**Essential Math for Data Science: Information Theory**

Entropy, cross-entropy, log loss, and KL divergence

[Hadrien Jean](https://medium.com/@hadrienj?source=post_page-----5d0380232ca1--------------------------------)

[Hadrien Jean](https://medium.com/@hadrienj?source=post_page-----5d0380232ca1--------------------------------)

[Dec 2, 2020·11 min read](https://towardsdatascience.com/essential-math-for-data-science-information-theory-5d0380232ca1?source=post_page-----5d0380232ca1--------------------------------)

(Image by author)

Information Theory

The field of *information theory* studies the quantification of information in signals. In the context of machine learning, some of these concepts are used to characterize or compare probability distributions. The ability to quantify information is also used in the decision tree algorithm, to select the variables associated with the maximum information gain. The concepts of entropy and cross-entropy are also important in machine learning because they lead to a widely used loss function in classification tasks: the cross-entropy loss or log loss.

Shannon Information

The first step to understanding information theory is to consider the concept of the quantity of information associated with a random variable. In information theory, this quantity of information is denoted as *I* and is called the *Shannon information*, *information content*, *self-information*, or *surprisal*. The main idea is that likely events convey less information than unlikely events (which are thus more *surprising*). For instance, if a friend from Los Angeles, California tells you: “It is sunny today”, this is less informative than if she tells you: “It is raining today”. For this reason, it can be helpful to think of the Shannon information as the amount of surprise associated with an outcome. You’ll also see in this section why it is also a quantity of information, and why likely events are associated with less information.

Bits represent variables that can take two different states (0 or 1). For instance, 1 bit is needed to encode the outcome of a coin flip. If you flip two coins, you’ll need at least two bits to encode the result. For instance, 00 for HH, 01 for HT, 10 for TH, and 11 for TT. You could use other codes, such as 0 for HH, 100 for HT, 101 for TH, and 111 for TT. However, this code uses a larger number of bits on average (considering that the probability distribution of the four events is uniform, as you’ll see)

Let’s take an example to see what a bit describes. Erica sends you a message containing the result of three coin flips, encoding ‘heads’ as 0 and ‘tails’ as 1. There are 8 possible sequences, such as 001, 101, etc. When you receive a message of one bit, it divides your uncertainty by a factor of 2. For instance, if the first bit tells you that the first roll was ‘heads’, the remaining possible sequences are 000, 001, 010, and 011. There are only 4 possible sequences instead of 8. Similarly, receiving a message of two bits will divide your uncertainty by a factor of *2²*; a message of three bits, by a factor of *2³*, and so on.

Note that we talk about “useful information”, but it is possible that the message is redundant and convey less information with the same number of bits.

Let’s say that we want to transmit the result of a sequence of eight tosses. You’ll allocate one bit per toss. You thus need eight bits to encode the sequence. The sequence might be for instance “00110110”, corresponding to HHTTHTTH (four “heads” and four “tails”).

However, let’s say that the coin is biased: the chance to get “tails” is only 1 over 8. You can find a better way to encode the sequence. One option is to encode the index of the outcomes “tails”: it will take more than one bit, but ‘tails’ occurs only for a small proportion of the trials. With this strategy, you allocate more bits to rare outcomes.

This example illustrates that more predictable information can be compressed: a biased coin sequence can be encoded with a smaller amount of information than a fair coin. This means that Shannon information depends on the probability of the event.

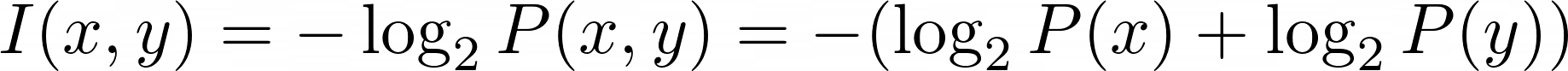
Shannon information encodes this idea and converts the probability that an event will occur into the associated quantity of information. Its characteristics are that, as you saw, likely events are less informative than unlikely events and also that information from different events is additive (if the events are independent).

