EE-508: Hardware Foundations for Machine Learning

Lecture 3: Quick Review of ML Algorithms

University of Southern California

Ming Hsieh Department of Electrical and Computer Engineering

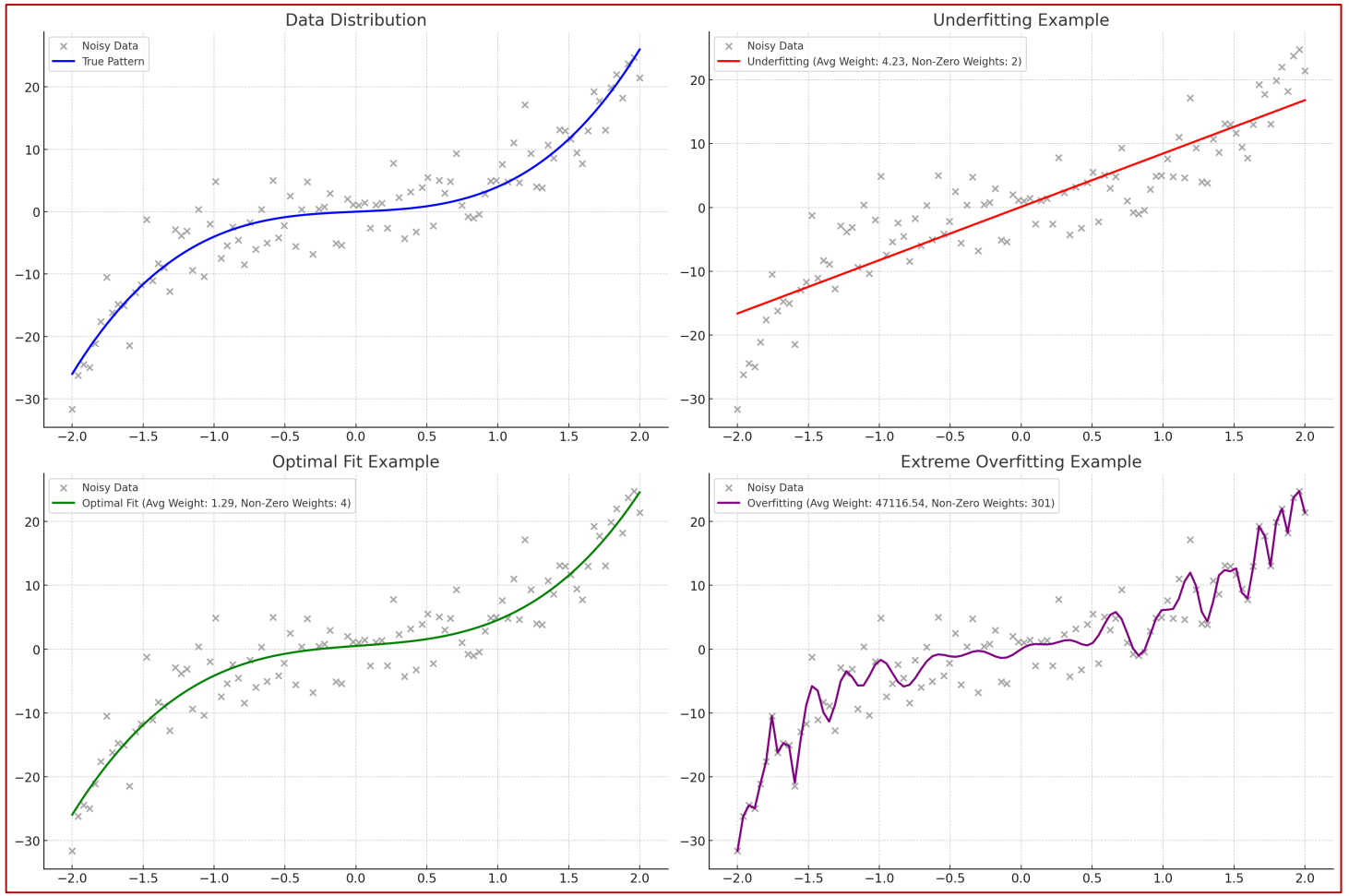
Instructor:
Arash Saifhashemi

Overfitting

- **Definition**: When the model model learns not only the underlying pattern but also the noise and specifics of the training data.
- **Result**: The model performs **well** on **training data** but **poorly** on **unseen/test data**.

