



Published in Towards Data Science



Daniel Godoy

Follow

Nov 21, 2018 · 9 min read · Listen



Save



# Understanding binary cross-entropy / log loss: a visual explanation

Photo by [G. Crescoli](#) on [Unsplash](#)

## Introduction

If you are training a **binary classifier**, chances are you are using **binary cross-entropy / log loss** as your loss function.



Have you ever thought about **what exactly does it mean** to use this loss function? The thing is, given the ease of use of today's libraries and frameworks, it is **very easy** to **overlook the true meaning** of the loss function used.

## Motivation

I was looking for a blog post that would explain the concepts behind **binary cross-entropy / log loss** in a **visually clear and concise manner**, so I could show it to my students at Data Science Retreat. Since I could not find any that would fit my purpose, I took the task of writing it myself :-)

## A Simple Classification Problem

Let's start with 10 random points:

$x = [-2.2, -1.4, -0.8, 0.2, 0.4, 0.8, 1.2, 2.2, 2.9, 4.6]$

This is our only **feature**:  $x$ .

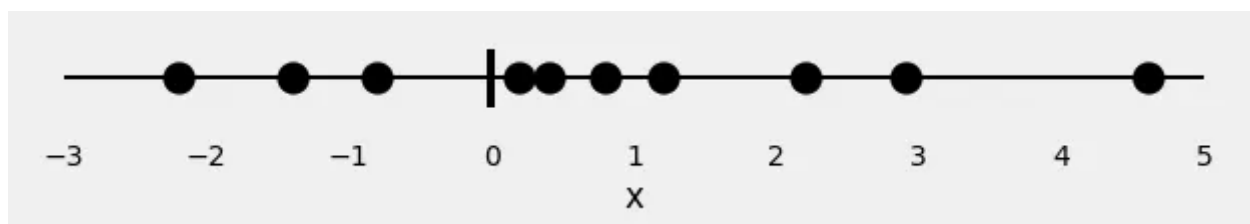


Figure 0: the feature

Now, let's assign some **colors** to our points: **red** and **green**. These are our **labels**.

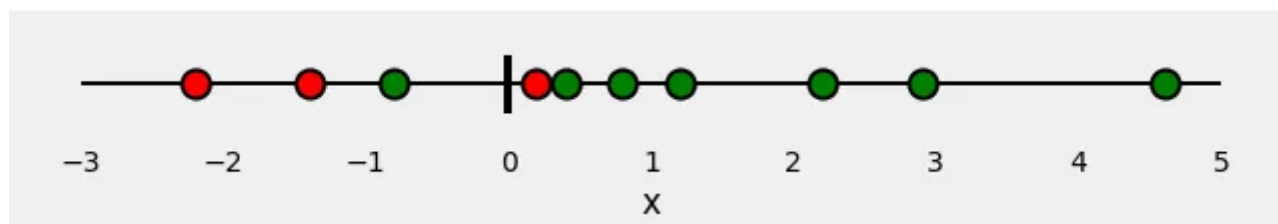


Figure 1: the data

So, our classification problem is quite straightforward: given our **feature**  $x$ , we need to predict its **label**: **red** or **green**.

Since this is a **binary classification**, we can also pose this problem as: “**is the point green**” or, even better, “**what is the probability of the point being green**”? Ideally,

**green points** would have a probability of **1.0** (of being green), while **red points** would have a probability of **0.0** (of being green).

In this setting, **green points** belong to the **positive class** (YES, they are green), while **red points** belong to the **negative class** (NO, they are not green).

If we **fit a model** to perform this classification, it will **predict a probability of being green** to each one of our points. Given what we know about the color of the points, how can we **evaluate** how good (or bad) are the predicted probabilities? This is the whole purpose of the **loss function**! It should return **high values for bad predictions** and **low values for good predictions**.

For a **binary classification** like our example, the **typical loss function** is the **binary cross-entropy / log loss**.

## Loss Function: Binary Cross-Entropy / Log Loss

If you look this **loss function** up, this is what you'll find:

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

where **y** is the **label** (1 for **green points** and 0 for **red points**) and **p(y)** is the predicted **probability of the point being green** for all **N points**.

