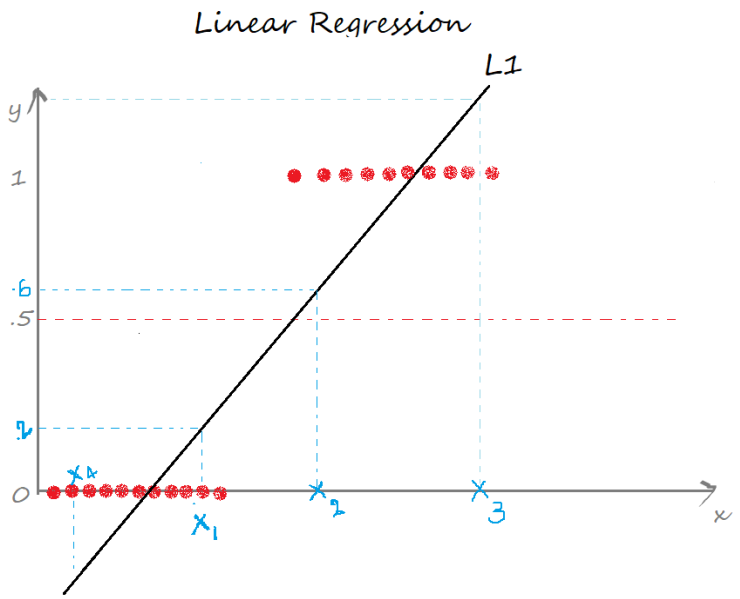
Logistic Regression

- **Type:** A fundamental **Classification** algorithm, not a regression algorithm despite its name.
- **Purpose:** Used to model the probability of a discrete outcome, typically binary (e.g., 0 or 1, Yes/No, Spam/Not Spam), given a set of input features (predictors).
- **Extension:** While primarily used for binary classification, it can be extended to multi-class classification problems (e.g., using Multinomial Logistic Regression or One-vs-Rest strategies).
    - Customer Rating: extremely dislike, dislike, neutral, like, extremely like.
    - Income level: low income, middle income, high income

2. Why Not Use Linear Regression for Classification?

Linear regression predicts continuous values and is unsuitable for classification tasks because:

- **Output Range:** Linear regression can output values outside the [0, 1] range, which doesn't make sense for probabilities.
- **Non-Linear Relationship:** The relationship between predictors and the *probability* of a binary outcome is often inherently non-linear (typically S-shaped). Linear regression assumes a linear relationship.
- **Assumption Violations:** The assumptions of linear regression (like normally distributed errors) are violated when the outcome variable is binary (Bernoulli distribution).
- Now, let us try if we can use linear regression to solve a binary class classification problem. Assume we have a dataset that is linearly separable and has the output that is discrete in two classes (0, 1).



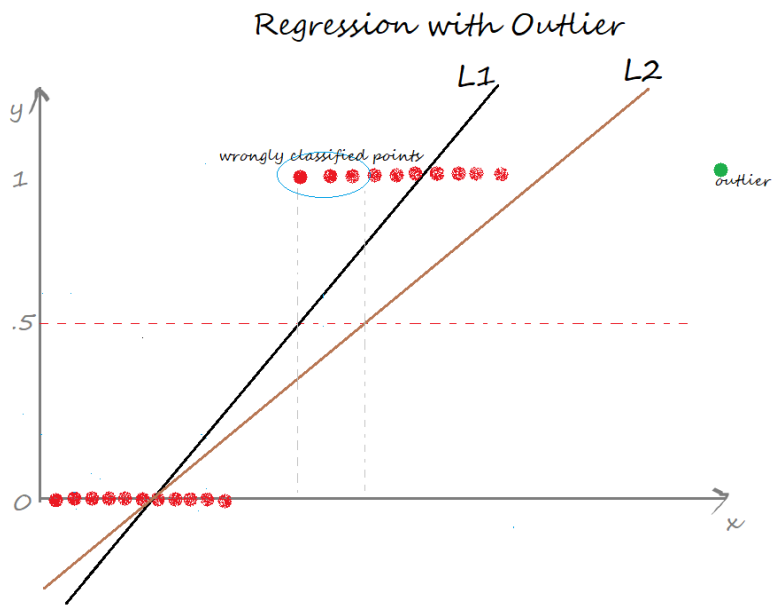
In Linear regression, we draw a straight line(the best fit line) L1 such that the sum of distances of all the data points to the line is minimal. The equation of the line L1 is y=mx+c, where m is the slope and c is the y-intercept.
We define a threshold T = 0.5, above which the output belongs to class 1 and class 0 otherwise.

