can compute the average number of symbols x_i in a code

$$\overline{l_{x_i}} = \sum_i p(s_i) l_{x_i}(s_i)$$

  - $l_{x_i}(s_i)$ = number of symbols x_i in the codeword of s_i
  - e.g.: average number of 0's and 1's in a code

- Divide by average length => probability (frequency) of symbol x_i

$$p(x_i) = \frac{\overline{l_{x_i}}}{\overline{l}}$$

- These are the probabilities of the input symbols for the transmission channel
  - they play an important role in Chapter IV (transmission channels)

Source coding as data compression

- Consider that the messages are already written in a binary code
  - Example: characters in ASCII code
- Source coding = remapping the original codewords to other codewords
  - The new codewords are shorter, on average
- This means data **compression**
  - Just like the example in lab session
- What does data compression remove?
  - Removes **redundancy**: unused bits, patterns, regularities etc.
  - If you can guess somehow the next bit in a sequence, it means the bit is not really necessary, so compression will remove it
  - The compressed sequence looks like random data: impossible to guess, no discernable patterns

Discussion: data compression with coding

- Consider data compression with Shannon or Huffman coding, like we did in lab
  - What property do we *exploit* in order to obtain compression?
  - How does *compressible data* look like?
  - How does *incompressible data* look like?
  - What are the limitation of our data compression method?
  - How could it be improved?

Other codes: arithmetic coding

- Other types of coding do exist (info only)
  - Arithmetic coding
  - Adaptive schemes
  - etc.

Chapter summary

- Average length: $\bar{l} = \sum_i p(s_i) l_i$
- Code types: instantaneous $\subset$ uniquely decodable $\subset$ non-singular
- All instantaneous or uniqualy decodable code must obey Kraft:

$$\sum_i D^{-l_i} \leq 1$$

- Optimal codes: $l_i = -\log(p(s_i))$, $\overline{l_{min}} = H(S)$
- Shannon's first theorem: use n-th order extension of S, S^n:

$$\boxed{H(S) \leq \overline{l_S} < H(S) + \frac{1}{n}}$$

  - average length always larger, but as close as desired to $H(S)$
- Coding techniques:
  - Shannon: ceil the optimal codeword lengths (round to upper)
  - Shannon-Fano: split in two groups approx. equal
  - Huffman: group last two. Is best of all.