Reading this formula, it tells you that, for each **green point** ( $y=1$ ), it adds  $\log(p(y))$  to the loss, that is, the **log probability of it being green**. Conversely, it adds  $\log(1-p(y))$ , that is, the **log probability of it being red**, for each **red point** ( $y=0$ ). Not necessarily difficult, sure, but no so intuitive too...

Besides, what does **entropy** have to do with all this? Why are we taking **log of probabilities** in the first place? These are valid questions and I hope to answer them on the “*Show me the math*” section below.

But, before going into more formulas, let me show you a **visual representation** of the formula above...

## Computing the Loss — the visual way

First, let's **split** the points according to their classes, **positive** or **negative**, like the figure below:

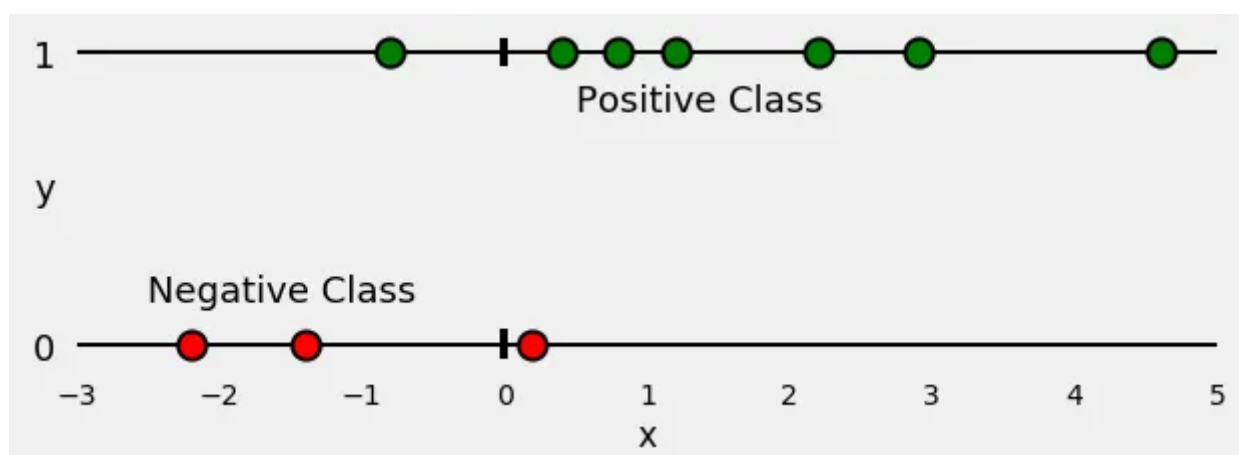


Figure 2: splitting the data!

Now, let's train a **Logistic Regression** to classify our points. The fitted regression is a *sigmoid curve* representing the **probability** of a point being green for any given  $x$ . It looks like this:

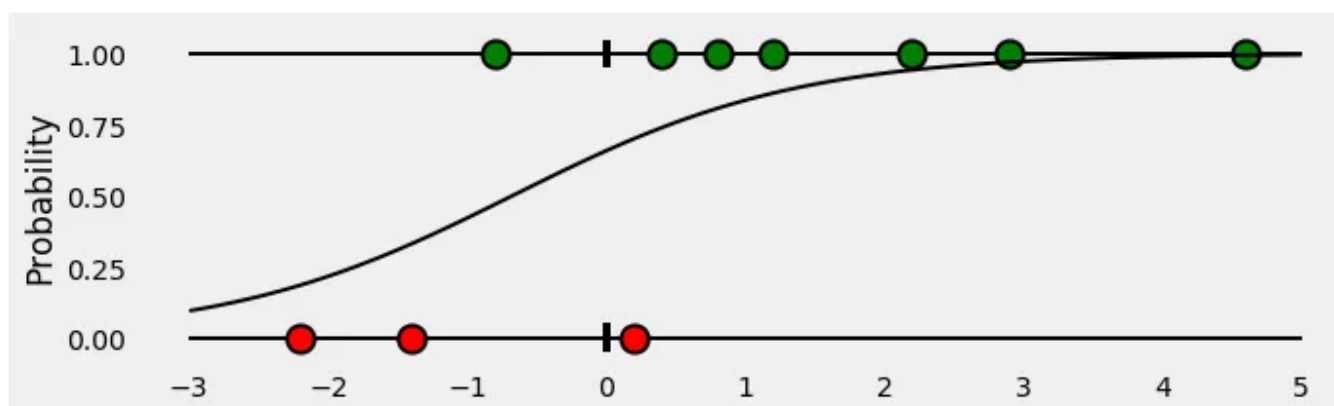


Figure 3: fitting a Logistic Regression

Then, for all points belonging to the **positive class** (green), what are the predicted **probabilities** given by our classifier? These are the **green bars under the sigmoid curve**, at the  $x$  coordinates corresponding to the points.

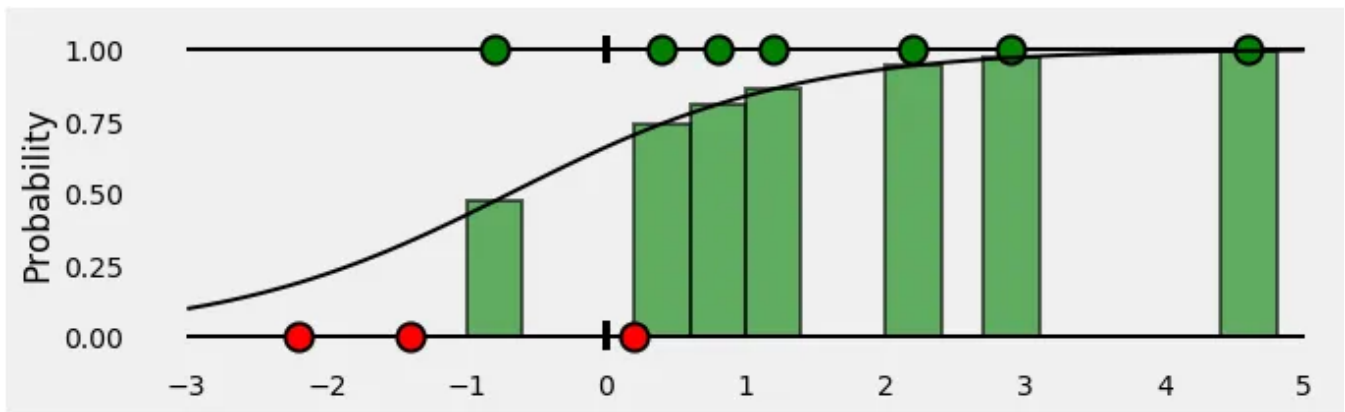


Figure 4: probabilities of classifying points in the POSITIVE class correctly

OK, so far, so good! What about the points in the **negative class**? Remember, the **green bars under the sigmoid curve** represent the probability of a given point being **green**. So, what is the probability of a given point being **red**? The **red bars ABOVE the sigmoid curve**, of course :-)

