LOGISTIC REGRESSION

INTRODUCTION TO LOGISTIC REGRESSION

Logistic regression is a fundamental statistical and machine learning technique used for solving binary classification problems. Despite its name containing "regression," it's actually a classification algorithm that predicts the probability of an instance belonging to a particular class. Unlike linear regression which predicts continuous values, logistic regression predicts discrete class labels by modeling the probability that a given input belongs to a specific category.

SOME BASIC GEOMETRY

1. Every Line Has a Positive Side and a Negative Side

In the figure, the line represents the equation:

$$4x+3y+5=0$$

The blue region corresponds to the positive side of the line, where:

$$4x+3y+5>0$$

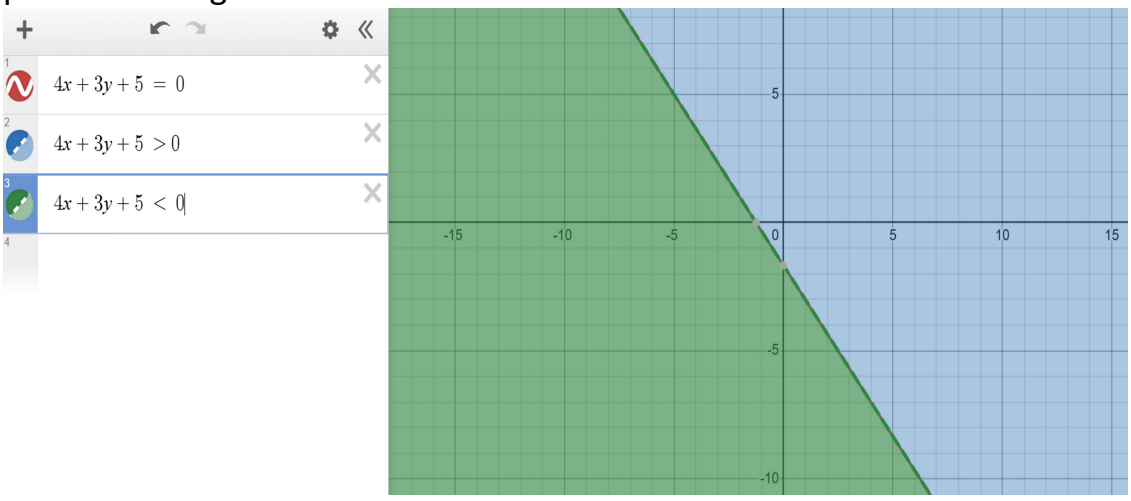
The green region corresponds to the negative side of the line, where:

$$4x+3y+5<0$$

The line itself (shown as the boundary between the two regions) represents the set of points satisfying:

$$4x+3y+5=0$$

Any linear equation in two variables (x and y) divides the 2D plane into two regions—called half-planes—based on whether the expression evaluates to a positive or negative value.



2. How to Determine if a Given Point Lies on a Given Line

Suppose we are given the equation of a line:

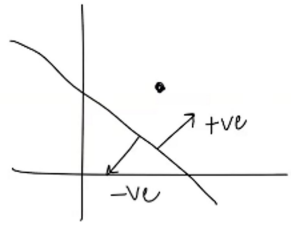
$$4x + 3y + 5 = 0$$

Now, let's say we have a point (5,2) and we want to check whether this point lies on the line.
To do this, substitute the x- and y-coordinates of the point into the equation:

$$4(5) + 3(2) + 5 = 20 + 6 + 5 = 31 \neq 0$$

Since the result is not zero, the point (5,2) does not lie on the line.
A point (x,y) lies on the line Ax + By + C = 0 if and only if substituting its coordinates into the equation results in a value of zero.

3. How to Determine if a Given Point Lies on the Positive or Negative Side of a Line



Suppose we are given a line represented by the equation:

$$Ax + By + C = 0$$

and a point (x_1, y_1). To determine whether this point lies on the positive side or the negative side of the line, substitute the coordinates into the left-hand side of the equation:

$$Ax_1 + By_1 + C$$

If the result is greater than 0, the point lies on the positive side of the line.

- If the result is less than 0, the point lies on the negative side.
- If the result is equal to 0, the point lies on the line.

Example 1:

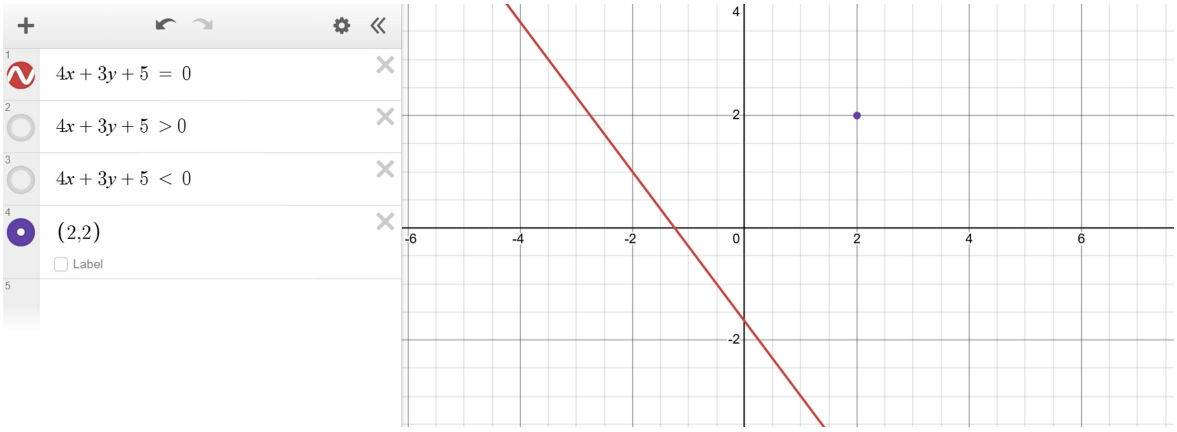
Given the line:

$$4x+3y+5=0$$

Let's test the point (2,2) :

$$4(2)+3(2)+5=8+6+5=19>0$$

So, the point (2,2) lies on the positive side of the line.

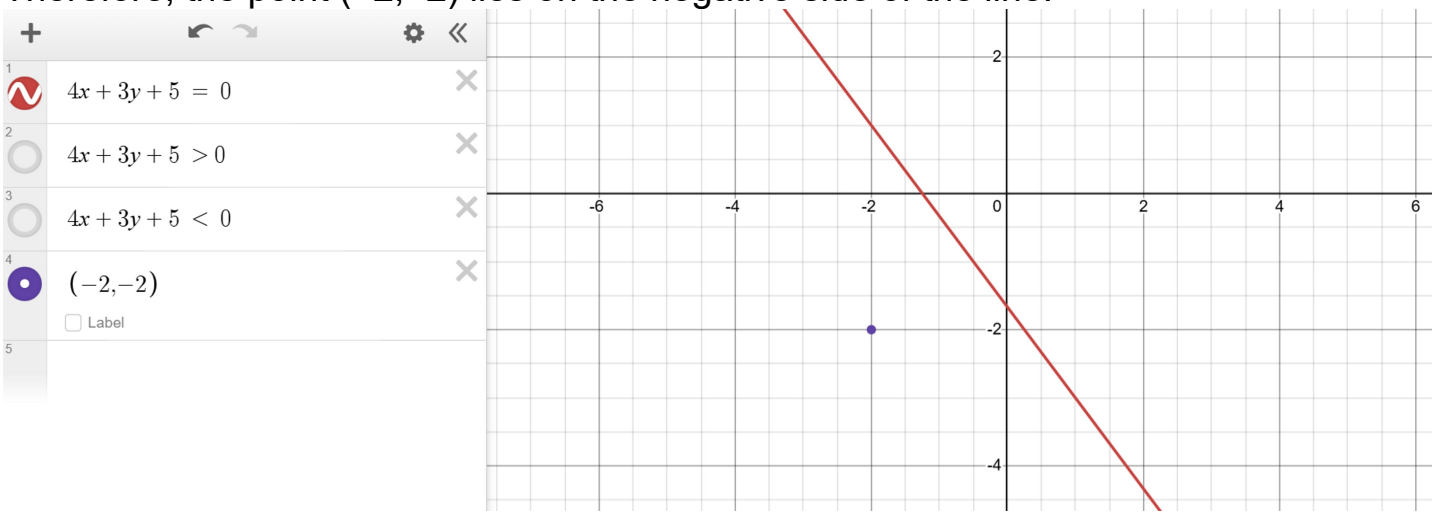


Example 2:

Now test the point (−2,−2) :

$$4(-2)+3(-2)+5 = -8-6+5 = -9 < 0$$

Therefore, the point (−2,−2) lies on the negative side of the line.



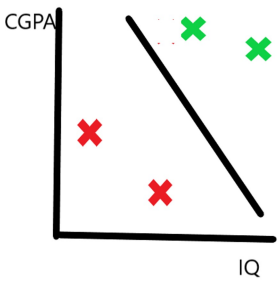
THE PROBLEM

Logistic Regression is a classification algorithm. Given two classes, it learns a decision boundary (a line in 2D) that separates the data points belonging to different classes.

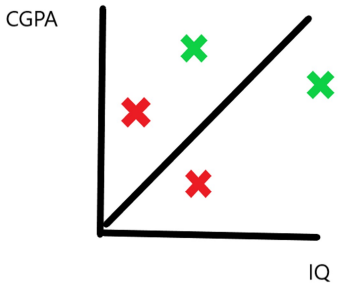
Let's say we have data for four students. Each student has two features: IQ and CGPA.

- If a student got placed, they are marked with a green dot.
- If a student did not get placed, they are marked with a red dot.

We want to train a logistic regression model to predict if a student will get placed, given their IQ and CGPA.



MODEL 1



MODEL 2

Comparing Two Models

We trained two different models. Each model draws a different decision boundary:

- Model 1: Separates green and red dots correctly (0 misclassifications).
- Model 2: Misclassifies two red dots as green.

Clearly, Model 1 is better because it has fewer misclassifications. So we might be tempted to always select the model with the least misclassification count.

Let the equation of the decision boundary be:

$$Ax + By + C = 0$$

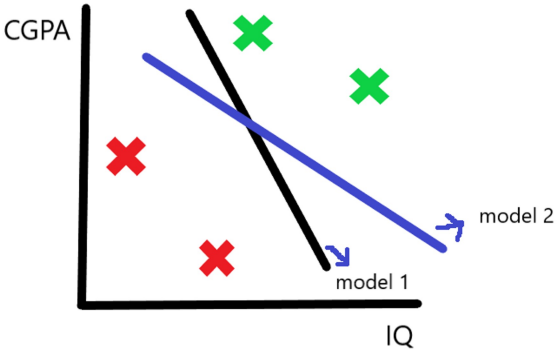
To classify any point (x_i, y_i) , compute:

$$Z = Ax_i + By_i + C$$

Then:

- If the point is green (should be positive) and z > 0 : correct
- If the point is red (should be negative) and z < 0 : correct
- Otherwise: misclassification

But There's a Problem…



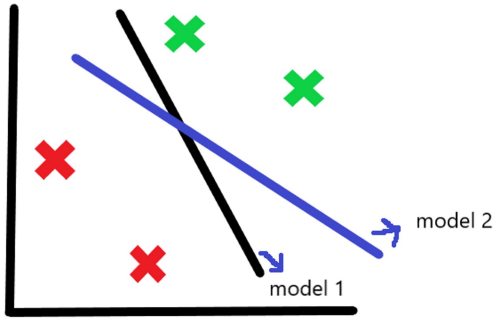
What if two or more models result in zero misclassification? For example:

- Model 1 and Model 2 both classify all training points correctly.
- Yet, the boundaries they draw are very different.

Now, which model should we choose?

We can't use misclassification count anymore because it’s the same for both.
This exposes a flaw in the naïve approach. Just using the count of misclassified points is not enough.

New Problem Formulation



As we know, there is a problem where we have two models—both with zero misclassification. Because of this, it's difficult to determine which model is better, as both seem equally good in terms of classification accuracy.
However, classification shouldn't be treated as strictly black and white. The current logic assumes that any point above a decision boundary (the line) lies in the positive region and is therefore classified as positive, while any point below the line is in the negative region and classified as negative. This binary perspective is overly simplistic and rigid.
A more nuanced approach would be to assign a score or value to each point instead of labeling them directly as positive or negative. For example, we can assign a value between 0 and 1 to each point based on how far it is from the decision boundary.

- Points that are farther from the line in the positive region can be assigned a higher score, indicating stronger confidence in the positive classification.
- Points closer to the line in the positive region get a lower score, indicating lower confidence.
- Similarly, for the negative region, points closer to the line can be assigned higher (less negative) values, while points farther from the line receive lower values (more negative), reflecting stronger confidence in the negative class.

This highlights that distance from the decision boundary matters, and classification should consider this.
In summary, this approach—where we use a continuous score based on distance from the boundary—is more informative than binary classification. Once these scores are assigned, classification can be performed more meaningfully based on thresholds or probability-like interpretations.

Understanding Logistic Regression: How 0 and 1 Are Predicted

When working on a classification problem, let's say we have training data and we apply logistic regression. The goal is to train the model so that it can make predictions for every data point. Each prediction is either 0 or 1 — where 1 might represent a "positive" class (e.g., green), and 0 represents a "negative" class (e.g., red).

