Scientific Machine Learning 09
Logistic Regression

Donghyun Ko

January 4, 2026

What you will learn in this posting. Why not just use a line for classification? We'll see why plain linear regression breaks—no valid probabilities, unstable 0.5 cutoffs, and fake label order—and how logistic regression fixes this by piping a linear score through a sigmoid so $p(x) \in (0, 1)$ and the decision boundary is simply $z = 0$. You'll learn to read coefficients on the *log-odds* (odds ratios $\exp(\beta_j)$, not linear changes in p), train the model via MLE $\leftrightarrow$ binary cross-entropy with the neat gradient $\nabla C = \frac{1}{m} X^\top (p - y)$, and tame complete separation with regularization. We'll also sketch the multiclass jump with softmax and wrap up with practical tips on threshold choice, class imbalance, and probability calibration—so your outputs are not just labels, but trustworthy probabilities.

Contents

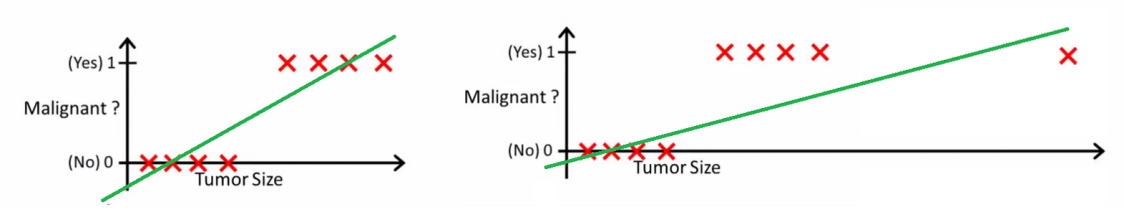
1 Motivation: "Why don't we just use a regression?"

In supervised learning, *regression* predicts a continuous target (e.g., rainfall, price, temperature), while *classification* forecasts a discrete label. Yet, there is a useful bridge: a regression model can also *predict probabilities*. If we can predict $P(Y = 1 \mid X = x)$ reliably, a simple *decision rule* (e.g., predict class 1 whenever the probability exceeds a threshold like 0.5) turns those probabilities into a classifier. This is exactly the lens we will use in a logistic regression.

Why plain linear regression fails for classification? Imagine a situation where we label the two classes by $\{0, 1\}$ and fit a line $h(x)$ with MSE:

1. **No probabilistic meaning.** $h(x)$ is an unconstrained real number; it may fall below 0 or above 1 as seen by the green line in Fig. 1, so it can't be interpreted as $P(Y = 1 \mid X = x)$.
2. **No stable threshold.** Because $h(x)$ is a linear interpolation, adding a single point can shift the best-fit line so much ("It is very sensitive to outlier!") that a fixed cut (e.g. $h(x) > 0.5$) flips many decisions. The "tumor-size" example in Fig. 1 shows this fragility: one extra sample makes the old 0.5 rule invalid.
3. **Multi-class mismatch;** When there are more than 2 classes, linear regression makes you give each class a number like $0, 1, 2, \ldots$ and then fit a line to those numbers. But these numbers are *made up*—they create a fake order and distance:
   - If the classes are cat, dog, bird and we code them as $0, 1, 2$, the model treats "the "dog" as halfway between "cat" and "bird" because 1 is between 0 and 2. In reality, there is no such middle class.
   - A prediction of 1.7 would be pulled towards code 2 ("bird") just because 1.7 is numerically closer to 2 than to 1, not because the features say "bird".

   So, linear regression imposes a *fake geometry* (ordering and distances) that the classes do not have. What we actually want is a *separate probability for each class* that sums to 1. That is exactly what the softmax regression gives: $a_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$.



2 What is a logistic regression?

In supervised learning, *regression* predicts a continuous target (rainfall, price, temperature). Importantly, a regression model can also *predict probabilities*—e.g., "how likely is this image

a cat?" Once we have $P(Y = 1 \mid X = x)$, a simple *decision rule* turns it into a classifier:

$$\hat{y} = \begin{cases} 1, & P(Y = 1 \mid X = x) \geq \tau, \\ 0, & P(Y = 1 \mid X = x) < \tau, \end{cases} \qquad \text{with a common choice } \tau = 0.5$$

Logistic regression is therefore a *regression algorithm that predicts probabilities*, but in ML, it is commonly introduced as a *classification method* because we use those probabilities to make class decisions.