$$y = mx + c, \text{ Threshold } T = 0.5$$

$$y = \begin{cases} 1, & mx+c \geq 0.5 \\ 0, & mx+c < 0.5 \end{cases}$$

- Case 1: the predicted value for x1 is ≈0.2 which is less than the threshold, so x1 belongs to class 0.
- Case 2: the predicted value for the point x2 is ≈0.6 which is greater than the threshold, so x2 belongs to class 1.
- So far so good, yeah!
- Case 3: the predicted value for the point x3 is beyond 1.
- Case 4: the predicted value for the point x4 is below 0.
  The predicted values for the points x3, x4 exceed the range (0,1) which doesn't make sense because the probability values always lie between 0 and 1. And our output can have only two values either 0 or 1. Hence, this is a problem with the linear regression model.

Now, introduce an outlier and see what happens. The regression line gets deviated to keep the distance of all the data points to the line to be minimal.



L2 is the new best-fit line after the addition of an outlier. Seems good till now. But the problem is, if we closely observe, some of the data points are wrongly classified. Certainly, it increases the error term 😕 This again is a problem with the linear regression model.

The two limitations of using a linear regression model for classification problems are:

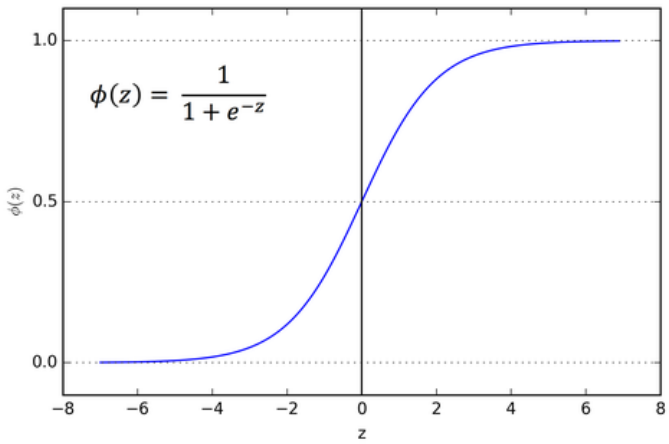
- the predicted value may exceed the range (0,1)
- error rate increases if the data has outliers

3. How Logistic Regression Works: The Sigmoid Function

Logistic Regression addresses these issues by using a **logistic function (or sigmoid function)** to transform a linear combination of inputs into a probability score.