Example Dataset

Suppose we have the following dataset:

IQ	CGPA	Placed
60	6	0
40	4	0
80	8	1
90	9	1

Now, we want to make a prediction for the fourth data point.
Let's assume our logistic regression model has learned the following linear equation:

$$3X + 5Y + 6 = 0$$

To make a prediction, we plug the values of CGPA = 9 and IQ = 90 into the equation:

$$3(9) + 5(90) + 6 = 483$$

Since the result is greater than 0, we predict the label as 1 (placed/green). If the result were less than 0, we would predict 0 (not placed/red).

Mathematical Interpretation in Machine Learning

The equation can be written in the standard linear model form:

$$z = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

Where:

- β_0 (intercept),
- β_1,β_2 are the weights (coefficients),
- $X_1 =$ CGPA, $X_2 =$IQ ,
- z is the linear combination of inputs.

If $z < 0$, we label the point as 0; if $z > 0$, we label it as 1.

This kind of thresholding is done using a step function:

- If z is positive → output = 1
- If z is negative → output = 0

Graphically, this looks like a sharp jump from 0 to 1, which is why it's called a step function.

Why Not Use the Step Function?

While the step function can classify, it doesn't give us a probabilistic or continuous output. Ideally, we want a model that:

- Outputs values close to 1 for points far from the decision boundary (more confident predictions).
- Outputs values close to 0.5 for points near the decision boundary (less confident predictions).

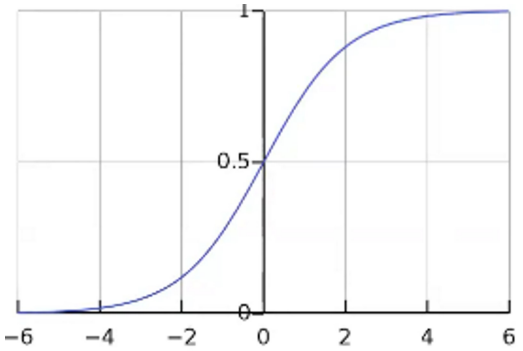
So instead of a step function, we use the sigmoid function to convert the value of z into a probability between 0 and 1.

$\sigma(z) = \frac{1}{1+e^{-z}}$

The sigmoid function smoothly maps any real-valued number into the (0, 1) range. This allows logistic regression to estimate the probability that a given input belongs to class 1.

1. We compute a linear combination:
   $z = \beta_0+\beta_1 X_1+\beta_2 X_2$
2. Instead of directly thresholding z with a step function, we pass it through the sigmoid:
   $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}$
3. We interpret the output $\hat{y}$ as the probability of belonging to class 1.
4. We can then use a threshold (e.g., 0.5) to decide whether to classify it as 0 or 1.

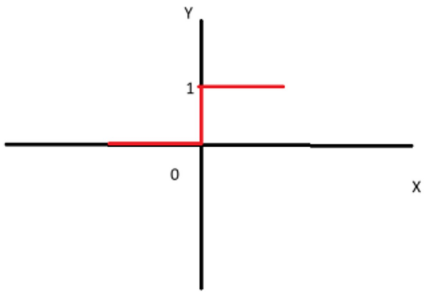
Sigmoid Function



Equation of the Sigmoid Function:

$Y = \frac{1}{1+e^{-X}}$

Step Function (for Comparison):



Understanding the Behavior:

- In the sigmoid function:
  - When the value of X is very large, the output Y approaches 1.
  - When X = 0 , the output Y = 0.5 .
  - When X is very small or approaches negative infinity, the output Y approaches 0.
  - This smooth transition is the key behavior of the sigmoid function.
- In the step function:
  - No matter how large or small the value of X :
    - If X is positive, the output Y=1 .
    - If X is negative, the output Y=0 .
  - It's a sharp, binary decision — either 0 or 1 — with no smooth transition.

Why Use Sigmoid Instead of Step Function?

- The step function is too rigid — it classifies values as either 0 or 1 with no sense of uncertainty.
- The sigmoid function provides a smooth and continuous output between 0 and 1.
- This makes it more suitable for machine learning models, especially when we want to interpret outputs probabilistically (e.g., “this input has a 70% chance of belonging to class 1”).

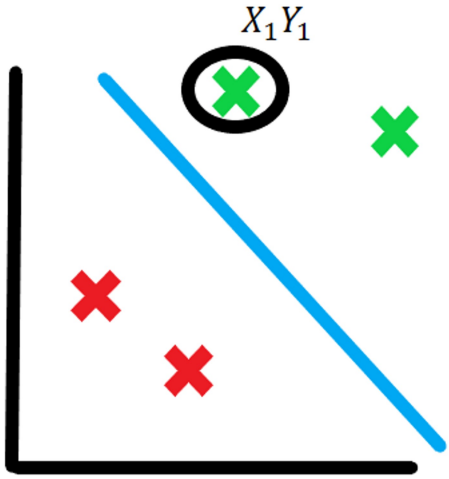
Let our equation of the line be:

$$Z = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

First, we calculate the value of Z.
Instead of passing this value to a step function, we pass it to the sigmoid function:

$$\sigma(z)$$

This gives us a value between 0 and 1 depending on the value of z . If z=0 , the output is 0.5. This means that any point on the line will have a sigmoid output of 0.5.
So, our point X lying on the line will also output 0.5 from the sigmoid.

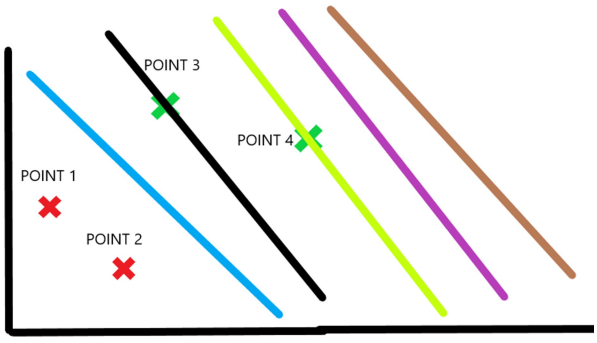


Now, consider the case where:

- Point 3 lies just above the blue line — its z value is slightly greater than 0, so the sigmoid output could be around 0.6.
- Point 4, further from the line, could have a sigmoid output around 0.7.

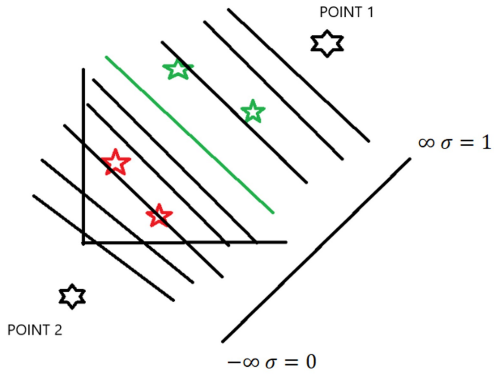
As we continue moving further in the positive direction, the sigmoid output approaches 1.
Conversely, if we move below the decision boundary into the negative side, sigmoid values decrease:

- At Point 1, we might get an output of 0.3.
- At Point 2, which is farther, the output might be 0.2.
- As we go further negative (towards infinity), the output approaches 0.



Essentially, our whole classification region gets converted into a gradient of probabilities.
The green decision boundary (our equation of the line) corresponds to a sigmoid value of 0.5.
As we move towards positive infinity, the sigmoid approaches 1.
As we move towards negative infinity, the sigmoid approaches 0.

So, if a point lies at positive infinity (far on the positive side), we are nearly certain it belongs to the positive class. Similarly, a point far in the negative side is almost certainly negative.



This gives us a probabilistic interpretation of classification:

- If a point has a probability of 0.6 of being positive, it has a 0.4 probability of being negative.
- If a point has a 0.7 probability of being positive, the probability of it being negative is 0.3.

This smooth gradient transforms our entire classification region into a probability landscape:

- On the line → probabilities are equal (0.5 positive, 0.5 negative)
- Farther in the positive direction → probability of positive increases
- Farther in the negative direction → probability of negative increases

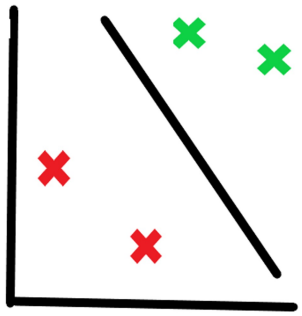
From Step Function to Sigmoid Function

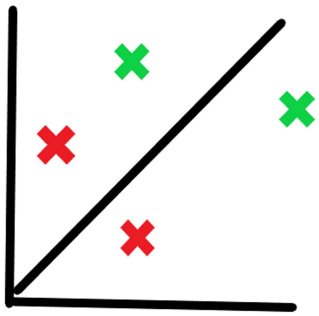
In the step function, if a point lies in the positive region of the line, we classify it as positive (1).
If it lies on the negative side, we classify it as negative (0).
But this is a rigid approach—it doesn't capture uncertainty.
With the sigmoid function, we now get a probability score for every point:

- The farther a point is in the positive region, the higher its chance of being 1.
- The farther a point is in the negative region, the higher its chance of being 0.

This turns our classification from black-and-white to a shaded gradient, allowing for a more nuanced, probabilistic understanding of each prediction.

Maximum Likelihood in Logistic Regression





When trying to classify data points using logistic regression, we want to find the best decision boundary (line) that separates the classes most accurately. But there are many possible lines—so how do we choose the best one?

We use a loss function.

The idea is to:

- Define a loss function that tells us how well a line (or model) performs.
- Minimize this loss function to find the best parameters (and hence the best line).

In logistic regression, the loss function used is based on maximum likelihood estimation (MLE).

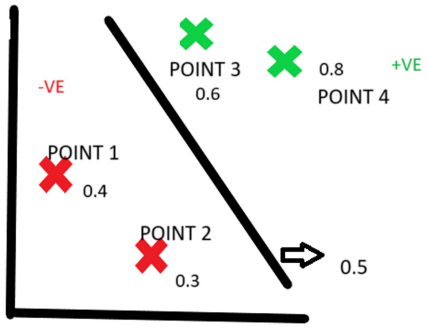
What is Maximum Likelihood?

Maximum Likelihood Estimation is a method that finds the parameters (i.e., the line) that make the observed data most probable.

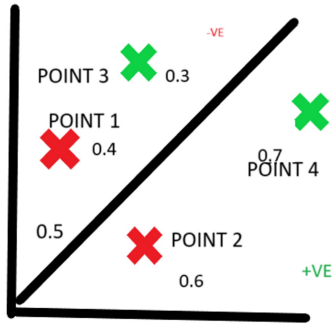
Instead of predicting exact labels (0 or 1), logistic regression gives probabilities. For each data point:

- If it's a positive class (e.g., green), we want its predicted probability to be close to 1.
- If it's a negative class (e.g., red), we want its predicted probability to be close to 0.

Comparing Two Models Using Likelihood



MODEL 1



MODEL 2

Suppose we have two models (Model 1 and Model 2), each with a different decision boundary. We calculate how likely each model is to produce the observed data by multiplying the predicted probabilities for all the points.
Let's walk through an example:

Model 1 Predictions:

Point	True Class	Predicted Probability	Contribution to Likelihood
1	Red	0.4 (→ red: 0.6)	0.6
2	Red	0.3 (→ red: 0.7)	0.7
3	Green	0.6	0.6
4	Green	0.8	0.8

Likelihood (Model 1) = 0.6 × 0.7 × 0.6 × 0.8 = 0.2016

Model 2 Predictions:

Point	True Class	Predicted Probability	Contribution to Likelihood
1	Red	0.4 (→ red: 0.6)	0.6
2	Red	0.6 (→ red: 0.4)	0.4
3	Green	0.3	0.3
4	Green	0.6	0.6

Likelihood (Model 2) = 0.6 × 0.4 × 0.3 × 0.6 = 0.0432

- Model 1 has a higher likelihood than Model 2.
- Therefore, Model 1 is a better fit according to maximum likelihood estimation.
- In logistic regression, the line that gives maximum likelihood becomes our final decision boundary.

Log Loss (Binary Cross-Entropy)

Let's say we have four data points, and a logistic regression model is used to predict the probabilities of those points belonging to the green (class = 1) or red (class = 0) classes.
Suppose the model gives the following predicted probabilities for the true classes:

- For a green point: 0.8
- For another green point: 0.6
- For a red point: 0.6

- For another red point: 0.7

The joint likelihood (i.e., the product of individual probabilities) for correct predictions would be:
ML = 0.8 × 0.6 × (1−0.6) × (1−0.7) = 0.8 × 0.6 × 0.4 × 0.3 = 0.0576
This is a small number — and in real-world datasets with thousands of samples, the product of so many probabilities (each between 0 and 1) becomes extremely small, approaching zero, which causes numerical underflow in computations.

Using Log to Avoid Underflow

To avoid underflow, instead of multiplying probabilities, we take the logarithm of the likelihood. The logarithm converts the product into a sum:

$$\log(ML) = \log(0.8) + \log(0.6) + \log(0.4) + \log(0.3)$$

However, since log values of numbers between 0 and 1 are negative, to make the loss value positive (since we aim to minimize it), we take the negative log-likelihood:

$$-\log(ML) = -\log(0.8) - \log(0.6) - \log(0.4) - \log(0.3)$$

So, instead of maximizing the likelihood (which was our original objective in Maximum Likelihood Estimation), we equivalently minimize the negative log-likelihood. This transformation makes the objective numerically stable and computationally feasible.