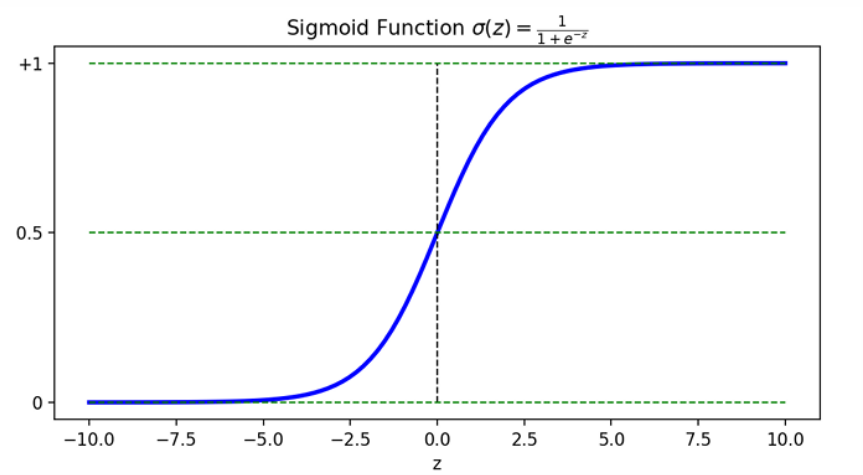
Key modeling trick of logistic regression: squeeze a linear score into $(0, 1)$. Instead of predicting with a bare line or hyperplane as in regression, we first compute a linear score

$$z \;=\; \beta_0 + \beta^\top x,$$

and then *wrap* it through the logistic (sigmoid) function s.t:

$$\sigma(z) \;=\; \frac{1}{1 + e^{-z}} \quad \Rightarrow \quad p(x) \equiv P(Y = 1 \mid X = x) = \sigma(\beta_0 + \beta^\top x) \in (0, 1)$$

The logistic curve is S–shaped (See Fig. 2): it maps any real z smoothly into $(0, 1)$, changes fastest near $z = 0$, and saturates as $z \to \pm\infty$. Historically, it was used to model population growth (rapid rise, then carrying-capacity saturation), which is why the name "logistic" was used." In the logistic regression, computing a linear score matches exactly a linear regression (i.e., this is exactly the same as for MLP with linear activation function), and then we pass the result through 'logistic (or, sigmoid)' activation function where the name of "logistic regression" comes from.



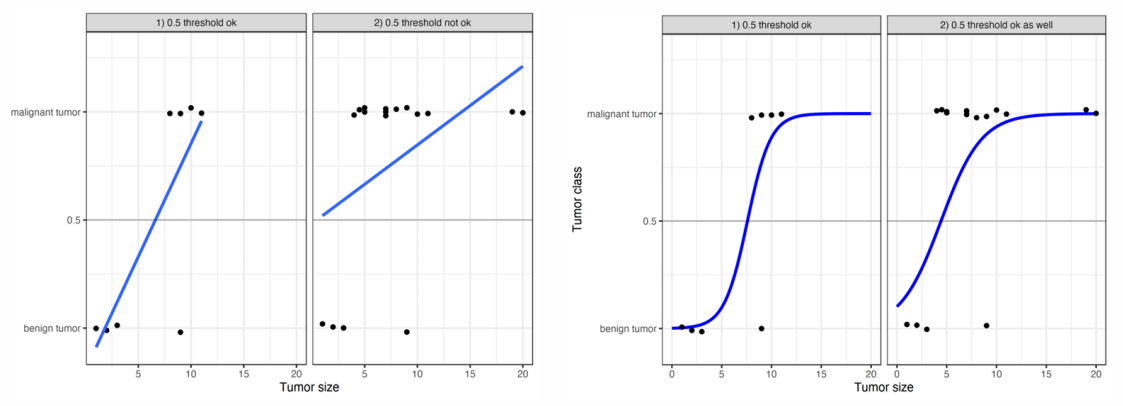
Decision boundary and why 0.5 is natural. A decision boundary is a threshold that we use to categorize the probabilities of logistic regression into discrete classes. A decision boundary with the threshold of $\tau = 0.5$ could take the form:

$$p(x) \;\equiv\; P(Y = 1 \mid X = x), \quad \text{and} \quad \hat{Y}(x) \;=\; \begin{cases} 0, & p(x) < \frac{1}{2}, \\ 1, & p(x) \geq \frac{1}{2} \end{cases}$$

Using threshold $\tau = 0.5$,

$$p(x) \geq 0.5 \iff \frac{1}{1+e^{-z}} \geq \frac{1}{2} \iff z = \beta_0 + \beta^\top x \geq 0$$

Hence, the decision boundary is the hyperplane $\{\beta_0+\beta^\top x = 0\}$ in the feature space. Compared with linear regression, the S–curve yields a *robust* 0.5 crossing—adding a few data points, even if outliers, barely moves the boundary, as seen in the tumor example (See Fig. 2).