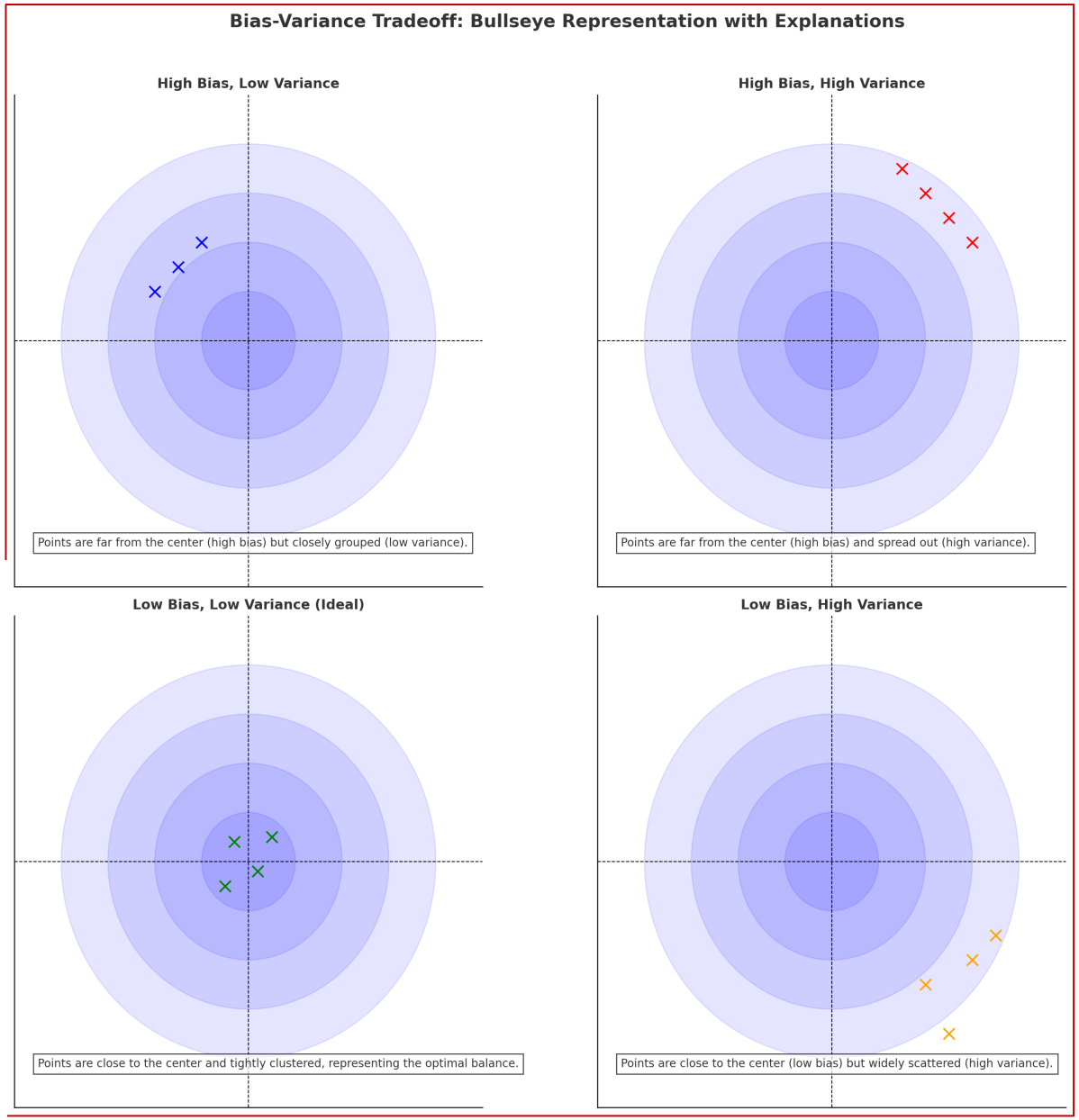
Causes of Overfitting

- Too complex models (e.g., very deep neural networks, high-degree polynomials).
- Insufficient training data.
- Noise or irrelevant features in the data.



