Induction, information, and encoding: the view from natural language

John Goldsmith

November 3, 2008

Contents

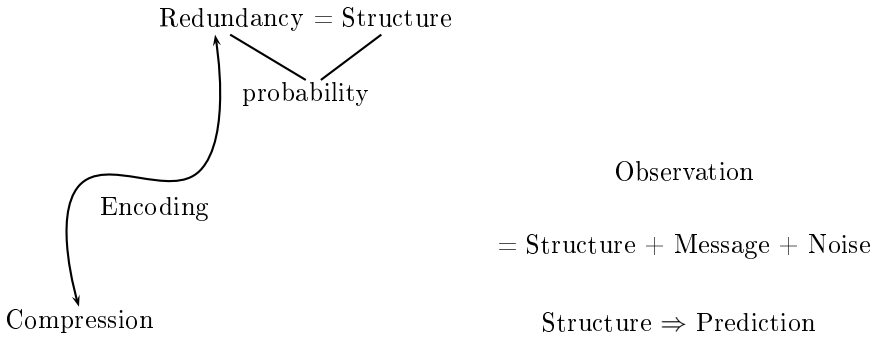
1 Introduction

In these presentations I would like to discuss the following ideas:

- Encoding for communication
- Encoding for compression

- The relationship between structure and redundancy
- Minimization of description length of scientific data ...
- Leading to a model that makes predictions

We begin with what seems like a technical engineering problem, and very quickly arrive at what seems to be a deep philosophical problem. We will just look briefly at that deep philosophical problem, and then consider the analysis of natural language—a very concrete problem—as an example of the application of these ideas. I'll also try to give a sense of the history and development of these ideas.



2 Shannon: encoding for communication

2.1 First notions of optimal encoding

> If we know (or think we know) the structure of the device of the device that produced the message M, we can compress M as Ann *sends* it to Betty.

Claude Shannon developed a set of ideas known as *information theory* which had their first concrete application in the transmission of messages by digital means. The basic idea concerned a situation in which Ann wants to sends Betty a message from a language: let's say the language is English, and as a first approximation, we will assume that the message is a concentation of words from a prespecified lexicon (word-list) called $\mathcal{L}$. Technology requires that the message be just a series of 0's and 1's—that is, a string from $\{0,1\}^+$. Ann therefore needs an **encoding**, i.e., a map from $\mathcal{L}$ to $\{0,1\}^+$, and the map must be **injective** (no two words are encoding by the same binary string) but not necessarily surjective. In fact, it will *not* be surjective, because there is a natural condition that both Ann and Betty agree is necessary: the encoding must be **prefix-free**: we place the constraint that the encoding of word w can never be the prefix of another word v (i.e., if w encodes as 0101, then no other word can

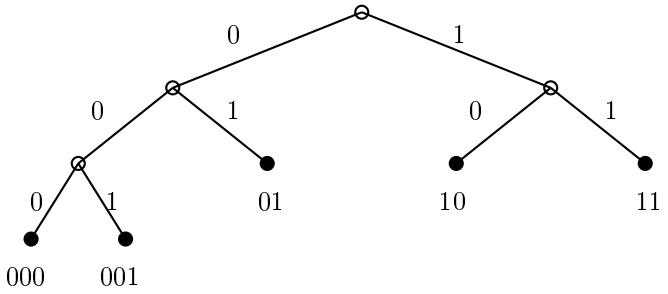


Figure 1: binary tree of encodings with prefix property

encode as 01011). In this way, the encoding is **instantaneous**: after any digit, Betty knows if she has gotten to the end of the encoding of an individual word.[1]

Prefix-free encoding systems can be organized into a binary tree in which all encodings correspond to terminal nodes of the tree, where the string indicates the path taken from the root to the leaf: a 0 means take a left branch, and a 1 means take a right branch. If any non-terminal node were used as an encoding and it dominated a terminal node that were used as an encoding, then clearly the system would not be prefix-free. When we are talking about a particular encoding, we will talk about the nodes of the tree that represents it, identifying strings and nodes in the natural way.

We are interested in properties of good encoding systems, and one of the important qualities of an encoding system is create relatively short strings (from $\{0,1\}^+$, *given* a particular message from $\mathcal{L}^*$). If we knew nothing about the frequencies of the different words in $\mathcal{L}$, we could easily create an encoding scheme which would limit the worst case length, by encoding each word by a binary string of length $\lceil log_2|\mathcal{L}|\rceil$ (where $|\mathcal{L}|$ is the number of items in $\mathcal{L}$). We can always enumerate the $|\mathcal{L}|$ different members of the lexicon and we will need no more than the base 2 log of their count to do so, rounding up as necessary.

