Cross Entropy

Signal Data Science

This is a short primer on the notion of the *cross entropy*. We'll develop it from the standpoint of information theory.

Uniquely decodable codes

We have to begin with some preliminary information about codes.

An *alphabet* is a set of symbols. For example, $\{a, b, c\}$ and $\{0, 1\}$ are both alphabets. *Words* are made up of concatenated symbols from some particular alphabet, *e.g.*, *cbacb* is a word constructed from the alphabet $\{a, b, c\}$.

Now, a *code* is a mapping from a *source alphabet* to the *words of a target alphabet*. For example, consider the code $C = \{a \mapsto 0, b \mapsto 01, c \mapsto 011\}$. This is a code which maps the symbols of the alphabet $\{a, b, c\}$ to words in the alphabet $\{0, 1\}$. The *code words* of C are the set $\{0, 01, 011\}$.

Let's consider a particular type of code: *prefix codes*, where there is no code word which is the prefix of a different code word. For example, a code with code words $\{9, 5, 59, 55\}$ is not a prefix code, because "5" is a prefix of "59" and "55". However, a code with code words $\{9, 55\}$ is a prefix code, because "9" is not a prefix of "55" and "55" is not a prefix of "9".

Prefix codes are part of a larger class of very special codes called *uniquely decodable codes*. A code is uniquely decodable if you can look at a sequence of code words, without any breaks in between, and reconstruct the source sequence (with knowledge of the code's mappings).[1]

To illustrate, if a code has code words $\{0, 1, 10\}$, then this code is *not* uniquely decodable; if someone receives the message "0110", it's impossible to tell whether the last two characters, "10", are supposed to be interpreted as the code word "10" or as "1" followed by "0". As such, "0110" could stand for "0, 1, 10" or for "0, 1, 1, 10". However, a code with code words $\{0, 1, 2, 32\}$ would be uniquely decodable.

[1] All prefix codes are uniquely decodable, but not all uniquely decodable codes are also prefix codes.

The Kraft–McMillan theorem

The Kraft–McMillan theorem says the following:[2]

Assumptions

Suppose we start with a finite alphabet $X = \{x_1, x_2, \ldots, x_n\}$ and that we have a uniquely decodable code $C = \{x_1 \mapsto c_1, x_2 \mapsto c_2, \ldots, x_n \mapsto c_n\}$.[3] Suppose also that l_i is the length of the code word c_i in *bits*.[4]

Results

It's necessarily true that $\sum_i 2^{-l_i} = 1$. The interpretation of this is that the function $p(x_i) = 2^{-l_i}$ can be interpreted as a *discrete probability distribution* over the set X (because each value of $p(x_i)$ is under 1, and they all sum to 1 together).

There is also a converse result. Suppose that we are given a discrete probability distribution with probabilities p_i. Then there exists a uniquely decodable code with code word lengths $l_i = -\log_2 p_i$.[5]

Speaking loosely, if you have a uniquely decodable code over X, then there is a probability distribution *implicit* in that code, and conversely if you have a probability distribution over X, then there is a uniquely decodable code *implicit* in that probability distribution. As such, one can think of a sort of correspondence between probability distributions and uniquely decodable codes over sets.

In particular, the uniquely decodable code C corresponding to a particular distribution P is in some sense the *optimal coding scheme* if elements of X are sampled via P and then encoded using C, because the most likely elements of X – those for which p_i is the greatest – are the same ones for which the length of the corresponding code word, $-\log_2 p_i$, is the smallest. As such, in expectation the length of the encoded message will be minimized using C (as opposed to using other coding schemes).

[2] This theorem is more properly formulated as an inequality, but it's presented here in a simplified form for the sake of exposition.

[3] We also need a non-redundancy assumption, but that's not as important: the general thrust of this expository passage does not suffer for its omission.

[4] Precisely, think of the code words of C as being constructed from the alphabet $\{0, 1\}$.

[5] We should be taking the *ceiling* of $-\log_2 p_i$, but I've avoided doing so here to cut back on notational clutter.

The cross entropy

Let us consider a set X with n elements $x_1, \ldots, x_n$, and suppose that we sample elements of X according to the probability distribution P with probabilities p_i. We would like to encode our sampled elements of X using the optimal coding scheme for P, which we will call C, mapping $x_i \mapsto c_i$.

In expectation, how many bits does it take to encode a randomly sampled element of X? Recall that each code word c_i has length $-\log_2 p_i$. Then we have the following:

$$L_1 = \mathbb{E}_P[-\log_2 P] = \sum_{i=1}^{n} \left[(p_i) \times (-\log_2 p_i)\right].$$

(Recall that the expected value of a discrete probability distribution is the sum over probability times outcome. p_i is the probability of getting x_i, and the length of the corresponding code word is $-\log_2 p_i$, so we multiply the two together and sum over all values of i.)

However, suppose that we mistakenly assume that the data are drawn from a *different* distribution Q, with probabilities q_i and an optimal coding scheme D that maps $x_i \mapsto d_i$. The length of the code word d_i is $-\log_2 q_i$. Of course, we'll want to use D to encode our sampled data. Then the expected number of bits it takes to encode a randomly sampled element of X will be

$$L_2 = \mathbb{E}_P[-\log_2 Q] = \sum_{i=1}^{n} \left[(p_i) \times (-\log_2 q_i)\right].$$

Here, we take the expectation *over* the the true probability distribution P, but we take the expectation *of* the length of the code words of the wrong probability distribution Q. In the sum, p_i remains, because the probability of getting x_i is still p_i, but $-\log_2 p_i$ has changed to $-\log_2 q_i$, because we are using the coding scheme optimized for Q rather than for P.

