DEEP LEARNING

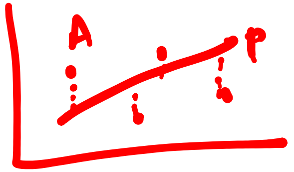
Trainer : Dr. Darshan Ingle

In-Depth Loss Functions

1. Mean Squared Error

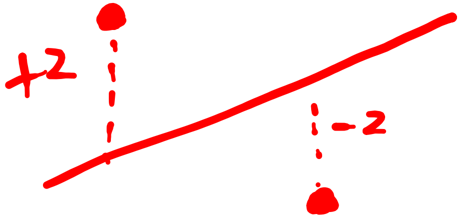
- from a probabilistic perspective.
- helps prepare us for cross-entropy loss

Note: Error = Cost = Loss = Objective

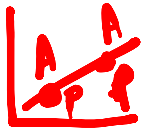


$$MSE = \frac{1}{N} \cdot \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

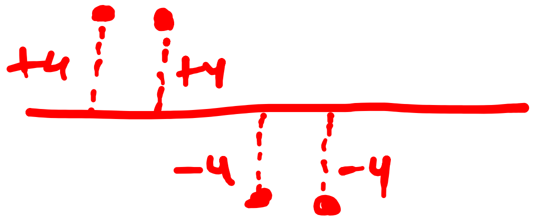
$$ME = \frac{\overset{=0}{-2+2}}{2} \quad \frac{\cancel{2}}{2} = 0$$



MSE- Why is it squared?



1



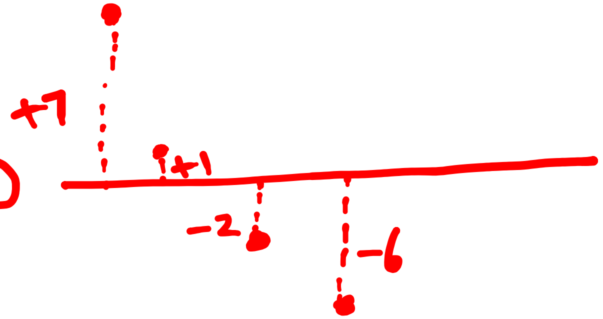
$$\text{Mean error} = \frac{(+4)+(+4)+(-4)+(-4)}{4} = \frac{0}{4} = 0$$

$[-4, +4]$

$$\text{Mean Absolute error} = \frac{|+4|+|+4|+|-4|+|-4|}{4} = \frac{4+4+4+4}{4} = 4$$

:)

2



$$\text{MAE} = \frac{|+7|+|+1|+|-2|+|-6|}{4} = \frac{7+1+2+6}{4} = \frac{16}{4} = 4$$

$[-6, +7]$

3

$$\text{Mean Squared error} = \frac{(4)^2+(4)^2+(-4)^2+(-4)^2}{4} = \frac{64}{4} = 16$$

$$= \frac{(7)^2+(1)^2+(-2)^2+(-6)^2}{4} = \frac{49+1+4+36}{4} = \frac{90}{4} = 22.5$$

MSE- Footnote 1

MSE → It very difficult to map to original values.

∴ we go for Root Mean Square error.

$$\boxed{RMSE = \sqrt{MSE}}$$

$RMSE_1 = \sqrt{16} = 4$ [-4, +4]

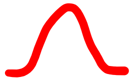
$RMSE_2 = \sqrt{22.5} \approx 4.74$ [-6, +7]

MSE- Footnote 2

2. Maximum Likelihood Estimation (MLE)

- https://www.youtube.com/watch?v=XepXtl9YKwc

MLE Formula

eg: We model the heights of the students in our class as a Gaussian distribution (Bell Curve) 

$$PDF = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot exp\left(-\frac{1}{2} \cdot \frac{(x-\mu)^2}{\sigma^2}\right)$$