In discussing communication systems of this sort, we typically assume, however, that we know certain statistical properties of the messages that Ann sends, such as the (time-averaged) frequency of his usage each individual word in $\mathcal{L}$, $fr(w_i)$. In that case, we can come up with much better encoding systems.

Consider again the binary tree of the encodings of a prefix-free encoding systems. If all nodes are either terminal or binary branching, we say that the tree is *complete* (every binary string either identifies a node in the tree, or has a prefix which identifies a terminal node in the tree). There is a natural way to associate each node in our tree with a subinterval of [0,1): if the string is s (e.g., 01010), then we associate it with the interval that begins with the binary

[1]For now, that conclusion comes immediately from looking at the sequence of bits Ann has just sent. When we get to arithmetic encoding, we will see that Betty has to do some computation to reach that conclusion, but she does not need to look ahead to future bits of the message.

When	Uniform prob	pr(x[i])	pr(x[i] | x[i-1])
Shannon era	0 order Markov	1st order Markov	2nd order Markov
Later (now)		0 order Markov	1st order Markov
Today	uniform	unigram	bigram

Table 1: Terminology: What order Markov model?

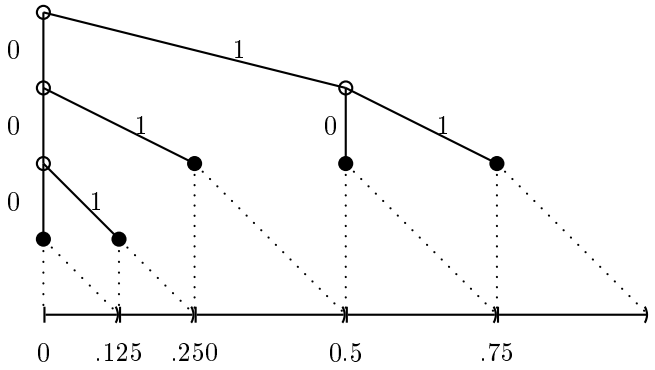


Figure 2: The canonical correspondence

fraction we would write as "0.s" (call that number p), of length $2^{-|s|}$; so the interval is $[p, p + 2^{-|s|})$. This is very natural, given the figure; see the figure on the canonical correspondence.[2]

Now, it is not hard to show that if we have a lexicon $\mathcal{L}$ whose members w_i all have frequencies which are (negative) powers of 2, then the very best encoding system that can be devised is one satisfying the simple property that the length of the encoding of word w_i is $-log_2 fr(w_i)$. Why?

Recall that a distribution π over a set S assigns a non-negative member of [0,1] to each member such that $\sum_x \pi(x) = 1$. Let's use the term "plog distribution $\pi()$" to mean the function $-log_2\pi()$ ("plog" standards for "positive log"). Plog distributions turn out to be central to information theory. You can see that a plog distribution maps from the members of the set to the positive reals. If we have a (complete, prefix-free) encoding $\mathcal{E}$, then the canonical correspondence defines a distribution, and *its* plog distribution is the lengths of each element of the encoding (i.e., the lengths of the strings corresponding to the paths through the tree).

The cross-entropy H(P,Q) between two distributions P and Q over the same set X is defined as $-\sum_{x\in X} P(x)log_2Q(x)$: you can see that this is the sum of the products of the corresponding values of the distribution P and the values of Q's *plog distribution*. And it turns out that this quantity is always strictly larger than H(P,P) for any distribution $Q \neq P$:

[2]See Li and Vitányi, *Introduction to Kolmogorov Complexity*, the standard book in this area.

$$\sum_x p(x) ln \frac{q(x)}{p(x)} \leq \sum_x p(x) \left(\frac{q(x)}{p(x)} - 1 \right) \quad (1)$$

[3]

$$= \sum_x p(x) \frac{q(x)}{p(x)} - \sum_x p(x) = 1 - 1 = 0. \quad (2)$$

So $\sum_x p(x) ln(\frac{q(x)}{p(x)} \leq 0$, and the same holds then if we change the base of the logarithm to 2. Therefore $\sum_x p(x) log_2 q(x) \leq \sum_x p(x) log_2 p(x)$, and multiplying both sides by -1, we see that $H(P,Q) \geq H(P,P)$.

Terminology: the quantity H(P,P) is known as the *self-entropy* of P, or simply as the *entropy* of P.

