Fundamentals of Machine Learning

Prepared by

Prince Thomas M.E., PhD

Associate Professor

Chapter 2: Supervised Machine Learning: Linear Models and Fundamentals

Linear Regression

- Simple Linear Regression and Multiple Linear Regression
- Cost Function (Mean Squared Error)
- Gradient Descent for Linear Regression

Logistic Regression (Linear Classification)

- Sigmoid Function and Decision Boundaries
- Binary Classification vs. Multi-class Classification

Gradient Descent Algorithm

- Stochastic Gradient Descent (SGD) vs. BGD
- Learning Rate and Convergence Issues

Overfitting and Underfitting

- Bias-Variance Trade-off
- Cross-Validation Techniques

Regularization Techniques (L1, L2)

- Lasso and Ridge Regression

- High Dimensional Data
  - Curse of Dimensionality and Feature Selection

Multivariate Methods

- Covariance Matrix
- Linear Models for Multivariate Regression

Parametric vs. Non-parametric Methods

- Differences, Examples, and Use Cases

Supervised Learning:

- Models learn from labeled datasets, where each data point contains both features (input) and corresponding labels (correct output).
- The objective is to predict the label for new, unseen data points using the patterns learned from the training data.

Example Applications:

- Predicting house prices based on features like area, number of rooms, etc.
- Identifying spam emails based on words in the email.

What are Linear Models:

- Linear models are the simplest and most widely used models in supervised learning.
- Relationship between the input (independent variable) and the output (dependent variable) by fitting a straight line or plane.

Types

- Linear Regression
- Logistic Regression
- Ridge Regression
- Lasso Regression
- Elastic Net
- Support Vector Machine (SVM) with Linear Kernel
- Generalized Linear Models (GLM)
- Multinomial Logistic Regression
- Quantile Regression

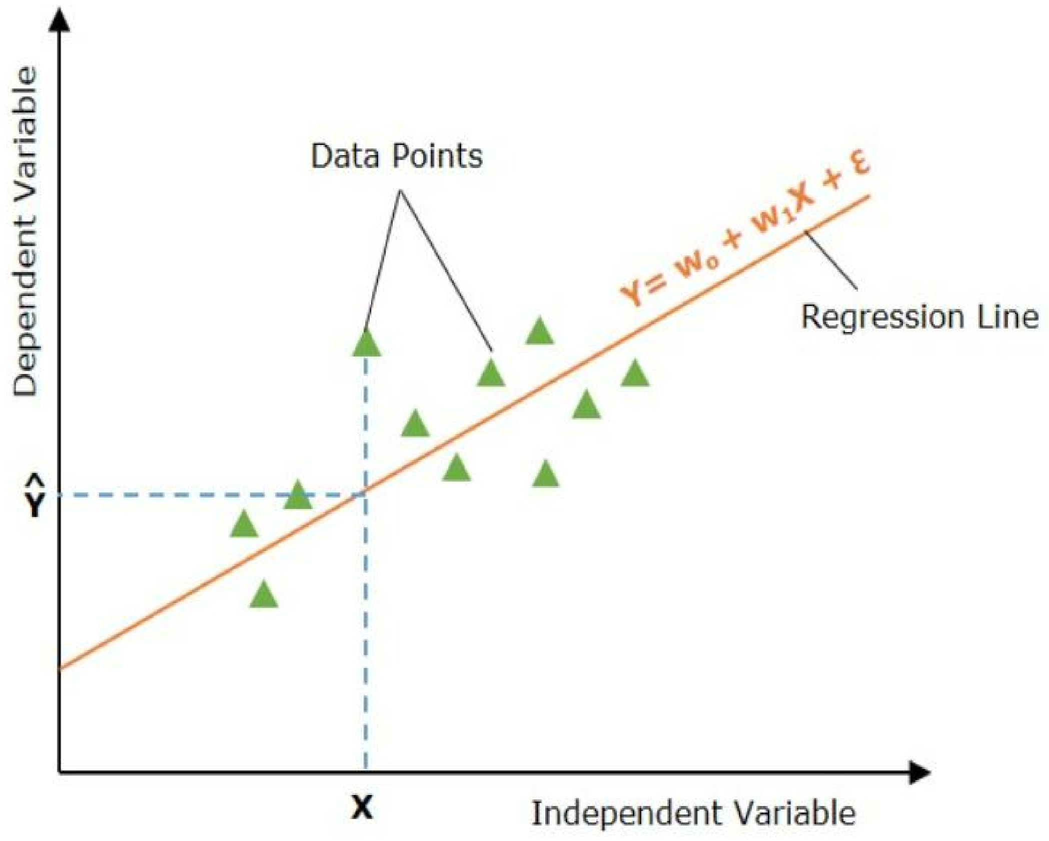
Introduction to Linear Regression

- Linear Regression is a fundamental statistical method used in machine learning and data analysis to model the relationship between a dependent variable and one or more independent variables.
- It assumes a linear relationship, which means the change in the dependent variable can be explained by a linear combination of the independent variables.

Types:

- Simple Linear Regression
- Multiple Linear Regression

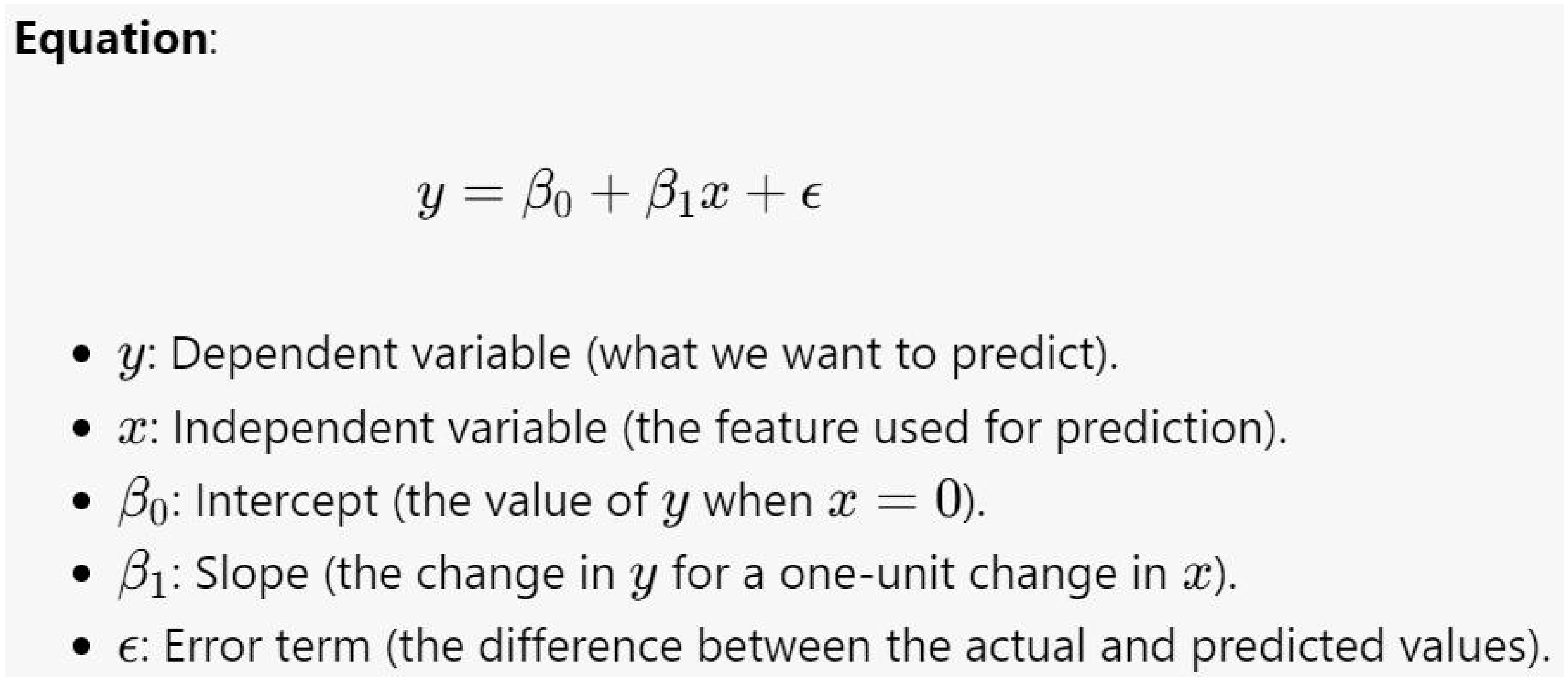
Square Feet (X)	House Price (Y)
1300	240
1500	320
1700	330
1830	295
1550	256
2350	409
1450	319



- Here X is input and Y is output.
- plotted these points in graph and drawing the best fit line using the data points.
- using the best fit line can find the prediction for new input value.

Simple Linear Regression

Definition: Simple Linear Regression involves one independent variable (feature) and one dependent variable (target).



Equation:

$$y = \beta_0 + \beta_1 x + \epsilon$$

- y: Dependent variable (what we want to predict).
- x: Independent variable (the feature used for prediction).
- β_0: Intercept (the value of y when $x = 0$).
- β_1: Slope (the change in y for a one-unit change in x).
- ϵ: Error term (the difference between the actual and predicted values).

Assumptions:

Linearity: The relationship between variables is linear.