MLE vs MSE Conclusion

Maximizing the likelihood is same as minimizing the squared error.
i.e. $(x_i - \mu)$

$$l = -\frac{1}{N} \cdot \sum_{i=1}^{N} (x_i - \mu)^2$$

$$MSE = +\frac{1}{N} \cdot \sum_{i=1}^{N} (x_i - \mu)^2$$

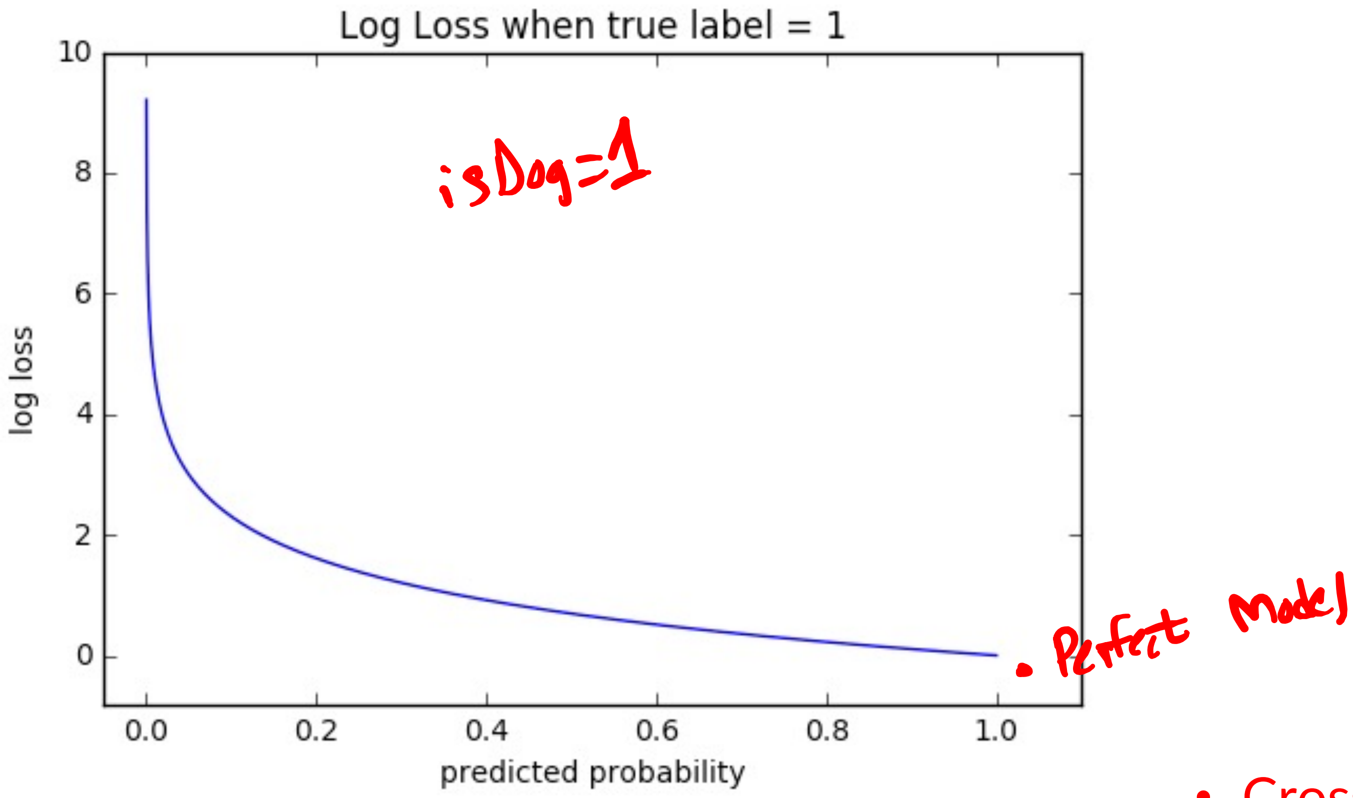
3. Cross-Entropy / Log Loss

- 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.

Cross-Entropy / Log Loss

Cross-Entropy / Log Loss

[0 - 1]



- Cross-entropy and log loss are slightly different depending on context, but in machine learning when calculating error rates between 0 and 1 they resolve to the same thing.

Cross-Entropy / Log Loss

Math

o/p

- In binary classification, where the number of classes M equals 2, cross-entropy can be calculated as:

$$-(y \log(p) + (1-y)\log(1-p))$$

- If M>2 (i.e. multiclass classification), we calculate a separate loss for each class label per observation and sum the result.