So what we have seen is this: any encoding system E maps to a distribution, and in particular to a *plog distribution* whose values are the lengths of the encodings. If we encode a message M of length $|M|$using E, then the expected number of bits of the encoded message will be $|M| \sum_{x \in \mathcal{L}} p(x)(-log\, q(x))$. Better put, the average encoding length of a message drawn by distribution P but encoded in a system derived from distribution Q in the way we have just sketched is the cross-entropy $H(P,Q)$. And we have already proven that the cross-entropy can never be shorter than the self-entropy (and will in fact be larger, unless P=Q).

We can now speak of the optimal compressed length of data d given a model (grammar) h that generates d and assigns a probability $pr(d)$: it is $-log_2 pr_h(d)$. We'll express that as $|d|_h$. It is a length, whose unit of measure is the bit.

2.2 Arithmetic encoding

Arithmetic encoding was independently discovered by about 5 different people during the 1970s.[4] It is conceptually elegant and useful in practice too, and clarifies some of the ideas we have touched on so far. Arithmetic coding does not use an encoding in the sense that we have defined it above; it employs an algorithm to map strings in Σ^* to strings in $\{0,1\}^+$.

The fundamental insight behind arithmetic encoding is that if the sender and the receiver share a model that assigns a probability to each possible message from a countable set, then we can use the model to associate with each possible message m an interval I_m, and these intervals partition [0,1). To send a message m, then, we need only send the shortest (shortest, not smallest) binary string that corresponds to a real in I_m.

We can do this with pretty much any way of assiging a probability distribution over a countable set. If we want to send message m which is the i^{th}, we sum the probabilities of the messages with lower index numbers: q = $\sum_{j=1}^{i-1} pr(j)$, and define m's interval as $[q, q + pr(m))$.

That's a bit abstract. It's clearer in a simple case, like a unigram model. Let's suppose we are sending the message *eaii* (i.e., *eaii#*). Then we drill down

[3] Why? Look at the plot of $ln(x)$, and compute its first and second derivatives, and its value at (1,0): $ln(x) \leq x - 1$.

[4] The very best source on text encoding of all sorts is *Text Compression*, by Bell, Cleary, and Witten.

Symbol	Probability	Range
a	0.2	[0, 0.2)
e	0.3	[0.2, 0.5)
i	0.1	[0.5, 0.6)
o	0.2	[0.6, 0.8)
u	0.1	[0.8, 0.9)
\$	0.1	[0.9, 1.0)

Table 2: BCW example: arithmetic encoding

to smaller and smaller intervals. First, the table tells us that to encode e, we want to restrict our attention to [0.2, 0.5). Now we take that interval, and divide it up with exactly the same proportions as before (since this is a unigram model: if we were using a bigram model, we would divide e's interval up in a way different than how we divide up i's interval). We see that the interval corresponding to ea is [0.20, 0.26). We then divide that smaller interval up, and we see that corresponding to i is the interval [0.23, 0.236), and next the subinterval corresponding to the 2nd i in $eaii$ is [0.233, 0.2336). Finally we break up that interval to find the part corresponding to #, which is [0.23354, 0.2336). Now we just send the shortest binary fraction in that interval.

3 Learning Structure

> Suppose we do *not* know the structure of the device that produced the message (= the observations). How can we learn about that structure, and quantify our success in that task?

3.1 Bayes' Rule

Simple manipulations of the definition of conditional probability. By definition,

$$pr(A|B) = \frac{pr(A \ \& \ B)}{pr(B)}$$

so

$$pr(A \ \& \ B) = pr(A|B)pr(B).$$

and for the very same reason

$$pr(A \ \& \ B) = pr(B|A)pr(A).$$

Hence

$$pr(A|B)pr(B) = pr(B|A)pr(A)$$

or

$$pr(A|B) = \frac{pr(B|A)pr(A)}{pr(B)}.$$

Things start to get tricky when we think of one of the "events" as a *hypothesis*, because we have to ask what we mean by saying that a hypothesis has a particular probability. That is the heart of Bayesian reasoning: a willingness to go *there.*

Two styles of answers:

1. There is no reasonable choice for a prior probability distribution over hypotheses, but if we start with a distribution anyway (even though we know it is not justifiable), and then iteratively use Bayes' rule to update the distribution after we see a large sequence of data, the result posterior will be just fine. Then we are rethinking $pr(h)$ on the right-hand side: that only means the prior probability of h on the first iteration; on all later iterations, it means $pr(h|$data seen up through yesterday$)$.

