Entropy and Its Relations to Measure Theory and Information Theory

Dallas Anderson

March 5 2024

Abstract

Entropy is an important concept in physics: The information of an event that occurs in a system. In this paper, we'll explore how it can be defined in measure-theoretic scenarios. Specifically, we'll consider measure-preserving dynamical systems $(X, \mathcal{A}, \mu, T)$. Then, we'll take a look at the information theory version, exploring different typographical languages and their structure. We'll see how they can be represented with graphs, explore a certain notion of *ergodicity* among them, and finally, define entropy on them.

1 Introduction

Definition 1.1 Take a countable (finite or countably infinite) partition $\alpha = \{A_1, A_2, \ldots\}$ of a space $(X, \mathcal{A}, \mu, T)$ as above. We define the *information function* of α to be

$$I(\alpha)(x) = -\sum_{A \in \alpha} \mathbb{1}_A(x) \cdot \log_2 \mu(A),$$

where $\mathbb{1}_A$ is the indicator function of A. We define the *entropy* of α to be

$$H(\alpha) = \int I(\alpha) d\mu = -\sum_{A \in \alpha} \mu(A) \log_2 \mu(\alpha).$$

What's the motivation behind these definitions?

(The following is from [2]) well, let's think about how we want entropy to be defined. Let's say we have a point $x \in X$ that we want to 'locate'. The fact that $x \in A_j$ gives us information about the location of x. We want to have the knowledge $x \in A_j$ always give the same amount of information, i.e. if x_1 and x_2 are two points we want to locate, then we want $x_1 \in A_j$ to have the same information value as $x_2 \in A_j$. Also, we want to consider $X \in A_j$ to be more informative if A_j is a smaller set (in the sense that $\mu(A_j)$ is small), and less informative if it's a bigger set. Basically, $I(\alpha)(x)$ is equal to the 'value' of the information $t \in A_j$ for the $A_j \in \alpha$ that contains x, whatever this 'value' is. We'd like the value of the information $t \in A_j$ to depend on $\mu(A_j)$, so we can take it to be $\phi(\mu(A_j))$ for some function ϕ we choose. So $I(\alpha)(x) = \phi(\mu(A_j))$ for the A_j containing x. Another way of writing this is

$$I(\alpha)(x) = \sum_{A \in \alpha} \mathbb{1}_A(x) \cdot \phi(\mu(A)).$$

Look familiar? This is the same as our definition except with $\log_2$ instead of ϕ, and a negative sign in front, or basically just $-\log_2$ replacing ϕ. Why is $-\log_2$ a good choice for our function ϕ?

Definition 1.2 Let $\alpha = \{A_1, A_2, \ldots\}$ and $\beta = \{B_1, B_2, \ldots\}$ be two partitions. Define the *join* of α and β to be

$$\alpha \vee \beta = \{A_i \cap B_j : A_i \in \alpha, B_j \in \beta\}$$

(it's not too hard to see that this is a partition as well). We say that two partitions α and β are *independent* if

$$\mu(A_i \cap B_j) = \mu(A_i)\mu(B_j)$$

for all i, j.

It seems nautral to require that the information we obtain by using two independent partitions together should be equal to the sum of the information received using each partition seperately, i.e.

$$I(\alpha \vee \beta) = I(\alpha) + I(\beta).$$

But then

$$\sum_{A\in\alpha, B\in\beta} \mathbb{1}_{A\cap B}(x) \cdot \phi(\mu(A \cap B)) = \sum_{A\in\alpha, B\in\beta} \mathbb{1}_{A\cap B}(x) \cdot \phi(\mu(A)\mu(B))$$

$$= I(\alpha \vee \beta) = I(\alpha) + I(\beta) = \sum_{A\in\alpha} \mathbb{1}_A(x) \cdot \phi(\mu(A)) + \sum_{B\in\beta} \mathbb{1}_B(x) \cdot \phi(\mu(B)) = \sum_{A\in\alpha, B\in\beta} \mathbb{1}_{A\cap B} \cdot (\phi(\mu(A)) + \phi(\mu(B))).$$