Left: linear fit is brittle; Right: logistic curve keeps the same boundary

Why is it called *Regression*. We use logistic *regression* to make class decisions, but the model itself *regresses* a continuous target—the **log-odds**. By defining:

$$p(x) \equiv P(Y = 1 \mid X = x), \qquad \text{odds}(x) = \frac{p(x)}{1-p(x)}, \qquad \text{logit}(x) = \log \frac{p(x)}{1-p(x)}.$$

the logistic regression models log-odds in the manner of linear regression:

$$\boxed{\log \frac{p(x)}{1-p(x)} = \beta_0 + \beta^\top x} \iff p(x) = \sigma(\beta_0 + \beta^\top x) = \frac{1}{1+e^{-(\beta_0+\beta^\top x)}}$$

Interpreting the results can be done this way: (1) *Linearity in log-odds:* increasing x_j by 1 adds β_j to the log-odds. (2) *Odds ratios:* odds multiply by e^{β_j}; e.g., $\beta_j = 0.7 \Rightarrow \times e^{0.7} \approx \times 2$. (3) *Probabilities after regression:* we predict log-odds linearly, then map to $p(x) \in (0, 1)$ via the inverse-logit.

Preview: multi-class extension. For K classes, we compute class-wise logits $z_j = W_{j:}^\top x + b_j$ and apply *softmax* s.t:

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}, \qquad \sum_j a_j = 1,$$

so each class gets its own probability, and no fake ordering is imposed.

Takeaway. Logistic regression answers exactly what a classifier needs: (i) valid probabilities in $(0, 1)$, (ii) a clear and stable boundary via a fixed threshold, (iii) a principled path to multiclass via softmax.

3 Interpretation: "How to read logistic regression?"

Let $p(x) = P(Y = 1 \mid X = x)$ and $z = \beta_0 + \beta^\top x$ so that $p(x) = \sigma(z) = \frac{1}{1+e^{-z}}$. Define the **odds** and **log-odds (logit)** such that:

$$\text{odds}(x) = \frac{p(x)}{1-p(x)}, \qquad \text{logit}(x) = \log \frac{p(x)}{1-p(x)}$$

With these definitions, logistic regression is a linear model for the log-odds:

$$\log \frac{p(x)}{1-p(x)} = \log \frac{P(Y=1)}{1-P(Y=1)} = \log \frac{P(Y=1)}{P(Y=0)} = \beta_0 + \beta_1 X_1 + \cdots + \beta_d X_d$$

$$\underbrace{\log \frac{p(x)}{1-p(x)}}_{\text{log-odds}} = \beta_0 + \sum_{k=1}^{d} \beta_k X_k \quad \xRightarrow{\text{"Exponentiate both sides"}} \quad \frac{p(x)}{1-p(x)} = \exp\Big(\beta_0 + \sum_{k=1}^{d} \beta_k X_k\Big)$$

$$\Downarrow \qquad (\text{since } p(x) = P(Y=1 \mid X=x),\ 1-p(x) = P(Y=0 \mid X=x))$$

$$\boxed{\frac{P(Y=1 \mid X=x)}{P(Y=0 \mid X=x)} = \text{odds}(x) = \exp\Big(\beta_0 + \beta_1 X_1 + \cdots + \beta_j X_j + \cdots + \beta_d X_d\Big)}$$

Reading a coefficient β_j

Odds form. One unit increase in X_j multiplies the odds by $\exp(\beta_j)$. A change in a feature X_j by 1 unit changes the odds ratio by a factor of $\exp(\beta_j)$:

$$\frac{\text{odds}(X_j+1)}{\text{odds}(X_j)} = \frac{\exp(\beta_0 + \cdots + \beta_j(X_j+1) + \cdots)}{\exp(\beta_0 + \cdots + \beta_j X_j + \cdots)} = \exp(\beta_j)$$

Log-odds form. The log-odds increases by exactly β_j (assuming others fixed). A change in X_j by 1 unit increases the log odds by the value of corresponding weight β_j.

Numerical example. If $\beta_j = 0.82$, then $\exp(0.82) \approx 2.27$: increasing X_j by one unit *multiplies* the odds of $Y = 1$ by about 2.27 (holding other features fixed).

Categorical example. For a binary dummy $D \in \{0, 1\}$ with coefficient β_D, switching $D : 0 \to 1$ multiplies the odds by $\exp(\beta_D)$. For instance, $\beta_D = -0.12$ gives $\exp(-0.12) \approx 0.89$, i.e., odds are 11% lower when $D = 1$.

Intercept β_0

When all features are 0 (at the chosen coding/centering), the model's baseline log-odds is β_0, so the baseline probability is

$$p_0 = \frac{1}{1+e^{-\beta_0}}$$

(If features are standardized/centered, β_0 is the probability at the mean feature vector.)