Mathematically, the function *I(x)* is the information of the event X*=x* that takes the outcome as input and returns the quantity of information. It is a monotonically decreasing function of the probability (that is, a function that never increases when the probability increases). Shannon information is described as:

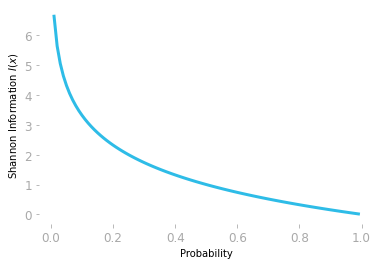
Image for post

The result is a lower bound on the number of bits, that is, the minimum amount of bits needed to encode a sequence with an optimal encoding.

The logarithm of a product is equal to the sum of the elements: *log2(ab)=log2(a)+log2(b)*. This property is useful to encode the additive property of the Shannon information. The probability of occurrence of two events is their individual probabilities multiplied together (because they are independent, as you saw in [Essential Math for Data Science](https://www.essentialmathfordatascience.com/)):



Let’s plot this function for a range of probability between 0 and 1 to see the shape of the curve:



*Figure 1: The quantity of information is given by the negative logarithm of the probability.*

As you can see in Figure 1, the negative logarithm function encodes the idea that a very unlikely event (probability around 0) is associated with a large quantity of information and a likely event (probability around 1) is associated with a quantity of information around 0.

Since you used a base-two logarithm np.log2(), the information *I(x)* is measured in *bits*.

Entropy

Consider for instance a biased coin, where you have a probability of 0.8 of getting ‘heads’.

1. Here is your distribution: you have a probability of 0.8 of getting ‘heads’ and a probability of *1–0.8 = 0.2* of getting ‘tails’.
2. These probabilities are respectively associated with a Shannon information of:

Image for post

and

Image for post

Image for post

and

Image for post

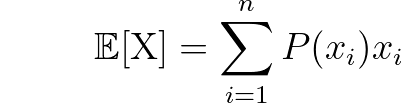
Image for post

The average number of bits required to describe a series of events from this distribution is 0.72 bits.

To summarize, you can consider the entropy as a summary of the information associated with the probabilities of the discrete distribution:

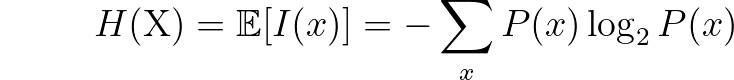
1. You calculate the Shannon information of each probability of your distribution.
2. You weight the Shannon information with the corresponding probability.
3. You sum the weighted results.

The entropy is the expectation of the information with respect to the probability distribution. Remember from [Essential Math for Data Science](https://www.essentialmathfordatascience.com/) that the expectation is the mean value you’ll get if you draw a large number of samples from the distribution:



with the random variable X having *n* possible outcomes, *x\_i* being the *i*th possible outcome corresponding to a probability of *P(x\_i)*. The expected value of the information of a distribution corresponds to the average of the information you’ll get.

