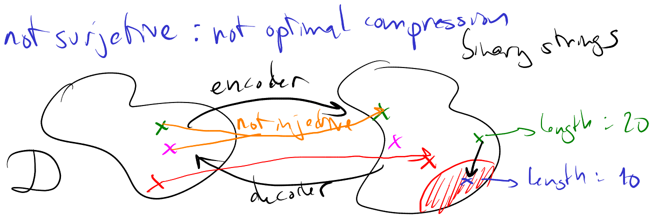
Chapter 2 : Compression / Prediction / Generation equivalence

I Concepts

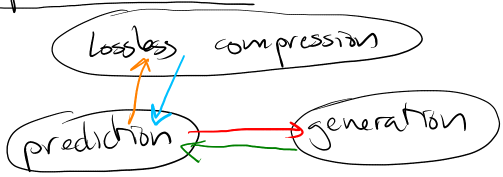
- Lossless compression
  - data = sequence of symbols, from a known predefined alphabet (discrete setting)
    ↳ set of symbols
    ex: {'a', 'b', 'c', ... 'z'}
    ex: {'1', '2', '3' ...}
    ex: {'α', 'β', 'γ', ...}
  - lossless compression:
    ↳ no information lost
  - encoder: data ⟼ binary string
    ⚠ injective!
    otherwise: not decodable
    ↳ "uniquely decodable code"
  - decoder: binary string ⟼ data
    if encoder not surjective: not optimal compression
  - lossless ⟺ decoder ∘ encoder = Id



- Prediction
  - given a sequence of observed variables: $x_1, x_2, \dots x_n$, predict x_{n+1}
  - i.e. give the probability distribution of each possible symbol (discrete setting → finite alphabet)
    ex: {a, b, c}
  - predictor: function: data $(x_1, \dots x_n)$ ⟼ $p(x_{n+1} | x_1, \dots x_n)$ probability of next variable
    ↳ ex: vector: $\begin{pmatrix} p('a' | x_1 \dots x_n) \\ p('b' | x_1 \dots x_n) \\ p('c' | x_1 \dots x_n) \end{pmatrix}$ $\Sigma = 1$
    probability distribution over the next variable x_{n+1}

- Generative model
  - model able to generate new data
  - data D is generated according to its model probability $p(D)$
    as often as
    2 points → distribution over data
    → sample from it

II From one concept to the other



- Prediction → generation [easy]
  - start from x_1 → given
    → or taken from a learned distrib°
  - choose x_{n+1} according to the probability law $p(x_{n+1} | x_1 \dots x_n)$ used to predict next symbol
  - iterate
    → generate data this way
  - $p(D) = \prod_i p(x_i | x_1, \dots x_{i-1})$ $= p(x_1) \times p(x_2|x_1) \times p(x_3|x_1,x_2) \times p(x_4|x_1 \dots x_3) \dots$
    ↳ $x_1 \dots x_n$
    $= p(D)$ Bayes theorem applied n-1 times

. Generation $\longrightarrow$ Prediction [by sampling... might be slow]

- given $\underline{x_1 \cdots x_n}$, how to estimate the probability distribution of x_{n+1}?
- generate new data D many many times, according to the generator law
- check how frequent each $(\underbrace{x_1 \cdots x_n}_{\text{fixed}}, \underbrace{x_{n+1}}_{\text{different possibilities}})$ is generated $\to p(x_1, \cdots x_{n+1})$
- compute $p(x_{n+1} | x_1 \cdots x_n) = p(x_1, \cdots x_{n+1}) / p(x_1, \cdots x_n)$ Bayes

. Compression $\longrightarrow$ Prediction

- idea: strings whose encoding is shorter are more probable
  ↳ coherent with $L = -\log p \to p = 2^{-L}$

↳ complete a string:
$$L(\text{string} + \text{'c'}) \to p(\text{'c'}) \propto 2^{-L(\text{string}+\text{'c'})}$$
($\text{'c'} = x_{n+1}$)