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

- **Note:**
- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

Cross-Entropy / Log Loss

Code:

```
def CrossEntropy(yHat, y):
    if y == 1:
      return -log(yHat)
    else:
      return -log(1 - yHat)
```

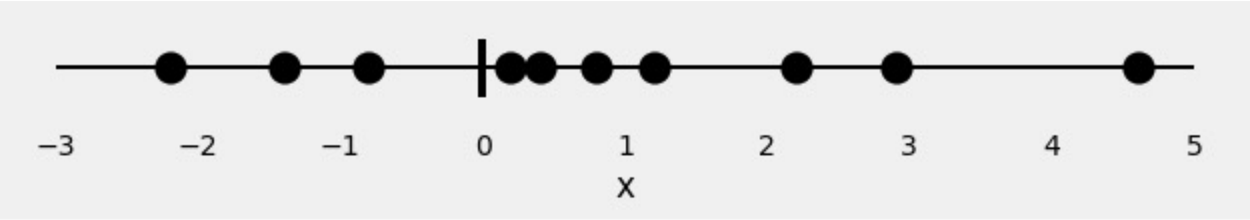
Cross-Entropy / Log Loss

A Simple Classification Problem

- Let's start with 10 random points:

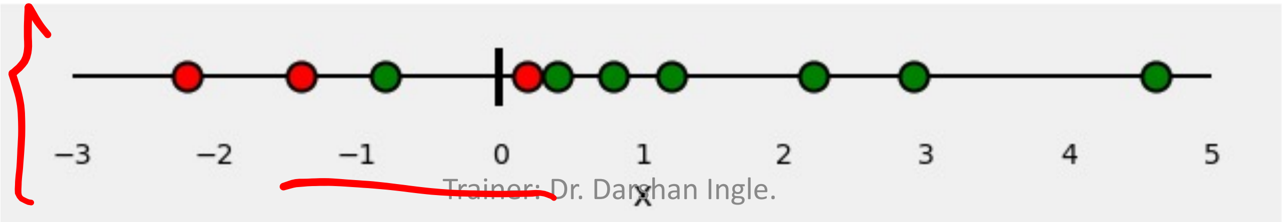
x = [-2.2, -1.4, -0.8, 0.2, 0.4, 0.8, 1.2, 2.2, 2.9, 4.6]

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

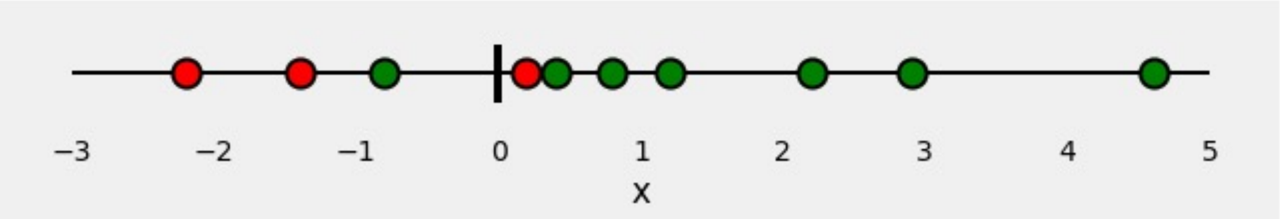


Now, let's assign some **colors** to our points: **red** and **green**. These are our **labels**.

Green =1.0
Red=0.0



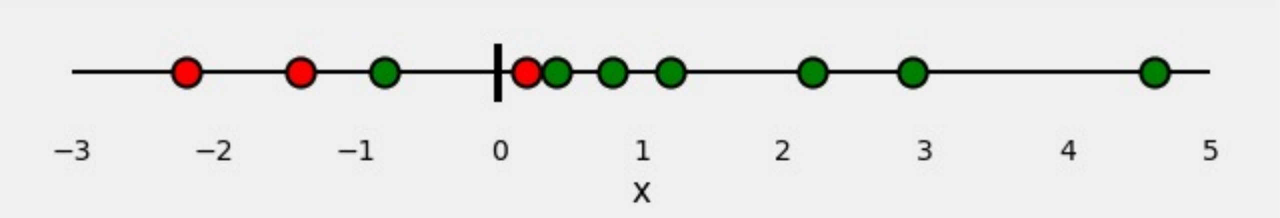
Cross-Entropy / Log Loss



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).

Cross-Entropy / Log Loss



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**.

Cross-Entropy / Log Loss

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

$$- \sum p \cdot \log p$$

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

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.