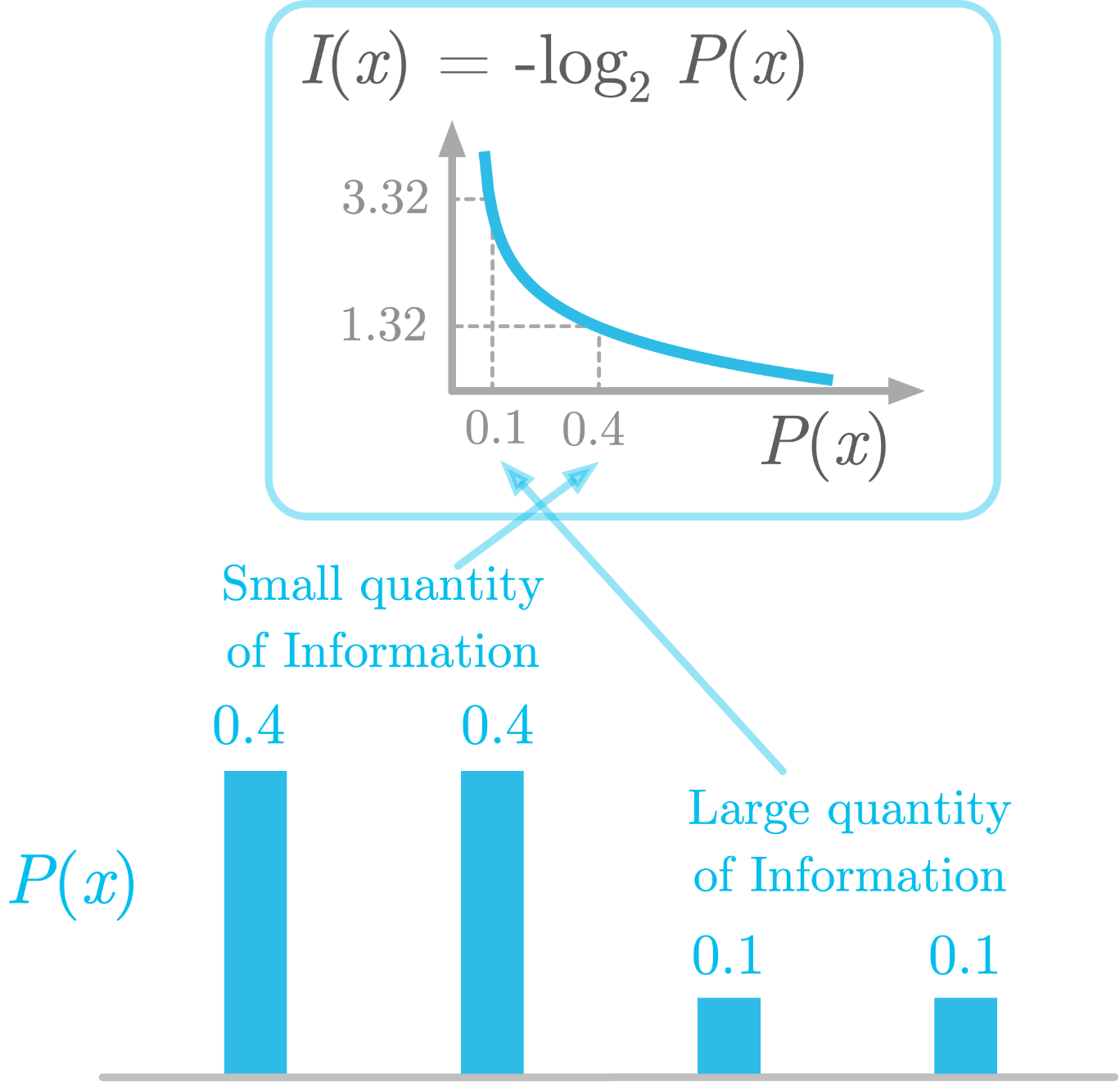
Following the formula of the expectation and the Shannon information, the entropy of the random variable X is defined as:



The entropy gives you the average quantity of information that you need to encode the states of the random variable X.

Note that the input of the function *H(*X*)* is the random variable X while *I(x)* denotes the Shannon information of the event X*=x*. You can also refer to the entropy of the random variable X which is distributed with respect to *P(x)* as *H(P)*.

Let’s take an example: as illustrated in Figure 2 in the bottom panel, you have a discrete distribution with four possible outcomes, associated with probabilities 0.4, 0.4, 0.1, and 0.1, respectively. As you saw previously, the information is obtained by log transforming the probabilities (top panel). This is the last part of the entropy formula: *log2 P(x)*.



*Figure 2: Illustration of the entropy as the weighted sum of the Shannon information.*(Image by author)

Each of these transformed probabilities is weighted by the corresponding raw probability. If an outcome occurs frequently, it will give more weight into the entropy of the distribution. This means that a low probability (like 0.1 in Figure 2) gives a large amount of information (3.32 bits) but has less influence on the final result. A larger probability (like 0.4 in Figure 2) is associated with less information (1.32 bits as shown in Figure 2) but has more weight.

In the example of a biased coin, you calculated the entropy of a Bernoulli process (more details about the Bernoulli distribution in [Essential Math for Data Science](https://www.essentialmathfordatascience.com/)). In this special case, the entropy is called the *binary entropy function*.

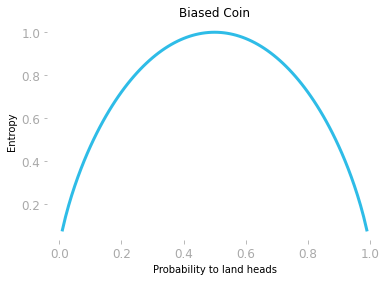
To characterize the binary entropy function, you’ll calculate the entropy of a biased coin described by various probability distributions (from heavily biased in favor of “tails” to heavily biased in favor of “heads”).