Bias-Variance Trade-off

- **Bias**: Error due to approximating a **complex** problem with a **simple model**.
  - High bias leads to **underfitting**, where the model fails to capture the underlying patterns in the data.



Bias vs Variance

Bias-Variance Trade-off

- **Bias**: Error due to approximating a **complex** problem with a **simple model**.
  - High bias leads to **underfitting**, where the model fails to capture the underlying patterns in the data.
- **Variance**: Error due to **sensitivity** to small **fluctuations** in the training data.
  - High variance leads to **overfitting**, where the model captures noise as if it were a real pattern.

Bias vs Variance

Bias-Variance Trade-off

- **Bias**: Error due to approximating a **complex** problem with a **simple model**.
  - High bias leads to **underfitting**, where the model fails to capture the underlying patterns in the data.
- **Variance**: Error due to **sensitivity** to small **fluctuations** in the training data.
  - High variance leads to **overfitting**, where the model captures noise as if it were a real pattern.

•**High Bias**: Simplify the model (e.g., linear regression for non-linear data).

•**High Variance**: often a result of a **complex model** with many parameters (e.g., deep neural networks, high-degree polynomials).

Bias vs Variance

How to Prevent Overfitting?

- A **high-variance model** is **too sensitive** to the training data. It captures even the **noise** in the data, mistaking it for meaningful patterns.
  - Use simpler models (reduce complexity).
  - Increase training data.
  - Apply regularization (e.g., L1/L2 penalties).
  - Other methods:
    - Use cross-validation to monitor performance on unseen data.
    - Prune decision trees or restrict model depth.
    - Apply dropout in neural networks.

Regularization

- **Definition**: Adding a penalty to the loss function, **discouraging large model weights**.
- **Purpose**:
  - Reduce overfitting.
  - Improve generalization to unseen data.
  - Simplify models by controlling complexity.

$$\text{Minimize: } ||y - Xw||^2 + \lambda||w||_1$$

L1 Regularization (Lasso)