Why We Cannot Use Only $\widehat{Y_i}$ in All Cases

We might be tempted to write:

$$-\log(\hat{Y}_1) - \log(\hat{Y}_2) - \log(\hat{Y}_3) - \log(\hat{Y}_4)$$

But this is incorrect. Why?
Because $\widehat{Y_i}$ represents the predicted probability of the positive class (e.g., green). If a data point actually belongs to the negative class (e.g., red), using $\widehat{Y_i}$ would incorrectly measure the probability of the wrong class. So we must account for the true label Y_i.

Unified Expression for All Data Points

To correctly handle both classes (0 and 1), we use this combined expression for each data point:

$$-\left[Y_i \cdot \log(\hat{Y}_i) + (1 - Y_i) \cdot \log(1 - \hat{Y}_i)\right]$$

This works as follows:

- If $Y_i = 1$: only the first term remains $\rightarrow -\log(\hat{Y}_i)$
- If $Y_i = 0$: only the second term remains $\rightarrow -\log(1 - \hat{Y}_i)$

Applying this to all four points, we compute:

$$-\log(\hat{Y}_1) - \log(\hat{Y}_2) - \log(1 - \hat{Y}_3) - \log(1 - \hat{Y}_4)$$

Log Loss Formula (Binary Cross-Entropy)

Now, generalizing for all n data points:

$$L = -\frac{1}{n} \sum_{i=1}^{n} \left[Y_i \cdot \log(\hat{Y}_i) + (1 - Y_i) \cdot \log(1 - \hat{Y}_i) \right]$$

Where:

- Y_i is the true label (0 or 1)
- $\hat{Y}_i = \sigma(z_i)$, with $z_i = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots$
- $\sigma(z_i)$ is the sigmoid output (predicted probability of class 1)

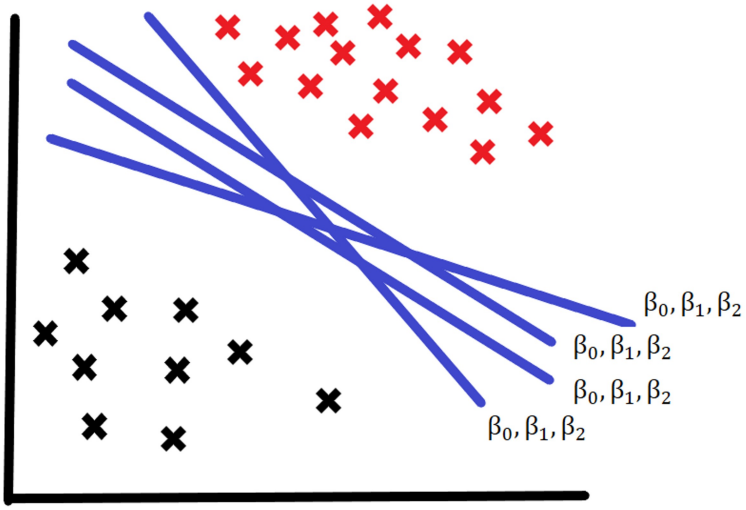
This function L is called:

- Log Loss
- Binary Cross-Entropy Loss
- It is the standard loss function for binary classification tasks using logistic regression.
- Minimizing Log Loss is equivalent to maximizing the likelihood of the observed data under the model.

We need to find the values of $\beta_0, \beta_1, \beta_2$ that minimize the value of our loss function.

$$\hat{Y}_i = \sigma(Z_i)$$

$$Z_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i}$$



Suppose we have data like the one shown above. When we apply logistic regression to this data, any of the blue lines could represent a possible logistic regression decision boundary. Each line corresponds to a specific set of coefficients $\beta_0, \beta_1, \beta_2$. However, we aim to find the line (i.e., the set of coefficients) that minimizes the loss function. That line will be our optimal logistic regression model.

GRADIENT DESCENT

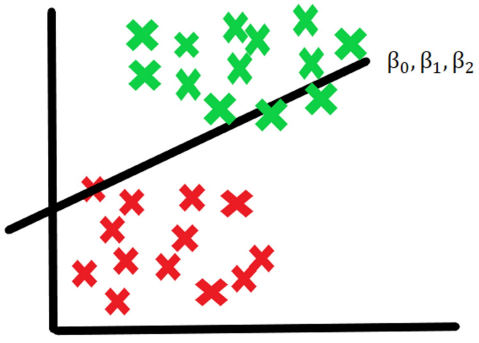
$$L = \frac{-1}{n} \sum_{i=1}^{n} Y_i \log(\hat{Y}_i) + (1 - Y_i) \log(1 - \hat{Y}_i)$$

Where,
n = Number of data points.
Y_i = Actual class label (0 or 1) for the i-th data point.
$\hat{Y}_i$ = Model's predicted probability of Y_i=1 (e.g., "green points").
$(1 - \hat{Y}_i)$ = Predicted probability of Y_i = 0 (e.g., "red points").

1. Interpretation of $1 - \hat{Y}_i$:
   Since logistic regression outputs the probability $\hat{Y}_i$ = P(Y_i=1), subtracting from 1 gives the complementary probability P(Y_i = 0).
2. No Closed-Form Solution:
   Unlike linear regression, logistic regression's loss function has no analytical solution for coefficients ($\beta_0, \beta_1, \beta_2$). Instead, we use iterative optimization methods like gradient descent to minimize the loss.
3. Gradient Descent:

- Updates coefficients iteratively by moving in the direction of the steepest descent of the loss function.
- Terminates when convergence (minimal loss) is achieved.

So what we do is:



Suppose we have a dataset, and we start with a random decision boundary

(a line). This line is defined by coefficients (parameters) like $\beta_0, \beta_1, \beta_2$
Initially, these coefficient values are randomly assigned.

Then we start updating these coefficient values using gradient descent.
The update rules are:

For β_0:

$$\beta_0 = \beta_0 - \eta \frac{\partial L}{\partial \beta_0}$$

Similarly, for other coefficients:
For β_1:

$$\beta_1 = \beta_1 - \eta \frac{\partial L}{\partial \beta_1}$$

For β_2:

$$\beta_2 = \beta_2 - \eta \frac{\partial L}{\partial \beta_2}$$

Eventually, through these updates, we will reach a point where the line fits the data best (i.e., the loss is minimized).

Now the challenge is to compute the gradients:

$$\frac{\partial L}{\partial \beta_0}, \frac{\partial L}{\partial \beta_1}, \frac{\partial L}{\partial \beta_2}$$

$$L = \frac{-1}{n} \sum_{i=1}^{n} Y_i \log(\hat{Y}_i) + (1 - Y_i) \log(1 - \hat{Y}_i)$$

Assume we have just one data point for simplicity. Then the loss becomes:

$$L = \left[-Y \log \hat{Y} - (1 - Y) \log(1 - \hat{Y})\right]$$

$$\frac{\partial L}{\partial \beta_1} = \frac{\partial}{\partial \beta_1} \left[-Y \log(\hat{Y}) - (1 - Y) \log(1 - \hat{Y})\right]$$

$$= -Y(1-\hat{Y})X_1 + (1 - Y)\hat{Y}X_1$$

$$= [-Y + Y\hat{Y} + \hat{Y} - Y\hat{Y}]X_1$$

$$= (\hat{Y} - Y)X_1$$

$$\frac{\partial L}{\partial \beta_1} = (\hat{Y} - Y)X_{1i}$$

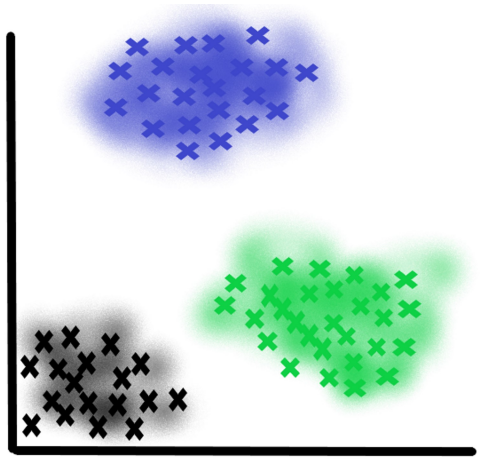
Other Partial Derivatives

Similarly:

$$\frac{\partial L}{\partial \beta_2} = (\hat{Y} - Y)X_{2i}$$

$$\frac{\partial L}{\partial \beta_0} = (\hat{Y} - Y)$$

What is multi-class classification ?



Multi-class classification refers to classification tasks that involve more than two classes, where each input is assigned to exactly one of the possible classes.

Examples:

1. Digit Recognition
   Predict which digit (0–9) is in an image → 10 classes
   (e.g., MNIST dataset)
2. Animal Classification
   Classify an image as cat, dog, or rabbit → 3 classes
3. Sentiment Analysis
   Predict whether a review is positive, neutral, or negative → 3 classes

How Logistic Regression Handles Multi-class Classification Problems

Logistic regression can be extended to handle multi-class classification using two main techniques:

1. One-vs-Rest (OvR), also known as One-vs-All (OvA)
2. Multinomial Logistic Regression, also known as Softmax Regression

OVR (One-vs-Rest) Approach

Suppose we have a dataset where the input features are cgpa and iq, and the target/output feature is placement, which can take one of three values: Yes, No, or Opt Out.

cgpa	iq	placement
-	-	Y
-	-	N
-	-	O

In the One-vs-Rest (OvR) approach, for each unique class, we train a

separate logistic regression model. So, if we have 3 classes, we train 3 logistic regression models — one for each class.

Steps:

1. Split the output into separate binary classification problems — one for each class:
   - Model 1: "Yes" vs (No + Opt Out)
   - Model 2: "No" vs (Yes + Opt Out)
   - Model 3: "Opt Out" vs (Yes + No)
2. Apply One-Hot Encoding to the output column. This converts the categorical target variable into multiple binary columns — one for each class:

cgpa	iq	Placement = Y	Placement = N	Placement = O
-	-	1	0	0
-	-	0	1	0
-	-	0	0	1

This is the transformed data used to train the three logistic regression models — each trying to predict whether a sample belongs to its respective class or not.

In the OvR approach, the dataset is split into three binary classification problems based on the target variable placement:

1. Class 1 (Placement = Yes): Dataset includes samples where placement = Yes, and the others (No, Opt Out) are considered negative examples.
2. Class 2 (Placement = No): Dataset includes samples where placement = No, and the others (Yes, Opt Out) are considered negative examples.
3. Class 3 (Placement = Opt Out): Dataset includes samples where placement = Opt Out, and the others (Yes, No) are considered negative examples.

Thus, the multiclass classification problem is transformed into three binary classification problems.

How the Prediction Works:

- Suppose we have a new data point with cgpa = 6.5 and iq = 65, and we want to predict the placement (whether the student gets placed, is not placed, or opts out).
- The point is passed through all three trained classifiers, and we get three probabilities:
  - P(Placed) = 0.6
  - P(Not Placed) = 0.3
  - P(Opt Out) = 0.5
- Next, we normalize the probabilities to make sure they sum to 1:

$$P(Y) = \frac{0.6}{0.6 + 0.3 + 0.5} = 0.4286$$

$$P(N) = \frac{0.3}{0.6 + 0.3 + 0.5} = 0.2143$$

$$P(O) = \frac{0.5}{0.6 + 0.3 + 0.5} = 0.3571$$

- Now, the sum of the probabilities is 1, and we have the normalized probabilities for each class:
  - P(Placed) = 0.4286
  - P(Not Placed) = 0.2143
  - P(Opt Out) = 0.3571
- The class with the highest probability (0.4286 for "Placed") is chosen as the predicted outcome.

Advantages and Disadvantages of the OvR Approach

Advantages:

1. Simplicity: Each classifier is a binary logistic regression model, which makes the approach easy to implement and understand.
2. Flexibility: The approach can be extended to a large number of classes, allowing us to handle problems with more than three classes by training additional binary classifiers.
3. Independence of Classifiers: Since each classifier is independent, you can optimize and train them separately, which can make the process more efficient in certain scenarios.
4. Interpretability: The coefficients of each logistic regression model are interpretable, making it easy to understand how each feature influences the decision for each class.

Disadvantages:

1. Imbalanced Data: If one class is significantly larger than the others, the classifiers for the smaller classes might perform poorly due to imbalance.
2. Multiple Predictions: During prediction, we need to compute multiple probabilities (one for each classifier) and then normalize them, which adds extra computational steps.
3. Overlapping Decision Boundaries: Each binary classifier might misclassify samples that are close to the decision boundary between two classes. As a result, the overall predictions can be less accurate if the classes are not well-separated.
4. Training Time: As the number of classes increases, the number of

binary classifiers grows, which can increase the training time and complexity.

5. No Global Probability Relationship: Since each classifier is trained independently, there is no direct relationship between the predicted probabilities across the classifiers. This can cause inconsistencies when comparing the probabilities.