From odds ratios to probability changes

Be cautious that *coefficients do not change the probability $p(x)$ linearly.* They act linearly on the *log-odds*; after the sigmoid transformation, effects on $p(x)$ are nonlinear and depend on the current baseline probability. For example, because $p(x) = \sigma(z)$ with $\sigma'(z) = p(1-p)$, a small change of ΔX_j changes the probability by $\Delta p \approx p(1-p)\,\beta_j\,\Delta X_j$. The multiplier $p(1-p)$ depends on the *current* probability level p: the same β_j yields the largest probability change around $p \approx 0.5$ and very small changes near $p \approx 0$ or 1. So, the same β_j moves p much around 0.5, but hardly at all when p is already very low or very high.

Example of analysis

If the coefficient is $\beta_j = 0.7$, then increasing X_j by 1 multiplies the odds by $e^{0.7} \approx 2.01$, so new odds $= 2 \times e^{0.7} \approx 4.03$ (≈ 4).

$$\text{odds} = \frac{p}{1-p} = 2 \implies p = \frac{2}{1+2} = \frac{2}{3} \approx 0.667$$

Why show "odds = 2 ⇒ p = 2/3?" Odds are often what the model manipulates, but readers think in *probabilities*. The identity $\boxed{o = \dfrac{p}{1-p} \iff p = \dfrac{o}{1+o}}$ lets us translate any reported odds (or odds ratio) back to a probability. So when the baseline odds are $o = 2$, the baseline probability is $p = \frac{2}{1+2} = \frac{2}{3} \approx 0.667$.

How much does p move when odds are multiplied? If increasing X_j by 1 multiplies the odds by the factor $r = \exp(\beta_j)$ (e.g., $r = e^{0.7} \approx 2.01$), then

$$p' = \frac{ro}{1+ro}, \qquad \Delta p \;=\; p' - p \;=\; \frac{(r-1)\,o}{(1+o)(1+ro)}$$

Hence, the same coefficient has a *baseline-dependent* effect: Δp is largest around $p \approx 0.5$ (i.e., $o \approx 1$) and much smaller when p is already near 0 or 1. For example, with $o = 2$ ($p \approx 0.667$) and $r = e^{0.7} \approx 2.01$,

$$p' = \frac{2.01 \times 2}{1 + 2.01 \times 2} \approx 0.803, \quad \Delta p \approx 0.136$$

(If $o = 1$ then p would move from 0.5 to ≈ 0.667; if $o = 4$ then $0.8 \to 0.889$)

Practical example. We use the logistic regression model to predict cervical cancer based on some risk factors. We will interpret each coefficient through its odds ratio $\exp(\beta)$, keeping the other features fixed (constant).

- **Num. of diagnosed STDs:** Odds-ratio(OR) = 2.27 ⇒ an increase in the number of diagnosed STDs multiplies the odds of cancer by ~ 2.27 (risk increases).
- **Smokes (y/n):** OR = 1.30 ⇒ smokers have about 30% higher odds than non-smokers.
- **Num. of pregnancies:** OR = 1.04 per unit ⇒ small positive effect on the odds.

- **Hormonal contraceptives (y/n):** OR $= 0.89 \Rightarrow$ users have $\sim 11\%$ lower odds than non-users (potential protective effect).
- **Intrauterine device (y/n):** OR $= 1.86 \Rightarrow$ about $1.86\times$ higher odds.
- **Intercept:** OR $= 0.05 \Rightarrow$ very low odds when all predictors are at reference values.
- **Uncertainty note:** Standard errors reflect estimation noise; report p-values or 95% CIs for odds ratios (e.g., $\exp(\beta \pm 1.96\,\mathrm{SE})$).
- **Probability caveat:** Coefficients are linear on *log-odds*; probability changes are nonlinear and depend on the current baseline p.

	Weight	Odds ratio	Std. Error
Intercept	-2.91	0.05	0.32
Hormonal contraceptives y/n	-0.12	0.89	0.30
Smokes y/n	0.26	1.30	0.37
Num. of pregnancies	0.04	1.04	0.10
Num. of diagnosed STDs	0.82	2.27	0.33
Intrauterine device y/n	0.62	1.86	0.40

4 Training: from MLE to cross-entropy

The aim of training the logistic regression model is to find the best weights/coefficients $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_d)^\top$ of a logistic regression so that the model assigns *high probability* to the labels actually observed in the data. In other words, among all possible $\boldsymbol{\beta}$, we prefer the one that makes our dataset look most plausible. The weights of the logistic regression can be estimated from the training data. This is done using maximum likelihood estimation (MLE).