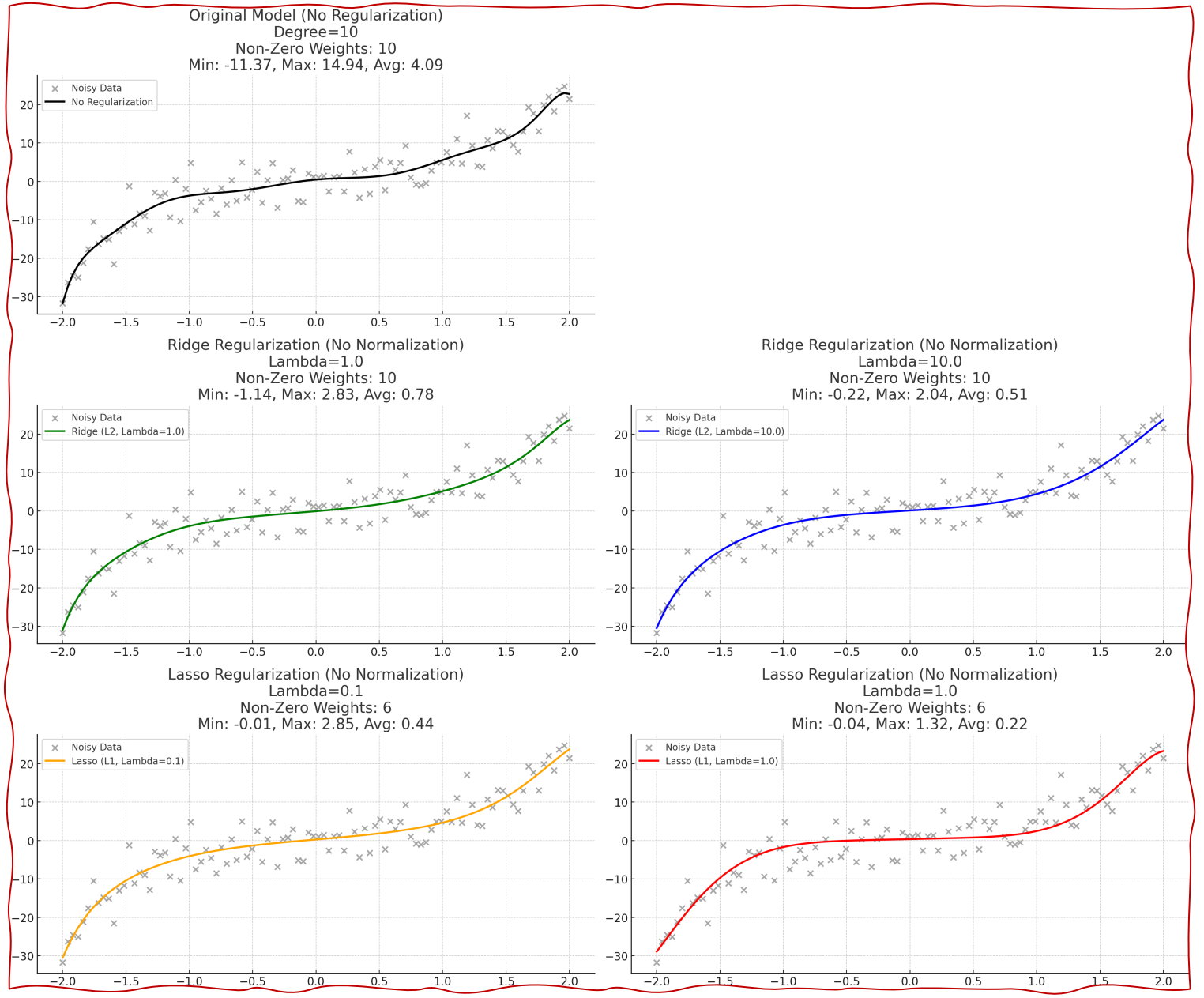
- **Effect**:
  - Shrinks some weights to zero, effectively selecting features.
  - Produces sparse models.

$$\text{Minimize: } ||y - Xw||^2 + \lambda||w||^2$$

L2 Regularization (Ridge)

- **Effect**: Reduces all weights proportionally.
- Retains all features.

By limiting the magnitude of model parameters (e.g., coefficients in linear models), it makes the model less sensitive to small noises in the training data, thus reducing variance.



- L1 (Lasso) attempts to shrink some weights to zero but is hindered by uneven feature scaling.
- L2 (Ridge) reduces all weights proportionally

L1 vs L2 Regularization

Feature	Lasso (L1)	Ridge (L2)
Regularization Term	Adds the sum of the absolute values of the coefficients	Adds the sum of the squared values of the coefficients
Effect on Coefficients	Shrinks some coefficients to exactly zero (feature selection)	Shrinks coefficients towards zero, but rarely to exactly zero
Sparsity	Creates sparse models (many coefficients are zero)	Does not create sparse models
Model Interpretability	Improves interpretability by selecting a subset of features	May provide some improvement in interpretability, but retains all features
Computational Efficiency	Can be more computationally expensive for very large datasets	Generally more computationally efficient for very large datasets
Use Cases	High-dimensional datasets Situations where feature selection is important Cases where model interpretability is crucial	Situations where all features are potentially relevant Improving model stability and generalization

- Lasso eliminates irrelevant features, resulting in a sparse model.
- Ridge reduces weight magnitudes but retains all features.