2. It is possible to find an unbiased prior over all hypotheses, or very close to one, if we use the insights of computer science.

Let's take the negative log of Bayes's Rule (and using the fact that negative log is monotonically decreasing):

$$-log\, pr(h|d) = -log\, pr(h) - log\, pr(d|h) - log\, pr(d).$$

where the pr(d) $= \sum_{g \in \mathcal{H}} pr(d|g)$, where $\mathcal{H}$ is the space of all conceivable hypotheses. That is, we want to find

$$\hat{h} = arg\, min_h\, (\, -logpr(h) - logpr(d|h) - logpr(d)\,)$$

Since the third term is a constant for all h, we can drop it; and we can make use of the fact that we know that $-log\, pr(d|h)$ is the optimal compressed length of d under the probability distribution determined by h:

$$\hat{h} = arg\, min_h\, (\, -log\, pr(h) + |d|_h)$$

We need to ask now what $-log\, pr(h)$ could be, and since we did so well in making sense of the information content, or length, of data by asking how long the shortest string was that represented that data, we might as well try something like that to answer the question of what a prior over hypotheses might be. We will go back to what Solomonoff, the first person to provide a serious answer to this question, was thinking about when he was a student at the University of Chicago in the very early 1950s.

4 Knowing the future and the problem of induction

The problem of induction was first recognized in its modern form by David Hume (1711-1776), about 100 years into the period of modern science (if we date that to have begun with Isaac Newton). The question is this: what justifies our belief that the world will continue to be as it has been in the past? And how do we know what vocabulary to use when describing nature, so that the terms we use to describe nature are indeed the right terms for capturing what will remain constant over time—the laws of Nature?

We will discuss this question in a computational context because the concepts and tools of contemporary computer science are among the most useful in figuring out useful answers to these questions. And I will discuss problems in computational linguistics, where these tools have been applied directly to answering questions about the nature of scientific induction.

In particular, we will explore the ways in which information theory and algorithmic complexity—two of the central ideas of computer science—have been applied to better understand the nature of linguistic reality, and to solve questions that are practical, or nearly practical. We will take as our central example the problem of determining what the words are of a language in which we have only a large sequence of symbols. This sample could be a text in an unknown human language; it could be the human genome. How do we discover structure in that string, and how do we infer something about the nature of the device that generated the string? Let's consider the question a bit more, before turning to the historical background.

We have a string $\mathcal{S}$; its length is $|\mathcal{S}|$. How big? It doesn't matter; it could be a thousand symbols, a million, or much, much larger. Let's define its length as a *zillion*. It is composed of an alphabet Σ; it is a member of Σ^*. Σ might be {0,1}, or it might be the alphabet a-z, or it might be the symbols established by the Unicode consortium. What is the nature of this string of symbols? A natural view (though not the only one possible[5]) is that it is a concatenation of pieces of symbols strings, pieces that we will call *words*, following tradition. There are two extreme hypotheses, neither of which is satisfactory. The first is that is composed of a zillion words, each word one symbol long. The second hypothesis is that it consists of a single word, which is a zillion letters long.

The first hypothesis underfits the data: it fails to discover structure that exists in the data. If $\mathcal{S}$ is (let's say) the King James version of the Bible in English, then it fails to discover that there are strings like *the, when, heaven*, and so forth, that reoccur with high frequency. On the other hand, the second hypothesis (which takes the whole string to be a single, simple word) fails a different version of the same reason: there are many other real, and possible, texts that are written in the same language and thus share a lot of common substrings, those pieces we call English words.

This is our problem in a nutshell: any time we have experience in the real world, it is partly composed of information about events that will not happen again and partly composed of information about structure that will recur often. In the case of $\mathcal{S}$, the particular order of all of the words will not happen again in another text, but the words themselves will be found in other texts, and many of the subsequences will be found in other texts as well. Our task is to find these two kinds of structure in $\mathcal{S}$ or any other long string of symbols that may be given to us by Nature.

5 Prior over hypotheses

In 1921, Wrinch and Jeffreys pointed out that simplicity considerations in scientific hypotheses have a natural home in probability distributions. They wrote,

> it will never be possible to attach appreciable probability to an inference if it is assumed that all laws of an infinite class, such as all relations involving only analytic functions, are equally probable *a priori*.[6]

[5] You would probably also think about describing $\mathcal{S}$ with an n-th order Markov model, for some n, right?

[6] Wrich and Jeffreys 1921, p. 389, cited in Keuzenkamp and McAleer 1995.