Model and notation. For any example $(\boldsymbol{x}, y)$, we compute a linear score and pass it through a sigmoid:

$$p(\boldsymbol{x}) \equiv P(Y = 1 \mid \boldsymbol{x}) = \sigma(z) = \frac{1}{1 + e^{-z}}, \quad \text{where} \quad z = \boldsymbol{\beta}^\top \boldsymbol{x} = \beta_0 + \sum_{j=1}^{d} \beta_j x_j$$

In other words,

$$P(Y = 1 \mid \boldsymbol{X} = \boldsymbol{x}) = \sigma(\boldsymbol{\beta}^\top \boldsymbol{x}), \qquad P(Y = 0 \mid \boldsymbol{X} = \boldsymbol{x}) = 1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{x})$$

For notational convenience, we fold the *intercept* β_0 into the dot product by augmenting the feature vector with a leading 1 s.t:

$$\tilde{\boldsymbol{x}} \;=\; (1,\, x_1, \ldots, x_d)^\top, \qquad \tilde{\boldsymbol{\beta}} \;=\; (\beta_0, \beta_1, \ldots, \beta_d)^\top,$$

so that

$$z = \boldsymbol{\beta}^\top \boldsymbol{x} \;=\; \beta_0 + \sum_{j=1}^{d} \beta_j x_j \;=\; \sum_{j=0}^{d} \beta_j x_j \;=\; \boldsymbol{\beta}^\top \boldsymbol{x}, \quad \text{with } x_0 \equiv 1$$

Here, $Y = 1$ is called the default label, and σ is the logistic (also called 'sigmoid') function. The linear part $\boldsymbol{\beta}^\top \boldsymbol{x}$ is easy to fit; the sigmoid merely converts that real number into a valid probability in $(0, 1)$. The negative class uses $P(Y = 0 \mid \boldsymbol{x}) = 1 - p(\boldsymbol{x})$.

Training data. We observe m independent pairs $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$ with $y^{(i)} \in \{0, 1\}$ and

$$Y^{(i)} \sim \text{Bernoulli}\Big(p^{(i)}\Big), \quad \text{where} \quad p^{(i)} = \sigma\Big(\boldsymbol{\beta}^\top \boldsymbol{x}^{(i)}\Big)$$

"Bernoulli" says each label is a single coin flip with success chance $p^{(i)}$ predicted by our model.

Likelihood: how plausible is the whole dataset? For a *single* example $(\boldsymbol{x}, y)$, the Bernoulli pmf with model probability $p(\boldsymbol{x}) = \sigma(\boldsymbol{\beta}^\top \boldsymbol{x})$ is:

$$P(Y = y \mid \boldsymbol{X} = \boldsymbol{x}) = \Big(\sigma(\boldsymbol{\beta}^\top \boldsymbol{x})\Big)^{y} \Big[1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{x})\Big]^{1-y}, \qquad y \in \{0, 1\}.$$

If $y = 1$, the term reduces to $\sigma(\boldsymbol{\beta}^\top \boldsymbol{x})$, and if $y = 0$, it reduces to $1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{x})$. With '$m$' independent training points $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$, the likelihood of the whole dataset is:

$$P(\mathcal{D} \mid \boldsymbol{\beta}) \;=\; \mathcal{L}(\boldsymbol{\beta}) \;=\; \prod_{i=1}^{m} P\Big(Y = y^{(i)} \mid \boldsymbol{X} = \boldsymbol{x}^{(i)}\Big) \;=\; \prod_{i=1}^{m} \Big(\sigma(\boldsymbol{\beta}^\top \boldsymbol{x}^{(i)})\Big)^{y^{(i)}} \Big[1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{x}^{(i)})\Big]^{1-y^{(i)}}$$

Due to independence assumption $\Rightarrow$ multiply per-example probabilities. Parameters $\boldsymbol{\beta}$ that assign larger probability to the actually observed labels yield a larger $\mathcal{L}(\boldsymbol{\beta})$.

Log-likelihood: products $\rightarrow$ sums for numerical safety. The factors in $\mathcal{L}(\boldsymbol{\beta})$ are probabilities (often tiny). Multiplying many tiny numbers can cause *arithmetic underflow* and is also inconvenient for calculus. We therefore take the logarithm of the likelihood and sum the per-example terms in order to prevent it:

$$\begin{aligned}
\ell(\boldsymbol{\beta}) &\equiv \log \mathcal{L}(\boldsymbol{\beta}) \\
&= \sum_{i=1}^{m} \Big[y^{(i)} \log \sigma(\boldsymbol{\beta}^\top \boldsymbol{x}^{(i)}) + \Big(1 - y^{(i)}\Big) \log\Big(1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{x}^{(i)})\Big) \Big] \\
&= \sum_{i=1}^{m} \Big[y^{(i)} \log p^{(i)} + \Big(1 - y^{(i)}\Big) \log\Big(1 - p^{(i)}\Big) \Big], \quad \text{with } p^{(i)} = \sigma(\boldsymbol{\beta}^\top \boldsymbol{x}^{(i)})
\end{aligned}$$

Taking logs turns the product into a sum, avoids underflow, and makes differentiation easy. Correct predictions contribute terms close to 0 (since log of a number near 1 is near 0); very wrong predictions contribute large negative values (since log of a tiny number is very negative), which the optimizer will try to avoid.