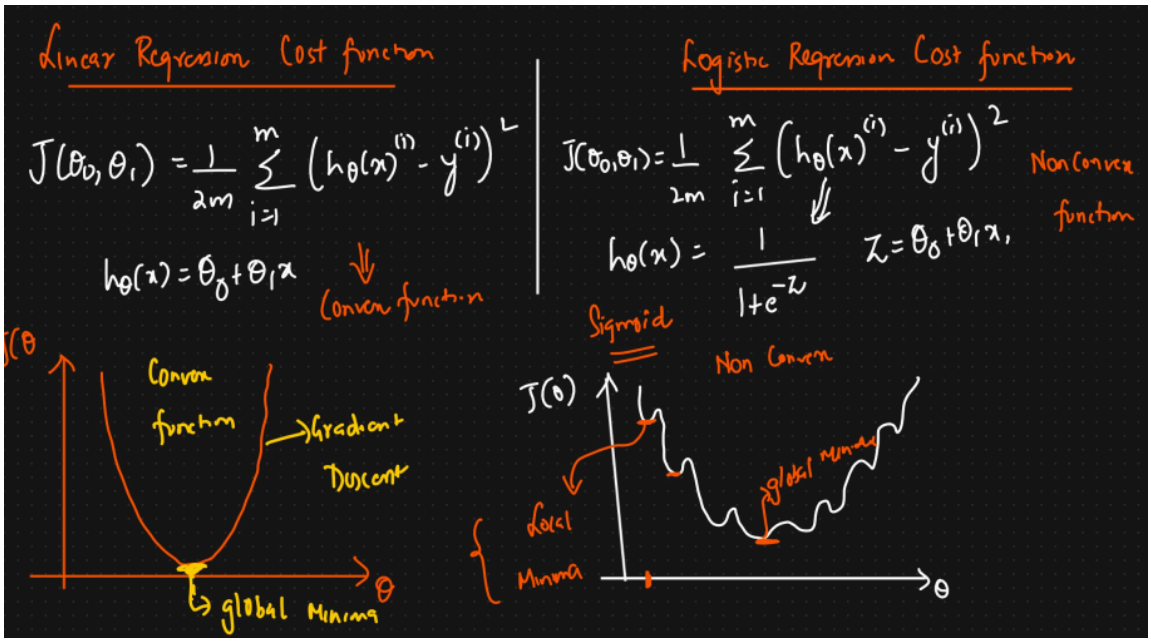
- **Linear Component:** First, calculate a linear combination of the input features and coefficients (similar to linear regression): $z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta x$ where β are the coefficients and x are the predictor variables.
- **Sigmoid Function:** Apply the sigmoid function $\sigma(z)$ to this linear combination: $\sigma(z) = 1 / (1 + e^{-z})$ The sigmoid function squashes any real-valued input z into the range (0, 1).
- **Probability Output:** The output $\sigma(z)$ is interpreted as the estimated probability of the positive class (Y=1) given the input features X: $P(Y=1 \mid X) = \sigma(z) = 1 / (1 + e^{-(\beta_0 + \beta_1 x_1 + ... + \beta x)})$
- **Decision Boundary:** To make a final classification, a threshold (commonly 0.5) is applied to the predicted probability:
    - If $P(Y=1 \mid X) \geq 0.5$, predict Class 1.

- If `P(Y=1 | X) < 0.5`, predict Class 0.
- Setting the probability to 0.5 corresponds to `z = 0`. Therefore, the equation $\beta_0 + \beta_1 x_1 + \ldots + \beta x = 0$ defines the **decision boundary**, which is linear in the feature space for standard logistic regression.



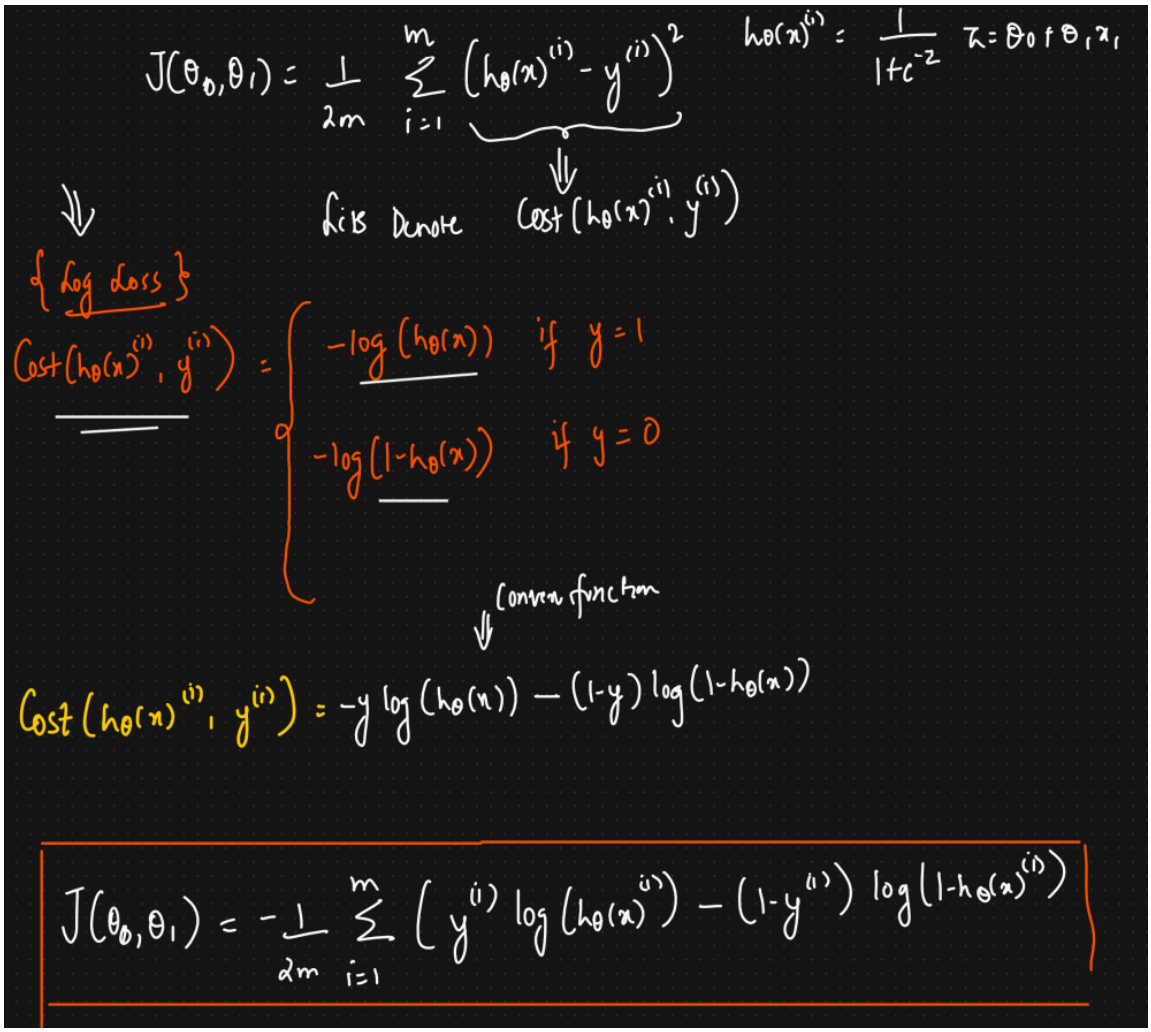
4. Cost Function: Log Loss (Binary Cross-Entropy)