Let’s start by creating a function to calculate the entropy of a distribution that takes an array with the probabilities as input and returns the corresponding entropy:

You can also use entropy() from scipy.stats, where you can specify the base of the logarithm used to calculate the entropy. Here, I have used the base-two logarithm.

Let’s take the example of a fair coin, with a probability of 0.5 of landing ‘heads’. The distribution is thus 0.5 and *1–0.5 = 0.5*. Let’s use the function we just defined to calculate the corresponding entropy is:

The function calculates the sum of P \* np.log2(P) over each element of the array you use as input. Using an array as input As you saw in the previous section, you can expect a lower entropy for a biased coin. Let's plot the entropy for various coin biases, from a coin landing only as 'tails' to a coin landing only as 'heads':

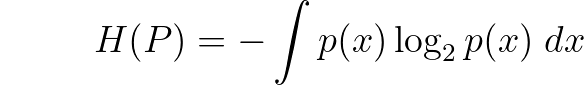


*Figure 3: Entropy as a function of the probability to land “heads”.*

Figure 3 shows that the entropy increases until you reach the more uncertain condition: that is, when the probability of landing ‘heads’ equals the probability of landing ‘tails’.

The entropy of a continuous distribution is called *differential entropy*. It is an extension of the entropy for discrete distribution, but it doesn’t satisfy the same requirements. The issue is that values have probability tending to zero with continuous distributions, and encoding this would require a number of bits tending to infinity.

It is defined as:



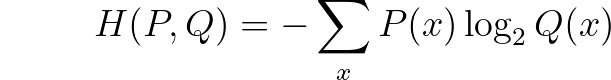
Differential entropy can be negative. The reason is that, as you saw in [Essential Math for Data Science](https://www.essentialmathfordatascience.com/), continuous distributions are not probabilities but probability densities, meaning that they don’t satisfy the requirements of probabilities. For instance, they are not constrained to be lower than 0.

Cross-Entropy

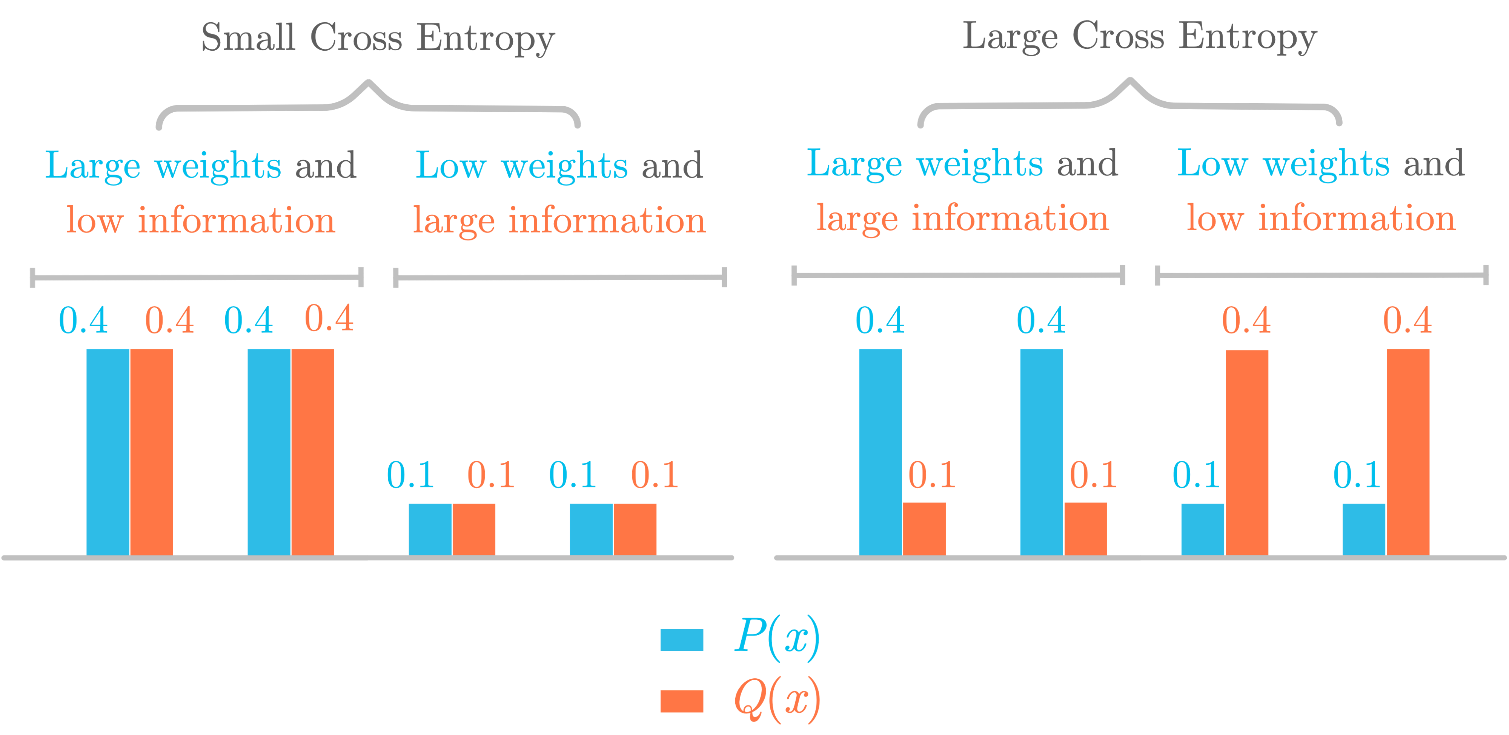
The concept of entropy can be used to compare two probability distributions: this is called the *cross-entropy* between two distributions, which measures how much they differ.