From MLE to the training loss and parameter updates How do we find parameters β that maximize $\ell(\boldsymbol{\beta})$? We can fit $\boldsymbol{\beta}$ by *MLE*: choose $\hat{\boldsymbol{\beta}}$ that maximizes the (log-)likelihood $\ell(\boldsymbol{\beta})$ like what we did in training ANN. In practice, we minimize a loss, so we take the *negative* log-likelihood(NLL) and average it over the dataset. This is exactly the binary cross-entropy used in ML libraries. The exact procedure for doing this is the following steps.

Step 1. Construct a loss function. We average the cross-entropy for a scaled negative log-likelihood. With $p^{(i)} = \sigma(\boldsymbol{\beta}^\top \boldsymbol{x}^{(i)})$ and $\ell(\boldsymbol{\beta}) = \sum_{i=1}^{m} \Big[y^{(i)} \log p^{(i)} + (1 - y^{(i)}) \log\Big(1 - p^{(i)}\Big)\Big]$, define the cost s.t:

$$C(\boldsymbol{\beta}) \;\equiv\; -\frac{1}{m}\,\ell(\boldsymbol{\beta}) = -\frac{1}{m}\sum_{i=1}^{m}\Big[y^{(i)} \log \sigma(\boldsymbol{\beta}^\top \boldsymbol{x}^{(i)}) + \Big(1 - y^{(i)}\Big) \log\Big(1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{x}^{(i)})\Big)\Big]$$

Why the factor $1/m$*?* Pure scaling. It turns the sum into an average so the loss has comparable magnitude across different dataset (or batch) sizes.

Step 2. Take the gradient of the loss. For the i-th single data point, $z^{(i)} = \boldsymbol{\beta}^\top \boldsymbol{x}^{(i)}$, $p^{(i)} = \sigma(z^{(i)})$, and $\sigma'(z) = \sigma(z)\Big(1 - \sigma(z)\Big)$.

(1) Chain rule through the sigmoid.

$$\frac{\partial z^{(i)}}{\partial \boldsymbol{\beta}} = \boldsymbol{x}^{(i)}, \qquad \frac{\partial p^{(i)}}{\partial \boldsymbol{\beta}} = \sigma'(z^{(i)})\,\boldsymbol{x}^{(i)} = p^{(i)}\Big(1 - p^{(i)}\Big)\boldsymbol{x}^{(i)}$$

(2) Derivatives of the log terms.

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\beta}} \log p^{(i)} &= \frac{1}{p^{(i)}}\,\frac{\partial p^{(i)}}{\partial \boldsymbol{\beta}} && \text{(derivative of log)} \\
&= \frac{1}{p^{(i)}}\,\frac{\partial p^{(i)}}{\partial z^{(i)}}\,\frac{\partial z^{(i)}}{\partial \boldsymbol{\beta}} && \text{(chain rule)} \\
&= \frac{1}{p^{(i)}}\Big[p^{(i)}(1 - p^{(i)})\Big]\,\boldsymbol{x}^{(i)} && \text{(since } \sigma'(z) = \sigma(1-\sigma),\ \partial z/\partial \boldsymbol{\beta} = \boldsymbol{x}^{(i)}\text{)} \\
&= (1 - p^{(i)})\,\boldsymbol{x}^{(i)}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\beta}} \log\Big(1 - p^{(i)}\Big) &= \frac{1}{1 - p^{(i)}}\,\frac{\partial (1 - p^{(i)})}{\partial \boldsymbol{\beta}} && \text{(derivative of log)} \\
&= \frac{1}{1 - p^{(i)}}\left[-\frac{\partial p^{(i)}}{\partial z^{(i)}}\right]\frac{\partial z^{(i)}}{\partial \boldsymbol{\beta}} && \text{(chain rule)} \\
&= \frac{1}{1 - p^{(i)}}\Big[-\,p^{(i)}(1 - p^{(i)})\Big]\,\boldsymbol{x}^{(i)} && \text{(use } \sigma'(z)\text{)} \\
&= -\,p^{(i)}\,\boldsymbol{x}^{(i)}
\end{aligned}$$