- given $x_1 \cdots x_n$, build all possible completed sequences $x_1 \cdots x_n, x_{n+1}$
  sequence s_k: $x_1 \cdots x_n$, k^{th} symbol
- compress each of them independently, and check their encoding length $L_k = \text{size}(\text{encode}(s_k))$
- define $p(k) = 2^{-L_k}$
- ⚠ p needs to be a probab. distrib: $\sum_k p(k) = 1$
  ↳ normalize $\left(p(k)\right)_{k \in \text{Alphabet}}$ by a global factor if necessary
  ↳ we'll see later that the norm is already 1 if the encoder is optimal
- define $p(x_{n+1} = k | x_1 \cdots x_n) = p(k)$

ex: $A = \{a, b, c \ldots\}$
$x_1 \cdots x_n$ = "This course is abou"
$p(x_{n+1} | x_1 \cdots x_n)$?
↳ This course is abou a → 1001...
b → 001...
c → ⋮
d → ⋮
⋮
This course is about → ...
(encoder)

. Predictor $\longrightarrow$ Compression [the main point of this lesson]

- Lesson from entropy: if we have good prior information on the distribution $\overbrace{x_{n+1} | x_1 \cdots x_n}^{p}$, then we don't need too much information to be communicated to know which symbol is chosen for x_{n+1}
- encode just that information $\longrightarrow$ so that it's decodable later
- entropy = average length of that code $\mathbb{E}_{x_{n+1} \sim p}[-\log p] = H(p)$
  ↳ not possible to be shorter, without losing information
  Can we prove that bound?
  ↳ is it possible to reach that bound? how?

III Encoders

- issue: when to stop reading data

$\underbrace{1011100011}_{\text{one sequence to decode}} | \underbrace{100}_{\text{another one}}$

* self-delimiting code
  - ex: to code an integer n (not bounded) $n \in \mathbb{N}$
    ↳ need $b = \lceil \log n \rceil$
    ↳ need to tell the length of the sequence to decode
    ↳ encode b: $\log b$ bits $= \log \log n$

rewrite in binary
$\hat{\in} \mathbb{N}$
2 → 10
3 → 11
72 → 64+8 → 1001000
$\lceil \log_2 n \rceil$

$\underbrace{\quad}_{c}\ \underbrace{\quad}_{b}\ \underbrace{\quad}_{n}$ recursively until $\log\log\ldots\log n \simeq 1$

c: $\log\log\log n$, b: $\log\log n$, n: $\log n$

total encoding length:

$\log n + \log\log n + \log\log\log n + \ldots + \underbrace{\log\log\ldots\log n}_{=1}$

$$10\ \underbrace{10}_{2}\ \underbrace{11}_{2}\ \underbrace{111}_{3}\ \underbrace{1001000}_{7}$$

$$\underbrace{10}_{3}\ \underbrace{111}_{7}\ \underbrace{1001000}_{72}$$

00
01
1 10
1 11

0
1 $\in \mathbb{N}$

or stop at first iteration and encode differently

$$\underbrace{001}_{\text{same length as}}\ \underbrace{111}_{3}\ \underbrace{1001000}_{7 \text{ bits}}$$

$$111 \to 7, \quad 1001000 \to 72 \in \mathbb{N}$$

$\lceil \log n \rceil + \lceil \log\log n \rceil \times 2$

$\log\log n \ll \log n$

0000 01
padding with 0
$\lceil \log\log n \rceil - 1$ '0' followed by '1'

* codes with "end of file" character
  - postfix: read the data, until you meet a special code meaning "end"
  - to do this: make binary blocks of fixed size and use one of them for "end"
    - say: 4 — '0000'

$$\underbrace{1100}_{1\text{ block}}\ \underbrace{0110}\ \underbrace{1010}\ \underbrace{0010}\ \underbrace{0000}$$

