AICC II - Notes and Summary

Faustine Flicoteaux

Fall Semester 2024

Contents

1 Entropy and Data Compression **5**
1.1 Entropy 5
1.1.1 Definition 5
1.1.2 Entropy in equiprobable spaces 5
1.1.3 Entropy in other spaces 6
1.1.4 Entropy Distribution 6
1.2 Cross-Entropy 6
1.3 Source Coding 6
1.3.1 A few definitions 7
1.3.2 The Kraft-McMillan inequality 7

2 Modular Arithmetic and Finite Fields **9**
2.1 Extended Euclid Algorithm 9
2.2 Euler's Totient Function 10

Introduction

These are my notes for the Advanced Information, Communication and Computation II (COM-102) course given during the spring semester of 2025 at EPFL. Please note that the content is not mine but belongs to Professor Michael Gastpar, who taught the class. I have however changed some formulations, added definitions from other sources and personal notes, when I thought it was useful.

This summary is not exempt of errors. If you find one, you can contact me at my EPFL e-mail address: `faustine.flicoteaux@epfl.ch` or through the GitHub page `https://github.com/FocusedFaust/LectureNotes`.

Note that the GitHub repository is also where I have the latest pdfs and TEXdocuments, for this course and others.

Chapter 1

Entropy and Data Compression

1.1 Entropy

Entropy is not an easy concept to grasp at first, so I decided to write down what I understand to help myself.

1.1.1 Definition

Entropy is part of Information Theory, which studies the quantification, storage and transmission of information. Formally, entropy is "the measure of the average uncertainty of a probability distribution".

When we share a bit of information, this bit must have different possibilities, otherwise it doesn't mean anything.

For example, if I say "I feel good", information is communicated because I could feel sad, or nostalgic, or grateful (and much more). Same thing if the weather forecast announces rain, because it could be sunny or cloudy too.

Example: fair coin Say that I flip a fair coin. I either get heads or tails, with equal probabilities. This means that I can encode the result of the flip in a single bit (say 1 for heads and 0 for tails). Because all of the information fits in one bit, the entropy of the experiment is equal to 1 bit.

Example: tournament Say now that we want to encode the result of a sport tournament. Furthermore, let us say that 8 teams have entered that tournament and are equally as good. Because we have 8 equally likely outcomes, we can assign a number to each team. For that, we will need 3 bits, as $2^3 = 8$. Therefore, the entropy of the system is equal to 3.

1.1.2 Entropy in equiprobable spaces

When all outcomes are equally likely, we can compute the number of bits needed as $H(X) = log_2(\text{number of outcomes})$. We can then store 2^n different states in a bitstring of length n.

However, it is not always the case that all outcomes are equally likely. Take a page of a book for example. Each letter is not as equally likely as the others. Then, how do we share information and how do we do it more efficiently?

1.1.3 Entropy in other spaces

I like to visualise this using the possibility tree, which we construct for example when asking questions to guess the answer. Remember one of the first exercises of Introduction to Programming, where we had to write a program to guess what mushroom we thought of.

For non random-distribution spaces, we use the Shannon formula of entropy.

$$\begin{aligned} H(X) &= \sum_i p_i \cdot \text{"how far down the possibility tree we are"} \\ &= -\sum_i p_i \cdot log_2(n) \\ &= -\sum_i p_i \cdot log_2(p_i) \end{aligned}$$

with n the number of outcomes or the number of bits needed.

Example: the letter machine Let us imagine a machine that outputs a letter out of four, $l \in \{A, B, C, D\}$. If all four outcomes are equally likely (25% chance), we could encode A as 00, B as 01, C as 10 and D as 11.

However, if A has 0.7 probability, B has 0.2 and C and D have 0.05 each, we could encode it differently. A would be 0, B would be 10, C would be 110 and D 111. The analogy here is the question tree when guessing the outcome. First, we ask if the letter is A. Then, if it is B, etc. Each answer corresponds to a digit in the encoding of the answer.

The entropy here would be $H(X) = \sum_l p_l \cdot$ number of bits required $= p_A \cdot 1 + p_B \cdot 2 + p_C \cdot 3 + p_D \cdot 3 = 1.4$. Therefore, we need on average 1.4 bits to encode the outcome of the machine.

Notice how, in the question tree, a "yes" corresponds to 0 and a "no" corresponds to 1. This is because left-hand zeroes can be dropped (001 = 1) and would change the meaning of the number.