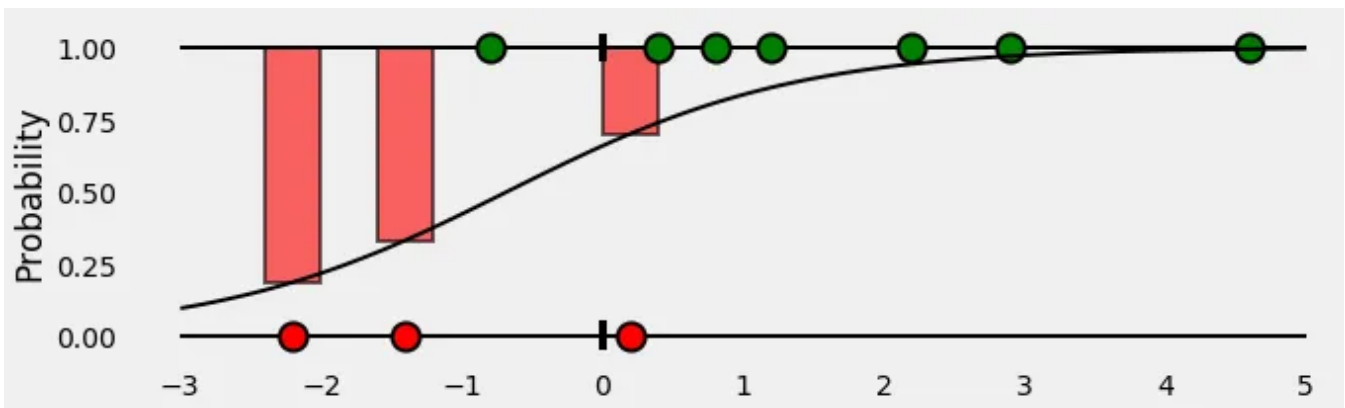


Figure 5: probabilities of classifying points in the NEGATIVE class correctly

Putting it all together, we end up with something like this:

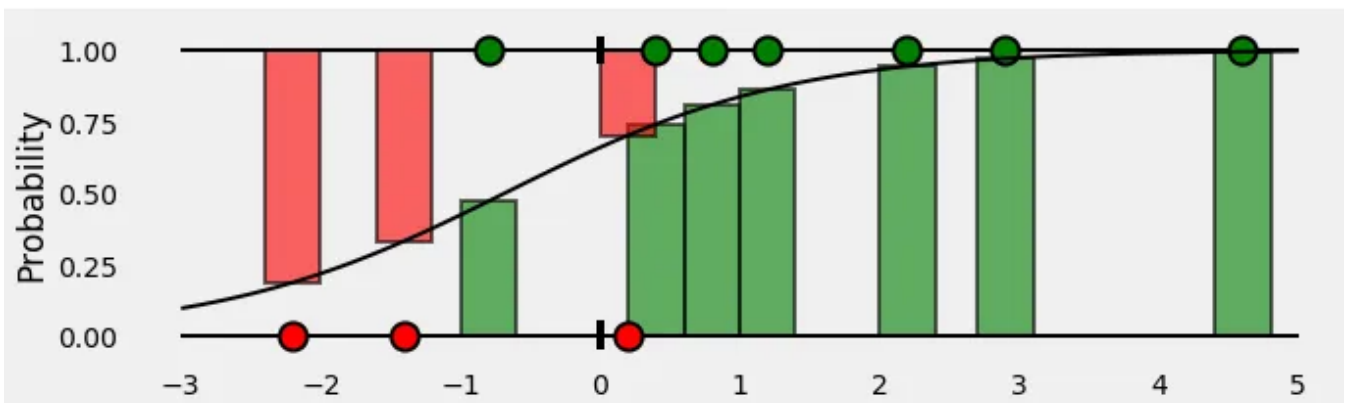


Figure 6: all probabilities put together!

The bars represent the **predicted probabilities** associated with the corresponding **true class** of each point!

OK, we have the predicted probabilities... time to **evaluate** them by computing the **binary cross-entropy / log loss**!

These **probabilities** are all we need, so, let's **get rid of the  $x$  axis** and bring the bars next to each other:

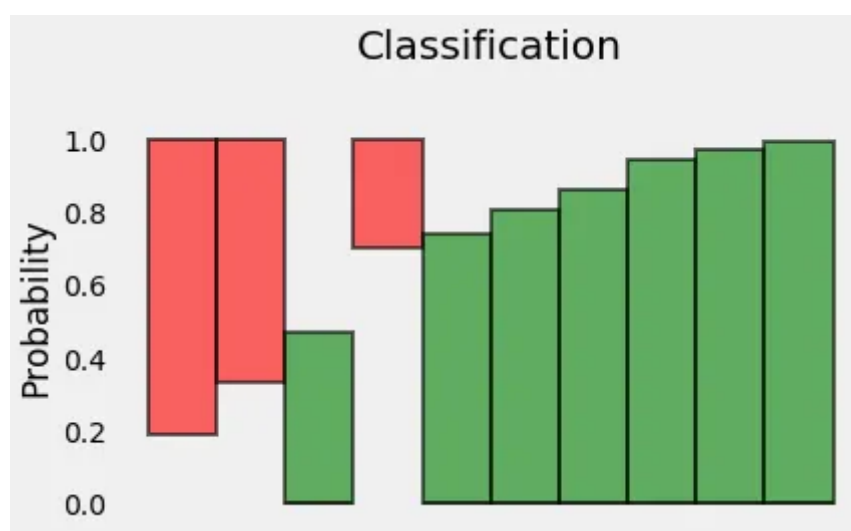


Figure 7: probabilities of all points

Well, the *hanging bars* don't make much sense anymore, so let's **reposition them**:

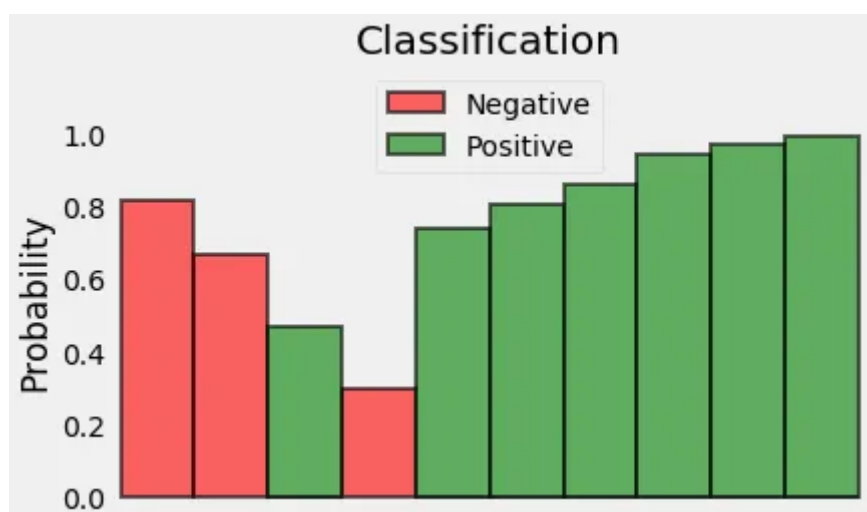


Figure 8: probabilities of all points — much better :-)

Since we're trying to compute a **loss**, we need to penalize bad predictions, right? If the **probability** associated with the **true class** is 1.0, we need its **loss** to be **zero**. Conversely, if that **probability** is low, say, 0.01, we need its **loss** to be **HUGE**!



It turns out, taking the **(negative) log of the probability** suits us well enough for this purpose (since the log of values between 0.0 and 1.0 is negative, we take the negative log to obtain a positive value for the loss).

Actually, the reason we use **log** for this comes from the definition of **cross-entropy**, please check the “**Show me the math**” section below for more details.

The plot below gives us a clear picture —as the **predicted probability** of the **true class** gets closer to zero, the **loss increases exponentially**:

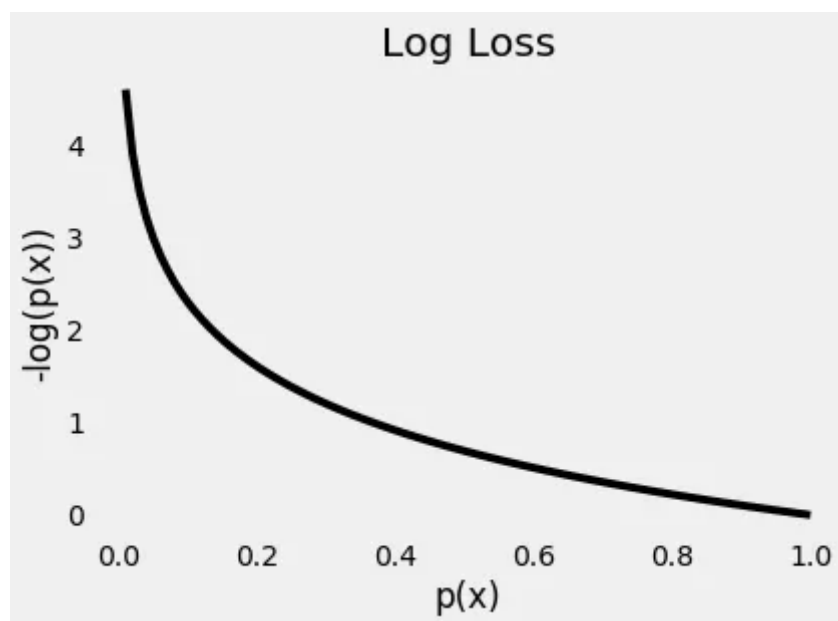


Figure 9: Log Loss for different probabilities

Fair enough! Let's take the **(negative) log of the probabilities** — these are the corresponding **losses** of each and every point.

Finally, we compute the **mean** of all these losses.

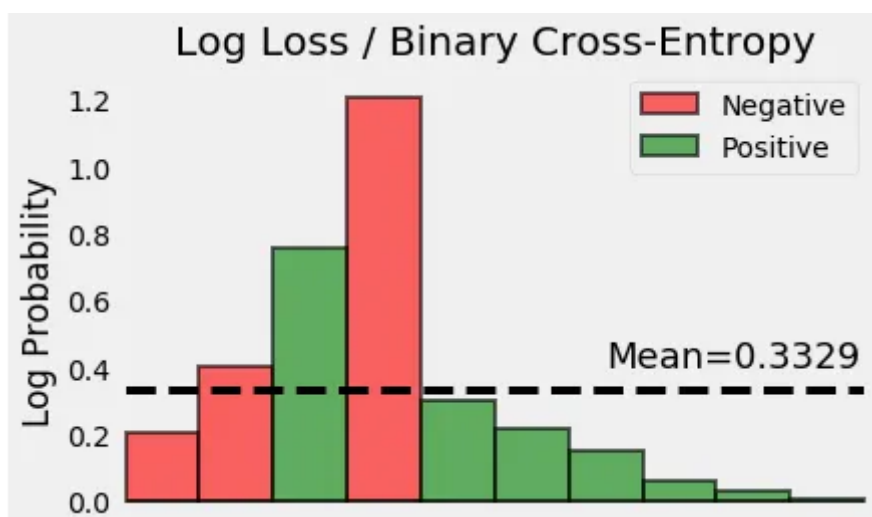


Figure 10: finally, the loss!

**Voilà!** We have successfully computed the **binary cross-entropy / log loss** of this toy example. It is **0.3329!**

### **Show me the code**

If you want to **double check the value** we found, just **run the code** below and see for yourself :-)

### **Show me the math (really?!)**

Jokes aside, this post is **not** intended to be very mathematically inclined... but for those of you, my readers, looking to understand the role of **entropy**, **logarithms** in



all this, here we go :-)

*If you want to go deeper into **information theory**, including all these concepts — entropy, cross-entropy and much, much more — check **Chris Olah's post** out, it is incredibly detailed!*

## Distribution

Let's start with the distribution of our points. Since  $y$  represents the **classes** of our points (we have 3 **red points** and 7 **green points**), this is what its distribution, let's call it  $q(y)$ , looks like:



Figure 11:  $q(y)$ , the distribution of our points

## Entropy

**Entropy** is a **measure of the uncertainty** associated with a given distribution  $q(y)$ .

What if **all our points were green**? What would be the **uncertainty of that distribution**? **ZERO**, right? After all, there would be **no doubt about the color** of a point: it is **always green**! So, **entropy is zero**!

On the other hand, what if we knew exactly **half of the points were green** and the **other half, red**? That's the **worst case scenario**, right? We would have absolutely **no edge on guessing the color** of a point: it is totally **random**! For that case, entropy is given by the formula below (*we have two classes (colors)— red or green — hence, 2*):

$$H(q) = -\sum_{y \in \mathcal{Y}} q(y) \log_2 q(y)$$

Open in app ↗

Sign up

Sign In



For every other case in between, we can compute the **entropy of a distribution**, like our  $q(y)$ , using the formula below, where  $C$  is the number of classes:

$$H(q) = - \sum_{c=1}^C q(y_c) \cdot \log(q(y_c))$$

Entropy

So, if we *know* the **true distribution** of a random variable, we can compute its **entropy**. But, if that's the case, *why bother training a classifier* in the first place? After all, we **KNOW** the true distribution...