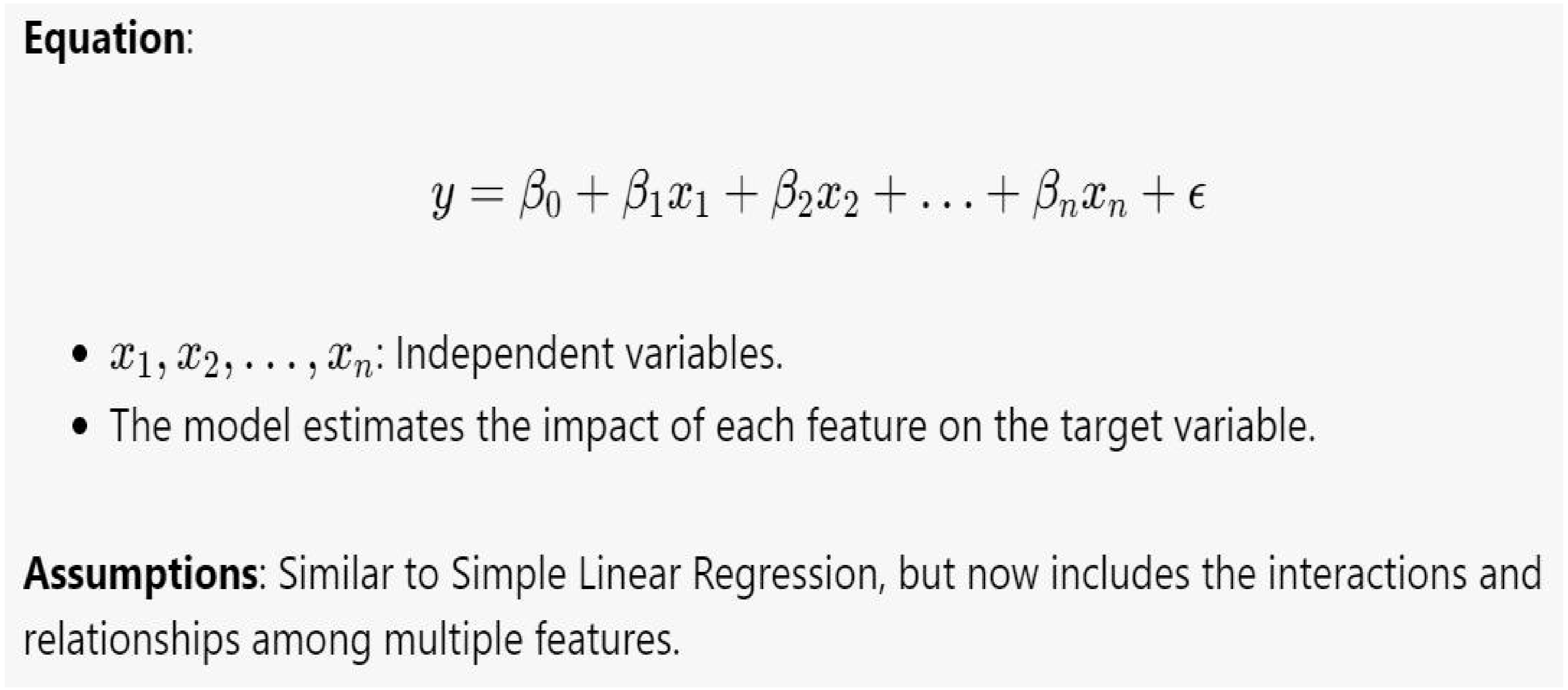
Independence: Observations are independent of each other.

Homoscedasticity: Constant variance of errors.

Normality: Errors are normally distributed.

Multiple Linear Regression

Definition: Multiple Linear Regression extends Simple Linear Regression by using two or more independent variables to predict the dependent variable.

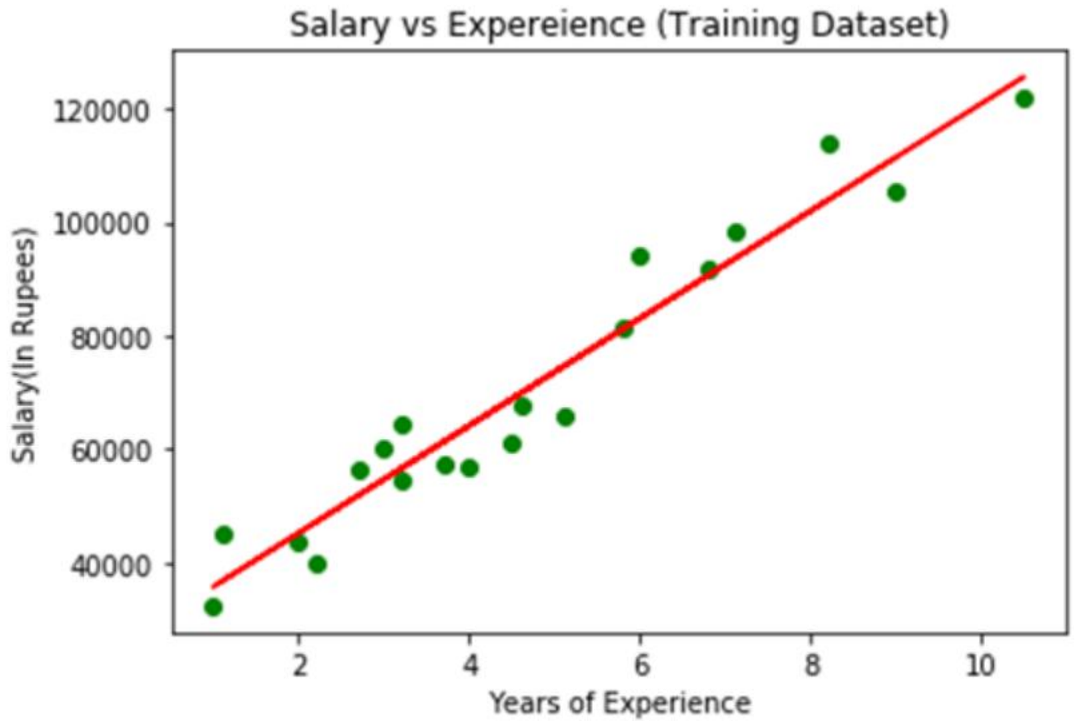


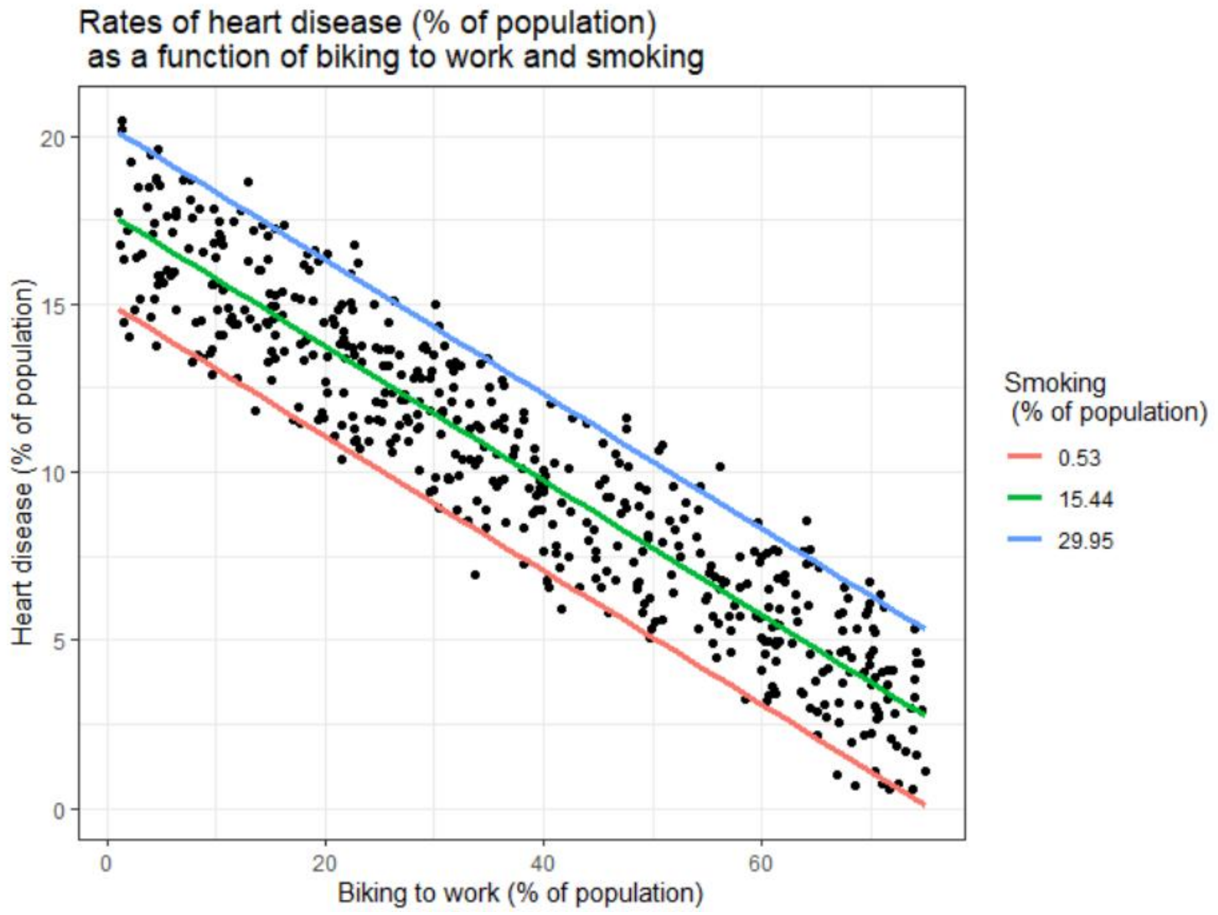
Equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilon$$

- $x_1, x_2, \ldots, x_n$: Independent variables.
- The model estimates the impact of each feature on the target variable.