1.1.4 Entropy Distribution

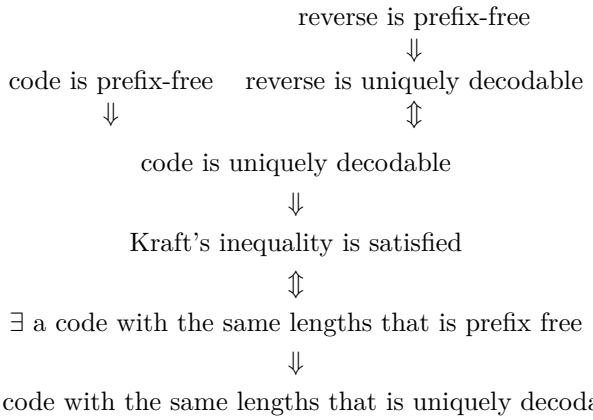
Entropy is maximal when all outcomes are equally likely, because we need the most bits to encode the outcome. When the entropy drops, it is the same as having to ask less questions to guess the result.

1.2 Cross-Entropy

If two random variables are not independent, then

1.3 Source Coding

When analysing a code, there exist some implications, that I have grouped here:



1.3.1 A few definitions

A code being **instantaneous** is the same as it being **prefix-free**, which means that no codeword is the prefix of another.

A code is **uniquely decodable** if there exist no sequence of codewords that can be decoded in more than one way.

1.3.2 The Kraft-McMillan inequality

This inequality gives a sufficient condition for a code to be uniquely decodable given a set of codeword lengths. Let the code $C = \{c_1, c_2, c_3, ..., c_n\}$ over an encoding alphabet of length r (and not n, which is the encoded alphabet's length) have the codeword lengths $l_1, l_2, l_3, ..., l_n$. The inequality is as follows:

$$\sum_{i=1}^{n} r^{-l_i} \leq 1$$

If this inequality is respected, then there exist a code with such codeword lengths that is uniquely decodable. However, it doesn't mean that this code is C, it could be another encoding.

Chapter 2

Modular Arithmetic and Finite Fields

"If you had nightmares about gcd computation in high school, your nightmares were true" - Michael Gastpar

(It's really not that scary but it's mostly annoying to do by hand)

2.1 Extended Euclid Algorithm

Euclid's algorithm allows us to compute the integers x and y of Bézout's identity such that

$$ax + by = \gcd(a, b)$$

To do so, we draw a table with 6 columns and as many rows as necessary. The leftmost column is used for the successive gcd computations. We know that $\gcd(x, a \cdot x + b) = \gcd(a \cdot x + b, b)$. Therefore, for each line, we simplify the previous calculus. Then, in the second column, we keep the result of the euclidean division, which we name q. Once we have rippled trough until we have a computation of the form $\gcd(c, 0) = c$, we go back up. The third ($\tilde{u}$) and fourth ($\tilde{v}$) columns hold the values 1 and 0, respectively.

Now, until our table is complete, we ripple trough upwards. At each step, we compute u and v using the values of $\tilde{u}$ and $\tilde{v}$ one row lower. Then, until we

Example We would like to find u, v such that $549u + 174v = \gcd(549, 174)$

		q	$\tilde{u}$	$\tilde{v}$	$u = \tilde{v}$	$v = (\tilde{u} - q\tilde{v})$
g	$= \gcd(549, 174)$	$549 = 3 \cdot 174 + 27 \rightarrow 3$			13	-41
	$= \gcd(174, 27)$	$174 = 6 \cdot 27 + 12 \rightarrow 6$	-2	13	-2	13
	$= \gcd(27, 12)$	$27 = 1 \cdot 12 + 3 \rightarrow 2$	1	-2	1	-2
	$= \gcd(12, 3)$	$12 = 4 \cdot 3 + 0 \rightarrow 4$	0	1	0	1
	$= \gcd(3, 0) = 3$	0	1	0		

In magenta is the first step of the upwards calculations, using the previously stored values. In teal we see the second step, which is storing the computed values. In the end, we computed $3 = 549 \cdot 13 + 174 \cdot (-41)$

Modular Inverse

This algorithm is particularly useful when trying to find the multiplicative inverse of a number in a finite field.

Let us say we are trying to find $b = a^{-1} \mod m$. We can rewrite it as $a \cdot b + m \cdot c = 1$. The c does not matter because any scalar product of m gets ditched trough the modularisation. Therefore, we can do the euclidean algorithm starting at $\gcd(m, a)$

2.2 Euler's Totient Function

Euler's Totient function $\phi(n)$ returns the number of integers $\{1, 2, ..., n-1\}$ that are relatively prime to n. Its formula is:

$$\phi(n) = n \cdot \prod_{p|n} \left(1 - \frac{1}{p}\right)$$