But, what if we **DON'T**? Can we try to **approximate the true distribution** with some **other distribution**, say,  $p(y)$ ? Sure we can! :-)

### Cross-Entropy

Let's assume our **points follow this other distribution**  $p(y)$ . But we know they are **actually coming from the true (unknown) distribution**  $q(y)$ , right?



If we compute **entropy** like this, we are actually computing the **cross-entropy** between both distributions:

$$H_p(q) = - \sum_{c=1}^C q(y_c) \cdot \log(p(y_c))$$

Cross-Entropy

If we, somewhat miraculously, *match  $p(y)$  to  $q(y)$  perfectly*, the computed values for both **cross-entropy** and **entropy** will match as well.

Since this is likely never happening, **cross-entropy will have a BIGGER value than the entropy** computed on the true distribution.

 9.5K |  55 |

$$H_p(q) - H(q) \geq 0$$

Cross-Entropy minus Entropy

It turns out, this difference between **cross-entropy** and **entropy** has a name...

### Kullback-Leibler Divergence

The **Kullback-Leibler Divergence**, or “*KL Divergence*” for short, is a measure of **dissimilarity** between two distributions:

$$D_{KL}(q||p) = H_p(q) - H(q) = \sum_{c=1}^C q(y_c) \cdot [\log(q(y_c)) - \log(p(y_c))]$$

KL Divergence

This means that, the **closer p(y) gets to q(y)**, the **lower the divergence** and, consequently, the **cross-entropy**, will be.

So, we need to find a good **p(y)** to use... but, this is what our **classifier** should do, isn't it?! **And indeed it does!** It looks for the **best possible p(y)**, which is the one that **minimizes the cross-entropy**.

### Loss Function

During its training, the **classifier** uses each of the **N points** in its training set to compute the **cross-entropy** loss, effectively **fitting the distribution p(y)**! Since the probability of each point is 1/N, cross-entropy is given by:

$$q(y_i) = \frac{1}{N} \Rightarrow H_p(q) = -\frac{1}{N} \sum_{i=1}^N \log(p(y_i))$$

Cross-Entropy —point by point

Remember Figures 6 to 10 above? We need to compute the **cross-entropy** on top of the *probabilities associated with the true class* of each point. It means using the **green bars** for the points in the **positive class (y=1)** and the **red hanging bars** for the points in the **negative class (y=0)** or, mathematically speaking:

$$\begin{aligned} y_i = 1 &\Rightarrow \log(p(y_i)) \\ y_i = 0 &\Rightarrow \log(1 - p(y_i)) \end{aligned}$$

Mathematical expression corresponding to Figure 10 :-)

The final step is to compute the **average** of all points in both classes, **positive** and **negative**:

$$H_p(q) = -\frac{1}{(N_{pos} + N_{neg})} \left[ \sum_{i=1}^{N_{pos}} \log(p(y_i)) + \sum_{i=1}^{N_{neg}} \log(1 - p(y_i)) \right]$$

Binary Cross-Entropy — computed over positive and negative classes

Finally, with a little bit of manipulation, we can take any point, **either from the positive or negative classes**, under the same formula:

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

Binary Cross-Entropy — the usual formula

Voilà! We got back to the **original formula** for **binary cross-entropy / log loss** :-)

## Final Thoughts

I truly hope this post was able **shine some new light** on a concept that is quite often taken for granted, that of **binary cross-entropy** as **loss function**. Moreover, I also hope it served to show you a little bit how **Machine Learning** and **Information Theory** are linked together.

*Update July 10th, 2022: I've just published a YouTube video with an animated version of this post — check it out!*

Understanding Binary Cross-Entropy / Log Loss in 5 minutes: a visual ex...



[Machine Learning](#)[Classification](#)[Entropy](#)[Loss Function](#)[Towards Data Science](#)

---

## Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.



Get this newsletter

[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app