Assumptions: Similar to Simple Linear Regression, but now includes the interactions and relationships among multiple features.





Year of Experience is input and Predicting salary is output

Here two independant values are biking to work and smoking. dependant value is prediction heart disease

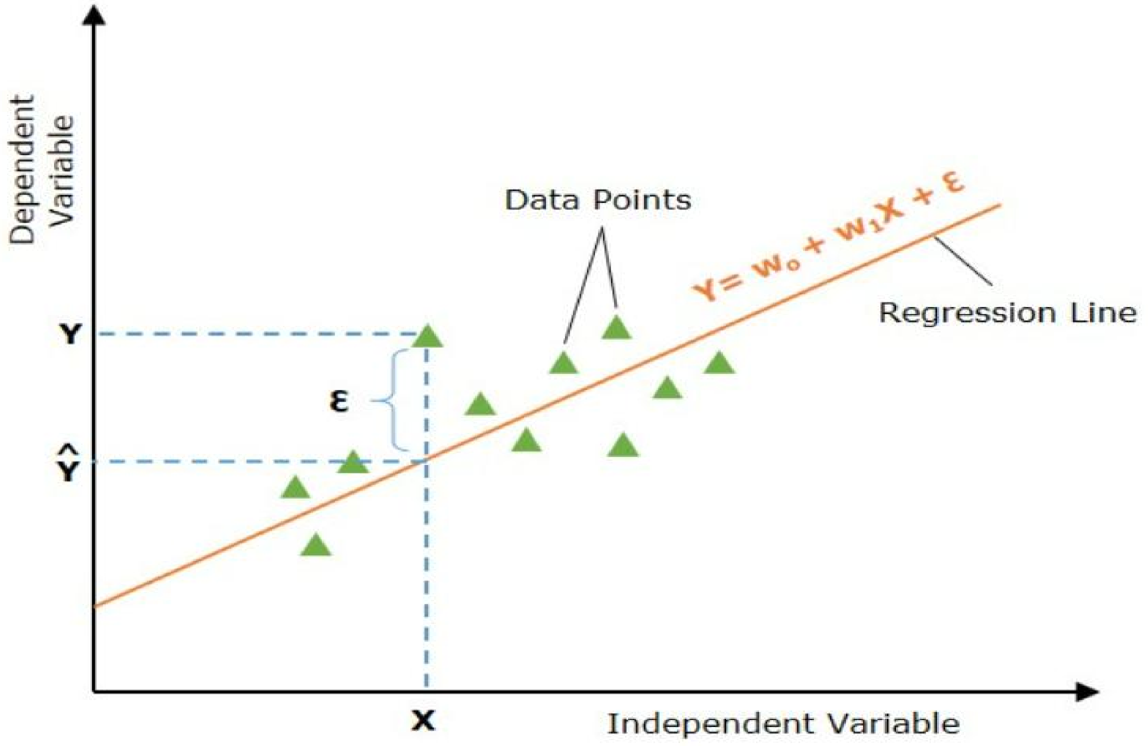
Cost Function

Definition: The cost function measures how well the linear regression model fits the data. It quantifies the error between predicted values and actual values.

Mean Squared Error (MSE):

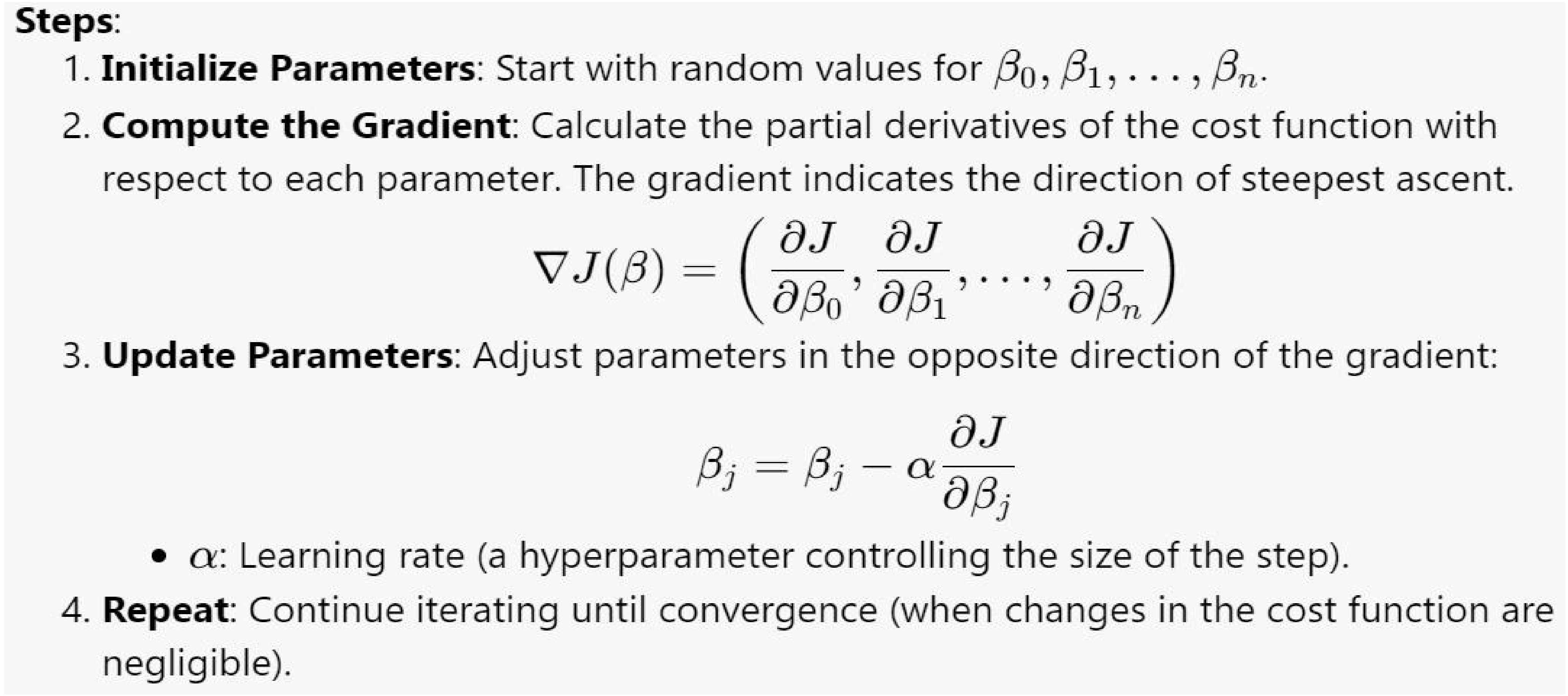
$$\mathrm{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

- m: Number of observations.
- y_i: Actual value of the dependent variable.
- $\hat{y}_i$: Predicted value from the model.



Purpose: The goal is to minimize the MSE by adjusting the model parameters ($\beta_0, \beta_1, \ldots, \beta_n$). A lower MSE indicates a better fit of the model to the data.

Gradient Descent: Gradient Descent is an optimization algorithm used to minimize the cost function by iteratively adjusting the model parameters.



Steps:

1. **Initialize Parameters**: Start with random values for $\beta_0, \beta_1, \ldots, \beta_n$.
2. **Compute the Gradient**: Calculate the partial derivatives of the cost function with respect to each parameter. The gradient indicates the direction of steepest ascent.

$$\nabla J(\beta) = \left(\frac{\partial J}{\partial \beta_0}, \frac{\partial J}{\partial \beta_1}, \ldots, \frac{\partial J}{\partial \beta_n} \right)$$

3. **Update Parameters**: Adjust parameters in the opposite direction of the gradient:

$$\beta_j = \beta_j - \alpha \frac{\partial J}{\partial \beta_j}$$

   - α: Learning rate (a hyperparameter controlling the size of the step).
4. **Repeat**: Continue iterating until convergence (when changes in the cost function are negligible).

learning rate (α) is a hyperparameter that determines the size of the steps taken towards the minimum of the cost function during the optimization process.

- If the learning rate is set too high, the algorithm may take large steps and overshoot the minimum
- If the learning rate is set too low, the algorithm will take very small steps. meaning it will take a long time to reach the minimum of the cost function

How Simple Linear Regression Works

1. Initialization:

- We start with random values for the intercept (Bo) and slope (B1). These values are our initial guesses for the best-fit line.

2. Cost Function Calculation:

- For each data point, calculate the predicted value y_hat using the current values of Bo and B1.
- Calculate the squared difference between the actual value y and the predicted value y_hat.
- Finally, average these squared differences to get the Mean Squared Error (MSE), which is our cost function.

3. Gradient Calculation:

- Calculate the partial derivatives of the MSE with respect to Bo and B1. These derivatives tell us how much the MSE changes when we slightly adjust Bo and B1.

4. Parameter Update:

- Update Bo and B1 by subtracting a small step in the direction opposite to the gradient. The size of this step is determined by the learning rate.
- A high learning rate means, it take larger steps, which can lead to faster convergence but also increase the risk of overshooting the minimum.
- A low learning rate means, it take smaller steps, which can lead to slower convergence but a more stable optimization process.