You can also consider cross-entropy as the expected quantity of information of events drawn from *P(x)* when you use *Q(x)* to encode them.

This is mathematically expressed as:



Let’s see how it works.



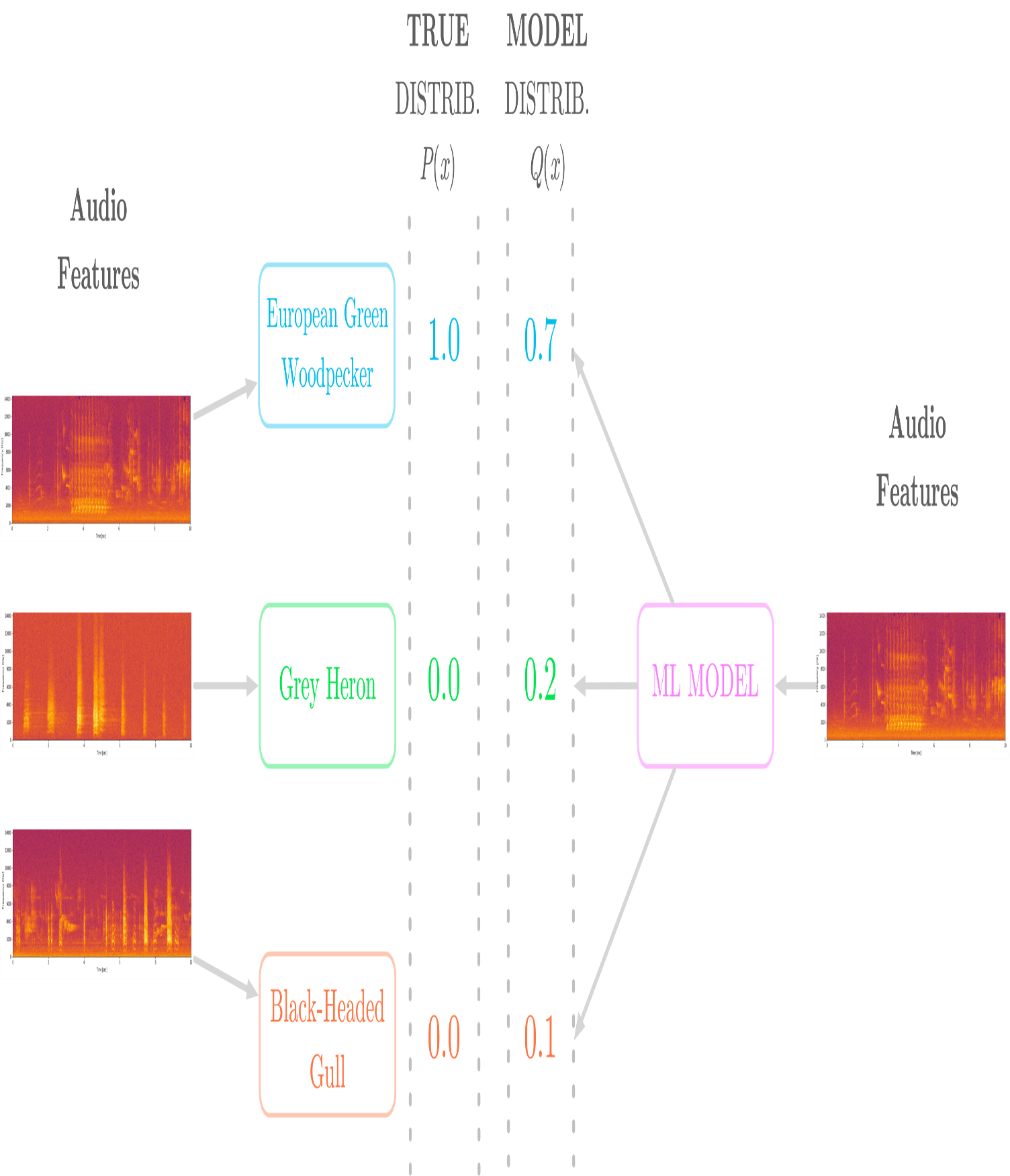
*Figure 4: Illustration of the cross-entropy as the Shannon information of Q(x) weighted according to the distribution of P(x).*(Image by author)