(3) Combine with the label $y^{(i)} \in \{0, 1\}$**.** We are going to differentiate a log-likelihood of a single data point 'i'

$$\ell^{(i)}(\boldsymbol{\beta}) = y^{(i)} \log p^{(i)} + \Big(1 - y^{(i)}\Big) \log\Big(1 - p^{(i)}\Big),$$

using the results from Step (2): $\frac{\partial}{\partial \boldsymbol{\beta}} \log p^{(i)} = (1-p^{(i)})\boldsymbol{x}^{(i)}$ and $\frac{\partial}{\partial \boldsymbol{\beta}} \log\Big(1-p^{(i)}\Big) = -p^{(i)}\boldsymbol{x}^{(i)}$.

$$\begin{aligned}
\frac{\partial \ell^{(i)}}{\partial \boldsymbol{\beta}} &= y^{(i)} \frac{\partial}{\partial \boldsymbol{\beta}} \log p^{(i)} + \Big(1-y^{(i)}\Big) \frac{\partial}{\partial \boldsymbol{\beta}} \log\Big(1-p^{(i)}\Big) \\
&= y^{(i)}(1-p^{(i)})\,\boldsymbol{x}^{(i)} \;+\; \Big(1-y^{(i)}\Big)\Big(-p^{(i)}\boldsymbol{x}^{(i)}\Big) \\
&= \Big(y^{(i)} - y^{(i)}p^{(i)} - p^{(i)} + y^{(i)}p^{(i)}\Big)\boldsymbol{x}^{(i)} \\
&= \boxed{\Big(y^{(i)} - p^{(i)}\Big)\,\boldsymbol{x}^{(i)}}
\end{aligned}$$

(4) Sum/average to obtain ∇C.

$$\nabla C(\boldsymbol{\beta}) \;\equiv\; -\frac{1}{m}\,\ell(\boldsymbol{\beta}) = -\frac{1}{m}\sum_{i=1}^{m}\Big(y^{(i)} - p^{(i)}\Big)\boldsymbol{x}^{(i)} = \frac{1}{m}\sum_{i=1}^{m}\Big(p^{(i)} - y^{(i)}\Big)\boldsymbol{x}^{(i)}$$

With $X \in \mathbb{R}^{m\times(d+1)}$, $\boldsymbol{p} = (p^{(i)})$, and $\boldsymbol{y} = (y^{(i)})$,

$$\boxed{\nabla C(\boldsymbol{\beta}) = \frac{1}{m}\,X^{\top}(\boldsymbol{p}-\boldsymbol{y})} \quad \text{(equivalently, } \nabla \ell(\boldsymbol{\beta}) = X^{\top}(\boldsymbol{y}-\boldsymbol{p})\text{)}$$

Step 3. Gradient updates. Using the averaged cross-entropy $C(\boldsymbol{\beta}) = -\frac{1}{m}\,\ell(\boldsymbol{\beta})$ and the result $\nabla C(\boldsymbol{\beta}) = \frac{1}{m}X^{\top}(\boldsymbol{p}-\boldsymbol{y})$, the two equivalent update rules are possible:

$$\begin{aligned}
&\text{(maximize log-likelihood } \ell) : \boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \eta\, X^{\top}(\boldsymbol{y}-\boldsymbol{p}), \\
&\text{(minimize loss } C) : \boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta\,\frac{1}{m}\,X^{\top}(\boldsymbol{p}-\boldsymbol{y})
\end{aligned}$$

Each sample contributes $(p^{(i)} - y^{(i)})\,\boldsymbol{x}^{(i)}$. If the model *overpredicts* the positive class ($p^{(i)} > y^{(i)}$), the term is positive and the descent step subtracts along $\boldsymbol{x}^{(i)}$, pushing the logit $\boldsymbol{\beta}^{\top}\boldsymbol{x}^{(i)}$ down. If it *underpredicts* ($p^{(i)} < y^{(i)}$), the step increases the logit. In general, Maximizing the log-likelihood has been more widely used such that:

$$\frac{\partial \ell(\boldsymbol{\beta})}{\partial \beta_j} = \sum_{i=1}^{m}\Big(y^{(i)} - \sigma(\boldsymbol{\beta}^{\top}\boldsymbol{x}^{(i)})\Big)\,x_j^{(i)} \quad \Rightarrow \quad \beta_j^{\text{new}} = \beta_j^{\text{old}} + \eta\sum_{i=1}^{m}\Big[y^{(i)} - \sigma\Big((\boldsymbol{\beta}^{\text{old}})^{\top}\boldsymbol{x}^{(i)}\Big)\Big]\,x_j^{(i)}$$

Dividing by m gives the ascent update on the *average* log-likelihood, and flipping the sign gives the descent update on C. The η is the learning rate and the effective step size self-adjust the whole algorithm. The Hessian of the negative log-likelihood is

$$\nabla^2\Big(-\ell(\boldsymbol{\beta})\Big) = \sum_{i=1}^{m} p^{(i)}\Big(1-p^{(i)}\Big)\;\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)\top} = X^{\top}WX, \quad W = \operatorname{diag}\Big(p^{(i)}(1-p^{(i)})\Big).$$

Since $p(1-p)$ peaks at $p = 0.5$ and is tiny near 0 or 1, ambiguous points (near 0.5) contribute more curvature, and hence more informative gradients, while near-certain points contribute less. This yields a smooth, convex objective with stable updates.