5. Iteration:

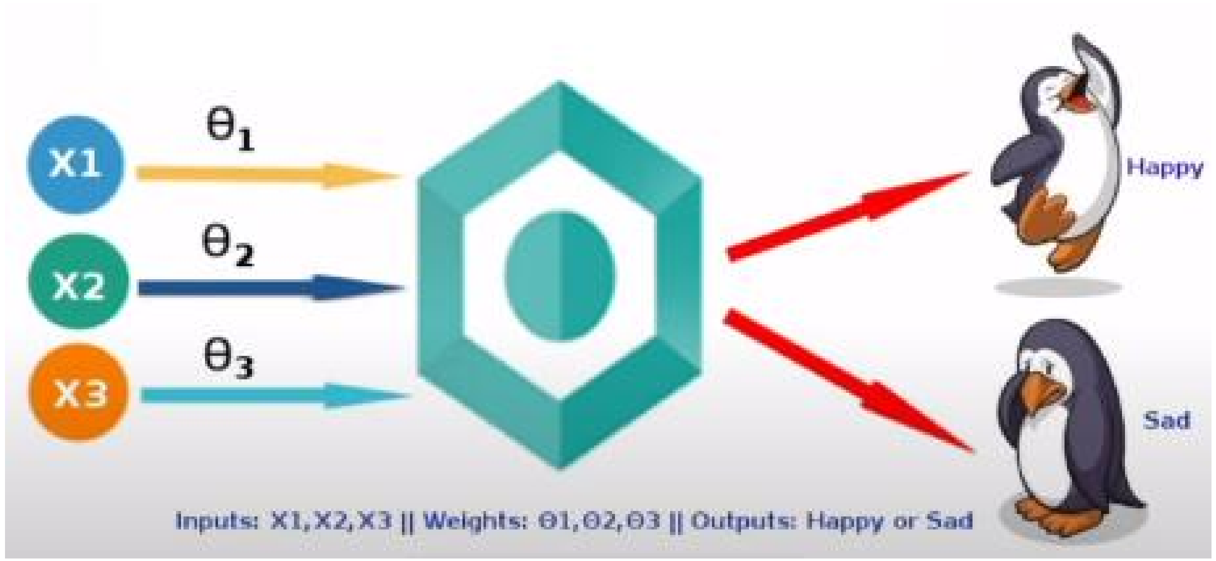
- Repeat steps 2-4 until the changes in the MSE become negligible, indicating that we have reached a minimum.

Applications of simple linear regression:

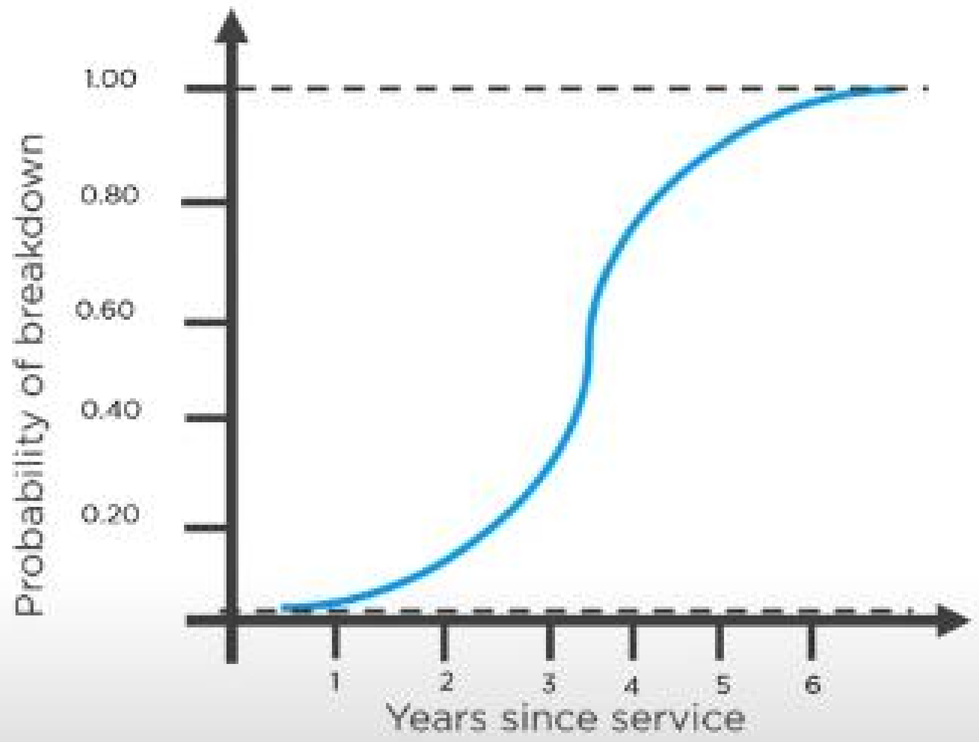
- Predictive Modeling
    - Sales Forecasting
    - Real Estate Pricing
- Trend Analysis
    - Economic Indicators
    - Market Research
- Quality Control
    - Manufacturing Output Monitoring
- Social Sciences
    - Education (Study Hours vs. Performance)
    - Health Studies (Lifestyle Factors vs. Health Outcomes)
- Finance
    - Risk Assessment (Market Factors vs. Stock Prices)
- Environmental Studies
    - Climate Data Analysis (CO2 Emissions vs. Temperature)
- Sports Analytics
    - Performance Analysis (Training vs. Performance Metrics)

Logistic Regression (Linear Classification)

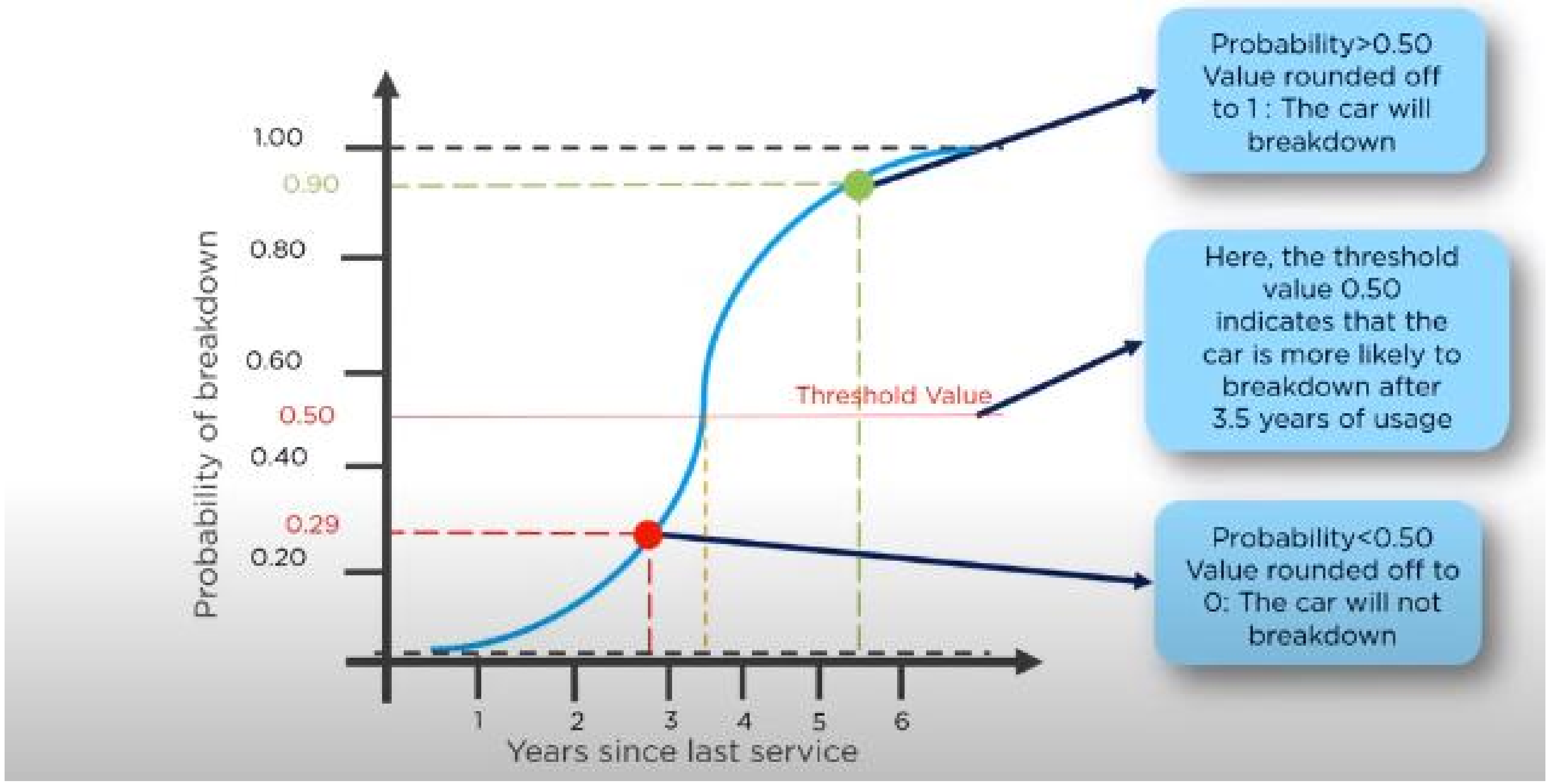
- Logistic Regression is a supervised learning used for binary classification problems.
- logistic regression gives the solution in classification manner using the input values with weight. Logistic regression predicts the probability that a given input belongs to a particular class, instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- Unlike linear regression, it predicts continuous values.


