A Tutorial introduction to the ideas behind Normalized cross-entropy and the information-theoretic idea of Entropy

Although the whole idea of entropy turns on Claude Shannon's theoretical idea of "information", we need not get into the idea of "information" here. Instead, we will talk about an idea that is more intuitive: uncertainty. Suppose we have a sequence of symbols being produced by something or another (say letters appearing on a ticker-tape), or suppose we have a sequence of events that are occurring. Suppose we know what the possible events are (for letters appearing on a ticker tape there are 26 letters in English, 26 possibilities). And suppose we want to guess which of the possible symbols or events will be the next in the sequence. We can say that we have n possible symbols or events (for example, if the symbols or events are English letters, then $n = 26$). Because we are trying to guess the next letter, there is uncertainty for us. Correspondingly, whatever is generating the symbols or events has a choice to make (what to produce next).

If each of the n possible symbols or events is equally likely, then the probability of each is $1/n$ (for our example of English letters, the probability that the next letter in a sequence will be an "A" is 1/26 if the letters are equally probable, and so forth). We can write this as $p_A = 1/26$, $p_B = 1/26$, . . ., $p_Z = 1/26$. To generalize this, let i and j be two possible events in our sequence. Then just as we wrote $p_A = 1/26$, we can likewise write $p_i = 1/n$ and $p_j = 1/n$. Notice that as n becomes larger, we can say that "the amount of choice" or the uncertainty about what symbol or event will be next becomes larger. Intuitively (and mathematically), for any given value of n, the maximum amount of choice or uncertainty (about the next symbol) will occur in the situation we have been considering here, when the probabilities of all the symbols are equal. If we try to guess the next equally-probable symbol or event when n is large, our prediction will usually be wrong. But if some symbols are more likely than others, the amount of choice or uncertainty will be less—if one symbol occurs 99.99% of the time, and the other symbols almost never occur, then we would predict that the next symbol will be the symbol that occurs 99.99% of the time and we would rarely be surprised by the actual next symbol.

Entropy:

We can formalize this notion and give it a mathematical analysis. We call the amount of choice or uncertainty about the next symbol "entropy" and (by historical convention) use the symbol H to refer to the entropy of the set of probabilities $p_1, p_2, p_3, \ldots, p_n$

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

Formula 1. Entropy.

In that formula, in the special case where all n symbols or events are equally likely, p_i is always $1/n$ and the formula reduces (via algebra) to $H = \log_2 n$, which is probably more intuitively obvious.

To give some actual examples

- for our example of 26 equally-probable letters, H is just barely over 4.7,
- for heads vs. tails of a fair coin $H = 1.0$, and
- where only one outcome is possible $H = 0.0$.

We must note that $H = 0.0$ not just when there is only one choice but also when there are multiple choices of which only one ever occurs. For example, say we have two choices: the first is that in the morning the sun will come up in the east, and the other is that in the morning the sun will come up in the west. The sun always comes up in the east, so H is 0.

Usually some of the actual choices are more likely than others, and in that case H will always be less than if the choices are all equally probable. For example, the letters in actual English text are not equally probably (E occurs often, but F, Z, Q, and P are less common.) So, the entropy of the set of probabilities of the letters in actual English text is considerably less than 4.7 (which is the maximum entropy possible). All 26 letters actually occur fairly frequently in English text, so there is considerable uncertainty about what letter will occur next in actual English text, and the entropy of the set of probabilities of the letters in actual English text is considerably greater than zero. So, we can see that the entropy H for the set of actual probabilities of the 26 letters in English is somewhat greater than 0.0 and somewhat less than 4.7

In many cases we are concerned with $n = 2$, for example when our choices are “correct” and “incorrect”. The entropy, H, of the probability distribution of these two choices will be

$$H = -p_{correct} \log_2 p_{correct} - (1 - p_{correct}) \log_2 (1 - p_{correct}) .$$

Formula 2. Entropy for just two choices.

Finally, since H is over some probability distribution, we can indicate that fact in our notation. If r is one probability distribution (a random variable) and s is another, we can write $H(r)$ and $H(s)$. Think, here, of r and s as two probability distributions over the same set of choices (for example, our set of 26 English letters, or the set of words in a document, or correct vs. incorrect).

Cross-entropy:

For two probability distributions over the same set of choices, we can extend the idea of entropy to a measure of the difference between the two probability distributions. This amounts to a measure of the difference between the two distributions, with values closer to zero where the differences between the distributions are less:

$$\sum_{i=1}^{n}\left(r_i \log_2 \frac{r_i}{s_i} \right).$$

Formula 3. Cross-entropy of two probability distributions.

Think of s_i here as a known distribution—note that swapping the two distributions will not give the same result. The cross-entropy value will be zero if distributions r and s are identical. Note that the value is undefined if any of the s_i values are zero. Note also that $\log_2(r_i / s_i)$ in that formula may be positive or negative, and hence the entire sum may be positive or negative. One useful application of cross-entropy is where s_i represents a known a-priori distribution and you have many candidate r_i distributions from which you wish to pick the r_i that most closely approximates s_i (you wish to find the minimum cross-entropy).