  - obvious cost: $-\log p(\text{"end"}) = 4$ +4 bits
  - hidden cost: can't use symbols forming "end" at other places
    cost: number of blocks $\times -\log(1 - p(\text{end}))$ → factor ~ 0,093 here
    → not negligible if encoding long sequences

    4-bit block: $2^4 = 16$ possible patterns
    → 1 for end
    15 remaining

* Prefix codes
  when the code of a string cannot be the beginning of the code of another string
  → avoids confusion when decoding

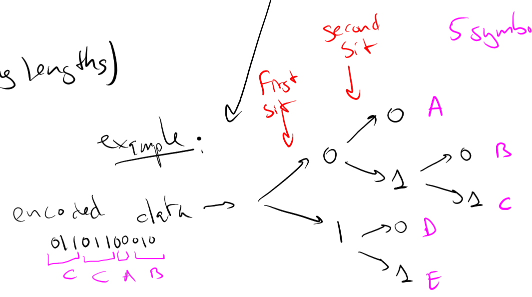
$$\underbrace{10011}_{23}\,|\,10110001$$

David Mackay's book page 104 for examples

→ such codes can be written as binary trees

→ can show later,
that without losing generality,
one can assume that a code is prefix
(otherwise, rebuild another code with same coding lengths)

example:



Alphabet with 5 symbols

symbol	encoding
A	00
B	010
C	011
D	10
E	11

IV Bounds, optimal codes and Kraft inequality

. entropy ⇒ make use of symbol probabilities (to produce shorter average encoding length)
↳ more frequent symbols: should have shorter encoding lengths (code)

. hint: $2^{-\text{Length}(\text{code of } s)}$ should be $\propto p(s)$

↳ expect $\text{Length}(\text{code of } s) = -\log p(s)$ ← cf. entropy

Kraft inequality

. Given an encoder, let's denote by (l_i) the list of the lengths of all possible codewords (with multiplicity) ← codes of each symbol in the alphabet

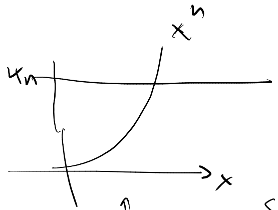
A: 001 → 3
B: 101 → 3

then:
the code is uniquely decodable $\Rightarrow \sum_i 2^{-l_i} \leq 1$

and reciprocally:
if (l_i) satisfies $\sum_i 2^{-l_i} \leq 1$, then there exists an encoder (prefix-code) whose codewords have this length distribution

last ex: $L = \{2, 3, 3, 2, 2\}$

Moreover: if equality ($\sum_i 2^{-l_i} = 1$), the code is said to be "complete"

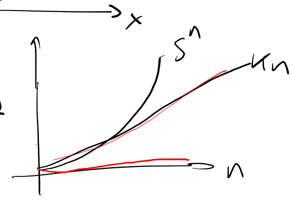


Math proof: . note $S = \sum_i 2^{-l_i}$

. choose any $n \in \mathbb{N}$

sketch . compute $S^n \to \left(\sum_i 2^{-l_i}\right)^n \to$ count terms → show it grows at most linearly with n

$S^n \leq Kn$

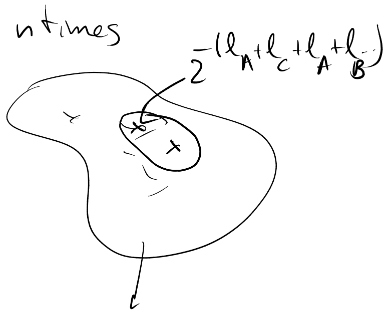


if $S > 1$
⇓
$S \leq 1$
$S \leq 1$

. $\begin{cases} l_{min} = \min_i l_i \\ l_{max} = \max_i l_i \end{cases}$ $\quad \forall i, \ l_{min} \leq l_i \leq l_{max}$

$$S^n = \left(\sum_i 2^{-l_i}\right)^n = \left(\sum_i 2^{-l_i}\right) \times \left(\sum_j 2^{-l_j}\right) \times \left(\sum_k 2^{-l_k}\right) \ldots \quad n \text{ times}$$