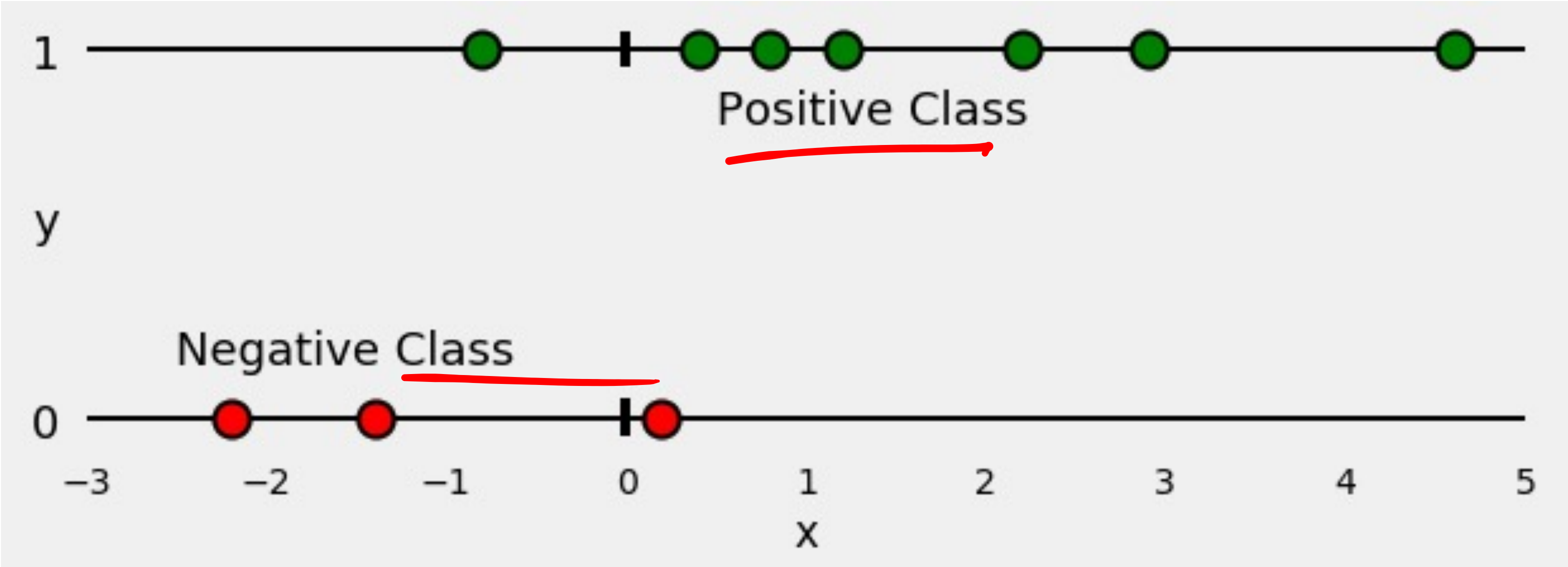
Green, y=1

Red, y=0

Cross-Entropy / Log Loss

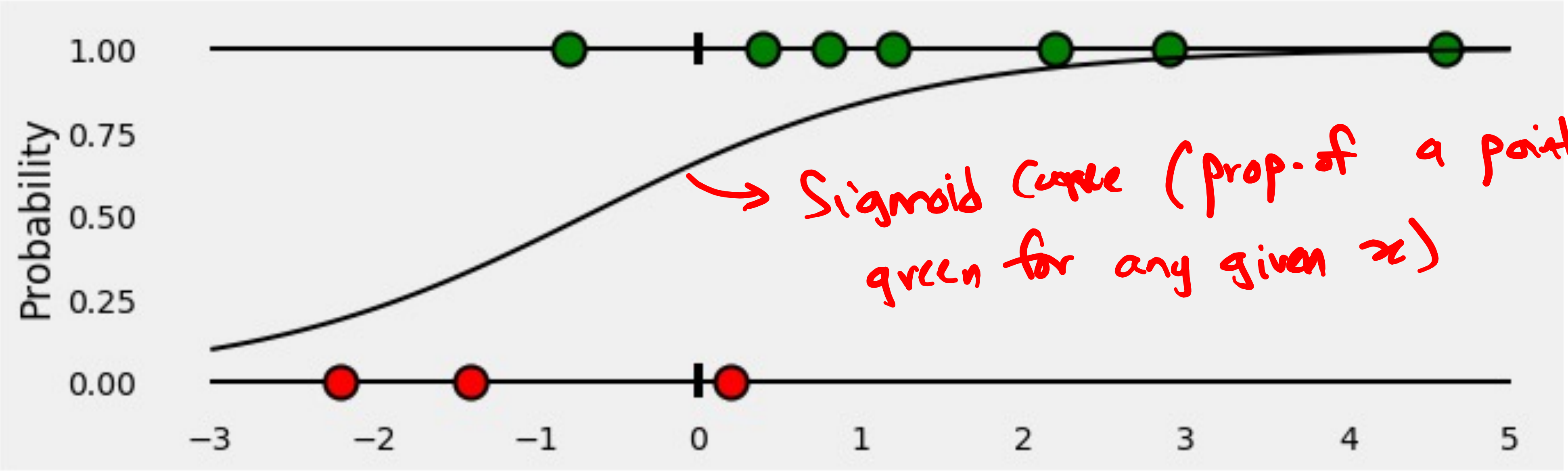
- **Computing the Loss — the visual way**