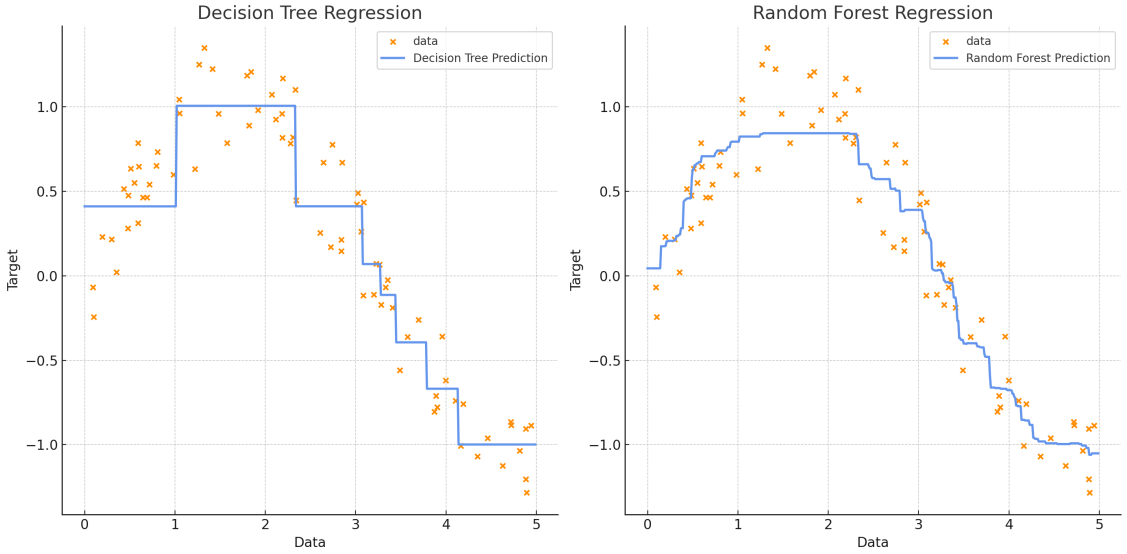
Regression

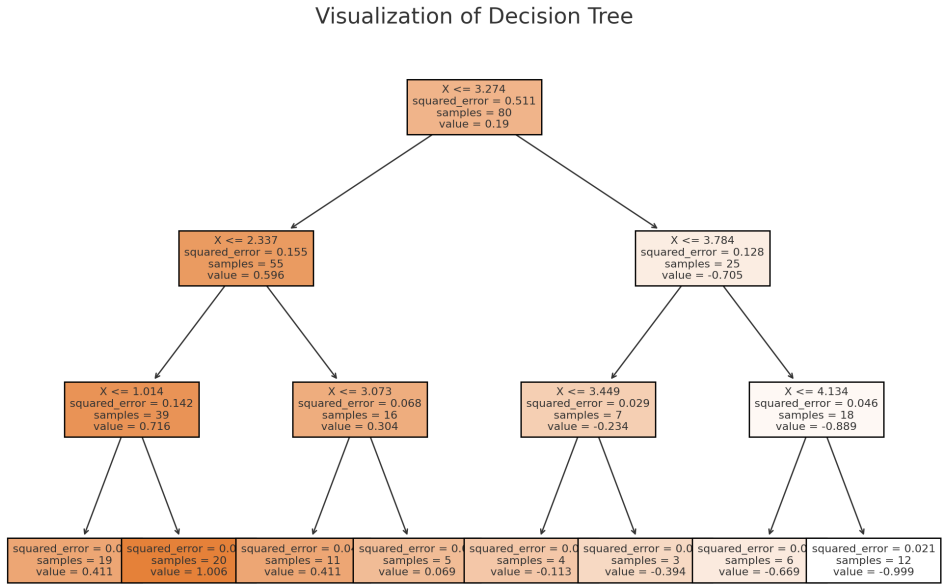
Some Famous Regression Algorithms

- **Linear Regression**
  - **Concept**: Models the relationship between inputs and output as a linear equation.
  - **Equation**: $y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$
  - **Use Cases**: Predicting sales, stock prices, etc.
- **Polynomial Regression**
  - **Concept**: Extends linear regression by fitting a polynomial curve to the data.
  - **Equation**: $y = w_0 + w_1 x + w_2 x^2 + \ldots + w_n x^n$
  - **Use Cases**: Capturing nonlinear trends.