$$= \sum_{ijk\ldots} \underbrace{2^{-l_i} \times 2^{-l_j} \times 2^{-l_k} \times \ldots}_{2^{-(l_i + l_j + l_k + \ldots)}}$$



$$= \sum_{l = n l_{min}}^{n l_{max}} A_l \, 2^{-l}$$

numbers of terms such that $l_i + l_j + l_k + \ldots = l$

001 10 1 110 → length l
i j k

$A_l \leq \underbrace{2^l}$ ← maximum number of binary chains of length l

how many terms

2^{-0} 2^{-1} 2^{-2} 2^{-3} 2^{-4} ?
↓ ↓ ↓ ↓ ↓

ex: $\overbrace{01100\cdots 2}^{l}$ ↳ $2 \times 2 \cdots = 2^l$

$$S^n \leq \sum_{l = n l_{min}}^{n l_{max}} \underbrace{2^l \times 2^{-l}}_{1}$$

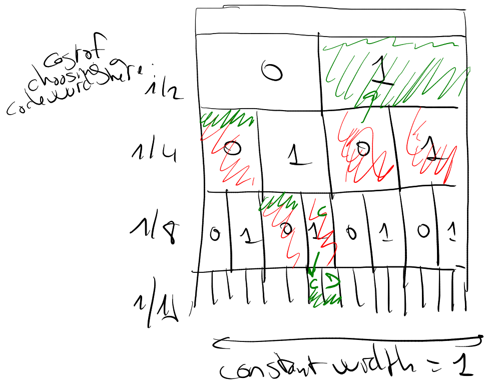
$$\leq n(l_{max} - l_{min}) \leq n \, l_{max}$$

Proof by drawing:

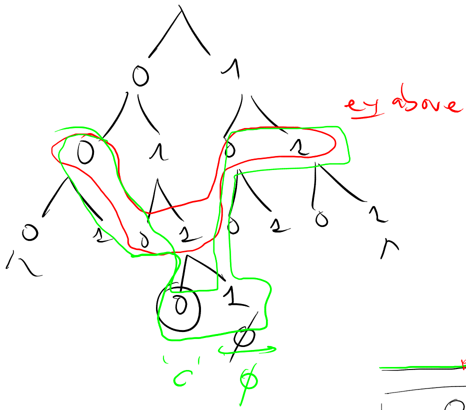
- list all the codewords

	A	B	C	D	E
	00	010	011	10	11
	00	01	10	11	?
ex:	0	1	?	?	?

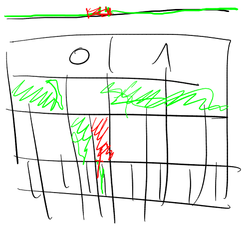
→ not possible to have too many short codewords



$\Sigma 2^{-l_i} \leq 1$



For every codeword taken, consider 2^{-l_i}



Lower bound on compression length

$$\underbrace{\text{Length (encoding } X)}_{\text{on average}} \geq H(X)$$

with equality only if the code length is coherent with the probability

$l_i \rightarrow (2^{-l_i})$ $\qquad$ $p(i)$ in the data $\rightarrow -\log p(i)$

Proof: Gibbs inequality + Kraft

$$\mathbb{E}_{X \sim p}\left[\,l(x)\,\right] = \sum_i p(i)\, l_i$$

p ↑ distribution over symbols of the alphabet

$q_i = \dfrac{2^{-l_i}}{\sum_j 2^{-l_j}}$ $\qquad$ $i \to l_i \to q_i \propto 2^{-l_i}$

$\Sigma q_i = 1$

q : proba distrib

$$= \sum_i p(i)(-\log S - \log q_i)$$

$q_i = \dfrac{2^{-l_i}}{S}$ $\qquad$ $S q_i = 2^{-l_i}$

$l_i = -\log S - \log q_i$

$$= \underbrace{-\log S}_{\substack{S \leq 1 \\ \log S \leq 0 \\ -\log S \geq 0}} \; \underbrace{- \sum p(i) \log q_i}_{\text{cross-entropy } p \| q}$$