Figure 4 shows two different situations to illustrate the cross-entropy. On the left, you have two identical distributions *P(x)* (in blue) and *Q(x)* (in red). Their cross-entropy is equal to the entropy because the information of *Q(x)* is weighted according to the distribution of *P(x)*, which is similar to *Q(x)*.

However, in the right panel, *P(x)* and *Q(x)* are different. This results in a larger cross-entropy, because probabilities associated with a large quantity of information have a small weight, while probabilities associated with a small quantity of information have large weights.

The cross-entropy can’t be smaller than the entropy. Still in the right panel, you can see that, when the probability *Q(x)* is larger than *P(x)* (and thus associated with a lower amount of information), it is counterbalanced by the low weights (resulting in low weights and low information). These low weights will be compensated with larger weights in other probabilities from the distribution (resulting in large weights and large information).

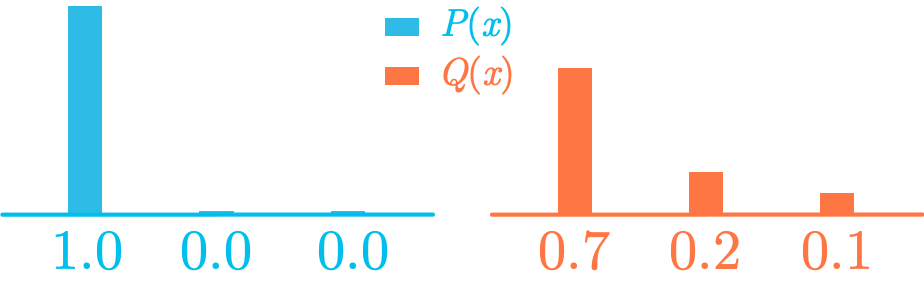
To summarize, the cross-entropy is minimum when the distributions are identical. As you’ll see in 0.1.4, this property makes the cross-entropy a useful metric. Note also that the result is different according to the distribution you choose as a reference: *H(P, Q) ≠ H(Q, P)*.



*Figure 5: Cross entropy can be used to compare the true distribution (probability of 1 for the correct class and 0 otherwise) and the distribution estimated by the model.*(Image by author)