X1
X2
X3
θ1
θ2
θ3
Happy
Sad
Inputs: X1,X2,X3 || Weights: θ1,θ2,θ3 || Outputs: Happy or Sad

You provide years since last service



Regression model created based on other users' experience



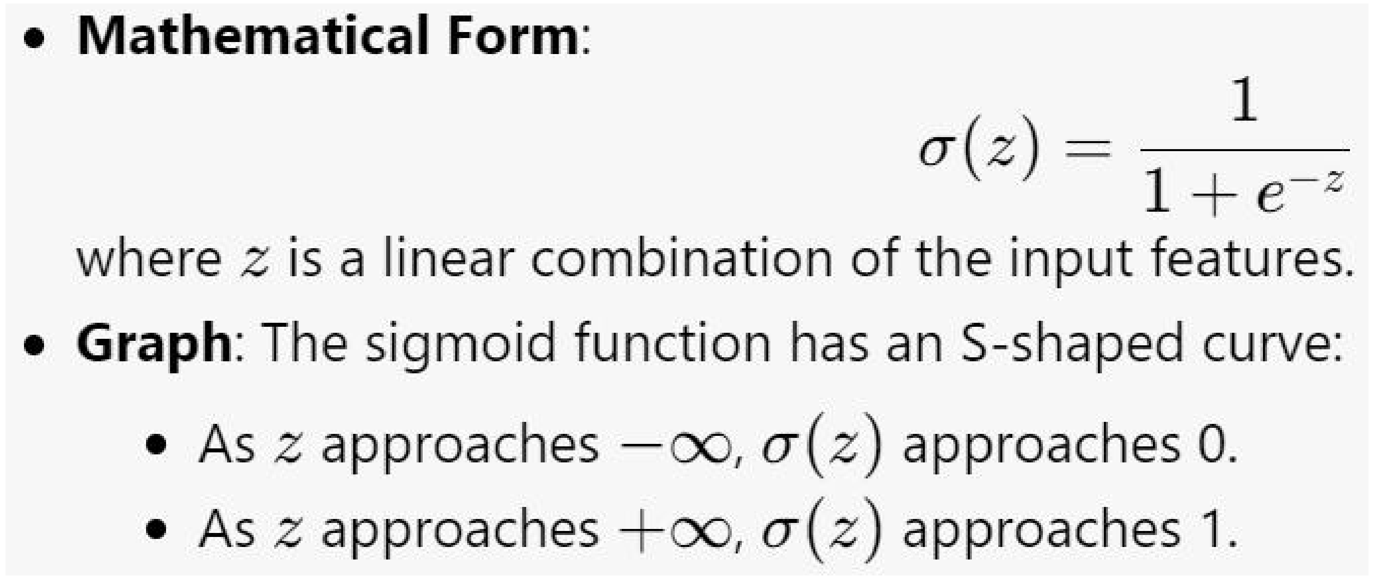
Probability>0.50 Value rounded off to 1 : The car will breakdown

Here, the threshold value 0.50 indicates that the car is more likely to breakdown after 3.5 years of usage

Probability<0.50 Value rounded off to 0: The car will not breakdown

Logistic Function – Sigmoid Function

- In simple linear regression best fit line is used to predict and here S shaped logistic function used to predict
- The sigmoid function is a mathematical function used to map the predicted values to probabilities. It maps any real value into another value within a range of 0 and 1.

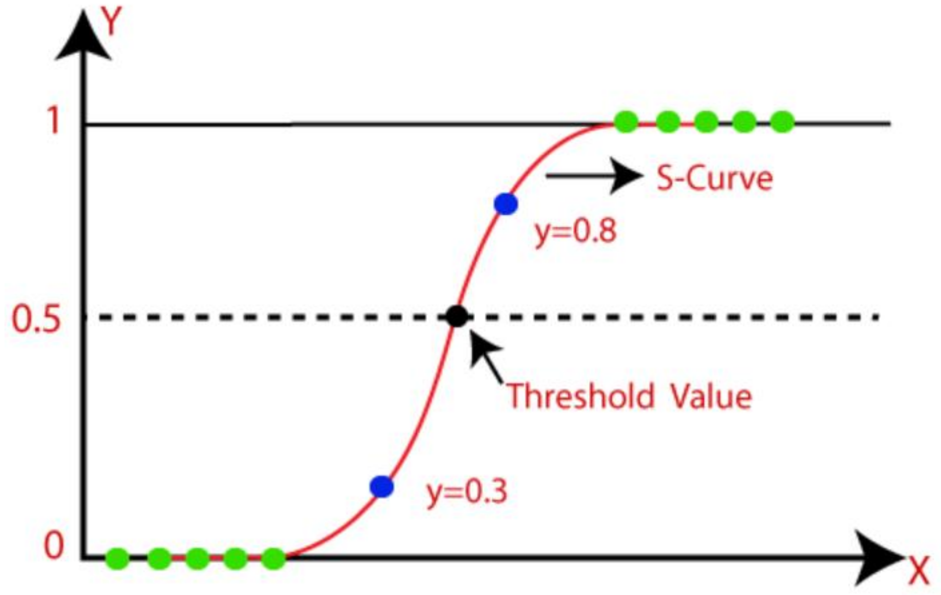


- **Mathematical Form**:

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

where z is a linear combination of the input features.

- **Graph**: The sigmoid function has an S-shaped curve:
  - As z approaches $-\infty$, $\sigma(z)$ approaches 0.
  - As z approaches $+\infty$, $\sigma(z)$ approaches 1.



- In logistic regression, used the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0

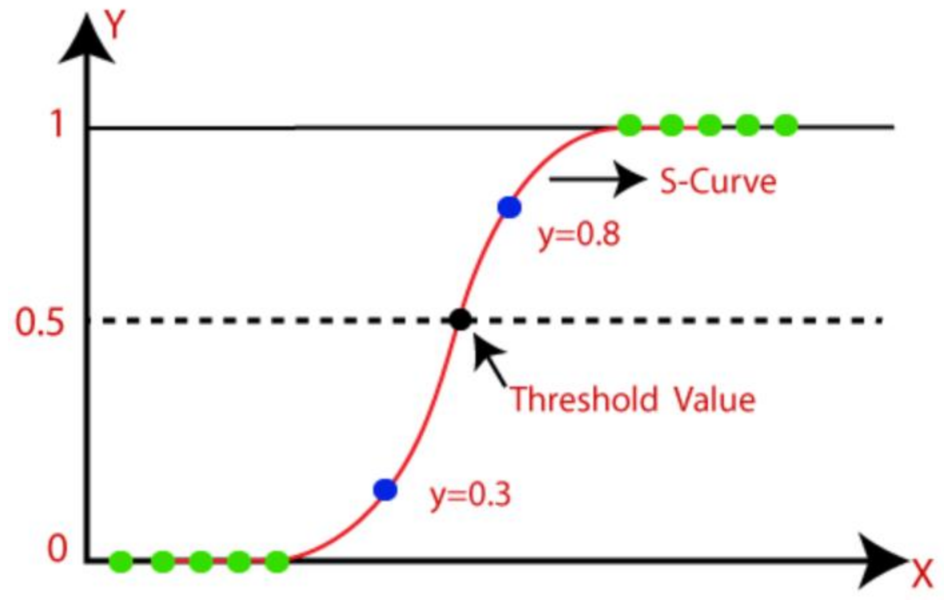
Decision Boundaries

Definition: The decision boundary is the threshold that separates different classes in the feature space.

For Logistic Regression:

- The decision boundary is defined where the predicted probability is 0.5: This implies that the linear combination of features results in a score of 0.

$$\sigma(z) = 0.5 \implies z = 0$$



Binary Classification

Overview: Logistic regression is primarily used for binary classification problems where there are two classes (e.g., 0 and 1). **Example:** Predicting if an email is spam (1) or not spam (0).