Some Famous Regression Algorithms

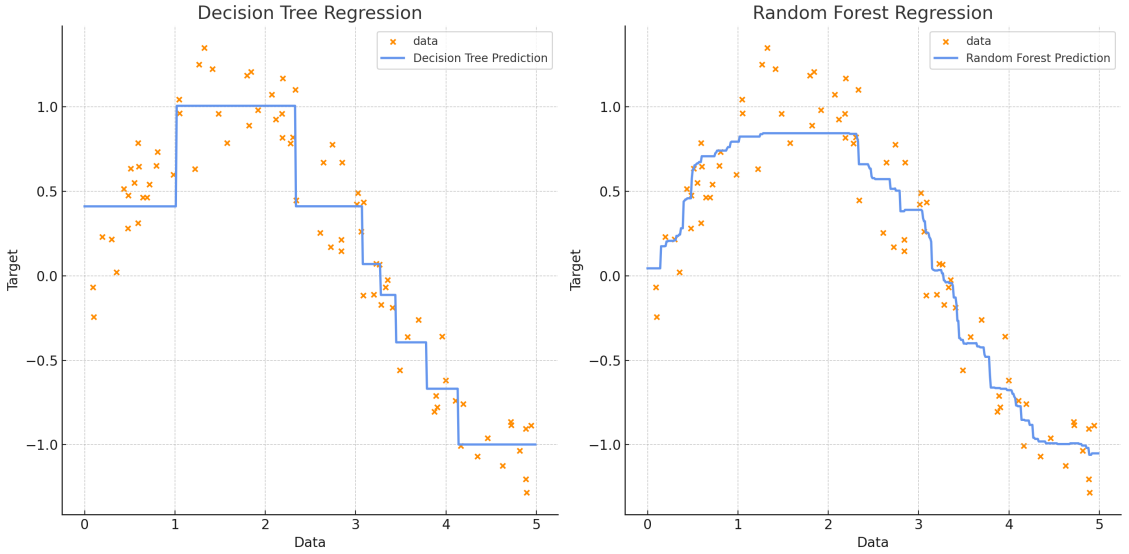
- **Decision Tree Regression**
  - **Concept**: Splits the data into regions and fits constant values in each region.
  - **Advantage**: Nonlinear, interpretable.
  - **Use Cases**: Price prediction, forecasting