↳ Gibbs: crossentropy $\geq$ entropy

$-\sum p(i) \log q(i) \geq H(p)$

$$\geq H(p)$$

Upper bound

It's possible to build a code such that:

Source coding theorem: (symbol codes)

There exists a variable-length encoding C of an ensemble X such that the average length of an encoded symbol, $L(C,X)$, satisfies:

$$L(C,X) \in [H(X),\ H(X)+1[$$

$\in \mathbb{R}$ $\qquad$ $[\qquad\qquad)$

Proof: Kraft with $l_i = \lceil -\log p_i \rceil$

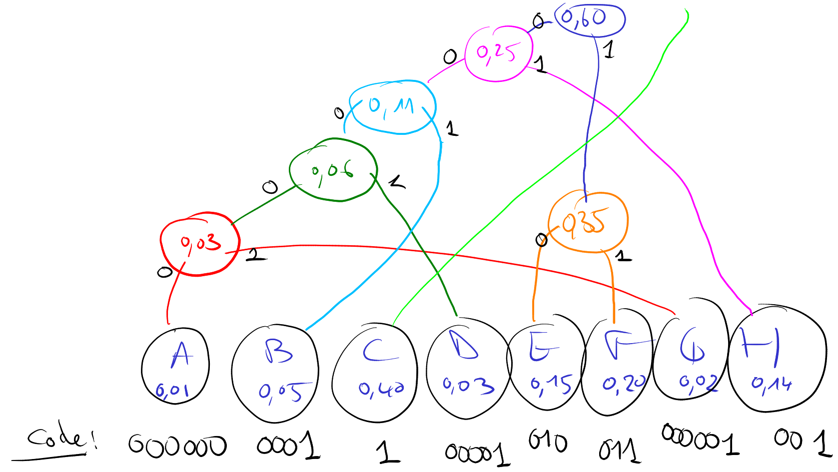
ex: 'A' : 1010

Constructive proof:
- → Huffman coding (best possible with integer nb of bits/symbol)
- ↳ arithmetic " (non-integer nb of bits)
  - ↳ reach $H(X)$ rate

V Huffman & arithmetic coding

Huffman coding

1) write down one node per symbol, with its probability
2) iteratively, join the 2 nodes with lowest probability, and form a new node, with proba = sum of the 2
3) this forms a binary tree
4) code = position in the tree



prefix code encoding

ex: BACH → 0001000000 1 001

Optimality proof:

→ frequencies match encoding lengths

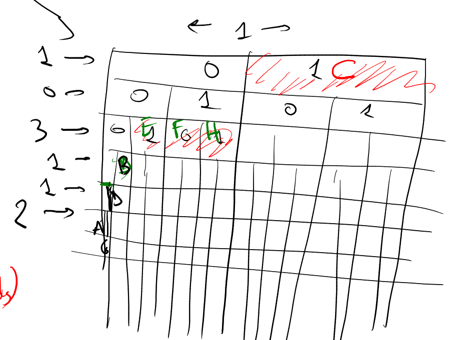
1) for a given set of code words lengths,
2) optimal set of codeword lengths

Suppose another prefix code is better with a ≠ set of codelengths

S1 ≠ S0 : codelengths for least probable symbols are ≠

never true

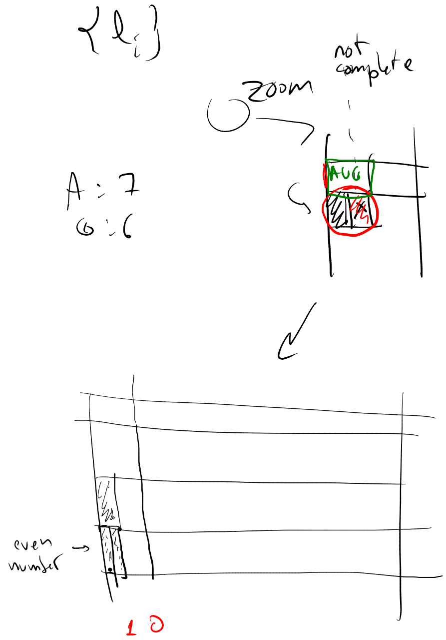
⇒ length (2 least frequent symbols) = the same