Split the points acc. to classes (pos or neg).



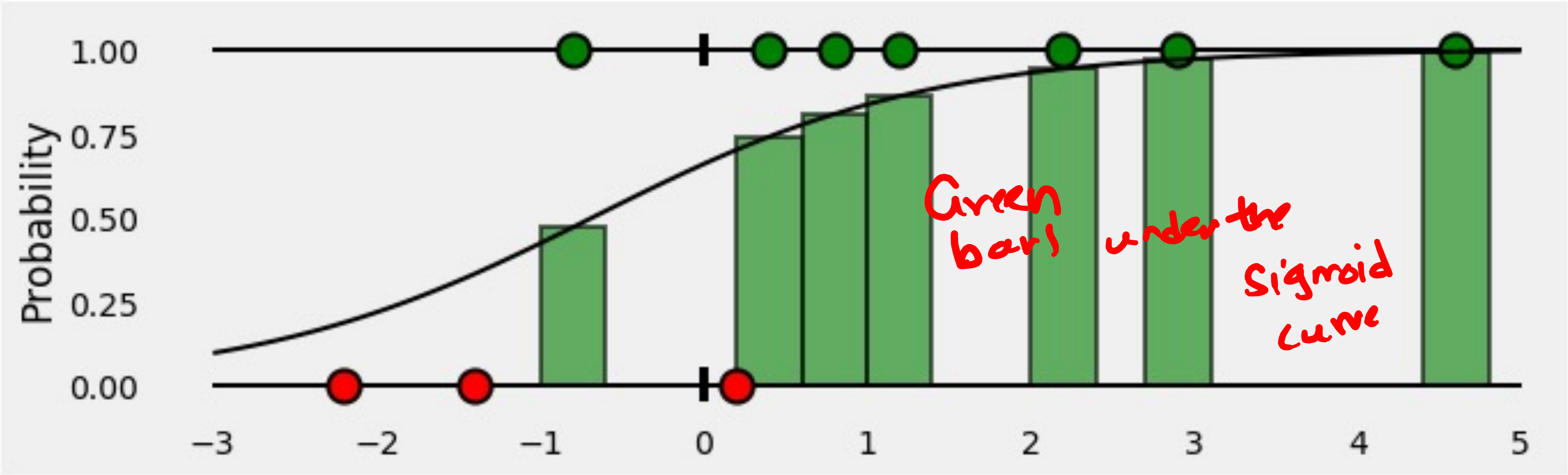
Cross-Entropy / Log Loss

Train a Log. Reg.