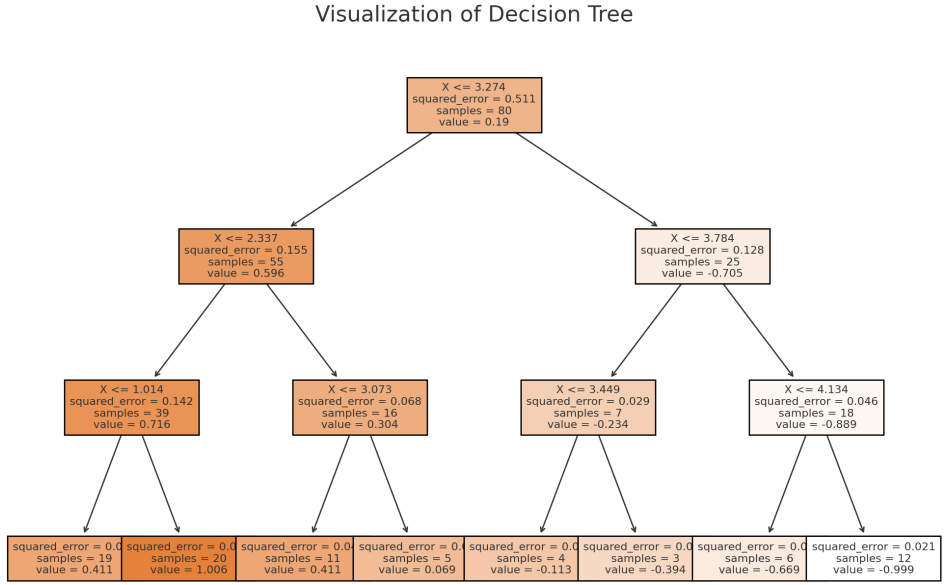




Some Famous Regression Algorithms

- **Decision Tree Regression**
  - **Concept**: Splits the data into regions and fits constant values in each region.
  - **Advantage**: Nonlinear, interpretable.
  - **Use Cases**: Price prediction, forecasting
- **Random Forest Regression**
  - **Concept**: Uses an ensemble of decision trees to improve robustness and reduce overfitting.
  - **Advantage**: High accuracy for many tasks.
  - **Use Cases**: Predicting housing prices, energy consumption.





Some Famous Regression Algorithms

- **Support Vector Regression (SVR)**
  - **Concept**: Extends SVM to regression by fitting a hyperplane within a margin of tolerance.
  - **Kernel Options**: Linear, polynomial, RBF.
  - **Use Cases**: Predicting time-series data.

Primal Form: Minimize:

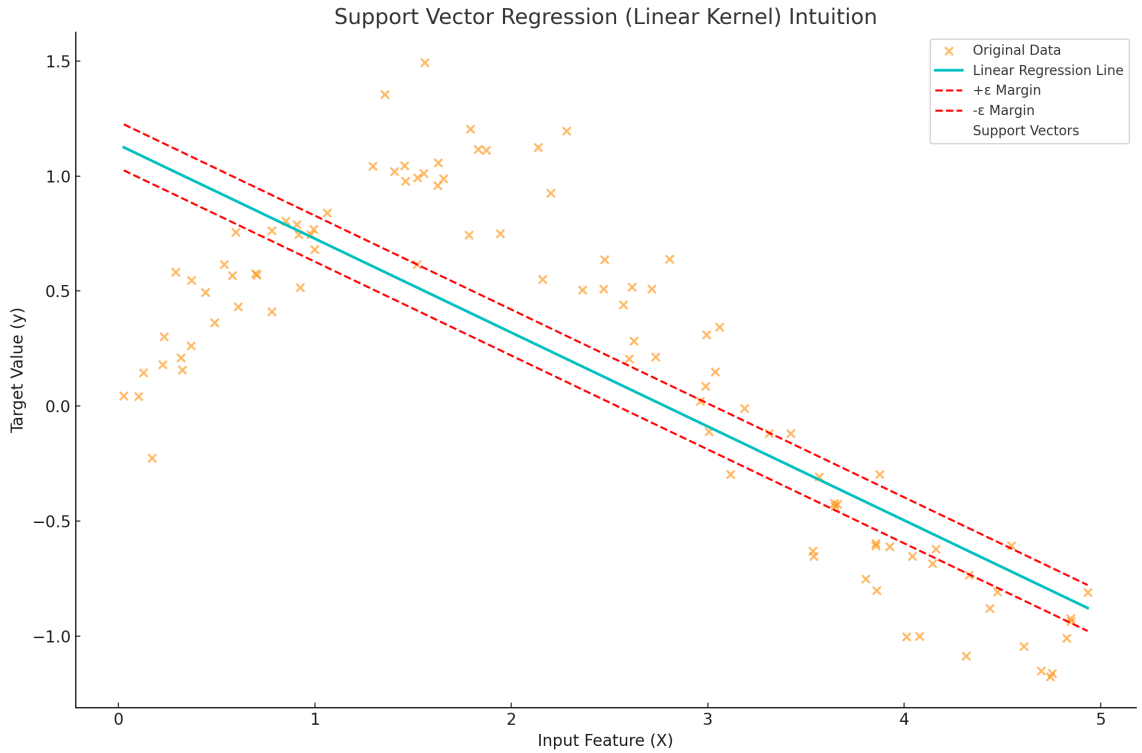
$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

