Cross-entropy is commonly used in machine learning as a loss function.

## **What Is Cross-Entropy?**

Cross-entropy is a measure of the difference between two probability distributions for a given random variable or set of events.

You might recall that **information** quantifies the number of bits required to encode and transmit an event. Lower probability events have more information, higher probability events have less information.

**Entropy** is the number of bits required to transmit a randomly selected event from a probability distribution. A skewed distribution has a low entropy, whereas a distribution where events have equal probability has a larger entropy.

In information theory, we like to describe the “*surprise*” of an event. Low probability events are more surprising therefore have a larger amount of information. Whereas probability distributions where the events are equally likely are more surprising and have larger entropy.

* **Skewed Probability Distribution** (*unsurprising*): Low entropy.
* **Balanced Probability Distribution** (*surprising*): High entropy.

Entropy can be calculated for a random variable with a set of *x* in *X* discrete states discrete states and their probability *P(x)* as follows:

* H(X) = – sum x in X P(x) \* log(P(x))

## **How to Calculate Cross-Entropy**

In this section we will make the calculation of cross-entropy concrete with a small example.

### **Two Discrete Probability Distributions**

Consider a random variable with three discrete events as different colors: red, green, and blue.

We may have two different probability distributions for this variable; for example:

...

# define distributions

events = ['red', 'green', 'blue']

p = [0.10, 0.40, 0.50]

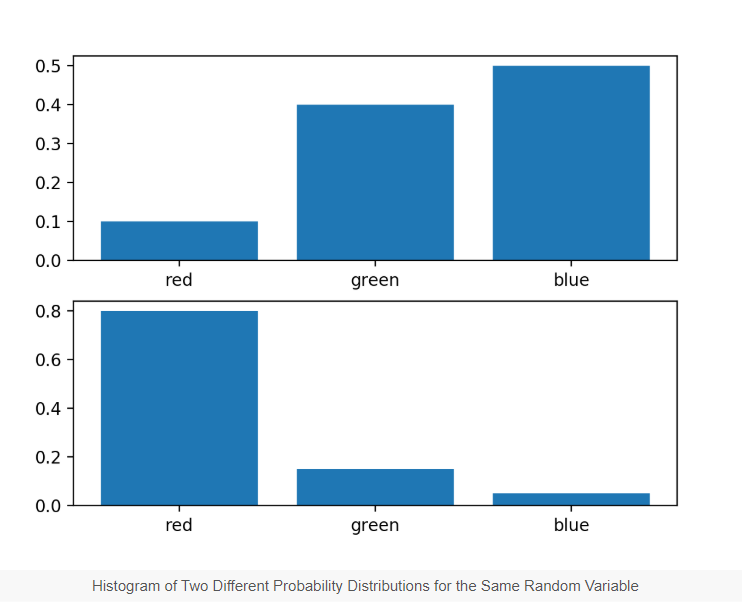
q = [0.80, 0.15, 0.05]

We can plot a bar chart of these probabilities to compare them directly as probability histograms.

The complete example is listed below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | # plot of distributions  from matplotlib import pyplot  # define distributions  events = ['red', 'green', 'blue']  p = [0.10, 0.40, 0.50]  q = [0.80, 0.15, 0.05]  print('P=%.3f Q=%.3f' % (sum(p), sum(q)))  # plot first distribution  pyplot.subplot(2,1,1)  pyplot.bar(events, p)  # plot second distribution  pyplot.subplot(2,1,2)  pyplot.bar(events, q)  # show the plot  pyplot.show() |

Running the example creates a histogram for each probability distribution, allowing the probabilities for each event to be directly compared.

We can see that indeed the distributions are different.

### **Calculate Cross-Entropy Between Distributions**

Next, we can develop a function to calculate the cross-entropy between the two distributions.We will use log base-2 to ensure the result has units in bits.

|  |  |
| --- | --- |
| 1  2  3 | # calculate cross entropydef cross\_entropy(p, q):  return -sum([p[i]\*log2(q[i]) for i in range(len(p))]) |

We can then use this function to calculate the cross-entropy of P from Q, as well as the reverse, Q from P.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | ...  # calculate cross entropy H(P, Q)  ce\_pq = cross\_entropy(p, q)  print('H(P, Q): %.3f bits' % ce\_pq)  # calculate cross entropy H(Q, P)  ce\_qp = cross\_entropy(q, p)  print('H(Q, P): %.3f bits' % ce\_qp) |