Multi-class Classification

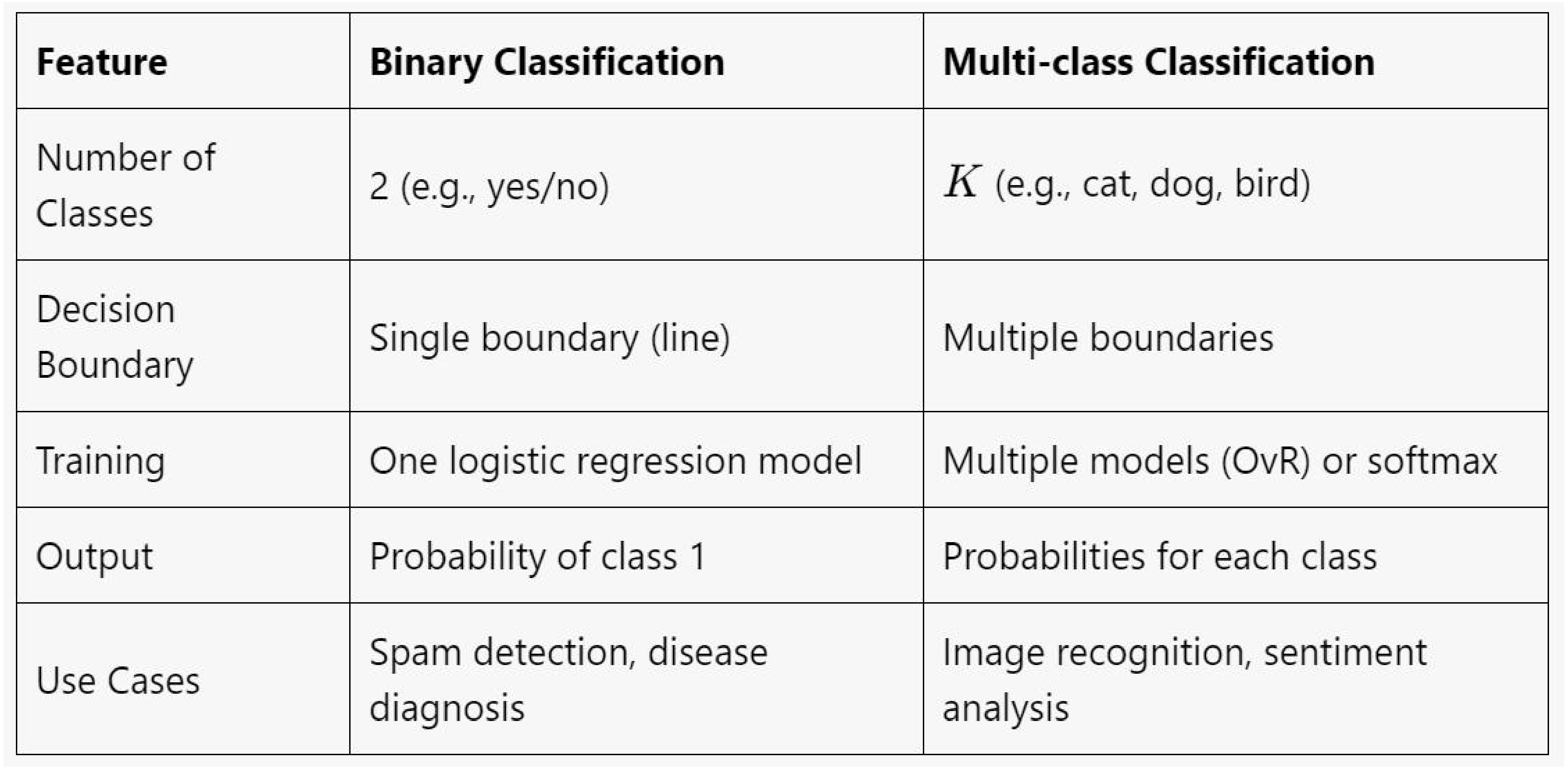
Overview: While logistic regression is inherently binary, it can be extended to multi-class classification using strategies like One-vs-Rest (OvR) or Softmax regression.

- **One-vs-Rest (OvR):** For K classes, K separate binary classifiers are trained. Each classifier predicts whether the input belongs to a particular class or not.
- **Softmax Regression:** Softmax generalizes logistic regression to multiple classes. The probability of class k is given by:

$$P(y = k|x) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

where zk is the linear combination of features for class k.

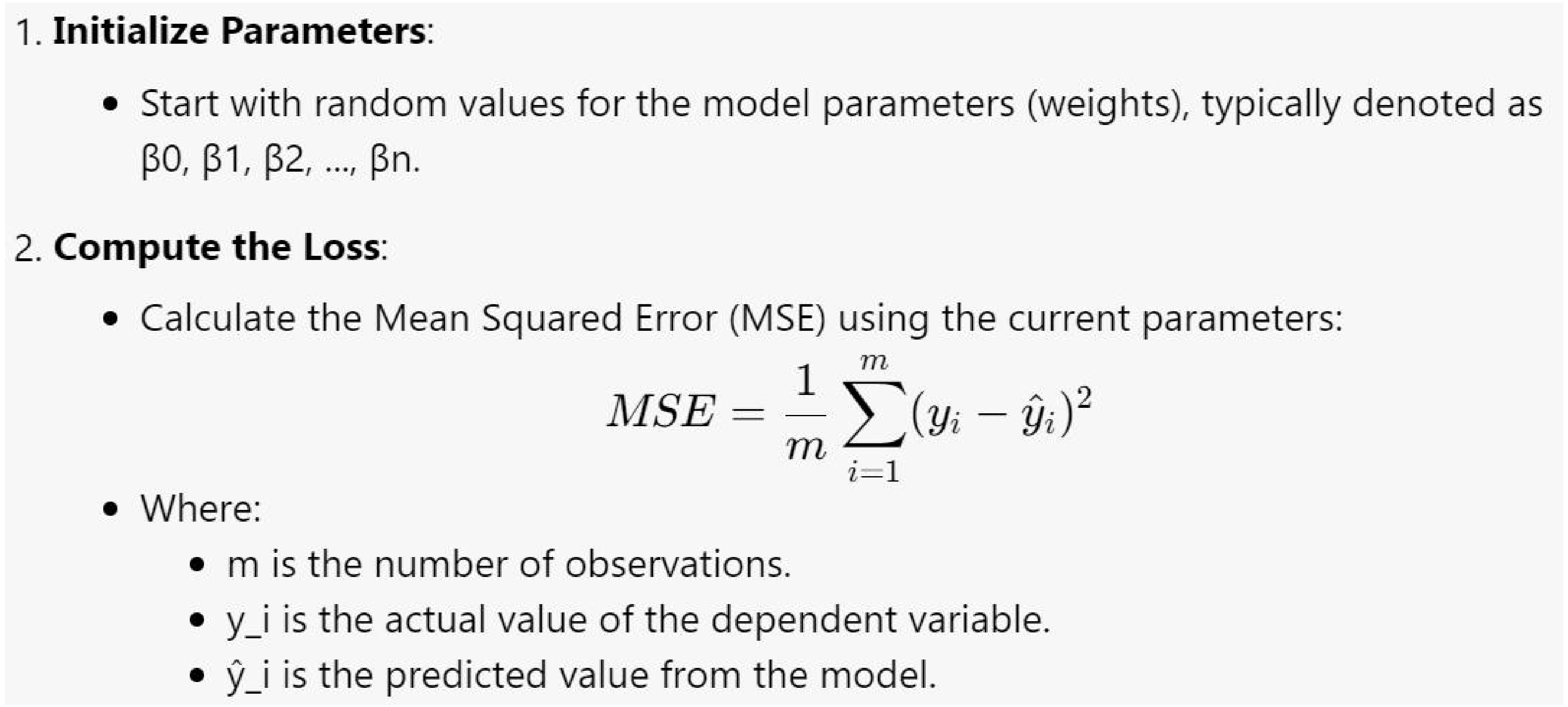
Comparison: Binary vs. Multi-class Classification



Feature	Binary Classification	Multi-class Classification
Number of Classes	2 (e.g., yes/no)	K (e.g., cat, dog, bird)
Decision Boundary	Single boundary (line)	Multiple boundaries
Training	One logistic regression model	Multiple models (OvR) or softmax
Output	Probability of class 1	Probabilities for each class
Use Cases	Spam detection, disease diagnosis	Image recognition, sentiment analysis

Gradient Descent Algorithm: Gradient descent is an optimization algorithm used to minimize a cost function. It iteratively updates the model's parameters (weights) in the direction that reduces the cost. Iteratively moving towards the steepest descent, which is determined by the gradient.

How does it works for Multiple Linear Regression



1. **Initialize Parameters**:
   - Start with random values for the model parameters (weights), typically denoted as β0, β1, β2, ..., βn.

2. **Compute the Loss**:
   - Calculate the Mean Squared Error (MSE) using the current parameters:

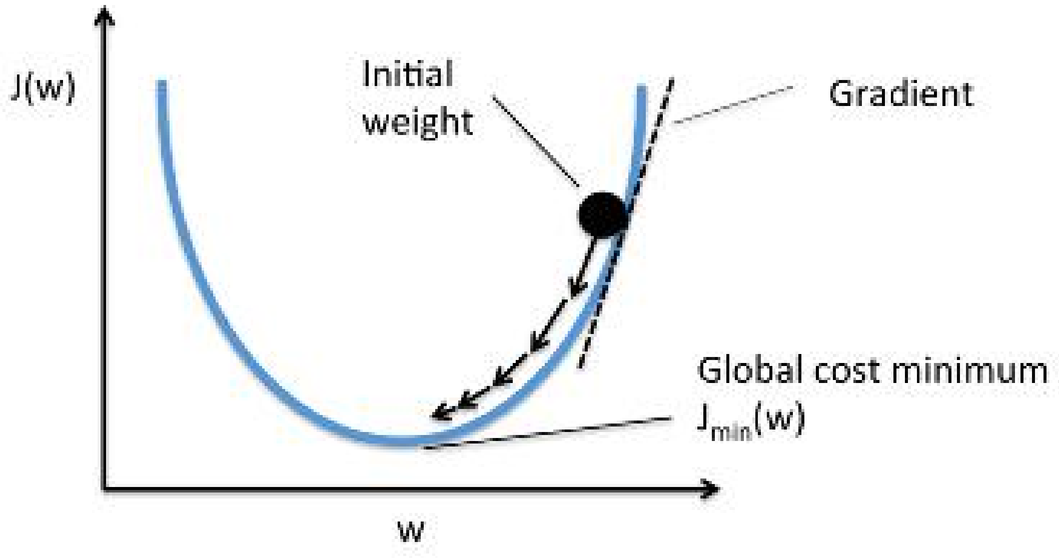
$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

   - Where:
     - m is the number of observations.
     - y_i is the actual value of the dependent variable.
     - ŷ_i is the predicted value from the model.

3. **Calculate the Gradient**:
    - Compute the gradients of the MSE with respect to each parameter (β0, β1, β2, ..., βn):
$$\frac{\partial MSE}{\partial \beta_j} = \frac{2}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i) \cdot x_i^{(j)}$$
    - Where:
        - x_i^{(j)} is the j-th feature for the i-th observation.