Rather, they said, "the simpler the law, the greater is its prior probability" (p. 386).

6 Ray Solomonoff's problem and his solution

Please: read at least the first half of Ray Solomonoff's "The discovery of algorithmic probability." Note the people whose influence Solomonoff cites: Rudolf Carnap, in philosophy, and Nicolas Rashevsky and Anatol Rapoport, mathematical biology, all here at the U of Chicago. Also: Percy Bridgman, physicist and philosopher of science, and Alfred Korzybski; also Freud and Poincaré, and Claude Shannon.

> My general conclusion was that Bayes' theorem was likely to be the key. That a person was born with a reasoanbly good built-in a priori probability distribution. The person would then make predictions and decisions based on this distribution. The distribution was then modified by their life experience.

From Shannon came the idea that information could be *quantified*.

Carnap was puzzling over the meaning of probability: the early 20th century view leaned toward the freqentist view of probability, but Carnap was also exploring the "degree of confidence" conception of probability: "the degree of confidence one had in a hypothesis with respect to a certain body of data."

Huffman encoding came along in 1952, and offered "an approximate solution to a special case of the 'information packing problem."' This provides "a short code from knowledge of probabilities". This work was continued over the next 30 years to eventually arrive at arithmetic encoding. Solomonoff, however, wanted to obtain "probabilities from knowledge of short codes."

Marvin Minsky and John McCarthy: from them, Solomonoff learned about formal logic and Turing machines. 1956: Summer Study Group in Artificial Intelligence at Dartmouth University.

Solomonoff's question: how can you extrapolate from a long sequence of symbols? This was, in part, in response to Minsky's claim that "all mathematical problsm coulbe formulated as problems of inverting Turing machines." Given a machine M, we want to find a string p such that M(p)= s.

1956: Solomonoff's inductive inference machine: give it some input/output pairs (in binary encoding), and test it by giving inputs not in the training data. The machine, M, tries various primitive (and then not primitive) abstractions over the training data to see if they satisfy the input/output pairs. "After each round of training, various of the abstractions are given utility scores, depending on how successful they were in the prediction process." (Reminiscent of the genetic algorithms that John Holland was just starting to develop at that moment!)

On each iteration, those abstractions which were most successful on the previous round are preferred.

Chomsky's "Three models for the description of language" 1956. Chomsky's grammars were "organized better, easier to understand, and easier to generalize...his formal languages were ideal for induction. Furthermore, they would give a kind of induction that was considerably different from techniques used in

statistics up to that time. The kinds of regularities it could recognize would be entirely new."

Given a set of positive data $\mathcal{D}$, how do we find the "best" grammar? The right answer is neither of the two extremes — not the "ad hoc" grammar, which generates exactly $\mathcal{D}$, nor the "promiscuous" grammar, which generates Σ^*. What criterion could be used to find the right inbetween answer?

Solomonoff invented probabilistic languages and grammars. These are grammars that assign a probability distribution over their languages. Then probability that a grammar G generated $\mathcal{D}$, given $\mathcal{D}$, is the probability of G times $pr_G(\mathcal{D})$. But we don't know yet what the expression "the probability of a grammar G" means.

Deterministic acceptance grammar, given a UTM M, which produces M(x), given string x. Then we are trying to develop a system whereby we can represent a grammar as a finite string a, and the output of M(as) is 1 (and M stops) for all grammatical strings s. A probabilistic version outputs a probability instead of "1".

A deterministic generative grammar takes a and a positive integer n (i.e., an) as input, and generates the n^{th} sentence of a as output. The probabilistic counterpart is not so obvious: we assign a distribution over sentences s such that "the probability that M(hr) will print out s and stop" when r is a random sequence of bits is the probability of s. Clearly, if M stops, it has not read all of the digits of r.

Consider some finite string r which generates s when fed to M after h: M(hr) = s, and M stops. Then the pr(s) (given grammar h) $\geq 2^{-|r|}$. Why not extend that by "forgetting" that h is a grammar, and say that the probability of s over *all* grammars is $\geq 2^{-|hr|}$?

Big question that remained: how sensible is it to think of such a quantity as "universal" in any sense if its value depends on the choice of universal Turing machine (UTM)?

6.1 Chaitin 1977

One of the crucial contributions that Chaitin made to this development was th realization that the program that embodied the hypotheses we are interested in must be encoded in a prefix-free fashion in order to have a well-defined probability distribution over hypotheses.