Softmax function :-> The softmax function is used to convert a vector of real numbers (logits) into a probability distribution. It is commonly used in classification problems where the goal is to assign probabilities to each class.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Suppose we have three real-valued scores (also called logits): z_1, z_2, z_3
The softmax of each score is calculated as:

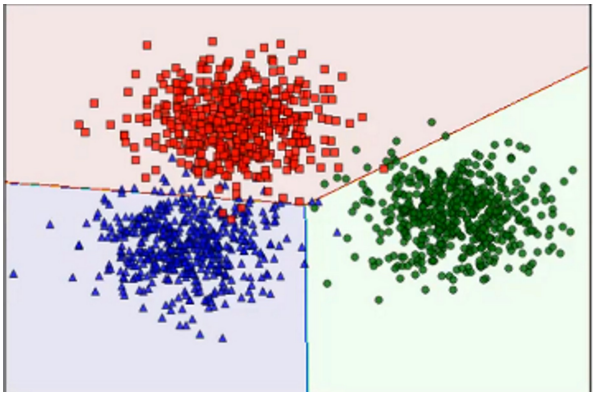
$$\sigma(z_1) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$
$$\sigma(z_2) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$
$$\sigma(z_3) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

Each output of the softmax function represents the predicted probability of the corresponding class. These probabilities are:

- Always positive
- Add up to 1, i.e.:

$$\sigma(z_1) + \sigma(z_2) + \sigma(z_3) = 1$$

Softmax Approach



Softmax regression (also known as multinomial logistic regression) is a generalization of logistic regression used for multi-class classification problems. While logistic regression is used to separate two classes using a decision boundary (like a line in 2D or a plane in 3D), softmax regression handles more than two classes.
Since softmax regression is still a linear model, it creates linear decision boundaries between classes.

Understanding the Model:

Suppose we have k classes. The softmax regression model will learn k sets of parameters, one for each class. Each set of parameters includes:

- A bias term $\beta_0^{(i)}$
- A weight vector $\beta^{(i)}=[\beta_1^{(i)},\beta_2^{(i)},\ldots,\beta_n^{(i)}]$, where n is the number of features, and i ∈ {1,2,...,k}

This means that for each class, the model learns a linear function:

$$z_i = \beta_0^{(i)} + \beta_1^{(i)}x_1 + \beta_2^{(i)}x_2 + \cdots + \beta_n^{(i)}x_n$$

These scores z_i are then passed through the softmax function to convert them into probabilities for each class.

Parameter Count:

- For k classes and n features, the model needs to learn:
  Total parameters=k×(n+1)
  (where "+1" accounts for the bias term)
- Example:
  - If we have 3 classes and 2 features, then the model will learn 3 × (2 + 1) = 9 parameters
  - If we increase to 3 features, then 3 × (3 + 1) = 12 parameters

Intuition:

Each class's parameter set defines a linear boundary. The model uses these boundaries to create decision regions, effectively assigning a class to every point in the feature space based on the highest softmax score.

Multi-class Classification Example using Softmax Regression

Suppose we have a dataset with the following features:

- cgpa (academic performance)
- iq (intelligence score)
- placement (target/output variable with three classes: Yes, No, and O for "Opt-out")

To handle multiple classes in the target variable, we first apply one-hot encoding to convert categorical labels into a numeric format.

One-Hot Encoding of the Target Variable:

cgpa	iq	Placement = Yes	Placement = No	Placement = O
—	—	1	0	0

—	—	0	1	0
—	—	0	0	1

Each row will now have a vector of size 3 representing the actual class label using 1s and 0s.

Loss Function: Categorical Cross-Entropy

To train the softmax regression model, we use the cross-entropy loss function. It measures the difference between the true distribution (from one-hot encoding) and the predicted distribution (from softmax outputs).

$$L = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} Y_i^{(k)} \log\left(\hat{Y}_i^{(k)}\right)$$

Where:

- n : Number of data points (samples)
- K : Number of classes (here, K=3)
- $Y_i^{(k)}$: True label (1 if sample iibelongs to class k , 0 otherwise)
- $\hat{Y}_i^{(k)}$: Predicted probability (from softmax) that sample I belongs to class k

Special Case – Binary Classification:
If there are only two classes (i.e., K=2), the categorical cross-entropy loss simplifies to the binary cross-entropy loss, commonly used in logistic regression.

Understanding Cross-Entropy Loss for One-Hot Encoded Data (Step-by-Step)

Let's assume, for simplicity, that our dataset contains only one data point, i.e., n=1 .
In that case, the categorical cross-entropy loss simplifies to:

$$L = -\sum_{k=1}^{K} Y^{(k)} \log\left(\hat{Y}^{(k)}\right)$$

Where:

- K = 3 (the number of classes: Yes, No, and Opt)
- $Y^{(k)}$: True label (1 for the correct class, 0 for others)
- $\hat{Y}^{(k)}$: Predicted probability (output of softmax for class k)

Expanding the Loss for a Single Data Point

Assume the true label for the first row is "Yes" (so the one-hot encoding is [1, 0, 0]).

Then the loss becomes:

$$L = -\left(Y^{\text{Yes}} \log\left(\hat{Y}^{\text{Yes}}\right) + Y^{\text{No}} \log\left(\hat{Y}^{\text{No}}\right) + Y^{\text{Opt}} \log\left(\hat{Y}^{\text{Opt}}\right)\right)$$

Since the one-hot vector is [1, 0, 0], this simplifies to:

$$L = -\log\left(\hat{Y}^{\text{Yes}}\right)$$

Extending to Multiple Data Points

Now assume we have three data points, each with different class labels:

1. First row → class: Yes
2. Second row → class: No
3. Third row → class: Opt

Then the total loss becomes:

$$L = -\frac{1}{3}\left(\log\left(\hat{Y}_1^{\text{Yes}}\right) + \log\left(\hat{Y}_2^{\text{No}}\right) + \log\left(\hat{Y}_3^{\text{Opt}}\right)\right)$$

Here:

- $\hat{Y}_1^{Yes}$: predicted probability of class “Yes” for the first data point
- $\hat{Y}_2^{Yes}$: predicted probability of class “No” for the second data point
- $\hat{Y}_3^{Yes}$: predicted probability of class “Opt” for the third data point

Out of the total 9 terms (3 data points × 3 classes), only 3 terms remain non-zero—the ones corresponding to the true class labels. All others are multiplied by 0 due to one-hot encoding.

The goal of training is to minimize this loss function. Minimizing the loss corresponds to maximizing the predicted probabilities of the correct classes.

How the Softmax Model Computes Predicted Probabilities

Earlier, we saw that our loss function for multi-class classification looks like:

$$L = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} Y_i^{(k)} \log\left(\hat{Y}_i^{(k)}\right)$$

Where:

- $\hat{Y}_i^{(k)}$ is the predicted probability that data point i belongs to class k
- But... how do we actually compute these $\hat{Y}_i^{(k)}$ values?

From Binary to Multi-class Logistic Regression

In binary logistic regression, the predicted probability is computed using the sigmoid function:

$$\hat{Y}_i = \sigma(z_i) = \frac{1}{1 + e^{-z_i}}, \quad \text{where } z_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i}$$

In multi-class classification (softmax regression), instead of the sigmoid, we use the softmax function.

Computing Softmax Scores:

Suppose we have three classes: Yes, No, and Opt. For each class, the model learns its own set of coefficients:

For class Yes: $\beta_0^{(1)}, \beta_1^{(1)}, \beta_2^{(1)}$

For class No: $\beta_0^{(2)}, \beta_1^{(2)}, \beta_2^{(2)}$

For class Opt: $\beta_0^{(3)}, \beta_1^{(3)}, \beta_2^{(3)}$

Assume we have a data point with features: cgpa = 8, iq = 80.
We compute the class scores (logits):

$$z_{\text{Yes}} = \beta_0^{(1)} + \beta_1^{(1)} \cdot 8 + \beta_2^{(1)} \cdot 80$$

$$z_{\text{No}} = \beta_0^{(2)} + \beta_1^{(2)} \cdot 8 + \beta_2^{(2)} \cdot 80$$

$$z_{\text{Opt}} = \beta_0^{(3)} + \beta_1^{(3)} \cdot 8 + \beta_2^{(3)} \cdot 80$$

These scores are then passed into the softmax function to get predicted probabilities:

$$\hat{Y}^{\text{Yes}} = \frac{e^{z_{\text{Yes}}}}{e^{z_{\text{Yes}}} + e^{z_{\text{No}}} + e^{z_{\text{Opt}}}}$$

$$\hat{Y}^{\text{No}} = \frac{e^{z_{\text{No}}}}{e^{z_{\text{Yes}}} + e^{z_{\text{No}}} + e^{z_{\text{Opt}}}}$$

$$\hat{Y}^{\text{Opt}} = \frac{e^{z_{\text{Opt}}}}{e^{z_{\text{Yes}}} + e^{z_{\text{No}}} + e^{z_{\text{Opt}}}}$$

Loss Function Depends on the Parameters

Now let's assume we have 3 data points, with respective true labels: Yes, No, Opt.
Then the loss function becomes:

$$L = -\frac{1}{3}\left[\log\left(\hat{Y}_1^{\text{Yes}}\right) + \log\left(\hat{Y}_2^{\text{No}}\right) + \log\left(\hat{Y}_3^{\text{Opt}}\right)\right]$$

Each predicted probability (e.g., $\hat{Y}_1^{Yes}$) depends on the values of:

- $\beta_0^{(1)}, \beta_1^{(1)}, \beta_2^{(1)}$ — the parameters for the "Yes" class

Similarly:

- $\hat{Y}_2^{\text{No}}$ depends on $\beta_0^{(2)}, \beta_1^{(2)}, \beta_2^{(2)}$
- $\hat{Y}_3^{\text{Opt}}$ depends on $\beta_0^{(3)}, \beta_1^{(3)}, \beta_2^{(3)}$

So, the loss ultimately depends on all 9 parameters:

$$\beta_0^{(1)}, \beta_1^{(1)}, \beta_2^{(1)}, \quad \beta_0^{(2)}, \beta_1^{(2)}, \beta_2^{(2)}, \quad \beta_0^{(3)}, \beta_1^{(3)}, \beta_2^{(3)}$$

Goal of Optimization

We aim to find the values of these 9 parameters that minimize the loss function.
This is typically done using gradient descent or other optimization algorithms.

Training Softmax Regression: An Optimization Problem

Softmax regression is essentially an optimization problem. Our objective is to find the best values for the model parameters (i.e., the coefficients for each class) that minimize the loss function (cross-entropy loss).

Step 1: Initialize Parameters

We begin by randomly initializing the model parameters. For a 3-class classification problem with 2 features (cgpa, iq) and a bias term, we have 9 parameters in total:

- Line 1 (Class 1): $\beta_0^{(1)}, \beta_1^{(1)}, \beta_2^{(1)}$
- Line 2 (Class 2): $\beta_0^{(2)}, \beta_1^{(2)}, \beta_2^{(2)}$
- Line 3 (Class 3): $\beta_0^{(3)}, \beta_1^{(3)}, \beta_2^{(3)}$

Step 2: Apply Gradient Descent

We use gradient descent to update the parameters and minimize the loss.
A general update rule for any parameter looks like:

$$\beta \leftarrow \beta - \eta \cdot \frac{\partial L}{\partial \beta}$$

Where:

- η is the learning rate
- $\frac{\partial L}{\partial \beta}$ is the gradient of the loss with respect to the parameter β

For example, for $\beta_0^{(1)}$:

$$\beta_0^{(1)} \leftarrow \beta_0^{(1)} - \eta \cdot \frac{\partial L}{\partial \beta_0^{(1)}}$$

We compute the gradients for all 9 parameters using the partial derivatives of the loss function:

$$\frac{\partial L}{\partial \beta_j^{(k)}} = -\frac{1}{n} \sum_{i=1}^{n} \left(Y_i^{(k)} - \hat{Y}_i^{(k)} \right) \cdot x_{ij}$$

Where:

- j ∈ {0,1,2} : refers to the bias, feature 1 (cgpa), and feature 2 (iq)
- k ∈ {1,2,3} : refers to the class index

We run this update iteratively for multiple epochs (e.g., 1000 times) until the loss converges.

Step 3: Final Model Parameters

After training, we obtain optimized values for the parameters:

- Line 1 (Class Yes): $\beta_0^{(1)}, \beta_1^{(1)}, \beta_2^{(1)}$
- Line 2 (Class No): $\beta_0^{(2)}, \beta_1^{(2)}, \beta_2^{(2)}$
- Line 3 (Class Opt): $\beta_0^{(3)}, \beta_1^{(3)}, \beta_2^{(3)}$

These parameter values are such that the loss function is minimized, i.e., the model has learned to best separate the classes.

Step 4: Making Predictions

Suppose we get a new data point with:

- cgpa = 6.5
- iq = 65

We compute the scores (logits) for each class:

$$z_1 = \beta_0^{(1)} + 6.5 \cdot \beta_1^{(1)} + 65 \cdot \beta_2^{(1)}$$

$$z_2 = \beta_0^{(2)} + 6.5 \cdot \beta_1^{(2)} + 65 \cdot \beta_2^{(2)}$$

$$z_3 = \beta_0^{(3)} + 6.5 \cdot \beta_1^{(3)} + 65 \cdot \beta_2^{(3)}$$

We then apply the softmax function to convert these scores into probabilities:

$$\hat{Y}^{(1)} = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \quad \hat{Y}^{(2)} = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \quad \hat{Y}^{(3)} = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

These three values represent the probabilities of the new point belonging to each class.

Final Prediction

Whichever class has the highest probability becomes the predicted class for the new data point.

For example:

- If $\hat{Y}^2$ is the highest, then the prediction is class "No".

From Multiclass Cross-Entropy to Binary Cross-Entropy

Let's now understand how the cross-entropy loss function for multiclass classification reduces to the binary cross-entropy loss when the number of classes k = 2 .