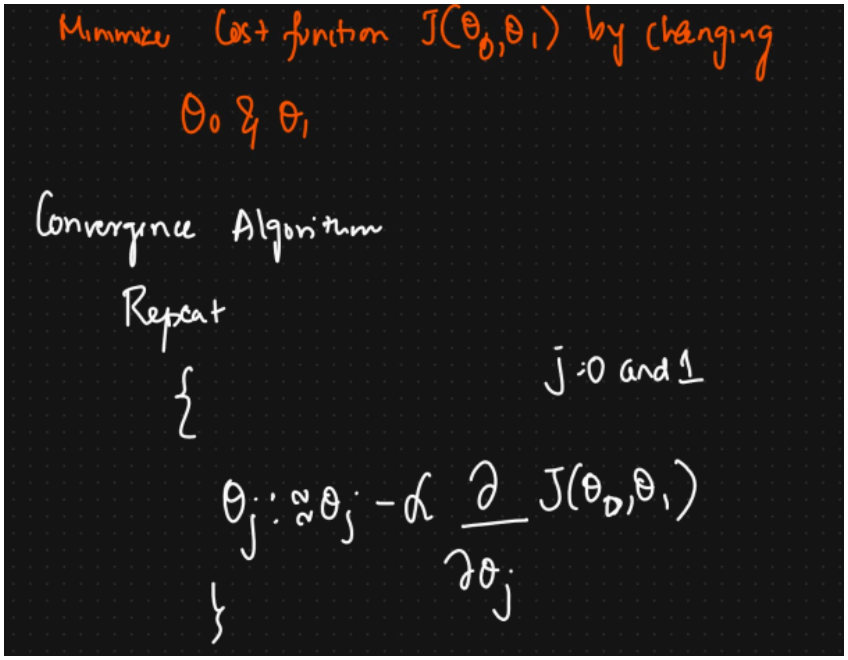
- Minimizing the Sum of Squared Errors (SSE) used in linear regression is problematic for logistic regression (leads to a non-convex cost function with multiple local minima).



- Instead, Logistic Regression uses a cost function derived from **Maximum Likelihood Estimation (MLE)**, commonly known as **Log Loss** or **Binary Cross-Entropy**.