7 Kolmogorov complexity

We now arrive at the definition of the Kolmogorov complexity of a grammar g, defined relative to a particular universal Turing machine (UTM_1). Suppose our language is Klingon, and the probability of message $m =$ "Xa" is $2-5$, and g assigns m the interval [0.1, 0.10001). We want a program p for UTM which will generate an ASCI (or Unicode) version of "Xa" as its output when p is followed on UTM's input tape by the string "0.100001", which is strictly within the interval—and then halts. That is what we mean for a UTM program to embody a grammar g – it must work that way, to generate every message from an appropriate binary string. Of course, p must be expressed in a self-delimiting

way (e.g., with a prefix-encoding), so that the UTM knows where p ends and where the string designating the real number begins.

The Kolmogorov complexity of a grammar g is, then, the length of the shortest program p for UTM that embodies g: and we write $\mathcal{C}_{UTM}(g) = |p|$; and its probability is $pr_{UTM}(g) = 2^{-|p|}$.

8 Language induction: Word chunking

8.1 Does word induction optimize some quantity?

9 Structure above and below the word

9.1 Below the word: morphology

9.2 Above the word: syntax

10 On the non-uniqueness of UTMs

How important is it that there are many UTMs, and we have not specified which particular one we should use? What is the impact of this observation on our conclusions?

11 Handout

We start with an alphabet Σ, and one or more strings from Σ^+, which we call messages, and which we notate as m or something orthographically like m.

In real life, we often have a "binary message in clear"—like ASCII or simple (2-byte) unicode.

$$m \in \Sigma^+ \stackrel{Tr}{\Rightarrow} \{0,1\}^+ \stackrel{E}{\Rightarrow} \{0,1\}^+$$

We typically think of encoding in two steps: the first providing a binary message "in clear": ASCII, etc.: where we typically apply a method of encoding that minimizes the length of the worst case (call this Tr for "trivial") by using an encoding whose size is $\lceil log_2 \Sigma \rceil$. But we're really interested in the second encoding E, when our goal is to minimize the length of $E(m)$ (or $E(Tr(m))$, more precisely).

Some terms:

- count
- frequency
- probability
- plog(): $-1 \times log_2()$.
- expected count

- distribution over S: a mapping pr from $S \rightarrow [0,1]$ such that $\sum_{x \in S} pr(s) = 1.0$. Since our sets are all countable, we can enumerate a set's members, and so a distribution pr can also be associated with a partitioning of [0,1), where each $x_i \in S$ is associated with the interval $[\sum_{j=1}^{i-1} pr(x_j), \sum_{j=1}^{i-1} pr(x_j) + pr(x_i))$.

- binary distribution: a set of intervals $I_i = (l_i, r_i)$ where the length of each interval $(r_i - l_i)$ is an integral power of 2 (a negative integer, of course!).

Classical encoding: function from $\Sigma \rightarrow \{0,1\}^+$ which is *self-delimiting*: i.e., *prefix-free*. An encoding e is prefix free if there are no two strings in the image of e where one is a prefix of the other. It follows that the images of e can be expressed as a binary tree, in which each left branch of the tree is labeled '0', and each right branch is labeled '1'. The prefix condition means that only terminal nodes in the tree correspond to images of e.

We normally care only about *complete* encodings, where we don't waste any possible strings. We can waste a possible string if we have non-branching non-terminal nodes or unused terminal nodes.

Each encoding can be mapped to a binary tree; each binary tree induces a binary distribution, where $pr(s) = 2^{-|e(s)|}$. So each encoding corresponds to a binary distribution. The length of the encoding corresponding to a value x in the distribution is plog (x).

Suppose we have a message m, and the counts of the symbol $l \in \Sigma$ in it are represented $[l]_m$. Then the length of the compressed version of m is $\sum_{l \in \Sigma} [l]_m |e(l)|$, and its average length (per symbol in m) is $\sum_{l \in \Sigma} freq_m(l)|e(l)|$.

The binary distribution corresponding to e we will call q. The average encoding length of m is then

$$-\sum_{x \in \Sigma} freq_m(x) \, log \, q(x)$$

This is an example of *cross-entropy* H(p,q) (where p, q are two distributions over the same set):

$$H(p,q) = -\sum_{x \in \Sigma} p(x) \, log \, q(x)$$

The cross-entropy is greater than or equal to the (self-entropy).

The difference between the cross-entropy and the self-entropy is the Kullback-Leibler divergence: $H(p,q) - H(p,p) = \sum p(x) log \frac{p(x)}{q(x)}$:

Arithmetic encoding.