4. **Update Parameters**:
    - Adjust the parameters by moving them in the opposite direction of the gradient:
$$\beta_j = \beta_j - \alpha \cdot \frac{\partial MSE}{\partial \beta_j}$$
    - Here, α is the learning rate that controls the size of the step taken towards the minimum.
5. **Check for Convergence**:
    - Determine if the algorithm has converged by checking if the change in MSE or parameters is below a certain threshold (e.g., |ΔMSE| < ε or |Δβ| < ε) or if a maximum number of iterations has been reached.
6. **Repeat**:
    - If not converged, return to step 2 and repeat the process until convergence is achieved.

Fixing Direction based on Gradient



- Diagram illustrate Gradient Descent, a optimization algorithm used in machine learning to minimize the cost function J(w), where w represents the model's parameters (weights).
- The black dot on the right side of the graph represents the initial weight. When we start training the model, the weights are initialized randomly (or based on some heuristic), and this initial weight is often far from the global minimum.

- The gradient is the slope of the cost function at the current point. The dashed line shows the slope at the position of the initial weight.
- Gradient descent calculates the gradient (or derivative) of the cost function with respect to the weight w.
- If the gradient $\nabla J(w)$ is positive at the current weight, this means the cost function is increasing as the weight increases. In this case, need to decrease the weight to move toward the minimum.
- If the gradient $\nabla J(w)$ is negative, the cost function is decreasing as the weight increases. Here, need to increase the weight to move toward the minimum.

How the Gradient Affects Direction:

- If $\nabla J(w)$ is **positive**, the update formula becomes:

  $$w_{\text{new}} = w_{\text{old}} - \alpha \cdot (\text{positive number})$$

  This results in a **smaller weight** (because we're subtracting a positive value), meaning we move left on the cost function curve (downhill).

- If $\nabla J(w)$ is **negative**, the formula becomes:

  $$w_{\text{new}} = w_{\text{old}} - \alpha \cdot (\text{negative number})$$

  This effectively **increases the weight** (because subtracting a negative number is equivalent to adding), moving us right on the cost curve (also downhill).

Stochastic Gradient Descent (SGD)

Computation: Computes the gradient using a single randomly selected training example.

Efficiency: Efficient for large datasets due to less computation.

Convergence: Can be noisy and slower due to high variance in gradient estimates.

Use Cases: Well-suited for large datasets and online learning.

- **Batch Gradient Descent (BGD)**

Computation: Computes the gradient using the entire dataset.

Stability: More stable updates and smoother convergence.

Efficiency: Can be computationally expensive for large datasets.

Use Cases: Suitable for smaller datasets or well-behaved objective functions.

Learning Rate

Learning Rate (α)

- The learning rate is a key hyperparameter in gradient descent
- It controls how much to change the model's weights in response to the estimated gradient during training. It defines the step size that the algorithm takes while moving towards the minimum of the cost function.

Definition: Learning rate (α) is a scalar value that determines the size of the steps taken during each iteration of gradient descent.

Update Formula: In the weight update rule:

$$W_{new} = W_{old} - \alpha \cdot \nabla J(w)$$

α dictates how much to adjust the weights, w in the direction of the negative gradient $\nabla J(w)$.

Choosing Learning Rate: Selecting learning rate is crucial because it significantly impacts the training process:

Too High Learning Rate:

- When it is too high, the algorithm might take large steps and overshoot the minimum. This can lead to divergence, where the cost function increases instead of decreasing.
- Large steps can cause the algorithm to oscillate around the minimum and fail to converge, as it constantly "jumps" over the optimal solution.
- **Example:** Imagine descending a hill in leaps, where each leap overshoots the bottom of the hill, taking you to the opposite side.

Too Low Learning Rate:

- A very small learning rate results in small steps, causing the algorithm to take a long time to converge. It may eventually reach the minimum, but the process will be slow and inefficient.
- Sometimes, a very low learning rate might cause the algorithm to get stuck in local minima

Optimal Learning Rate:

- The ideal learning rate leads to fast and stable convergence. It allows the model to reach the global minimum efficiently without overshooting or slowing down.

To address the challenge of selecting the right learning rate, several strategies are used:

Learning Rate Decay: Gradually reduces the learning rate over time during training to allow for faster convergence initially and more precise steps later.

Adaptive Learning Rates: Algorithms like AdaGrad, RMSprop, and Adam automatically adjust the learning rate for each parameter, making the training more robust and efficient.

Bias-Variance Trade-off: In machine learning, the bias-variance trade-off is a fundamental concept that describes the relationship between a model's complexity, its accuracy on training data, and its ability to generalize to unseen data.

Key Concepts:

Bias: Systematic error introduced by a model's assumptions. A high-bias model makes strong assumptions about the data and tends to underfit, failing to capture the underlying patterns.

Variance: This measures the sensitivity of a model to variations in the training data. A high-variance model is highly sensitive to noise in the training data and tends to overfit, performing well on the training data but poorly on new data.

The Trade-off:

- **Simple Models:** These models have high bias and low variance. They make strong assumptions about the data, leading to underfitting. While they are less sensitive to noise in the training data, they may not capture the underlying patterns effectively.
- **Complex Models:** These models have low bias and high variance. They are flexible and can capture complex patterns in the data, but they are also more susceptible to noise. This can lead to overfitting, where the model performs well on the training data but poorly on new data.

The Goal: The ideal model strikes a balance between bias and variance, minimizing both to achieve optimal performance. This balance is often referred to as the "sweet spot."

Strategies to Address the Trade-off:

Regularization: L1 and L2 regularization techniques can be used to penalize complex models and reduce overfitting.

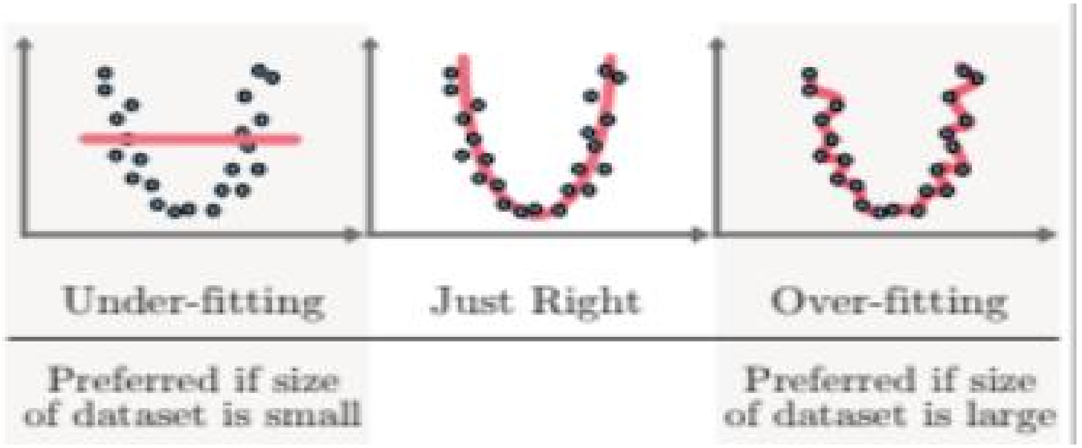
Feature Selection: By selecting the most relevant features, we can reduce the model's complexity and improve its generalization ability.

Ensemble Methods: Combining multiple models (e.g., bagging, boosting) can reduce variance and improve overall performance.

Cross-Validation: This technique helps assess a model's performance on unseen data and can be used to tune hyperparameters and select the optimal model complexity.

Overfitting and Underfitting

Overfitting and underfitting are two common problems that can occur when training machine learning models.



Overfitting:

- Occurs when a model learns not only the underlying patterns in the data but also the noise and irrelevant details.
- Results in high accuracy on training data but poor performance on unseen (test) data.

Underfitting:

- Happens when a model is too simple to capture the underlying patterns in the data.
- The model performs poorly on both training and test sets.

Ways to avoid overfitting and underfitting:

Right model complexity: By using a model that is not too complex or too simple.

Regularization: It can be used to prevent overfitting by adding a penalty term to the model's loss function.

Cross-validation: It can be used to evaluate the performance of a model on new data.

Tips for avoiding overfitting and underfitting:

Data augmentation techniques: Data augmentation can be used to create new training data. This can help to make the model more robust to noise and variations in the data.

Early stopping: It can be used to stop training a model when it starts to overfit.

Ensemble learning: It can be used to combine multiple models in order to improve performance.

Cross-Validation Techniques:

- Cross-validation is a powerful technique in machine learning to assess the performance of a model on unseen data.
- It helps prevent overfitting and provides a more reliable estimate of the model's generalization ability.

Types of Cross-Validation Techniques

1. Hold-Out Method:

- A simpler approach where data is split into training and test sets, usually in a 70-30 or 80-20 ratio.
- Efficient but may not provide as robust an estimate as k-fold cross-validation.