Tying this all together, the complete example is listed below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | # example of calculating cross entropy  from math import log2  # calculate cross entropy  def cross\_entropy(p, q):  return -sum([p[i]\*log2(q[i]) for i in range(len(p))])  # define data  p = [0.10, 0.40, 0.50]  q = [0.80, 0.15, 0.05]  # calculate cross entropy H(P, Q)  ce\_pq = cross\_entropy(p, q)  print('H(P, Q): %.3f bits' % ce\_pq)  # calculate cross entropy H(Q, P)  ce\_qp = cross\_entropy(q, p)  print('H(Q, P): %.3f bits' % ce\_qp) |

Running the example first calculates the cross-entropy of Q from P as just over 3 bits, then P from Q as just under 3 bits.

|  |  |
| --- | --- |
| 1  2 | H(P, Q): 3.288 bits  H(Q, P): 2.906 bits |

### **Calculate Cross-Entropy Between a Distribution and Itself**

If two probability distributions are the same, then the cross-entropy between them will be the entropy of the distribution.

We can demonstrate this by calculating the cross-entropy of P vs P and Q vs Q.

The complete example is listed below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # example of calculating cross entropy for identical distributions  from math import log2    # calculate cross entropy  def cross\_entropy(p, q):  return -sum([p[i]\*log2(q[i]) for i in range(len(p))])    # define data  p = [0.10, 0.40, 0.50]  q = [0.80, 0.15, 0.05]  # calculate cross entropy H(P, P)  ce\_pp = cross\_entropy(p, p)  print('H(P, P): %.3f bits' % ce\_pp)  # calculate cross entropy H(Q, Q)  ce\_qq = cross\_entropy(q, q)  print('H(Q, Q): %.3f bits' % ce\_qq) |

Running the example first calculates the cross-entropy of Q vs Q which is calculated as the entropy for Q, and P vs P which is calculated as the entropy for P.

|  |  |
| --- | --- |
| 1  2 | H(P, P): 1.361 bits  H(Q, Q): 0.884 bits |

### **Intuition for Cross-Entropy on Predicted Probabilities**

We can further develop the intuition for the cross-entropy for predicted class probabilities.

For example, given that an average cross-entropy loss of 0.0 is a perfect model, what do average cross-entropy values greater than zero mean exactly?

We can explore this question no. a binary classification problem where the class labels as 0 and 1. This is a discrete probability distribution with two events and a certain probability for one event and an impossible probability for the other event.

We can then calculate the cross entropy for different “*predicted*” probability distributions transitioning from a perfect match of the target distribution to the exact opposite probability distribution.

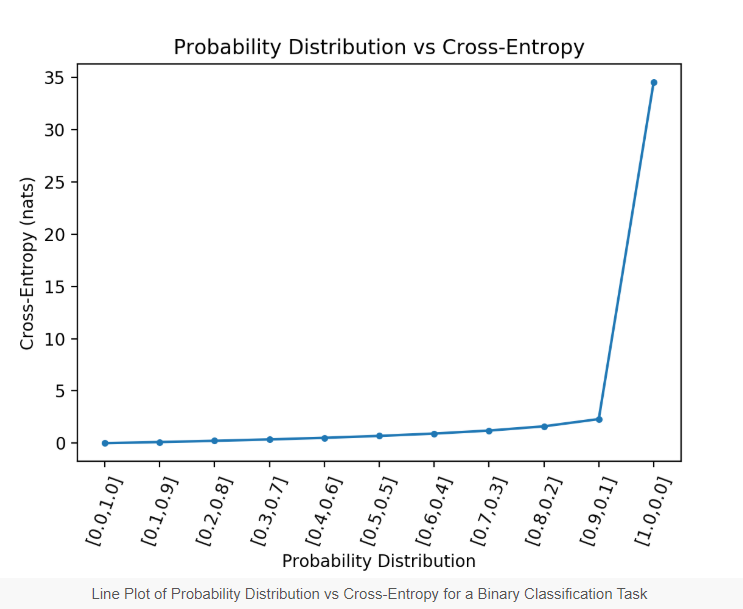
We would expect that as the predicted probability distribution diverges further from the target distribution that the cross-entropy calculated will increase.