Of course, we have $L_1 \leq L_2$. Also, L_1 is called the *entropy* of P, whereas L_2 is the *cross entropy* between P and Q. Note that the cross entropy is not symmetric, *i.e.*, $H(P, Q)! = H(Q, P)!$

In short:

- The **entropy** of a distribution P over a set X is the number of bits it takes to encode information drawn from P using the optimal coding scheme for P. It is denoted $H(P)$.
- The **cross-entropy** between two distributions P and Q, both over a set X, is the numebr of bits it takes to encode information drawn from P using the optimal coding scheme for Q, *i.e.*, the expected length of a code word

when we incorrectly assume that the data are drawn from Q. It is denoted $H(P, Q)$.

As a bonus, we can also get the **Kullback-Leibler divergence** of Q from P, which is given by

$H(P, Q) - H(P) = \sum_i p_i(\log p_i - \log q_i) = \sum_i p_i \log \frac{p_i}{q_i} = \mathbb{E}_P[\log(P/Q)]$.$

It is usually denoted $D_{\mathrm{KL}}(P||Q)$. It is true (you can just take this as a given) that the KL divergence is always nonnegative, which means that $H(P, Q) \geq H(P)$, precisely as one might expect.

Cross entropy as a loss function

The cross entropy is a particularly natural loss function for machine learning. Suppose that we have a multiclass classification problem for which we use, say, multinomial logistic regression (or something more complicated), where we classify our data points $\mathbf{x}_i$ and assign each one a label $y_i \in \{1, 2, 3\}$.[6]

For each i, we can consider each of the possible labels to be an *event*, analogous to our set X in the previous sections. Then we can call the *true probability distribution* p_i a distribution which is 1 for the correct label and 0 otherwise, and we call the *estimated probability distribution* q_i the estimated probability for each label.

Next, for each i, we can calculate the corresponding cross entropy:

$$H(p_i, q_i) = \sum_{y \in \{1,2,3\}} \left[-p_i(y) \log q_i(y)\right].$$

The loss function we can use is just the *average cross entropy* over all of the data points, that is,

$$L = \frac{1}{n} \sum_{i=1}^{n} H(p_i, q_i) = \frac{1}{n} \sum_{i=1}^{n} \sum_{y \in \{1,2,3\}} \left[-p_i(y) \log q_i(y)\right].$$

This is sometimes equivalently called **multinomial log loss**.

Binary classification

In the case of binary classification, where $y \in \{0, 1\}$, let's use the convenient shorthand $y_i = p_i(1)$ and $\hat{y}_i = q_i(1)$. Then we can simplify the notation a little bit to get

[6] More or fewer labels are possible, but we use 3 for illustrative purposes.

$$H(p_i, q_i) = \sum_{y \in \{0,1\}} [-p_i(y) \log q_i(y)] = -y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i).$$

Averaging over all data points, we obtain the typical **log loss** function, with which you may be familiar:

$$L = -\frac{1}{n} \sum_{i=1}^{n} \left(y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \right).$$

Relation to likelihood function

Note that optimizing the log loss is equivalent to optimizing the log-likelihood. For simplicity, let's restrict to the context of binary classification. Suppose that we have n pieces of data $y_i \in \{0, 1\}$. Now suppose that we have some generative model which tells us that the ith datum is equal to 1 with probability p_i. Then under this model, the probability of observing the actual data, *i.e.* the **likelihood**, is

$$\mathcal{L} = \prod_{i=1}^{n} p_i^{y_i} (1 - p_i)^{1 - y_i}.$$

Then the **log-likelihood** is just

$$\log \mathcal{L} = \sum_{i=1}^{n} \left[y_i \log p_i + (1 - y_i) \log(1 - p_i) \right],$$

because the logarithm converts the product to a sum.

Compare this expression with the log loss for binomial classification in the previous section. It should be fairly evident that maximizing the log-likelihood is equivalent to minimizing the log loss! (The only difference is a sign change and a scaling factor of $1/n$.)

Additional reading

- Shalizi's Information Theory is a good explanation of information-theoretic entropy as well as an excellent source for further reading.

The Kraft–McMillan theorem: more rigor

Feel free to skip this section. It's a development of the Kraft–McMillan theorem in more precise terms.

Formal statement

First, we'll give a formal statement of the theorem. This may not be immediately intelligible to you, but that's fine; try to relate it back to the discussion above (along with the footnotes).

Consider the alphabet $S = \{s_1, s_2, \ldots, s_n\}$ and a code which maps s_i to a code word with length l_i constructed from an alphabet of size r. Then

$$\sum_{i=1}^{n} \left(\frac{1}{r}\right)^{l_i} \leq 1.$$

Conversely, for natural numbers $l_1, l_2, \ldots, l_n$ satisfying the above inequality, there exists a uniquely decodable code over an alphabet of size r with those code word lengths.

Proof for prefix codes

To be continued …

See https://en.wikipedia.org/wiki/Kraft%27s_inequality#Proof_for_prefix_codes

Proof for the general case

To be continued …