Normalized Cross-entropy:

Suppose we want to look at the cross-entropy of the distribution of correct vs. incorrect words in the output of a speech-to-text (STT) system compared to the distribution of confidence factors that the STT system assigns to those output words (a function â(w) of the output word w). The values of $\hat{a}(w)$ are assigned by the STT system. These values are in the range $0.0 < \hat{a}(w) < 1.0$ (0.0 would mean that the system is sure the word is incorrect, and 1.0 would mean the system is sure the word is correct—and *note well,* the system cannot be sure that either of these is the case). Values around 0.5 indicate a 50-50 chance that the word is correct or not (alternatively, 0.5 means the system has no idea whether the word is correct or not). So, we want to look at whether there is some connection between the STT system actually getting a word correct and the system's opinion about whether it got the word correct. Although one of these (correct vs. incorrect) has just two values and the other (confidence) is a floating-point value, you will note that the formula for normalized cross-entropy (NCE) that is coming up will take that into account.

You will note that the formulas below involve the log of the confidence factors. In practice (because $\log(0.0) = -\infty$) NIST's code that calculates normalized cross-entropy (i.e., SCLite) will replace any confidence factor of 0.0 with 0.0000001 and will replace any confidence factor of 1.0 with 0.9999999 — but systems should not generate values of 0.0 or of 1.0 for $\hat{a}(w)$.

Let M be the total number of words in the STT output, and let m be the number of those M words that are correct. Then the average probability that a word in the STT output will actually be correct is $p_c = m/M$. As suggested by the earlier discussions, given p_c there is a maximum value for the entropy for STT actually getting the words correct vs. incorrect, and we multiply that entropy by M giving a value that we will call $H_{\max}$:

$$H_{\max} = -m \log_2 (p_c) - (M - m) \log_2 (1 - p_c)$$

Formula 4. $H_{\max}$ for *NCE*

$H_{\max}$ is equivalent to Formula 2 above, using p_c, and multiplied by M. This value will be positive, and pertains to the two choices (correct vs. incorrect), not to the $\hat{a}(w)$ confidence factors. Our formula for normalized cross-entropy is:

$$NCE = \frac{H_{\max} + \sum_{correct\_w} \log_2(\hat{a}(w)) + \sum_{incorrect\_w} \log_2((1-\hat{a}(w))}{H_{\max}}$$

Formula 5. *NCE*

Looking at Formula 5, we see that the numerator consists of three terms: the first is $H_{\max}$ as given in formula 4, the second deals with the confidence factors for the words that STT got correct, and the third deals with the confidence factors for the words that STT got incorrect. Note that $H_{\max}$ will always be positive and that the two terms dealing with the confidence factors will always be negative if the confidence factors are less than 1.0

As the confidence factors for the correct words become much larger than the confidence factors for the incorrect words, the numerator will tend to become positive. If the confidence factors for the correct words approach 1.0 and the confidence factors for the incorrect words approach 0.0, then *NCE* will be dominated by $H_{\max}$ (which can approach 1.0 in the limit—and because we divide by $H_{\max}$, *NCE* too will approach 1.0 in the limit). If the confidence factors were reversed (high confidence factors for the incorrect words, and vice-versa), then *NCE* would be quite negative, dominated by the log of the incorrectly small confidence factors for the correct words and the incorrectly large confidence factors for the incorrect words.

There is nothing "magic" about getting a value for *NCE* greater than zero. In practice, if the confidence factors correspond with whether the words are actually correct but have considerable random scatter (with the random scatter having a large magnitude compared with the average difference between the confidence factors for the correct words vs. for the incorrect words), the numerator (and thus *NCE*) may still be negative. It may interesting to note that if the system could somehow know p_c then assigning a confidence factor of p_c to every word would give exactly *NCE* = 0.0 (thus, assigning a confidence factor greater than p_c to every correct word, and less than p_c to every incorrect word will give a positive value for *NCE*). But of course, the system will not know, in advance, the value of p_c for a particular run.

<u>How to get the best possible score on NCE:</u>

If the random scatter of the system's system-internal confidence scores is large (so that the confidence scores are not informative), the system's *NCE* score will likely be better if the confidence scores that it puts out are around the range of the system's likely p_c values (avoiding low confidence scores on correct words and high confidence scores on incorrect words, since both these kinds of errors will hurt the *NCE* score).

On the other hand, if your system-internal confidence factors are really good at predicting whether the word is accurate or not, then your system will get the best possible score on the *NCE* metric if the confidence factors that it puts out for the correct words are almost 1.0, and the confidence factors that it puts out for the incorrect words are close to 0.0—so you would want your system to generate confidence factors that cover the full range (0.0, 1.0). In practice, we do the *NCE* calculations in double-precision, so it is generally safe to assume that there are at least nine (base-10) digits of precision, and thus the full range that systems could generate would be 0.000000001 to 0.999999999.

If your system-internal confidence scores are informative, it is also important that the distribution of the confidence factor values that your system puts out correspond as closely as possible to the actual probability that the words are correct (for example, if some value of your system-internal confidence factor corresponds to a 92% probability that the word is correct, put out a value of 0.92 rather than some other value). If there is an informative but not precise correspondence between your system-internal confidence factor and the actual probability that the word is correct, then it would probably be best to "bin" your system-internal confidence values—for example, if some range of your system-internal confidence factors corresponds to about a 95% to 100% probability that the word is correct, then you could put out a value of 0.975 when your system-internal confidence factors are in that bin (although it can be shown that you will score best on *NCE* if you put out a value that is equal to the actual p_c for that bin, any value reasonably close to the actual p_c will get you a good *NCE* score).