The last equality is true because if $x \in A \cap B$, then the left side is just $\phi(\mu(A)) + \phi(\mu(B))$ and the right side is also $\phi(\mu(A)) + \phi(\mu(B))$. Hence we require that

$$\phi(\mu(A \cap B)) = \phi(\mu(A)\mu(B)) = \phi(\mu(A)) + \phi(\mu(B)).$$

If we assume that ϕ is continuous, then one can check that we are forced to take $\phi(t) = \log_a(t) = \frac{\log_2(t)}{\log_2(a)}$ for some a. It is nautral to choose $-\log_2(t) = \log_{\frac{1}{2}}(t)$ (the negative sign is there since $0 \leq t \leq 1$ so that the logarithim is positive).

2 Information-Theoretic Entropy

We can also think of entropy in terms of information theory. Let's say you want to generate a string of letters, say your alphabet is A, B, C, D, and E, and you want to assign a probability for each letter's occurance at any given time. You could assign them equal probabilities, so you have no information about what will happen, or, say, give A a really low probability of occuring. These different options give different amounts of information about what will happen, and there's different amouts of "choice", i.e. different degrees of freedom in your choice for each letter.

To quantify this, we want $H(p_{\mathrm{A}}, p_{\mathrm{B}}, p_{\mathrm{C}}, p_{\mathrm{D}}, p_{\mathrm{E}})$, where the p's are the probabilities of each letter's occurance, that signifies how uncertain we are of the selection of the letter at a given point. We define H as a function on sets of random events with probabilities $p_1, \ldots, p_n$.

(The following comes from [1].) There are a few properties we want H to satisfy:

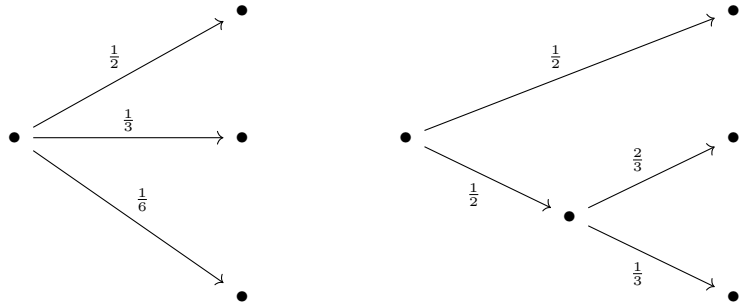
1) H should be continuous in the p_i (where p_i are the probabilities of the events in the system, as before).

2) If all the p_i are equal, i.e. $p_i = \frac{1}{n}$, then H should be a monotonic increasing function of n: With equally likely events there's more uncertainty when there are more possible events.

3) If a choice can be broken down into two succesive choices, the original H should be the weighted sum of the individual values of H.

The meaning of 3) is illustrated in the diagram below. On the left there's three possibilities, say a, b, and c, with probabilities $p_1 = \frac{1}{2}$, $p_2 = \frac{1}{3}$, and $p_3 = \frac{1}{6}$ respectively. On the right we first choose between two possibilities each with probability $\frac{1}{2}$, and if the second occurs we make another choice with probabilities $\frac{2}{3}$ and $\frac{1}{3}$. The first two events represent chossing between having a_1 or having either of a_2 and a_3 happen, and if a_2 or a_3 happens, choose which one of those. The final results have the same probabilities as before, and our property

states that $H\left(\frac{1}{2}, \frac{1}{3}, \frac{1}{6}\right) = H\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{1}{2}H\left(\frac{2}{3}, \frac{1}{3}\right)$ because of this.



The functions

$$H = -K \sum_{i=1}^{n} p_i \log_2 p_i$$

(where n is the amount of events) for some positive constant K turn out to be the only choices satisfying these. We choose $K = 1$ since K merely anounts to a coice of base for the logarithm.

That was the simpler case where the only structure in the sequence generated is the probabilities of each letter's occurance. What if, say we add that E can never come after A? What if the probability of a letter being typed is affected by the previous letter, or even the previous two letters? We're going to explore another aspect of these systems: The *state* the system is in at a given slot.