- For a single training example: `Cost(hθ(x), y) = - [ y * log(hθ(x)) + (1 - y) * log(1 - hθ(x)) ]` where:
  - `hθ(x)` is the predicted probability `P(Y=1 | X)`.
  - `y` is the actual class label (0 or 1).
- **Intuition:** This cost function heavily penalizes predictions that are confident (`hθ(x)` near 0 or 1) but wrong (`y` is the opposite). It is zero if the prediction is correct and confident.
- **Overall Cost:** The goal is to minimize the average Log Loss over all training examples. Optimization algorithms like **Gradient Descent** are used to find the coefficients (β values) that minimize this cost function.



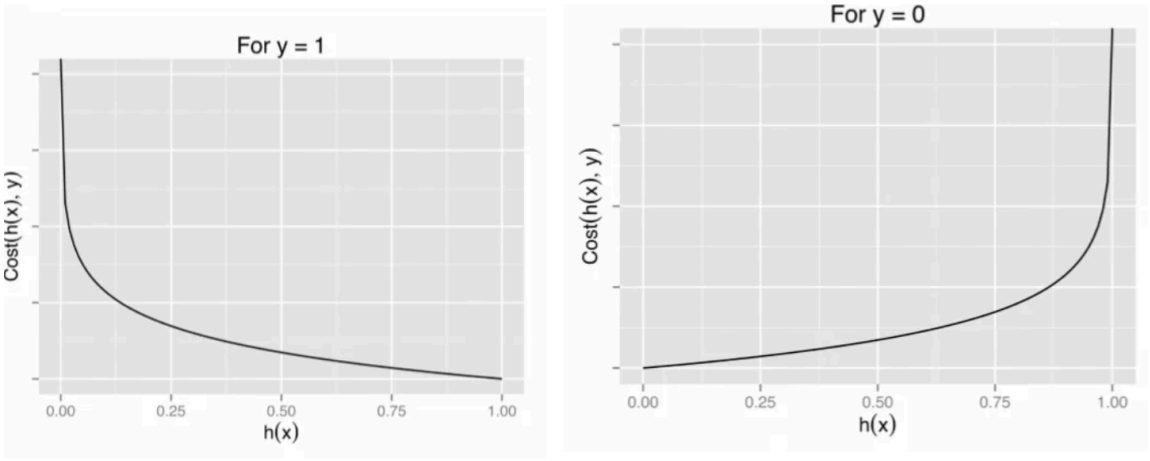


For logistic regression, the Cost function is defined as:

> $-log(h\theta(x))$ *if* $y = 1$
>
> $-log(1-h\theta(x))$ *if* $y = 0$

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Cost function of Logistic Regression



Graph of logistic regression

The above two functions can be compressed into a single function i.e.

$$J(\theta) = -\frac{1}{m}\sum\left[y^{(i)}\log(h\theta(x(i))) + \left(1 - y^{(i)}\right)\log(1 - h\theta(x(i)))\right]$$

Above functions compressed into one cost function

Gradient Descent

Now the question arises, how do we reduce the cost value. Well, this can be done by using Gradient Descent. The main goal of Gradient descent is to minimize the cost value. i.e. min J(θ).

Now to minimize our cost function we need to run the gradient descent function on each parameter i.e.

$$\theta j := \theta j - \alpha \frac{\partial}{\partial \theta j} J(\theta)$$

Objective: To minimize the cost function we have to run the gradient descent function on each parameter

Want $\min_\theta J(\theta)$:

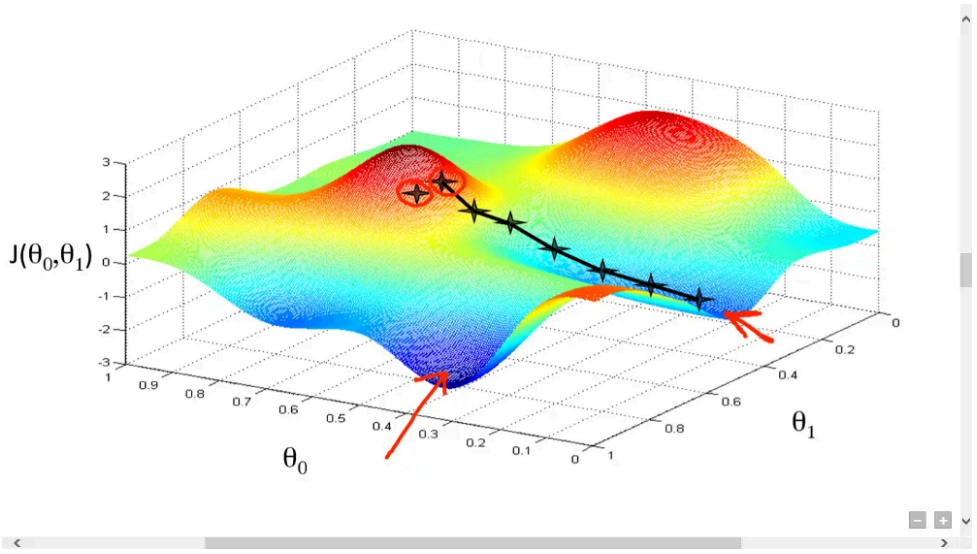
Repeat $\{$

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

$\}$

Gradient Descent Simplified | Image: Andrew Ng Course



5. Interpreting Coefficients: Log-Odds and Odds Ratios

Coefficients (β) in logistic regression have a specific interpretation related to the **log-odds** (also called the **logit**).

- **Odds:** The ratio of the probability of the event happening (Y=1) to the probability of it not happening (Y=0): `Odds = P(Y=1 | X) / P(Y=0 | X) = P(Y=1 | X) / (1 - P(Y=1 | X))`

- **Log-Odds (Logit):** The natural logarithm of the odds: $\log(\text{Odds}) = \log(P / (1-P)) = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n = z$ This shows that the model assumes a *linear* relationship between the predictors and the *log-odds* of the outcome.
- **Interpretation:** A one-unit increase in predictor x_i, holding all other predictors constant, changes the *log-odds* of the outcome by β_i.
- **Odds Ratio (OR):** For easier interpretation, we often exponentiate the coefficient: $\exp(\beta_i)$. This gives the **Odds Ratio** associated with a one-unit increase in x_i.
    - If $\exp(\beta_i) > 1$: A one-unit increase in x_i multiplies the odds of the outcome by $\exp(\beta_i)$ (i.e., increases the odds).
    - If $\exp(\beta_i) < 1$: A one-unit increase in x_i multiplies the odds of the outcome by $\exp(\beta_i)$ (i.e., decreases the odds).
    - If $\exp(\beta_i) = 1$ ($\beta_i = 0$): The predictor x_i has no effect on the odds of the outcome.

6. Assumptions

- **Binary Outcome:** The dependent variable is binary (or categorical for multinomial extensions).
- **Independence:** Observations are independent of each other.
- **No Severe Multicollinearity:** Predictor variables should not be highly correlated. High multicollinearity can inflate the variance of coefficient estimates.
- **Linearity of Log-Odds:** Assumes a linear relationship between each predictor variable and the log-odds of the outcome. Violations mean the model may not fit the data well.
- **Large Sample Size:** Relies on MLE properties, which are asymptotic (perform best with larger sample sizes).

7. Regularization

Like linear regression, logistic regression can suffer from overfitting, especially with many features. **L1 (Lasso) and L2 (Ridge) regularization** can be applied by adding penalty terms ($\alpha * \Sigma|\beta|$ for L1, $\alpha * \Sigma(\beta)^2$ for L2) to the log-loss cost function. This helps shrink coefficients, reduce overfitting, and handle multicollinearity.

8. Advantages

- Provides probability estimates for classification, useful for understanding confidence and ranking.
- Coefficients are interpretable in terms of odds ratios.
- Relatively simple, computationally efficient, and easy to implement.
- Performs well when the decision boundary is close to linear.
- Widely used and forms a basis for more complex algorithms (like neural networks).

9. Disadvantages

- Assumes linearity between predictors and the log-odds of the outcome. May not capture complex, non-linear relationships effectively.
- Can be sensitive to outliers.
- May underperform if the decision boundary is highly non-linear.
- Requires careful consideration of multicollinearity.

10. When to Use Logistic Regression

- For binary classification problems (the most common use case).
- When you need probability estimates alongside class predictions.
- As a baseline model for classification tasks before trying more complex methods.
- When interpretability of feature influence (via odds ratios) is important.
- When the relationship between features and the log-odds of the outcome is expected to be roughly linear.