Optional: Newton/IRLS as a reference

Use curvature to take a second-order step that solves a weighted least-squares subproblem each iteration. With $W = \mathrm{diag}(p^{(i)}(1-p^{(i)}))$ and residual $\boldsymbol{r} = \boldsymbol{p} - \boldsymbol{y}$,

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - (X^\top W X)^{-1} X^\top \boldsymbol{r}$$

This is the classical *Iteratively Reweighted Least Squares (IRLS)* algorithm for logistic regression. Each iteration: (i) compute $p^{(i)} = \sigma(\boldsymbol{\beta}^\top \boldsymbol{x}^{(i)})$ and W; (ii) solve the weighted normal equations; (iii) update $\boldsymbol{\beta}$.

Training logistic regression by **MLE** is exactly the same as minimizing **binary cross-entropy**. Gradients have the clean form $\nabla C = \frac{1}{m} X^\top (\boldsymbol{p} - \boldsymbol{y})$, and the objective is *convex*, so both the first-order (SGD) and second-order (IRLS) optimizers work reliably.

5 Expanding logistic regression to *multiclass*

Before moving to K classes, it helps to recall two (binary) practical notes and one upside that the slides emphasize:

- **How to read coefficients (multiplicative, not additive).** In logistic regression, the weights do *not* change the probability p linearly. They add linearly to the *log-odds* and thus scale the *odds*. If x_j increases by 1, then

  $$\log \frac{p}{1-p} \;\mapsto\; \log \frac{p}{1-p} + \beta_j \quad\Longrightarrow\quad \text{odds} \;\mapsto\; e^{\beta_j} \times \text{odds}.$$

  This "odds ratio" view is why interpretation is more involved than in linear regression.

- **Complete separation (why training may fail).** If some feature (or a linear combination of features) perfectly separates the two classes, the MLE drives one or more coefficients to $\pm\infty$ and gradient updates will not converge. This is a bit ironic—such a feature is extremely informative, but then you do not really need ML: a simple rule already classifies all training points. In practice, we fix this by (i) adding regularization (ℓ_2/ℓ_1) to keep coefficients finite, or (ii) using a Bayesian prior on coefficients.

- **Good news: we get probabilities.** Logistic regression is not just a classifier; it yields calibrated probabilities. Knowing an instance is 99% likely to be class 1 conveys much more than a bare label (e.g. 51%).

With those in mind, a single sigmoid models $P(Y = 1 \mid x)$ in the binary case. For $K > 2$ classes, we need *a vector of probabilities* that each lie in $(0, 1)$ and sum to 1. The **softmax** layer provides exactly that: for logits $z_j = W_j^\top x + b_j$,

$$a_j = \mathrm{softmax}(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \qquad a_j \in (0,1), \;\; \sum_{j=1}^{K} a_j = 1.$$

We will pair softmax with the negative log-likelihood (a.k.a. cross-entropy) loss to train the multiclass model as we saw before.

6 Thresholds, class imbalance, and calibration

Decision thresholds are task-dependent. The default rule "predict class 1 if $p \geq 0.5$" is convenient but not sacred. If false negatives are costly (e.g., medical screening), lower the threshold; if false positives are costly (e.g., fraud alerts), raise it. Use ROC or Precision–Recall curves to *see* the trade-off and pick a threshold that matches your goal.

Class imbalance needs attention. When one class is rare, a high accuracy can be misleading. Practical fixes:

- *Weighted loss:* give the rare class a larger weight in NLL.
- *Resampling:* oversample the minority or undersample the majority.
- *Report the right metrics:* PR-AUC, F1, recall/precision—alongside ROC-AUC—so success is measured on what you care about.

Calibration tells you if probabilities can be trusted. A calibrated model's "70%" means "about 70 out of 100 are truly positive." Logistic regression is often well-calibrated out of the box, but if you stack it in larger systems or post-process scores, check with reliability diagrams and, if needed, apply Platt scaling or isotonic regression.

This is a posting that I summarized with study-purpose and is adapted from lecture notes of NE-795(Scientific Machine Learning), given by professor Xu Wu, NC State University.