3 States of systems

As an example, let's try to create a program that is able to generate English sentences. We will say there's a 27-character alphabet, the 26 letters and space as the 27th character (this ignores capitals and punctuation, we're just focused on words and spaces here). The program can't just type in any string of symbols it wants, like "Vdvj er eg sfjsnfuijiuew drgersiksr ersuer." That's not a proper English sentence. To be able to generate only actual sentences, we need to know what exactly distinguishes it from proper sentences.

Well, one thing is that there's three spaces between "eg" and "sfjsnfuijiuew"; there can't be spaces after other spaces in English sentences. We could create *states* of our system representing this rule. Like, say, being in state A means you can put anything you want, and being in state B means you can put any letter, just not a space. Then the system is in state A if the previous character was anything except a space, and it's in state B if the previous character was a space (or at the beggining of the program; you can't start with a space). This prevents spaces coming after spaces (or, like I said, starting with a space), because after a space you're in state B and you can't put another space.

Also, another rule is that there's always a "u" after any "q" (which our "sentence" doesn't break but this still is a rule). We can create rules for states representing this, or better, a combined system for both of our rules. State A and state B are as before, and state C means you can only put a "u". State A is active if the previous character was a letter other than "q" this time, state B is active if the previous character was a space or it's the start of the program (like before), and state C is active if the previous letter was "q".

There's also a probabilistic element to this. Some letters are more likely to appear in words than others. For example, "e" is the most common letter in the English language. That sentence had eight e's (not counting the one in the quote)! Letters like "q", "j", and "z" are not so common– this sentence has none of those. There's also letter combinations (and character combinations possibly including spaces) that are more likely to appear than others.

For this, instead of states determining just which characters are *possible*, we have states determining the *probabilities* of each character occuring (which may include the probability 0). We not only consider the probability of a given character c occuring, call this $p(c)$ (I don't mean of the letter c, I mean Latex c standing for a variable

character), but $p_S(c)$, the probability of typing the character c given that we're in state S.

Now, let's say we have a system like our thing with states A, B, and C, in that states of the system are determined by whether there was a previous character and what it was if so. We can let $p_{c_1}(c_2)$ for characters $c_{1,2}$ be just $p_S(c_2)$, where S is the only state that's activated if c_1 was the previous letter (it can be activated by more than one letter, just c_1 at least). There's also a thing called $p(c_1, c_2)$, the general probability of the *digram* c_1c_2 occuring (its relative frequency, basically).

$p(c)$, $p_{c_1}(c_2)$, and $p(c_1, c_2)$ are related by the following formulas:

$$p(c_1) = \sum_{c_2} p(c_1, c_2) = \sum_{c_2} p(c_2, c_1) = \sum_{c_2} p(c_2)p_{c_2}(c_1)$$

$$p(c_1, c_2) = p(c_1)p_{c_1}(c_2)$$

$$\sum_{c_2} p_{c_1}(c_2) = \sum_{c_1} p(c_1) = \sum_{c_1, c_2} p(c_1, c_2) = 1.$$

But there can be other systems where states have different rules for being activated. For example, take the letters A, B, C, D, and E again for the characters. Suppose State 1 is activated by having the previous *two* letters be DB, State 2 is activated by being at least the third character but the previous two letters not being DB, and State 3 is activated by being the first or second letter. Note that I made the states numbered this time so they don't get confused with the characters, but it doesn't matter what they're called, just how they behave. I also didn't specify what happens when a state is activated, but that doesn't matter either for this particular example.

What matters here is that the states depend on the previous *two* characters (and if there are two characters before), not just the previous *one*. In cases like these, we can consider $p_{c_1,c_2,\ldots,c_{n-1}}(c_n)$, $p_S(c_n)$ where S is the state activated by $c_1c_2 \cdots c_{n-1}$ being the previous characters. We can also consider $p(c_1, \ldots, c_n)$, the relative frequency of the sequence $c_1 \cdots c_n$ (called an n-gram).

4 Ergodicity

Now, these systems can be represented using graphs. The junction points in the graphs will represent the different states of the system, the arrows between points will represent characters changing the system from one state to another, and the probabilities of the character occuring (given the state) will be given beside the corresponding arrow. Note that sometimes, not all characters will have an arrow between any two points. This represents there being a 0% chance of that character being typed given the state. There's also the detail about what state are we in at the start of the program, but we'll ignore that for now.