$\{l_i\}$

A : 7

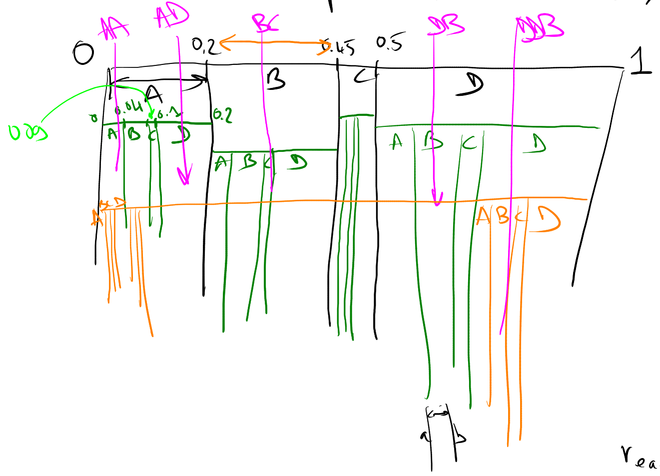
G : 6

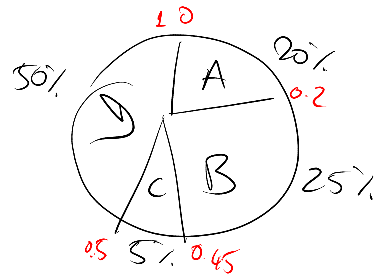


Arithmetic coding

$\mathcal{A} = \{A, B, C, \dots\}$ alphabet

↳ with predefined order : A, B, C, D . .





Length associated to symbol 'B' is its frequency

draw a random number uniformly in [0,1]

↳ will have 25% of chances to be in the 'B' bin

↳ can pick randomly symbols with the frequency of the distribution

read: the probability that a string starts with "DD" is the width of the bin coding for "DD"

1) Encoding

given string: DBAC → yields an interval [a,b] → chose the binary representation of a real number that falls into that bin, i.e. $x \in [a,b]$

2) Generation

pick randomly (uniformly) a real number $\in [0,1]$ and decode it (its binary representation)

0.0110110001...

$1/4 + 1/8 + 1/32 + 1/64 \in [a,b]$ stops when no ambiguity anymore (between bins)

→ when to stop decoding (a real number in binary format)
↳ use end-of-file character ⇝ cost?

$H(p)$

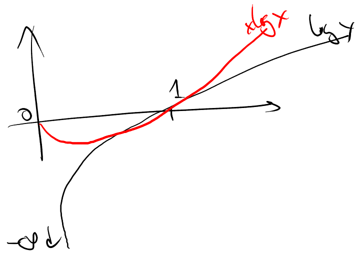
p' : $\begin{cases} \varepsilon \text{ for EoF} \\ 1-\varepsilon \text{ for standard symbols} \end{cases}$ ⟶ $p(A)$ for symbol A, $p(B)$ .... ⟶ $\begin{cases} p'(A) = (1-\varepsilon)\,p(A) \\ p'(\text{EoF}) = \varepsilon \end{cases}$

$$H(p') = -\varepsilon \log \varepsilon - (1-\varepsilon)\log(1-\varepsilon) + (1-\varepsilon) H(p)$$

continuous in ε

↳ choose ε : take the best for your data

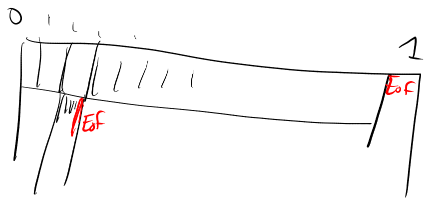
$\simeq H(p)$



Ex for EoF (End of File)