Multiclass Cross-Entropy Loss

The general form of the cross-entropy loss for k classes is:

$$L = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} Y_i^{(k)} \log(\hat{Y}_i^{(k)})$$

Where:

- n : number of data points
- K : number of classes
- $Y_i^{(k)}$: 1 if sample iibelongs to class kk, otherwise 0
- $\hat{Y}_i^{(k)}$: predicted probability that sample iibelongs to class kk

Binary Cross-Entropy Case (K = 2)

When the number of classes is 2, say class 0 and class 1, the formula becomes:

$$L = -\frac{1}{n}\sum_{i=1}^{n}\left[Y_i^{(1)} \log(\hat{Y}_i^{(1)}) + Y_i^{(0)} \log(\hat{Y}_i^{(0)})\right]$$

Since $Y_i^{(0)} = 1 - Y_i^{(1)}$ and $\hat{Y}_i^{(0)} = 1 - \hat{Y}_i^{(1)}$, we can substitute:

$$L = -\frac{1}{n}\sum_{i=1}^{n}\left[Y_i \log(\hat{Y}_i) + (1 - Y_i) \log(1 - \hat{Y}_i)\right]$$

This is the binary cross-entropy loss, where:

- $Y_i \in \{0,1\}$
- $\hat{Y}_i$: predicted probability of class 1 (from sigmoid)

- For binary classification, we use the sigmoid function and binary cross-entropy loss.
- For multiclass classification, we use the softmax function and categorical cross-entropy loss.

In Our Case

We have:

- 2 input features (cgpa, iq)
- 3 output classes (Yes, No, Opt-out)

This means:

- We use the softmax function for prediction
- We use the categorical cross-entropy loss for training
- There are 9 parameters to optimize (3 weights + 1 bias for each class)

We minimize the loss using gradient descent to find the optimal parameter values that best separate the classes.

When to use what?

Use One-vs-Rest (OVR) when:

1. Classes are Non-Mutually Exclusive: OVR is appropriate if an instance can belong to more than one class, as each classifier provides an independent probability foreach class.
2. Dealing with Imbalanced Data: OVR might perform better when class distribution is highly imbalanced since each class gets a dedicated model.

Use Multinomial Logistic Regression (SoftMax Regression) when:

1. Computational Efficiency is Required: Softmax Regression is generally more efficient for large datasets and a high number of classes.
2. Classes are Mutually Exclusive: SoftMax Regression is a good choice when each instance can only belong to one class. The SoftMax function provides a set of probabilities that sum to 1, fitting well with mutually exclusive classes.
3. Interpretability is Important: The probabilities output by SoftMax Regression are more interpretable than those from OVR, as they always sum to 1. This can make model predictions easier to explain.

Understanding Probability and Likelihood through Examples

Example 1: Coin Toss (Bernoulli Distribution)

Let’s begin with a simple random experiment: flipping a coin.
Suppose we are given a fair coin, which means it has two equally likely outcomes:

- Head (H)
- Tail (T)

We are told that the probability of getting a head is 0.5.
Now, what is the probability of getting tail in one flip?
Since the coin is fair, the answer is straightforward:
P(T) = 1−P(H) = 1−0.5 = 0.5
We can also model this situation using a Bernoulli distribution, which is used to model experiments with only two outcomes (success/failure, yes/no, head/tail).
The Bernoulli distribution has one parameter:

- p : probability of success (in our case, getting a head)
- q = 1−p : probability of failure (getting a tail)

The probability mass function (PMF) of the Bernoulli distribution is:

$$P(X = k) = p^k(1-p)^{1-k}, \quad \text{where } k \in \{0, 1\}$$

- If k = 1 , we are finding the probability of head → P(X = 1) = p
- If k=0 , we are finding the probability of tail → P(X = 0) = 1−p

So, using this function, we can determine the probability of an outcome based on a known parameter pp.
This process—using known parameters to compute the chance of an event—is what we call probability.
Definition of Probability:
Given a known distribution and known parameters, probability measures the chance of observing a particular event.

What is Likelihood?

Now let's look at the same experiment from a different perspective.
Suppose you flipped a coin 5 times and observed 5 heads in a row.
You begin to wonder: *Is the coin really fair?*
If the coin were truly fair, then the probability of getting 5 heads in a row would be:
L=$(0.5)^5$=0.03125
This value is very small, which raises doubt about whether the coin is actually fair.
This is the essence of likelihood.
In this case:

- The data is fixed: 5 heads
- The parameter (p) is uncertain
- We evaluate how likely it is that the observed data came from a distribution where p = 0.5

This is likelihood:
Definition of Likelihood:
Given observed data, likelihood measures how plausible a particular value of the model parameter is.
We can also try other values of p . Suppose we consider a biased coin where:

p = 0.7(probability of head)
Then the likelihood becomes:
L= $(0.7)^5 = 0.16807$
Now compare:
$(0.7)^5 > (0.5)^5$
This means:
Observing 5 heads is more likely if the coin has p = 0.7 than if it had p = 0.5
This demonstrates how likelihood helps us evaluate different parameter values based on the data we observe.

- When you know the parameter, and you want to compute the chance of some event → this is probability.
- When you observe the event, and you want to assess how well a parameter explains it → this is likelihood.

Both concepts are deeply connected, but the direction of inference is reversed.

Example 2: Drawing Balls from a Bag

Let's consider a bag that contains:

- 3 red balls
- 2 green balls

This is a closed bag, and we are randomly drawing one ball at a time.
Based on this setup, the probability of drawing a green ball is:
P(Green) = $\frac{2}{5}$

What kind of distribution does this random experiment follow?

Since we are drawing one ball and there are only two possible outcomes (Red or Green), this experiment follows a Bernoulli distribution.
In the Bernoulli distribution:

- Let success be defined as drawing a green ball.
- Then, the parameter pp(probability of success) is:
  p=P(Green)=$\frac{2}{5}$
- Accordingly, the probability of failure (drawing a red ball) is:
  1−p=$\frac{3}{5}$

The probability mass function (PMF) is again:

$$P(X = k) = p^k(1 - p)^{1-k}, \quad \text{where } k \in \{0, 1\}$$

Let's say we want to compute the probability of drawing a red ball:

- Let k=0 (representing red)

- Then:

$$P(X = 0) = (1 - p)^1 = 1 - \frac{2}{5} = \frac{3}{5}$$

So, we have:

- Known distribution: Bernoulli
- Known parameter: $p = \frac{2}{5}$
- Question: What is the chance of drawing a red ball?

This is a classic use of probability, because:

We know the parameter, and we are calculating the chance of an event.

Now, Let's Understand Likelihood

Suppose we perform this experiment 5 times and observe the following outcome:
All 5 draws result in green balls
This raises a natural question:
Is it still reasonable to believe that the probability of green is only $\frac{2}{5}$ *?*
Let's calculate the likelihood of seeing 5 green balls, assuming $p = \frac{2}{5}$:

$$L(p = 2/5 \mid 5 \text{ greens}) = \left(\frac{2}{5}\right)^5 = 0.01024$$

This is a very small number, indicating that:

- It is highly unlikely to observe 5 green balls in a row if the probability of green is truly $\frac{2}{5}$

Now, suppose instead the bag had:

- 4 green balls
- 1 red ball

Then, $p = \frac{4}{5}$. Let's compute the likelihood under this assumption:

$$L(p = 4/5 \mid 5 \text{ greens}) = \left(\frac{4}{5}\right)^5 = 0.32768$$

Compare the two likelihoods:

$$\left(\frac{4}{5}\right)^5 > \left(\frac{2}{5}\right)^5$$

This tells us:

- Observing 5 green balls is more likely if the bag has a higher proportion of green balls
- Therefore, the likelihood that $p = \frac{4}{5}$ is greater than the likelihood that

$$p = \frac{2}{5}$$

- If we know the parameter and ask about an event → this is probability.
- If we observe an event and ask about the parameter → this is likelihood.

Both are built on the same mathematical foundation but serve different purposes in statistical reasoning.

Example 3: Normal Distribution – Height of a Person

Let's take a real-world example of a normal distribution, which is a continuous probability distribution.

Scenario:

Suppose the heights of people in a population are normally distributed with:

- Mean μ= 150cm
- Standard deviation σ= 10cm

So the distribution is:

$X \sim N(150, 10^2)$

Probability

Let's say we are asked:
What is the probability that a randomly selected person has a height between 170 cm and 180 cm?
Since we know:

- The type of distribution (Normal)
- The parameters (μ=150 , σ=10)
- And we are calculating the chance of a specific event (height in a certain range),

This is a probability problem.
To calculate:
$P(170 < X < 180)$
This corresponds to the area under the curve of the normal distribution from 170 to 180. When we compute this area (either through z-scores or a statistical tool), we find:
$P(170 < X < 180) \approx 0.02$
So there is a 2% chance that a randomly chosen person has height between 170 cm and 180 cm.

Likelihood

Now let's flip the scenario.
Suppose we observe a person whose height is 100 cm.

Now the question becomes:
How likely is it that this person's height was drawn from a normal distribution with μ=150, σ=10 ?
This is a likelihood problem because we already have the data, and we are asking about how plausible a specific set of parameters is.
To compute the likelihood, we use the probability density function (PDF) of the normal distribution:

$$L(\mu, \sigma \mid X = 100) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Plugging in the values:

$$L(150, 10 \mid X = 100) = \frac{1}{\sqrt{2\pi(10^2)}} \cdot e^{-\frac{(100-150)^2}{2(10^2)}}$$

$$= \frac{1}{\sqrt{2\pi \cdot 100}} \cdot e^{-\frac{2500}{200}} = \frac{1}{\sqrt{628.318}} \cdot e^{-12.5}$$

$$\approx 1.47 \times 10^{-7}$$

This is a very small value, suggesting that:
Given a height of 100 cm, the likelihood that this data came from a normal distribution with μ=150, σ=10 is extremely low.

Varying the Data Point

Let's see how likelihood changes as we change the observed height:

Height (x)	Likelihood L(μ=150,σ=10|x)	Interpretation
100 cm	1.47×10^{-7}	Very unlikely under this distribution
130 cm	≈0.005	Still low, but more plausible
140 cm	≈0.2419	Much more likely
150 cm	≈ 0.3989(maximum)	Peak of the distribution
200 cm	$\ll 0.005$	Very unlikely

As expected, likelihood is maximum when the observed data is equal to the mean of the distribution (i.e., 150 cm), and drops off as we move further away.

Final Insight: Probability vs Likelihood

Concept	What is Given	What is Being Evaluated	Type
Probabil	Known distribution +	Find the chance of observing an	Forwar

ity	parameters	event	d
Likelihood	Known observed data (event)	Evaluate how well a parameter explains the observed data	Reverse

- Probability: You know the distribution and its parameters. You're finding how likely it is to observe a certain event.
- Likelihood: You observe the event (data), and you ask: *How well does a certain parameter explain this data?*

Probability: This is a measure of the chance that a certain event will occur out of all possible events. It's usually presented as a ratio or fraction, and it ranges from 0 (meaning the event will not happen) to 1 (meaning the event is certain to happen).

Likelihood: In a statistical context, likelihood is a function that measures the plausibility of a particular parameter value given some observed data. It quantifies how well a specific outcome supports specific parameter values.

More Definitions

A probability quantifies how often you observe a certain outcome of a test, given a certain understanding of the underlying data.

A likelihood quantifies how good one's model is, given a set of data that's been observed.

Probabilities describe test outcomes, while likelihoods describe models.

Understanding Likelihood and Probability through Examples

We discussed three examples to understand the concept of likelihood and probability:

1. **Coin Toss**
2. **Balls in a Bag**
3. **Normal Distribution**

In all these cases, we deal with a **parameter**.

- In the **coin toss** example, the parameter is the probability of getting a head (say, p).
- In the **balls in a bag** example, the parameter might be the probability of drawing a green ball.
- In the **normal distribution**, the parameters are typically the **mean (μ)** and **standard deviation (σ)**.

Let’s focus on the coin toss to understand likelihood in more depth.

Example: Coin Toss and Likelihood

Assume the probability of getting a head with a fair coin is p=0.5p = 0.5. Now, suppose we toss the coin 5 times and observe the outcome: **HHHHH** (i.e., 5 heads in a row).

We want to calculate the **likelihood** of observing this outcome under the assumption that the true probability of head is 0.5.

L(p=0.5 | HHHHH) = $(0.5)^5$

This gives the likelihood of observing the data given the parameter p = 0.5.

What if the Coin is Biased?

Let's assume the coin is **slightly biased**, and the probability of getting a head is p = 0.6.

Then the likelihood becomes:

L(p=0.6 | HHHHH) = $(0.6)^5$

Now compare:

- $(0.6)^5 > (0.5)^5$

This suggests that the data (5 heads in a row) is **more likely** under the assumption that p = 0.6 than p = 0.5.

Hence, the value p = 0.6 **better explains** the observed data.

Towards Maximum Likelihood Estimation (MLE)

Now, consider what value of ppwould **maximize** the likelihood of the observed outcome (HHHHH).

Clearly, if the coin is **completely biased** towards heads, i.e., p=1, then:

L(p=1 | HHHHH) = $(1)^5$ = 1

This is the **maximum possible value** for likelihood in this scenario, since all heads are guaranteed when p=1.

This leads us to the key idea:

Maximum Likelihood Estimation (MLE) is the process of choosing the parameter value that **maximizes** the likelihood of the observed data.

In our example, given the data HHHHH, the **likelihood is maximized when p = 1**.