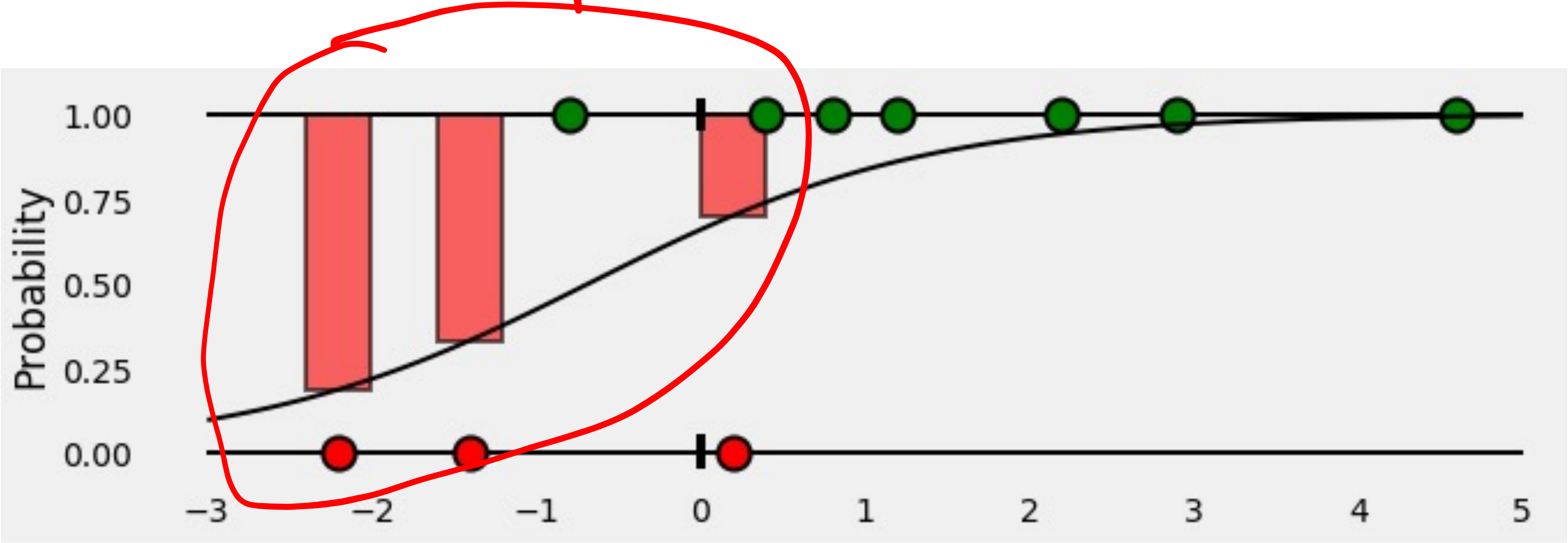
→ Sigmoid curve (prob. of a point being green for any given x)