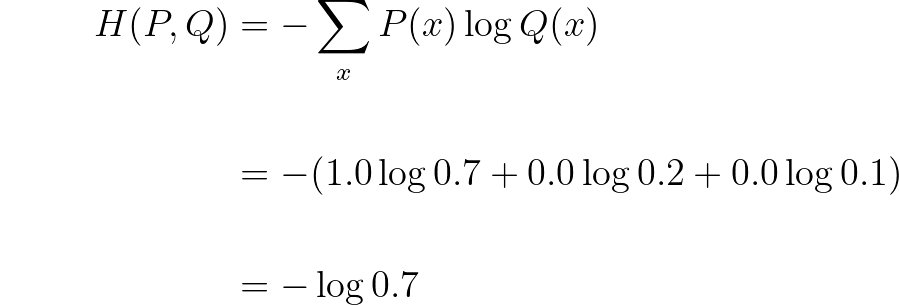
Say you want to build a model that classifies three different bird species from audio samples. As illustrated in Figure 5, the audio samples are converted in features (here spectrograms) and the possible classes (the three different birds) are *one-hot encoded*, that is, encoded as 1 for the correct class and 0 otherwise. Furthermore, the machine learning model outputs probabilities for each class.

Figure 5 shows that the true class corresponding to the sample you consider in this example is “European Green Woodpecker”. The model outputs a probability distribution and you’ll compute the cross-entropy loss associated with this estimation. Figure 6 shows both distributions.



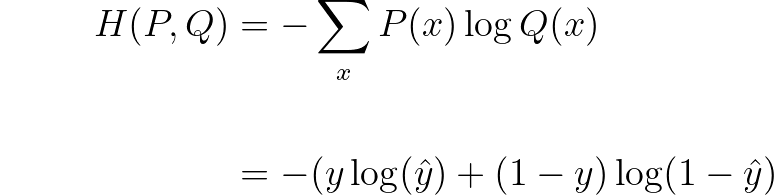
*Figure 6: Comparison of the true distribution P(x) and the estimated distribution Q(x).*(Image by author)

Let’s manually calculate the cross-entropy between these two distributions:



Since you one-hot encoded the classes (1 for the true class and 0 otherwise), the cross-entropy is simply the negative logarithm of the estimated probability for the true class.

In machine learning, the cross-entropy is widely used as a loss for binary classification: the log loss.



which is the formula of the log loss.

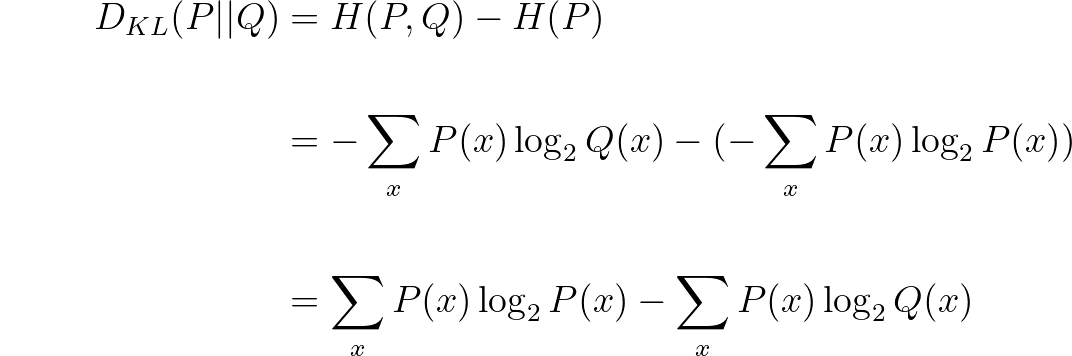
Kullback-Leibler Divergence (KL Divergence)

Intuitively, the KL divergence is the supplemental amount of information associated with the encoding of the distribution *Q(x)* compared to the true distribution *P(x)*. It tells you how different the two distributions are.

Mathematically, the KL divergence between two distributions *P(x)* and *Q(x)*, denoted as *D\_KL(P||Q)*, is expressed as the difference between the cross-entropy of *P(x)* and *Q(x)* and the entropy of *P(x)*:

Image for post

Replacing with the expressions of the cross-entropy and the entropy, you get:



The KL divergence is always non-negative. Since the entropy *H(P)* is identical to the cross-entropy *H(P, P)*, and because the smallest cross-entropy is between identical distributions ( *H(P, P)*), *H(P, Q)* is necessarily larger than *H(P)*. In addition, the KL divergence is equal to zero when the two distributions are identical.

*Originally published at*[*https://hadrienj.github.io/posts/Essential-Math-information\_theory/*](https://hadrienj.github.io/posts/Essential-Math-information_theory/)*.*

**This post is a sample of my book *Essential Math for Data Science*!**