Key Concepts Recap

Probability: Given a model (i.e., a known p), what is the chance of observing a specific outcome?

$$P(\text{data} \mid \text{parameter})$$

Likelihood: Given the observed data, how likely is a particular value of the parameter?

$$L(\text{parameter} \mid \text{data})$$

MLE: Find the parameter value that **maximizes** the likelihood of the observed

data.

Maximum Likelihood Estimation (MLE) – Example 2

We have a bag that contains:

- 2 green balls
- 3 red balls

So, the total number of balls = 5

The probability of drawing a green ball in a single draw (with replacement) is:
P(green) = 2/5
This scenario follows a **Bernoulli distribution** for each draw (success = green ball).
Now, we perform **5 independent draws with replacement**, and each time, we get a **green ball**. So, the observed data is:
G G G G G

Likelihood Calculation

Let's define the parameter p as the probability of drawing a green ball.
We want to evaluate how well different values of p explain our observed data (GGGGG).

Case 1: p = 2/5

Given this value of p, the likelihood of observing five greens in a row is:

$$L\ (p = 2/5 \mid GGGGG) = \left(\frac{2}{5}\right)^5$$

Case 2: p = 3/5

$$L(p = 3/5 \mid GGGGG) = \left(\frac{3}{5}\right)^5$$

Clearly:

$$\left(\frac{3}{5}\right)^5 > \left(\frac{2}{5}\right)^5$$

So, the likelihood is higher for p = 3/5 than for p = 2/5 , given the data.

Case 3: p = 1

If we assume the probability of getting a green ball is 1 (i.e., all balls are green):

$$L(p=1 \mid GGGGG) = (1)^5 = 1$$

Now,

$$1 > (3/5)^5 > (2/5)^5$$

The value of p that **maximizes** the likelihood of observing the data is:

$$\hat{p} = 1$$

This is called the **Maximum Likelihood Estimate (MLE)** of p based on the observed data.

It means that **given all 5 observed balls were green**, the best explanation (highest likelihood) is that the bag only contains green balls — i.e., probability of green = 1.

Conceptual Notes:

- **Likelihood vs Probability:**
  - **Probability**: Given a parameter value, what's the chance of some data?
  - **Likelihood**: Given some data, how likely is a parameter value?
- **MLE** always finds the parameter value that **maximizes the likelihood** of the observed data.

Example 3: Maximum Likelihood Estimation in a Normal Distribution

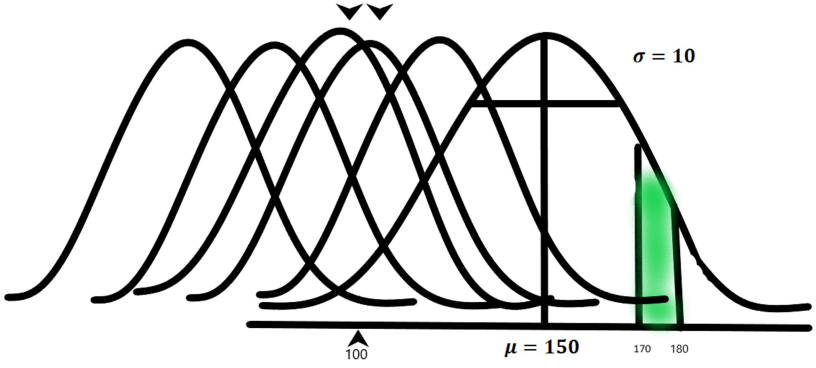
Let's consider a third example to understand Maximum Likelihood Estimation (MLE) more clearly.
Suppose we have a normal distribution with a mean μ = 150 and a standard deviation σ = 10.
Now, we randomly select one individual and observe that their height is 100. We want to evaluate how likely it is that this height of 100 came from the given normal distribution. This is where we calculate the likelihood of the observed data point under the specified distribution.
The likelihood function for a normal distribution is given by:

$$L(\mu = 150, \sigma = 10 \mid X = 100) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

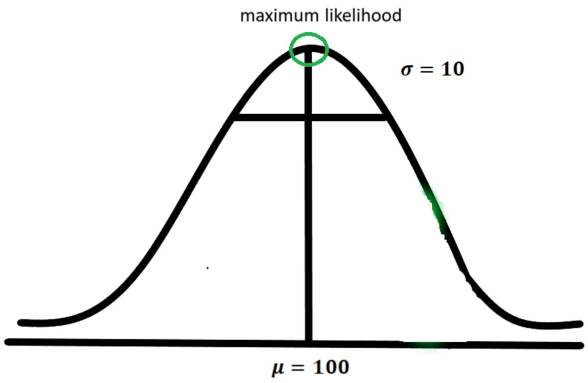
Substituting the values:

$$L(150, 10 \mid 100) = \frac{1}{\sqrt{2\pi \cdot 10^2}} e^{-\frac{(100-150)^2}{2 \cdot 10^2}} = \frac{1}{\sqrt{200\pi}} e^{-12.5}$$

This value turns out to be very small, indicating that the observed data point (height = 100) is quite unlikely under the distribution with μ = 150 , σ = 10.
However, this does not mean that the data doesn't come from a normal distribution at all — it might be coming from a different normal distribution with a different mean and/or standard deviation.



So we ask: Which normal distribution makes this data point most likely?
In other words, we try to find the parameters (like μ) of a normal distribution that maximize the likelihood of the observed data. This process is known as Maximum Likelihood Estimation (MLE).
In our example, clearly, a normal distribution with μ=100 and the same σ=10 would give the highest likelihood for the data point 100. So, under MLE, we would estimate that the mean μ of the distribution is 100 based on this observed data.



What is Maximum Likelihood Estimation?

Maximum Likelihood Estimation (MLE) is a method for estimating the parameters of a statistical model. Given observed data, it finds the values of parameters that maximize the likelihood of the data under that model.

- In the coin toss example, we used the observed outcomes (e.g., HHHH or HTHT) to estimate the probability of heads pp, such that the probability (likelihood) of the observed sequence is maximized.
- In the balls in a bag example, we found the proportion (or count) of colored balls that makes the observed draws most likely.
- In the normal distribution example, we estimated the mean μ that makes the observed height (100) most likely under a normal distribution.

In all cases, the observed data is fixed. The likelihood is treated as a function of the parameters, and we adjust those parameters to maximize the likelihood.
This is the core idea of MLE:
Find the parameter values that make the observed data most probable.

Understanding Maximum Likelihood Estimation with Normal Distribution

Recently, we discussed the concept of a normal distribution. Suppose we take a single random data point with a value of 100, and we know that it follows a normal distribution. If it indeed follows a normal distribution, then the most likely

value of the mean (μ) is 100. At this point, the distribution achieves its maximum likelihood because the mean matches the data point exactly. In essence, maximum likelihood occurs when the value of μ equals the value of the observed data point.

However, a valid question arises: in real-world scenarios, we rarely deal with just one data point. Instead, we often have multiple observations. For example, assume we collect age data for 100 individuals. We determine that this dataset follows a normal distribution using techniques like the Shapiro-Wilk test, Q-Q plot, etc. But while we confirm the data follows a normal distribution, we don't know which specific normal distribution it follows.

In other words, we don't know the actual values of the mean (μ) and standard deviation (σ). For instance, it could be a distribution where μ = 20 and σ = 5, or μ = 40 and σ = 5, or μ = 50 and σ = 3. These are all valid normal distributions but differ based on the values of μ and σ.

So, even if we know the data follows a normal distribution, the exact parameters (μ and σ) are unknown. A normal distribution is fully defined by its mean and standard deviation. To determine these parameters, we apply Maximum Likelihood Estimation (MLE).

Based on the observed data, we use MLE to estimate the values of μ and σ that maximize the likelihood function. In other words, we are trying to find the specific values of μ and σ that make the observed data most probable under the normal distribution. These estimates give us the best-fitting normal distribution for our dataset.

Strategy to Calculate Maximum Likelihood Estimation (MLE)

Let's learn how to calculate the parameters (mean and standard deviation) using Maximum Likelihood Estimation (MLE).

We need to compute the likelihood:

$$L(\mu, \sigma \mid X_1, X_2, X_3, \ldots, X_N)$$

We start by choosing random values for μ (mean) and σ (standard deviation). For example:

- μ = 100
- σ = 10

Then we compute the likelihood for this pair:

$$L(\mu = 100, \sigma = 10 \mid X_1 = 61, X_2 = 32, \ldots, X_N = 89) = 0.89$$

Next, we change the value of σ and calculate the likelihood again:

$$L(\mu = 100, \sigma = 20 \mid X_1 = 61, X_2 = 32, \ldots, X_N = 89) = 0.79$$

We repeat this process for more combinations of μ and σ. Among all the computed likelihood values, we select the pair (μ, σ) that gives the maximum likelihood.

The values of μ and σ that maximize the likelihood function are our estimated

parameter values.
This is the strategy we use to perform Maximum Likelihood Estimation.

Step 1: Likelihood with a Single Data Point

Assume that for the time being, we do not have the entire dataset—only one data point, x_1.
The likelihood function is:

$$\mathrm{L}(\mu, \sigma \mid \mathrm{x}_1) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_1-\mu)^2}{2\sigma^2}}$$

Step 2: Likelihood with Multiple Data Points

Now, suppose we have multiple data points: $x_1, x_2, x_3, \ldots, x_n$.
Only the x values change in the formula.
Let's assume that all data points are independent.
Hence, the joint likelihood is the product of the individual likelihoods:

$$L(\mu, \sigma \mid x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$$

This expands to:

$$= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_1-\mu)^2}{2\sigma^2}} * \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_2-\mu)^2}{2\sigma^2}} *\ldots* \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_n-\mu)^2}{2\sigma^2}}$$

So if we have n independent observations, we multiply their individual likelihoods to get the overall likelihood.

Step 3: Taking the Logarithm (Log-Likelihood)

To simplify this product, we take the natural logarithm of both sides:

$$\log\left(L(\mu, \sigma \mid x_1, x_2, \ldots, x_n)\right)$$

Using the logarithmic identity log(ab) = log a + log b, the expression becomes:

$$= \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_1-\mu)^2}{2\sigma^2}}\right) + \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_2-\mu)^2}{2\sigma^2}}\right) + \cdots + \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_n-\mu)^2}{2\sigma^2}}\right)$$

We will now simplify one term, and since all terms follow the same pattern, the same simplification applies to all of them.

$$\log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_1-\mu)^2}{2\sigma^2}}\right)$$

Take a = $\frac{1}{\sqrt{2\pi\sigma^2}}$

B = $e^{-\frac{(x_1-\mu)^2}{2\sigma^2}}$

Log(ab) = log(a) + log(b)

$$\text{Log}\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \log\left(e^{-\frac{(x_1-\mu)^2}{2\sigma^2}}\right)$$

$$\text{Log}\,(2\pi\sigma^2)^{-\frac{1}{2}} - \frac{(x_1-\mu)^2}{2\sigma^2}$$

$$-\frac{1}{2}\text{Log}\,(2\pi\sigma^2) - \frac{(x_1-\mu)^2}{2\sigma^2}$$

Take a = 2π , b = σ^2

$$-\frac{1}{2}\text{Log}\,2\pi - \frac{1}{2}log\sigma^2 - \frac{(x_1-\mu)^2}{2\sigma^2}$$

$$-\frac{1}{2}\text{Log}\,2\pi - \frac{2}{2}log\sigma - \frac{(x_1-\mu)^2}{2\sigma^2}$$

For first term we get

$$-\frac{1}{2}\text{Log}\,2\pi - log\sigma - \frac{(x_1-\mu)^2}{2\sigma^2}$$

Similarly for other terms we will also get

$$= -\frac{1}{2}\text{Log}\,2\pi - log\sigma - \frac{(x_1-\mu)^2}{2\sigma^2} - \frac{1}{2}\text{Log}\,2\pi - log\sigma - \frac{(x_2-\mu)^2}{2\sigma^2} \ldots\ldots - \frac{1}{2}\text{Log}\,2\pi - log\sigma - \frac{(x_n-\mu)^2}{2\sigma^2}$$

$$= -\frac{n}{2}\text{Log}\,2\pi - log\sigma - \frac{(x_1-\mu)^2}{2\sigma^2} - \frac{(x_2-\mu)^2}{2\sigma^2} \ldots\ldots \frac{(x_n-\mu)^2}{2\sigma^2}$$

To find the best values of μ (mu) and σ (sigma), we compute the log-likelihood for different combinations of μ and σ. By evaluating the log-likelihood over a range (theoretically infinite) of values, we aim to identify the combination of μ and σ that maximizes the log-likelihood function.

If we have only one data point, say 100, we can try different normal distributions to evaluate how likely it is to observe that data point under each distribution:

- First, we try a normal distribution with some μ and σ, and the likelihood is low.

- Then, we try a second distribution, and the likelihood increases.
- After that, we try a third distribution, and the likelihood decreases again.

This pattern gives us an important insight. Imagine a graph where:

- The x-axis represents the value of μ (mu).
- The y-axis represents the likelihood of the data under the normal distribution.

As we try different values of μ:

- When μ is small, the likelihood is low.
- As μ moves closer to the data point, the likelihood increases.
- After a certain point, increasing μ further causes the likelihood to decrease.

This implies that the graph of likelihood vs. μ has a maximum point—there is a peak in the curve.