2. K-Fold Cross-Validation

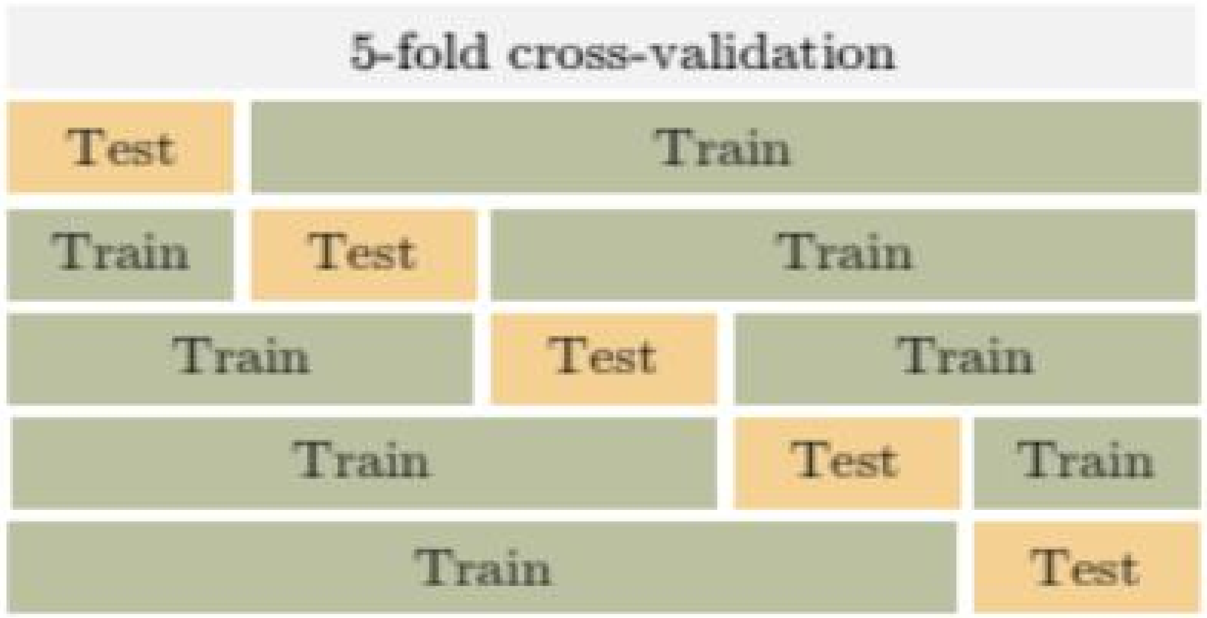
- Divides the dataset into k equal-sized folds. Common choice: k=5 or k=10.
- Train on k-1 folds and test on the remaining one fold.
- Averages the performance across all folds to get a more robust estimate.
- Repeats this process k times, using each fold as the validation set once.

3. Stratified k-Fold Cross-Validation

- Ensures that each fold contains a representative proportion of samples from each class (for classification problems).
- Useful for imbalanced datasets.

4. Leave-One-Out Cross-Validation (LOOCV)

- A special case of k-fold cross-validation where k equals the number of fold data points.
- Each fold data point is used once as a test, while the rest are used to train the model.
- It’s like testing the model with one point at a time, using all the others to build it.



Regularization Techniques (L1 and L2)

- Regularization is a technique used to prevent overfitting in machine learning models.
- It introduces a penalty term to the loss function, which discourages complex models.

Types of Regularization:

1. Lasso Regression (L1)(Least Absolute Shrinkage and Selection Operator):

Adds a penalty equivalent to the absolute value of the magnitude of coefficients.

Formula: Loss = Loss original $+\lambda\sum| \ w|$

Effect: L1 regularization can reduce some coefficients to exactly zero, effectively performing feature selection by eliminating features with little predictive power.

Application: Effective when there are many features, and it's beneficial to identify the most relevant ones.

L2 Regularization (Ridge):

- Adds a penalty equivalent to the square of the magnitude of coefficients.
- **Formula:** Loss = Loss original $+\lambda\sum w^2$
- **Effect:** L2 regularization shrinks all coefficients towards zero, but it does not reduce them to zero. This technique helps to keep all features but with smaller coefficients.
- Suitable when all features have some predictive value, even if small.

High Dimensional Data: High-dimensional data refers to datasets with a large number of features (or dimensions) relative to the number of observations.

Challenges in High Dimensions:

- Models can easily overfit due to the high complexity of the feature space.
- Increased computational cost and memory requirements.
- Visualization and interpretation become challenging.

Examples:

- Genomic data (e.g., thousands of genes as features).
- Text data represented by word frequencies or embeddings (e.g., thousands of words as features).

Curse of Dimensionality:

- Refers to the exponential increase in computational difficulty and sparsity of data as the number of dimensions increases.
- In high-dimensional spaces, distances between points become less meaningful because all points become equidistant.

Consequences:

- Poor model generalization due to overfitting.
- Increased computation and memory requirements.
- Diminished interpretability of results.

Example:

- In a 2D space, adding one more point provides useful information about the space. In 100D, the data becomes sparse, and each new point adds less useful information.

Covariance: A measure of how two variables change together. If the covariance between two variables is positive, they increase together; if negative, one increases as the other decreases. Formula for covariance between two variables X and Y:

$$\mathrm{Cov}(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})$$

Covariance Matrix: A square matrix that captures the covariance between each pair of variables in a dataset with multiple variables. For a dataset with variables X1, X2,…,Xn, the covariance matrix Σ is:

$$\Sigma = \begin{pmatrix} \mathrm{Cov}(X_1,X_1) & \mathrm{Cov}(X_1,X_2) & \cdots & \mathrm{Cov}(X_1,X_n) \\ \mathrm{Cov}(X_2,X_1) & \mathrm{Cov}(X_2,X_2) & \cdots & \mathrm{Cov}(X_2,X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(X_n,X_1) & \mathrm{Cov}(X_n,X_2) & \cdots & \mathrm{Cov}(X_n,X_n) \end{pmatrix}$$

Interpretation: Diagonal elements represent the variance of each variable (e.g., Var(Xi)=Cov(Xi, Xi)). Off-diagonal elements represent the covariance between pairs of different variables.

Uses: Helps in understanding correlations among multiple variables.

Multivariate Regression: A regression model where multiple dependent variables are predicted based on a set of independent variables. Useful when there are multiple outcomes or response variables.

Model Representation: The model for multivariate regression with m response variables can be written as:

$$Y=XB+E$$

Y is an $n \times m$ matrix of dependent variables (response matrix).

X is an $n \times p$ matrix of independent variables (design matrix).

B is a $p \times m$ matrix of coefficients (parameter matrix) for each response.

E is an $n \times m$ matrix of residuals or errors.

Objectives:

Estimate the coefficient matrix B to predict multiple dependent variables from the independent variables.

Minimize the difference between predicted and actual values of the dependent variables.

Multicollinearity in Multivariate Regression

Definition:

- Multicollinearity occurs when independent variables are highly correlated with each other, which can make it difficult to determine the individual effect of each predictor on the dependent variable(s).
- In multivariate settings, multicollinearity can affect the stability and interpretability of the coefficient matrix B.

Detection: Variance Inflation Factor (VIF) is common tools to detect multicollinearity.

Solutions: Ridge regression and Lasso regression (as regularization techniques) can help mitigate multicollinearity by penalizing large coefficients.

Parametric vs. Non-Parametric Methods

Parametric Methods

Characteristics:

- Relies on a predefined function to model the data (e.g., linear, polynomial).
- Parameter estimation is often done through methods like Maximum Likelihood Estimation (MLE) or Least Squares.

Examples:

Linear Regression: Assumes a linear relationship between the dependent and independent variables.

Logistic Regression: Models the probability of a binary outcome using a logistic function.

Naive Bayes: Assumes independence between features and is based on Bayes' theorem.

Assumptions: Parametric models often assume normality, independence, and homoscedasticity (equal variance).

Use Cases:

Linear Regression: Predicting outcomes where a linear relationship is expected (e.g., predicting salary based on years of experience).

Logistic Regression: Classifying binary outcomes (e.g., predicting if a customer will make a purchase or not).

Naive Bayes: Text classification tasks like spam filtering and sentiment analysis.

Advantages:

- Simple and interpretable.
- Requires less data, making it suitable for small datasets.
- Fast training and prediction.

Disadvantages:

- Limited flexibility due to strong assumptions.
- Performance declines if data violates model assumptions (e.g., non-linearity in linear regression).

Non-Parametric Methods

Characteristics:

- Do not rely on a specific functional form or distribution.
- Often rely on local information and are more data-driven, making them flexible and adaptable to various data shapes.

Examples:

- **k-Nearest Neighbors (k-NN):** Classifies a data point based on the classes of its nearest neighbors.
- **Decision Trees:** Divides data into subgroups based on feature values without assuming a distribution.
- **Kernel Density Estimation (KDE):** Estimates the probability density function of a random variable without assuming a specific form.
- **Support Vector Machine (SVM):** Can be non-parametric if using a kernel function, enabling complex decision boundaries.

Use Cases:

- **k-NN:** Suitable for classification problems where class boundaries are complex and unknown.
- **Decision Trees:** Useful for both classification and regression tasks with non-linear relationships.
- **Kernel Density Estimation:** Estimating the distribution of data points (e.g., to understand income distribution in a population).
- **SVM with Non-linear Kernels:** Effective in text classification, image recognition, and complex classification tasks.

Advantages:

- High flexibility, allowing the model to fit complex patterns in the data.
- Can handle non-linear relationships effectively.
- Suitable for a variety of applications, from classification to density estimation.

Disadvantages:

- Require larger datasets to perform well.
- Can be computationally intensive, especially with large datasets.
- Higher risk of overfitting, particularly with high-dimensional data.

Feature	Parametric Methods	Non-Parametric Methods
Assumptions	Assume specific distribution (e.g., normal)	Minimal or no distributional assumptions
Parameters	Fixed number of parameters	Number of parameters can grow with data
Flexibility	Less flexible; limited by the chosen distribution	More flexible; can adapt to complex data patterns
Data Requirements	Perform well with small to medium-sized datasets	Typically require larger datasets for accuracy
Computational Cost	Generally lower	Can be computationally intensive
Examples	Linear regression, logistic regression, Naive Bayes	k-Nearest Neighbors, decision trees, kernel density estimation
Overfitting Risk	Lower risk if assumptions hold	Higher risk, especially with complex models