Cross-Entropy / Log Loss



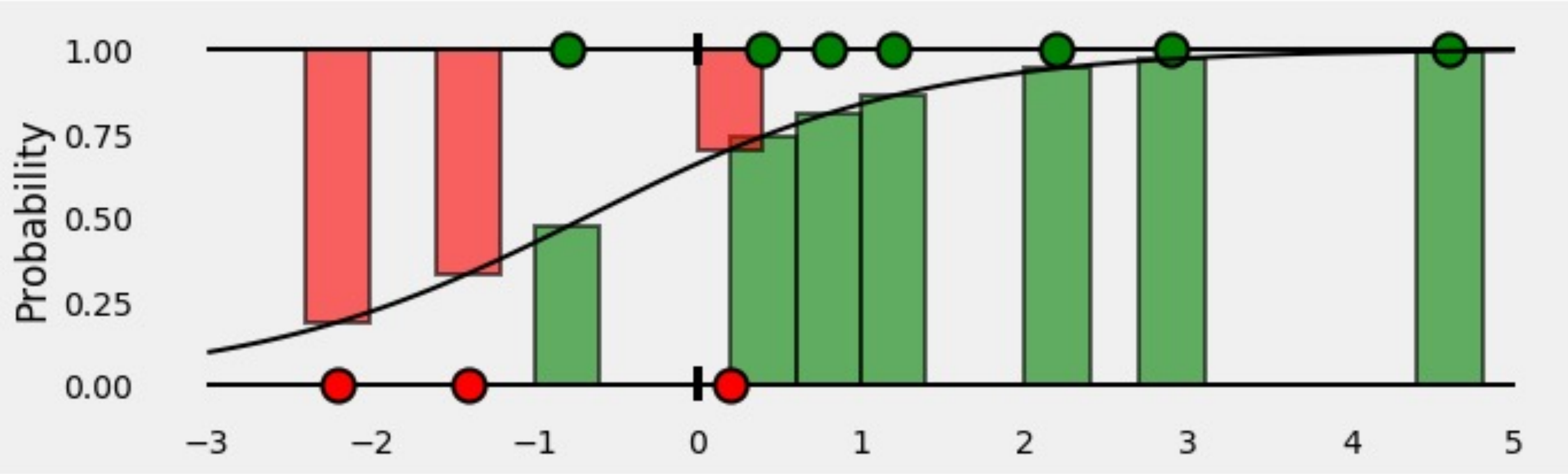
Cross-Entropy / Log Loss

Negative class



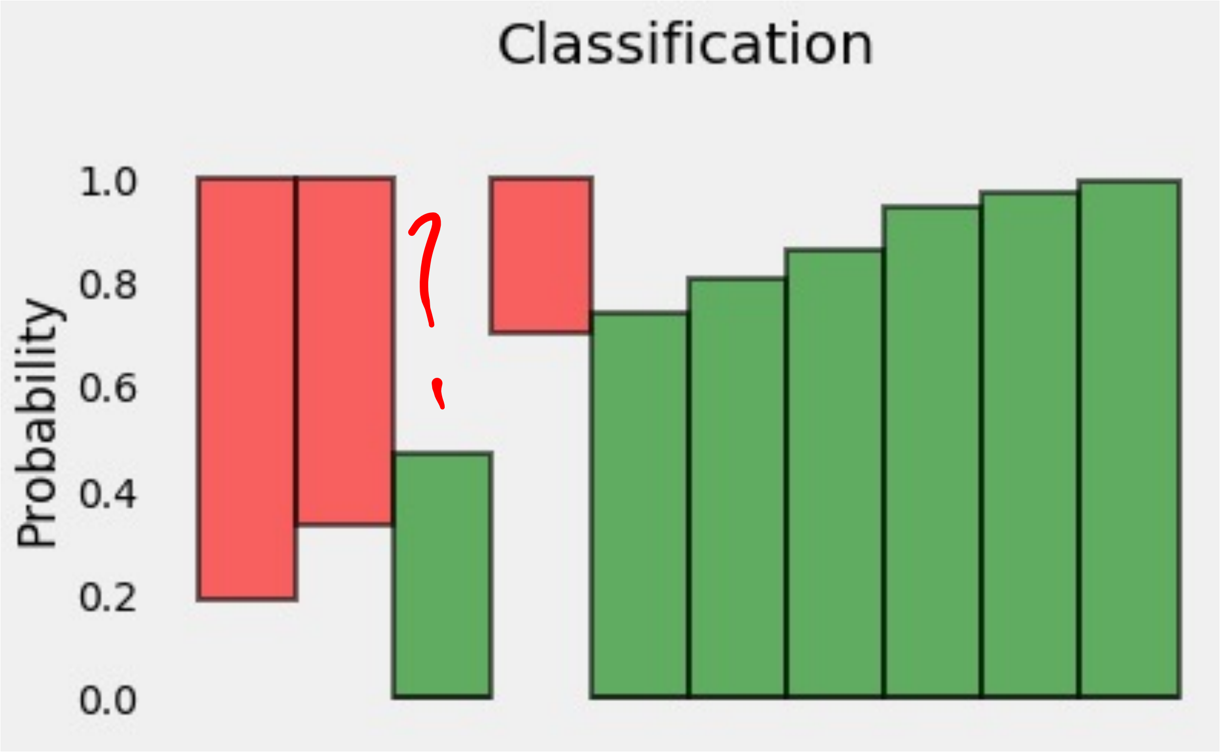
Cross-Entropy / Log Loss

Put it altogether



Cross-Entropy / Log Loss

We only & only need probabilities



Cross-Entropy / Log Loss

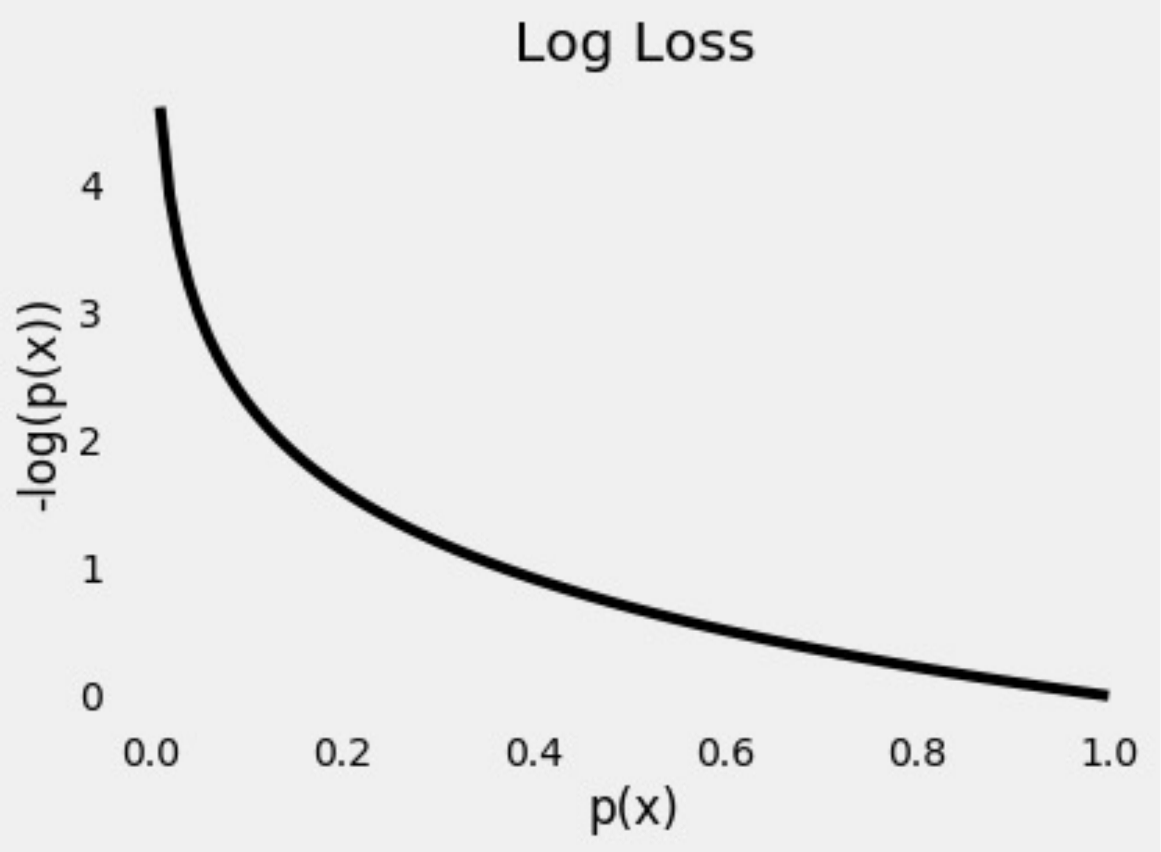