To find this maximum point mathematically, we differentiate the log-likelihood function with respect to μ. The derivative represents the slope of the log-likelihood curve. The slope is zero at the maximum. So, to find the value of μ that gives us the highest likelihood, we solve:

$$\frac{\partial \log(L)}{\partial \mu} = 0$$

$$\frac{\partial \log(L)}{\partial m} = \frac{(\mathrm{x}_1 - \mu)^2}{\sigma^2} - \frac{(x_2 - \mu)^2}{\sigma^2} \ldots\ldots \frac{(x_n - \mu)^2}{\sigma^2}$$

$$(\mathrm{x}_1 - \mu) + (\mathrm{x}_2 - \mu) + \ldots\ldots + (\mathrm{x}_\mathrm{n} - \mu) = 0$$

$$\mathrm{n}\mu = \mathrm{x}_1 + \mathrm{x}_2 + \mathrm{x}_3 + \mathrm{x}_4 + \ldots\ldots + \mathrm{x}_\mathrm{n}$$

$$\mu = \frac{\mathrm{x}_1 + \mathrm{x}_2 + \mathrm{x}_3 + \mathrm{x}_4 + \ldots\ldots + \mathrm{x}_\mathrm{n}}{\mathrm{n}}$$

This derivation shows that if we have n data points and we assume they follow a normal distribution, then the value of μ that maximizes the likelihood is simply the mean of the observed data points. This is a key result in Maximum Likelihood Estimation (MLE).

Now, we will apply the same logic to σ (sigma) that we previously applied to μ (mu).

We try different values of σ and observe how the shape of the normal distribution changes. Specifically, as σ changes, the width (spread) of the curve increases or decreases. Just like with μ, if we plot the likelihood as a function of σ, we will again observe a maximum point.

To find the value of σ that maximizes the likelihood, we again differentiate the log-likelihood function with respect to σ, set the derivative (slope) to zero, and solve for σ.

$$\log(L) = -\frac{n}{2}\log(2\pi) - n\log(\sigma) - \frac{(x_1-\mu)^2}{2\sigma^2} - \frac{(x_2-\mu)^2}{2\sigma^2} - \ldots - \frac{(x_n-\mu)^2}{2\sigma^2}$$

Taking the Derivative with Respect to σ:

$$\frac{\partial \log(L)}{\partial \sigma} = -\frac{n}{\sigma} + \frac{(x_1-\mu)^2}{\sigma^3} + \frac{(x_2-\mu)^2}{\sigma^3} + \ldots + \frac{(x_n-\mu)^2}{\sigma^3}$$

Set the derivative equal to zero:

$$\frac{(x_1-\mu)^2}{\sigma^3} + \frac{(x_2-\mu)^2}{\sigma^3} + \ldots + \frac{(x_n-\mu)^2}{\sigma^3} = \frac{n}{\sigma}$$

Multiply both sides by σ:

$$\frac{(x_1-\mu)^2}{\sigma^2} + \frac{(x_2-\mu)^2}{\sigma^2} + \ldots + \frac{(x_n-\mu)^2}{\sigma^2} = n$$

Solving for Variance:

$$\sigma^2 = \frac{(x_1-\mu)^2 + (x_2-\mu)^2 + \ldots + (x_n-\mu)^2}{n}$$

Solving for Standard Deviation:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n}}$$

This means that the value of σ (standard deviation) which maximizes the likelihood is the standard deviation of the observed data.
So, if we are given a dataset (for example, a list of ages) and we assume it follows a normal distribution, then:

- μ (mu) is the mean of all data points.
- σ (sigma) is the standard deviation of all data points.

These values are the maximum likelihood estimates of the parameters of the normal distribution.

MLE IN MACHINE LEARNING

In any dataset where the distribution is known, we can use Maximum Likelihood Estimation (MLE) to determine the parameters of that distribution.

Given a set of data points, we first assume a specific distribution. We then construct a likelihood function and find the parameter values that maximize this function. This process can be broken down into three steps.

Now, the question arises: how can we apply this flow in machine learning?

In machine learning, we work with training data, which consists of input features $(x_1, x_2, x_3, \ldots\ldots, x_n)$ and an output variable (y). If (y) has two classes, such as "yes" and "no," it follows a Bernoulli distribution. If (y) has multiple classes, it follows a multinomial or multinoulli distribution.

The process of applying Maximum Likelihood Estimation in machine learning can be outlined in the following steps:

1. Identify the Distribution : We first aim to determine the distribution of (y) given (x), denoted as (P(y|x)).

1. Select a Model : Based on the identified distribution, we choose a parametric machine learning model. MLE techniques are applicable only to models that are parametric in nature.

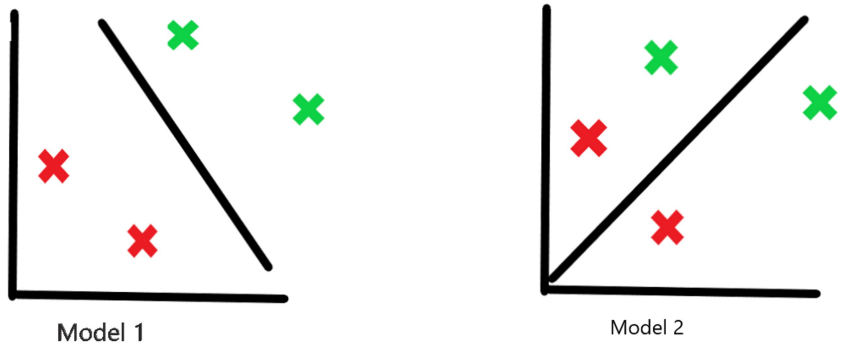
1. Initialize Parameters : We randomly assign initial values to our model parameters.

1. Choose a Likelihood Function : We select a likelihood function that corresponds to the distribution of the data.

1. Optimize Parameters : We then find the parameter values that maximize the likelihood function. At this point, our model is considered trained.

we have utilized the concept of Maximum Likelihood Estimation to train our model. This means that if we have a parametric function, we can train our model using MLE to find the parameter values that yield the maximum likelihood.

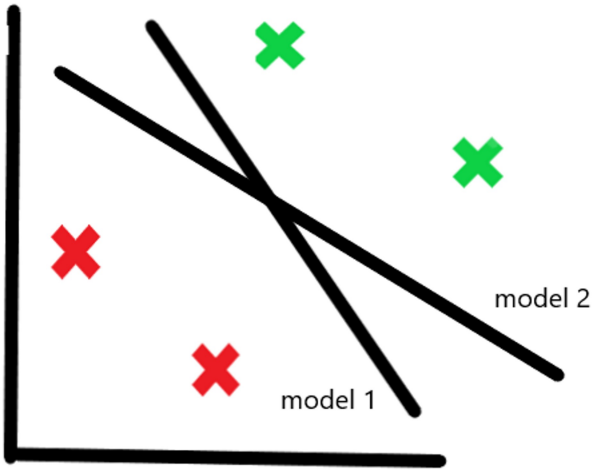
MLE IN LOGISTIC REGRESSION



We have four training points and two models. The question is: Which model is best?

The answer is: Model 1 is better because it is able to separate red and green points onto different sides of the decision boundary. Model 2 fails to do this.

Now, if we have two models like this:



How do we choose between them?
Both models correctly classify all points, so how do we differentiate which one is better?
We can use the sigmoid function on each point to calculate the probability of that point being green. Once we have the probabilities for all points, we multiply the probabilities of the actual outcomes:

- Probability of green points being predicted as green
- Probability of red points being predicted as red

This concept is known as Maximum Likelihood Estimation (MLE).
We take the probability of the actual class:

- For green points → probability of being predicted as green
- For red points → probability of being predicted as red

We calculate the MLE for each line. The line with the highest MLE is considered the best.

MLE Derivation: Step-by-Step

Suppose we have data like x_1 , x_2 , x_3 , … | y
There are n input columns and one output column.
We first need to understand what the output depends on: it clearly depends on the input features x . So, we write this relationship as:

y | x

Step 1: Assume a Distribution for the Output (y)
In logistic regression, this step is straightforward.
Since the output column consists of 0s and 1s (binary classification), the natural

choice is the Bernoulli distribution.
You might wonder — why not a Binomial distribution?
Here's the reason:

- Each row in the dataset represents a single observation or query point
- Each prediction is made independently
- Each output can only take two values: 0 or 1

While it may look like a binomial distribution because we have many rows, the actual prediction happens one point at a time. So each output is best modeled using the Bernoulli distribution.
However, if we were asked: *How many 1s are predicted out of 100 points?* — then we'd use the Binomial distribution.
Summary:
We assume that the output variable y follows a Bernoulli distribution in logistic regression.

Step 2: Choose a Parametric Model
We choose a parametric machine learning model — specifically, Logistic Regression.
This means that the relationship between x and y is modeled using a function (sigmoid), and the function depends on parameters (the coefficients). These parameters will be learned using the MLE method.

Step 3: Initialize Coefficients
We randomly choose the initial values for the coefficients (also called model parameters). This gives us a starting point for optimization.

Step 4: Assume a Likelihood Function
We need to assume a likelihood function based on the distribution of the output variable y . Since y is binary (0 or 1), we assume it follows a Bernoulli distribution.
So, for a single data point:

$$L(Y|X;\beta) = p^k(1-p)^{(1-k)}$$

Where:

- P is the predicted probability of y = 1
- 1 − p is the predicted probability of y = 0
- k ∈ {0,1} is the actual observed value from the output column

Likelihood Function for Multiple Data Points

Let us define the likelihood of y given x , parameterized by β:

$$L (Y | X ; \beta)$$

Assume all data points are independent. Then, for n data points:

$$L(Y|X;\beta) = [p_1^{y_1}(1-p_1)^{(1-y_1)}] \cdot [p_2^{y_2}(1-p_2)^{(1-y_2)}] \cdot \dots \cdot [p_n^{y_n}(1-p_n)^{(1-y_n)}]$$

For mathematical convenience, we transform the Bernoulli probability mass function (PMF) from:

pk + (1−p) (1−k)

to the equivalent exponential form:

$$p^y(1-p)^{1-y}$$

This transformation is helpful because both expressions give the same result, but the exponential form simplifies the process of taking the logarithm, which we need for optimization.

L(Y|X;β) = $p_1^{y_1}(1-p_1)^{(1-y_1)} * p_2^{y_2}(1-p_2)^{(1-y_2)} \dots . * \ p_n^{y_n}(1-p_n)^{(1-y_n)}$

$$\mathrm{L(Y|X;\beta)} = \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{(1-y_i)}$$

Now we take the logarithm of both sides of the likelihood function:

$$\log(\mathrm{l}) = \sum_{i=1}^{n} log(p_i^{y_i}(1-p_i)^{(1-y_i)})$$

We break this product into two parts using logarithmic properties:

$$\log(\mathrm{l}) = \sum_{i=1}^{n} \left[\log\left(p_i^{y_i}\right) + log((1-p_i)^{(1-y_i)})\right]$$

Applying the power rule of logarithms:

$$\log(\mathrm{l}) = \sum_{i=1}^{n} [y_i \log(p_i) + (1-y_i)\, log(1-p_i)]$$

This is our log-likelihood function

$$\log(\mathrm{l}) = \sum_{i=1}^{n} [y_i \log(p_i) + (1-y_i)\, log(1-p_i)]$$

We have to maximize this term , we want that values of coefficients for which this will give maximum value , we can apply gradient descent technique to find that values of coefficients.We aim to maximize this log-likelihood function. In other words, we want to find the values of the model coefficients (parameters) that maximize this expression. To achieve this, we can apply an optimization technique such as gradient descent, which iteratively adjusts the coefficients to reach the point where the log-likelihood is at its maximum.

SOME IMPORTANT QUESTIONS

1. Is MLE a general concept applicable to all machine learning algorithms?

- Maximum Likelihood Estimation (MLE) is a general statistical concept that can be applied to many machine learning algorithms, particularly those that are parametric (i.e., defined by a set of parameters), but it's not applicable to all machine learning algorithms. MLE is commonly used in algorithms such as linear regression, logistic regression, and neural networks to find the optimal values of the parameters that best fit the training data. However, some machine learning algorithms don't rely on MLE. For example:

1. Non-parametric methods: Some machine learning methods, such as k-Nearest Neighbors (k-NN) and Decision Trees, are non-parametric and do not make strong assumptions about the underlying data distribution. These methods don't have a fixed set of parameters that can be optimized using MLE.
2. Unsupervised learning algorithms: Some unsupervised learning algorithms, like K-means clustering, use different objective functions, not necessarily tied to a probability distribution.
3. Reinforcement Learning: Reinforcement Learning methods generally don't use MLE, as they are more focused on learning from rewards and punishments over a sequence of actions rather than fitting to a specific data distribution.

2. How is MLE related to the concept of loss functions?

- In machine learning, a loss function measures how well a model's predictions align with the actual values. The goal of training a machine learning model is often to find the model parameters that minimize the loss function. Maximum Likelihood Estimation (MLE) is a method of estimating the parameters of a statistical model to maximize the likelihood function, which is conceptually similar to minimizing a loss function. In fact, for many common models, minimizing the loss function is equivalent to maximizing the likelihood function. MLE and the concept of loss functions in machine learning are closely related. Many common loss functions can be derived from the principle of maximum likelihood estimation under certain assumptions about the data or the model. By minimizing these loss functions, we're effectively performing maximum likelihood estimation.