There's a notion of *ergodicity* among these systems which we'll define. First,

Definition 4.1 A closed series of lines in the graph with all arrows on the lines pointing in the same orientation will be called a *circut*. The *length* of a circut is the number of lines in it.

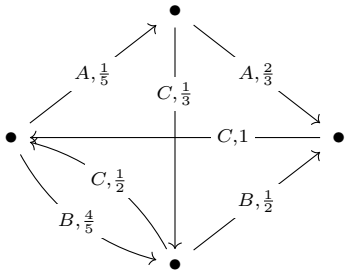
Now for *ergodicity*:

Definition 4.2 A system is said to be *ergodic* if it satisfies two properties:

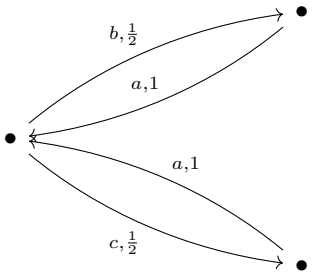
1) The graph doesn't consist of two isolated parts A and B such that it's impossible to go between junction points in one part to points in the other along lines of the graph in the direction of arrows.

2) The greatest common divisor, or gcd of the lengths of all circuts in the graph is 1.

Below is an example of an ergodic system.



(The following is from [1].) Now, if 1) is satisfied but 2) isn't, then the sequences will have a certain periodic structure. If d is the gcd of the lengths of all the circuts, then this means that $d > 1$, which makes the sequences have a certain type of periodic structure. The sequences you can make fall into d different categories which are statistically the same apart from a shift of the origin, i.e. which letter in the sequence is called letter 1. By a shift of from 0 up to $d - 1$ any sequence can be made statistically equivalent to any other. A simple example with $d = 2$ is the following: We have three characters a, b, and c. If the previous letter was a, it types either b or c with probabilities $\frac{1}{3}$ and $\frac{2}{3}$ respectively. If the previous letter was b or c, it always types a (the diagram below shows this).



A typical sequence would be something like

abacacabacacacabacabacac....

We're not going to be focusing on this type of situation here.

If the first condition, 1) isn't satisfied, the graph can be seperated into a set of subgraphs each of which satisfies the first condition. We will assume the second condition is satisfied for each subgraph, those are the cases we're going to work with for this last part.

This notion of seperating a graph can be thought of as a *sum* of subgraphs. Basically, if we call the graph L and call its subgraphs $L_1, L_2, L_3, \cdots$, we represent

$$L = L_1 + L_2 + L_3 + \cdots$$

...sort of. We also have to take into account the probability of the system's state at the start being in a given L_n; you always stay in the same subgraph. Call this probability p_n. You could write

$$L = p_1 L_1 + p_2 L_2 + \cdots$$

since you only get a 'portion' p_n of L_n– the probability that you end up in L_n in the first place. And of course, we require that the sum of these p_n be 1. Basically, to generate a sequence in L, you start by choosing an L_n with probability p_n, then generating a sequence from whichever L_n was chosen.

We now define entropy in the more general case. For any possible state S we have our set of probabilities $p_S(c)$ from before for each character c. So there's an H_S for each S, recalling that

$$H_S = \sum_c p_S(c) \log_2 p_S(c).$$

We will define the entropy of the system as the average of these H_S weighted in accordance with the probability of occurence of the states in question, or:

Definition 4.3 We define the *entropy* of a system to be

$$H = \sum_S P_S H_S = -\sum_{S,c} P_S p_S(c) \log_2 p_S(c),$$

where P_S is the general probability of being in state S.

For more details on measure-theoretic entropy, see [2], or for more of the information-theoretic stance, see [1].

Acknowledgements. I would like to thank my teacher Simon Rubenstein-Salzedo for making the writing of this paper possible, and I thank my T.A. Carson Mitchell for providing guidance along the way.

References

[1] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[2] Charles Waldken. Entropy. *Ergodic Theory lecture notes*, 2013.