Cross-Entropy / Log Loss

- 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*).

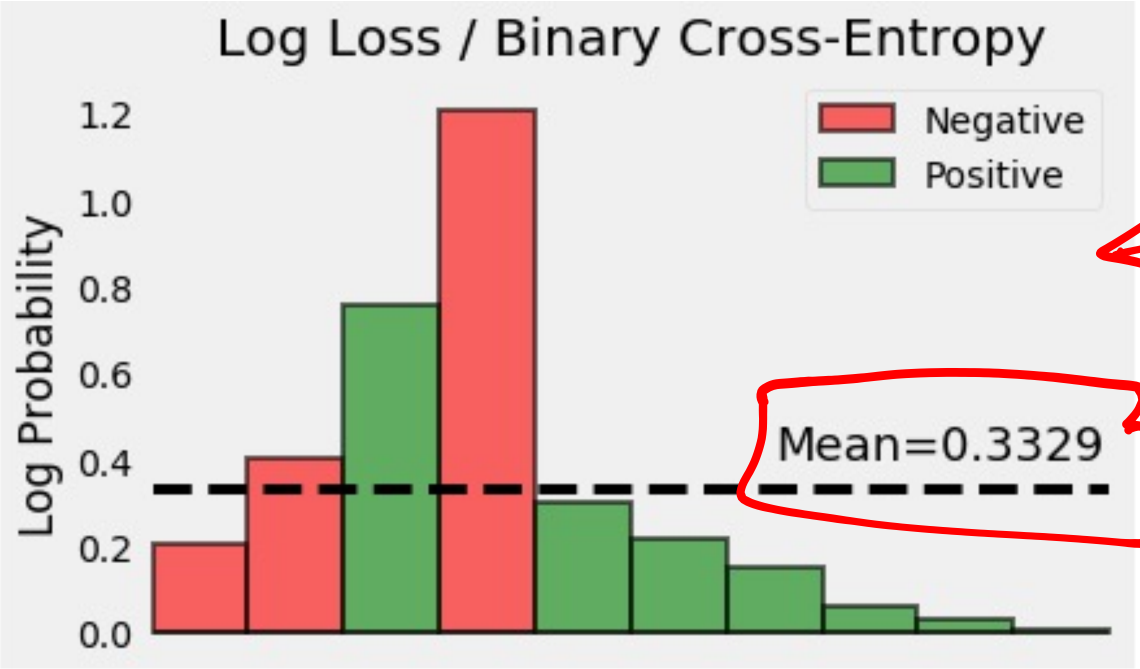
Cross-Entropy / Log Loss

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



Cross-Entropy / Log Loss

Take -ve log of prob.



Binary Cross entropy | log loss