3. Then why does loss function exist, why don't we maximize likelihood?

- The confusion arises from the fact that we're using two different perspectives to look at the same problem. In many machine learning algorithms, the aim is to minimize the difference between the predicted and actual values, and this is typically represented by a loss function. When we talk about minimizing the loss function, it's essentially the same as saying we're trying to find the best model parameters that give us the closest predictions to the actual values. On the other

hand, when we look at the problem from a statistical perspective, we talk in terms of maximizing the likelihood of seeing the observed data given the model parameters. This is represented by a likelihood function. For many models, these two perspectives are equivalent - minimizing the loss function is the same as maximizing the likelihood function. In fact, many common loss functions can be derived from the principle of MLE under certain assumptions about the data. So why do we often talk about minimizing the loss function instead of maximizing the likelihood? There are a few reasons:

1. Computational reasons: It's often easier and more computationally efficient to minimize a loss function than to maximize a likelihood function. This is particularly true when working with complex models like neural networks.

2. Generalization: The concept of a loss function is more general and can be applied to a wider range of problems. Not all machine learning problems can be framed in terms of maximizing a likelihood. For example, many non-parametric methods and unsupervised learning algorithms don't involve likelihood.

3. Flexibility: Loss functions can be easily customized to the specific needs of a problem. For instance, we might want to give more weight to certain types of errors, or we might want to use a loss function that is robust to outliers.

4. Then why study about maximum likelihood at all?

- The study of Maximum Likelihood Estimation (MLE) is essential for several reasons, despite the prevalence of loss functions in machine learning:

1. Statistical Foundation: MLE provides a strong statistical foundation for understanding machine learning models. It gives a principled way of deriving the loss functions used in many common machine learning algorithms, and it helps us understand why these loss functions work and under what assumptions.

2. Interpretability: The MLE framework gives us a way to interpret our model parameters. The MLEs are the parameters that make the observed data most likely under our model, which can be a powerful way of understanding what our model has learned.

3. Model Comparison: MLE gives us a way to compare different models on the same dataset. This can be done using tools like the Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC), which are based on the likelihood function and can help us choose the best model for our data.

4. Generalization to Other Methods: MLE is a specific case of more general methods, like Expectation-Maximization and Bayesian inference, which are used in more complex statistical modeling. Understanding MLE can provide a stepping stone to these more advanced topics.

5. Deeper Understanding: Lastly, understanding MLE can give us a deeper understanding of our models, leading to better intuition, better model selection, and ultimately, better performance on our machine learning tasks. In short, while

you can often get by with a practical understanding of loss functions and optimization algorithms in applied machine learning, understanding MLE can be extremely valuable for gaining a deeper understanding of how and why these models work.

In short, while you can often get by with a practical understanding of loss functions and optimization algorithms in applied machine learning, understanding MLE can be extremely valuable for gaining a deeper understanding of how and why these models work.

Assumptions of Logistic Regression

Logistic regression, like other statistical methods, relies on certain assumptions. Here are the main assumptions of logistic regression:

1. Binary Logistic Regression requires the dependent variable to be binary:
   That means the outcome variable must have two possible outcomes, such as "yes" vs "no", "success" vs "failure", "spam" vs "not spam", etc.
2. Independence of observations:
   The observations should be independent of each other. In other words, the outcome of one instance should not affect the outcome of another.
3. Linearity of independent variables and log odds:
   Although logistic regression does not require the dependent and independent variables to be related linearly, it requires that the independent variables are linearly related to the log odds.
4. Absence of multicollinearity:
   The independent variables should not be too highly correlated with each other, a condition known as multicollinearity. While not a strict assumption, multicollinearity can be a problem because it can make the model unstable and difficult to interpret.
5. Large sample size:
   Logistic regression requires a large sample size. A general guideline is that you need at least 10 cases with the least frequent outcome for each independent variable.
   For example, if you have 5 independent variables and the expected probability of your least frequent outcome is 0.10, then you would need a minimum sample size of:
   $500 = (10 \times 5) / 0.10$

Note: That violating these assumptions doesn't mean you can't or shouldn't use logistic regression, but it may impact the validity of the results and you should proceed with caution.

Odds and log(odds)

Odds:
The odds of an event is the ratio of the probability of the event happening (P) to the probability of the event not happening (1 - P). It's a way of expressing the likelihood of an event. If the odds are greater than 1, the event is more likely to happen than not, and vice versa.

Suppose you roll a die, and you are asked to find the odds of getting a 3.

1. Find the probability of getting a 3:
   The probability of rolling a 3 on a fair die is P(3) = $\frac{1}{6}$
2. Find the probability of not getting a 3:
   The probability of not rolling a 3 is P (not 3) = 1 − P(3) = $\frac{5}{6}$
3. Calculate the odds of getting a 3:
   The odds are the ratio of the probability of success (getting a 3) to the probability of failure (not getting a 3). So, the odds are:
   $$\text{Odds} = \frac{P(\text{success})}{P(\text{failure})} = \frac{\frac{1}{6}}{\frac{5}{6}} = \frac{1}{5} = 0.2$$
4. This means, "the odds of getting a 3 are 1 in 5 times."

General Formula for Odds:

In general, if p is the probability of success, the odds are given by the formula:
$$\text{Odds} = \frac{p}{1-p}$$
where p lies between 0 and 1, and the odds range from 0 to infinity.

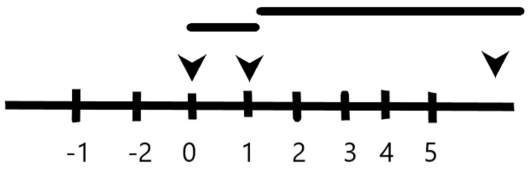
Log(Odds)

Sometimes, when dealing with odds, it can be difficult to interpret and compare them directly, especially when they vary widely. To address this, we take the logarithm of the odds, also known as log-odds, to make comparisons easier and to transform the scale into a more symmetric one.

1. Understanding the Concept of Odds:
   - Suppose we have a team that loses 1 match and wins 4 matches. The odds of winning for this team are:
     Odds of winning = 4/1 = 4
   - On the other hand, another team wins 1 match and loses 4. The odds of winning for this team are:

Odds of winning = 1/4 = 0.25

2. Problem with Unevenness:
    - If we plot these odds on a number line:
        - The odds of the first team (4) would be a large value.
        - The odds of the second team (0.25) would be a small value.
    - This creates an asymmetry. The team with more wins has odds between 1 and infinity, while the team with more losses has odds between 0 and 1. Understanding and comparing these two on the same scale can be challenging.
3. Symmetry through Logarithms:
    - To make the scale symmetric and easier to compare, we take the logarithm of the odds.
    - Applying the logarithm to the odds:
      log (4) = 0.602
      log (0.25) = −0.602
    - After applying the logarithm, the difference between the two values becomes smaller, and the scale becomes more symmetric. Now both values can be compared more easily.



Why Take Log(Odds)?

- Taking the log of odds transforms the scale, making it symmetrical and easier to interpret. It helps overcome the challenge of comparing probabilities or odds that differ significantly.
- In practice, log-odds are often used in logistic regression to model probabilities, as they can take any real value, making them more useful for analysis.

Another interpretation of logistic regression

Understanding How Log(Odds) is Connected to Logistic Regression
In logistic regression, the relationship between probability and predictors is modeled using the log of the odds (log-odds):

$$\text{Log(odds)} = \log\left(\frac{p}{1-p}\right)$$

Dataset Example

Suppose we have a dataset that contains two input features — CGPA and IQ, and one output column called placement. The output column contains binary values: 1 for placed and 0 for not placed.
We apply logistic regression to this dataset for prediction.
Let $\hat{Y}$ represent the predicted probability of getting placed, i.e., $P(1)$, which we denote as p (a small p).

cgpa	iq	placement	$\hat{Y}$ (Predicted Probability)
8	80	1	0.93
-	-	0	0.37
-	-	0	0.21

Logistic Regression Formula
$\hat{Y} = P(1) = p = \frac{1}{1+e^{-\beta X}}$

Where:
βX = $\beta_0 + \beta_1 X_1 + \beta_2 X_2$

Suppose we train our model and get:
$\beta_0 = 0$, $\beta_1 = 1$, $\beta_2 = 2$

For the first student:

βX = 0 + (1)(8) + (2)(80) = 168

Now, we plug this into the sigmoid (logistic) function to calculate the probability of being placed:

$$p = \frac{1}{1+e^{-168}}$$

Connecting to Log-Odds

We start with the sigmoid expression and manipulate it to derive the log-odds relation:

$p = \frac{1}{1+e^{-\beta X}}$
$1 + e^{-\beta X} = \frac{1}{p}$

$e^{-\beta X} = \frac{1}{p} - 1$

$$e^{-\beta X} = \frac{1-p}{p}$$

$$\frac{1}{e^{\beta X}} = \frac{1-p}{p}$$

$$\frac{p}{1-p} = e^{\beta X}$$

Taking the logarithm on both sides:

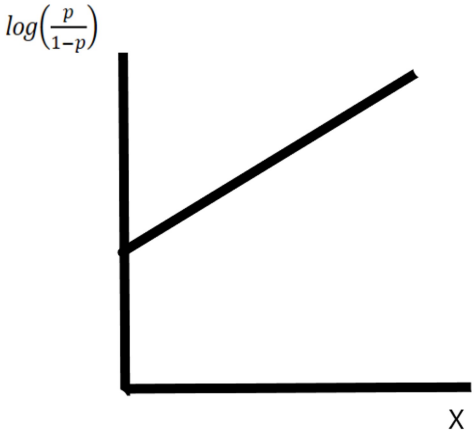
$$Log\left(\frac{p}{1-p}\right) = log(e^{\beta X})$$

$$Log\left(\frac{p}{1-p}\right) = \beta X$$

Interpretation
So we conclude:

$$log(odds) = log\left(\frac{p}{1-p}\right) = \beta X$$

This means there is a linear relationship between the log of odds and the input features (X).
If we plot a graph of log(odds) vs X, the relationship should be linear:



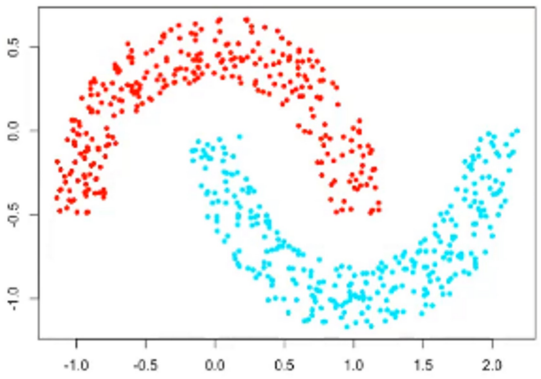
Important Insight

It should be linear. If it is not linear, then our logistic regression model may not provide accurate results.
Since $\hat{Y} =$ p, we can write:

$$Log\left(\frac{\hat{Y}}{1-\hat{Y}}\right) = X$$

Thus, if we plot $Log\left(\frac{\hat{Y}}{1-\hat{Y}}\right)$ against X, the graph should be linear.

POLYNOMIAL FEATURES



Logistic regression essentially performs classification by drawing a linear decision boundary. However, if the classes in our dataset are not linearly separable, a simple linear boundary may not yield good accuracy.
The good news is that even though logistic regression is a linear model, we can still model non-linear decision boundaries by using polynomial features.

How It Works

Suppose we have one input feature x , and we want to capture non-linearity by transforming it using polynomial terms.
If we choose a polynomial degree of 2, then we transform our input feature into:
x^0 , x^1, x^2
So instead of having just one input column, we now have three input columns.
Originally, we had parameters β_0 and β_1 , but after polynomial transformation, we now have:
β_0 , β_1 , β_2
This transformation allows the logistic regression model to fit non-linear decision boundaries, making it more flexible for complex data.
Important Consideration

- If we choose a very high polynomial degree, the model may overfit — it will fit the training data very well but perform poorly on unseen data.
- If we choose a very low degree, the model may underfit — it won't capture the complexity of the data.

Therefore, it's important to find an optimal polynomial degree — not too high and

not too low — to balance bias and variance.

REGULARIZATION IN LOGISTIC REGRESSION

Regularization is a technique used in machine learning models to prevent overfitting, which occurs when a model learns the noise along with the underlying pattern in the training data. Overfitting leads to poor generalization performance when the model is exposed to unseen data.

In the context of linear models like linear regression and logistic regression, regularization works by adding a penalty term to the loss function that the model tries to minimize. This penalty term discourages the model from assigning too much importance to any single feature, which helps to prevent overfitting.

The most common types of regularization in linear models are L1 and L2 regularization:

1. *L1 regularization (Lasso Regression): This technique adds a penalty term equal to the absolute value of the magnitude of the coefficients. Mathematically, it's represented as the sum of the absolute values of the weights ($|| w ||_1$). This can lead to sparse models, where some feature weights can become exactly zero. This property makes L1 regularization useful for feature selection.*
2. *L2 regularization (Ridge Regression): This technique adds a penalty term equal to the square of the magnitude of the coefficients. Mathematically, it's represented as the sum of the squared values of the weights ($|| w ||_2^2$). L2 regularization tends to spread the weight values more evenly across features, leading to smaller, but non-zero, weights.*

There's also Elastic Net regularization, which is a combination of L1 and L2 regularization. The contribution of each type can be controlled with a separate hyperparameter.

In all these techniques, the amount of regularization to apply is controlled by a hyperparameter, often denoted as λ (lambda). Higher values of λ mean more regularization, leading to simpler models that might underfit the data. Lower values of λ mean less regularization, leading to more complex models that might overfit the data. The optimal value of λ is typically found through cross-validation.