The example below implements this and plots the cross-entropy result for the predicted probability distribution compared to the target of [0, 1] for two events as we would see for the cross-entropy in a binary classification task.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | # cross-entropy for predicted probability distribution vs label  from math import log  from matplotlib import pyplot  # calculate cross-entropy  def cross\_entropy(p, q, ets=1e-15):  return -sum([p[i]\*log(q[i]+ets) for i in range(len(p))])  # define the target distribution for two events  target = [0.0, 1.0]  # define probabilities for the first event  probs = [1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0]  # create probability distributions for the two events  dists = [[1.0 - p, p] for p in probs]  # calculate cross-entropy for each distribution  ents = [cross\_entropy(target, d) for d in dists]  # plot probability distribution vs cross-entropy  pyplot.plot([1-p for p in probs], ents, marker='.')  pyplot.title('Probability Distribution vs Cross-Entropy')  pyplot.xticks([1-p for p in probs], ['[%.1f,%.1f]'%(d[0],d[1]) for d in dists], rotation=70)  pyplot.subplots\_adjust(bottom=0.2)  pyplot.xlabel('Probability Distribution')  pyplot.ylabel('Cross-Entropy (nats)')  pyplot.show() |

Running the example calculates the cross-entropy score for each probability distribution then plots the results as a line plot.

We can see that as expected, cross-entropy starts at 0.0 (far left point) when the predicted probability distribution matches the target distribution, then steadily increases as the predicted probability distribution diverges.

We can also see a dramatic leap in cross-entropy when the predicted probability distribution is the exact opposite of the target distribution, that is, [1, 0] compared to the target of [0, 1].

We are not going to have a model that predicts the exact opposite probability distribution for all cases on a binary classification task.

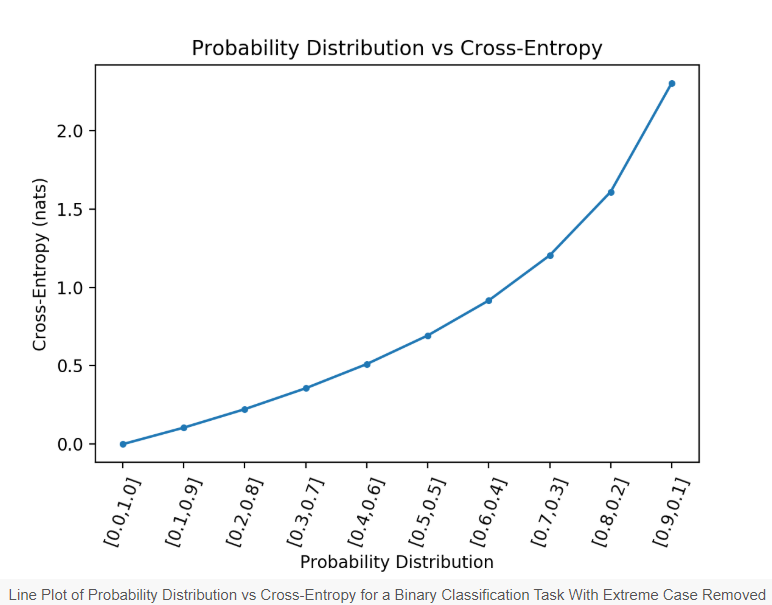
As such, we can remove this case and re-calculate the plot.

The updated version of the code is listed below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | # cross-entropy for predicted probability distribution vs label  from math import log  from matplotlib import pyplot  # calculate cross-entropy  def cross\_entropy(p, q, ets=1e-15):  return -sum([p[i]\*log(q[i]+ets) for i in range(len(p))])  # define the target distribution for two events  target = [0.0, 1.0]  # define probabilities for the first event  probs = [1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1]  # create probability distributions for the two events  dists = [[1.0 - p, p] for p in probs]  # calculate cross-entropy for each distribution  ents = [cross\_entropy(target, d) for d in dists]  # plot probability distribution vs cross-entropy  pyplot.plot([1-p for p in probs], ents, marker='.')  pyplot.title('Probability Distribution vs Cross-Entropy')  pyplot.xticks([1-p for p in probs], ['[%.1f,%.1f]'%(d[0],d[1]) for d in dists], rotation=70)  pyplot.subplots\_adjust(bottom=0.2)  pyplot.xlabel('Probability Distribution')  pyplot.ylabel('Cross-Entropy (nats)')  pyplot.show() |

Running the example gives a much better idea of the relationship between the divergence in probability distribution and the calculated cross-entropy.

We can see a super-linear relationship where the more the predicted probability distribution diverges from the target, the larger the increase in cross-entropy.



We can summarise these intuitions for the mean cross-entropy as follows:

* **Cross-Entropy = 0.00**: Perfect probabilities.
* **Cross-Entropy < 0.02**: Great probabilities.
* **Cross-Entropy < 0.05**: On the right track.
* **Cross-Entropy < 0.20**: Fine.
* **Cross-Entropy > 0.30**: Not great.
* **Cross-Entropy > 1.00**: Terrible.
* **Cross-Entropy > 2.00** Something is broken.