Alphabet = { latin characters + ' ' (space) + punctuation marks }

"Hello world!" ⟶ "Hello world! EoF"



Huffman coding: is an approximation of arithmetic coding with the constraint that bin sizes should be powers of 2 (i.e. 2^{-n})

VI Additional remarks

Law over integers

· code 1 : encoding length for number n : $\log n + 2\log\log n$

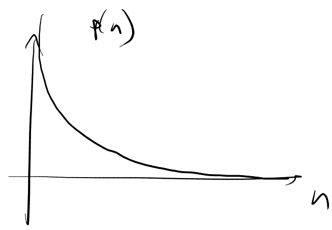
$$p(n) \propto 2^{-\log n - 2\log\log n} = \frac{1}{n(\log n)^2}$$

↳ the sum $\sum_n p(n)$ does converge ⟶ can be normalized as a probability distrib°

↳ $\neq 1$

· code 2 : encoding length : $\log n + \log\log n + \log\log\log n + \dots$

↳ $p(n) \propto \dfrac{1}{n \times \log n \times \log\log n \times \dots}$



↳ $\sum_n p(n)$ converge? ⟶ Kraft says yes

limit case

· code? s.t. $p(n) \propto \frac{1}{n}$ or $\frac{1}{n \log n}$? ⟶ not possible because $\sum_n p(n) \to +\infty$

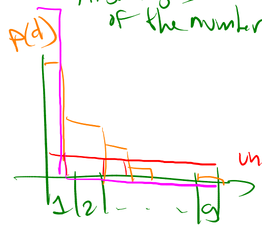
· Example : tax cheaters ⇝ fill the forms with fake numbers

↳ don't know information theory ... they pick the first digit randomly uniformly in {1, 2, ..., 9}

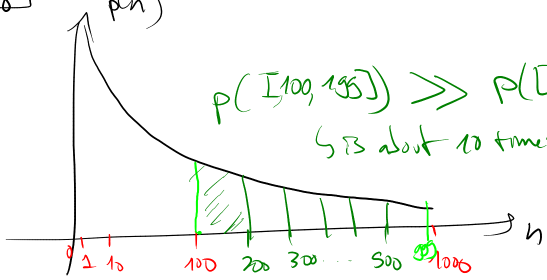
| 3 | 5 | 5 | 8 |

p(3) = 1/9

histogram of first digits of the numbers in the form



$\sim \frac{1}{n \log n^2}$



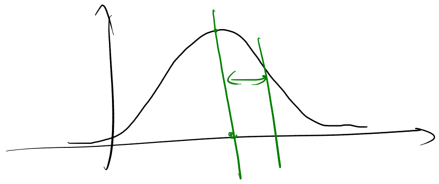
p([100, 199]) >> p([900, 999])

↳ is about 10 times more probable than

set of 3 digit numbers

· About generation : notion of typicality

data (made with n symbols) is typical of a model if its $\underbrace{\text{average coding length / symbol}}_{\text{encoding rate for that data}}$ is close to the entropy of the model



$$\mathbb{E}_{x \sim p}\left[(x-\mu)^2\right] = \sigma^2 \text{ standard deviation}$$

· Measure model efficiency or error : in bits

↳ Fit a model to a ML task → encoder

p → $-\log p$ unit : bits

VII Ex : text compression / prediction / generation with Markov Chains

→ cf practical session

↳ IID model

↳ Markov chains with various orders

– compression : Hutter prize → prize.hutter1.net

thousands € if able to compress file by 1% more

↳ Wikipedia

Conclusion

- good model for data : good for every task : prediction, generation, compression
- measure model suitability in bits → see generative models as compressors

↳ baseline : gzip (Lempel-Ziv)

- $-\log p$ is reachable with non-integer number of bits, thanks to arithmetic coding

not $\lceil -\log p \rceil$