Subject to:

$$y_i - (w \cdot x_i + b) \leq \epsilon + \xi_i$$
$$(w \cdot x_i + b) - y_i \leq \epsilon + \xi_i^*$$
$$\xi_i, \xi_i^* \geq 0$$

- w: Weight vector.
- b: Bias term.
- ξ_i, ξ_i^*: Slack variables for deviations outside the ϵ-tube.
- C: Regularization parameter controlling the trade-off between margin width and slack penalties.

Formulation for linear function $f(x) = w \cdot x + b$



Some Famous Regression Algorithms

- **Support Vector Regression (SVR)**
- **Idea**: Predicts target values within a margin of tolerance ϵ (the **epsilon-tube**).
  - Minimizes the error for points outside the epsilon-tube.
- **How It Works**
  - **Epsilon-Tube**:
    - A band of width 2ϵ around the regression function f(x).
    - Predictions within this tube are not penalized, allowing some tolerance for error.
  - **Support Vectors**:
    - Points outside or on the boundaries of the epsilon-tube.
    - These points "support" the regression model, influencing the shape of f(x).
  - **Slack Variables**:
    - Allow for deviations outside the epsilon-tube.

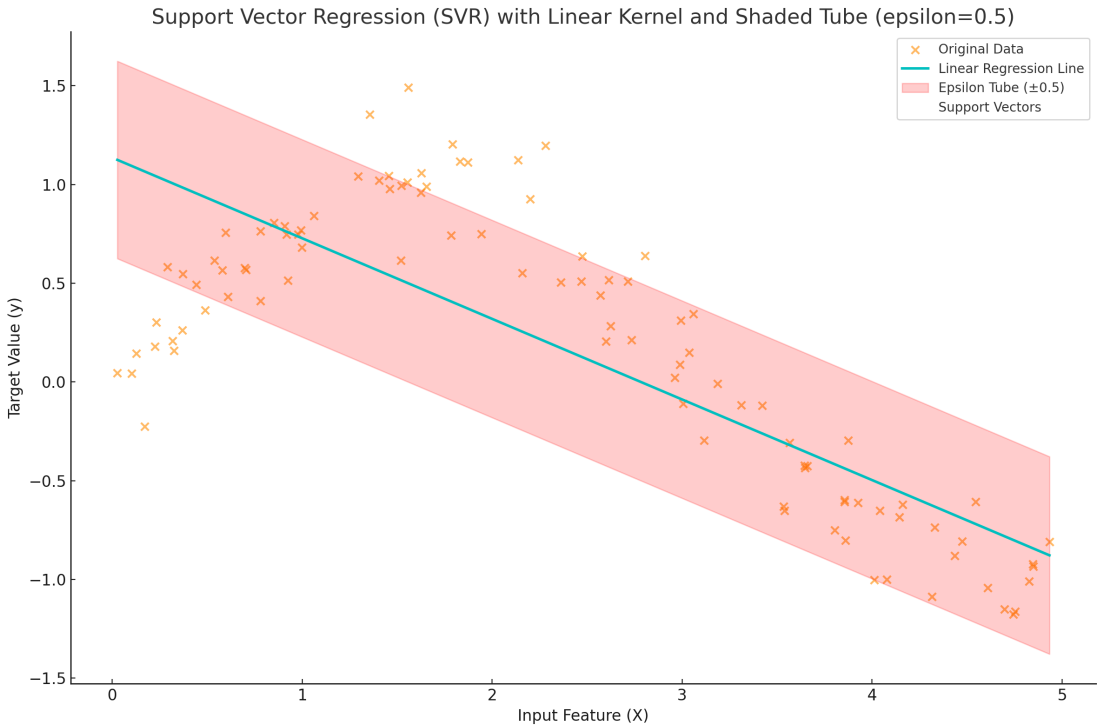
Optimization Goal:

- Minimize:

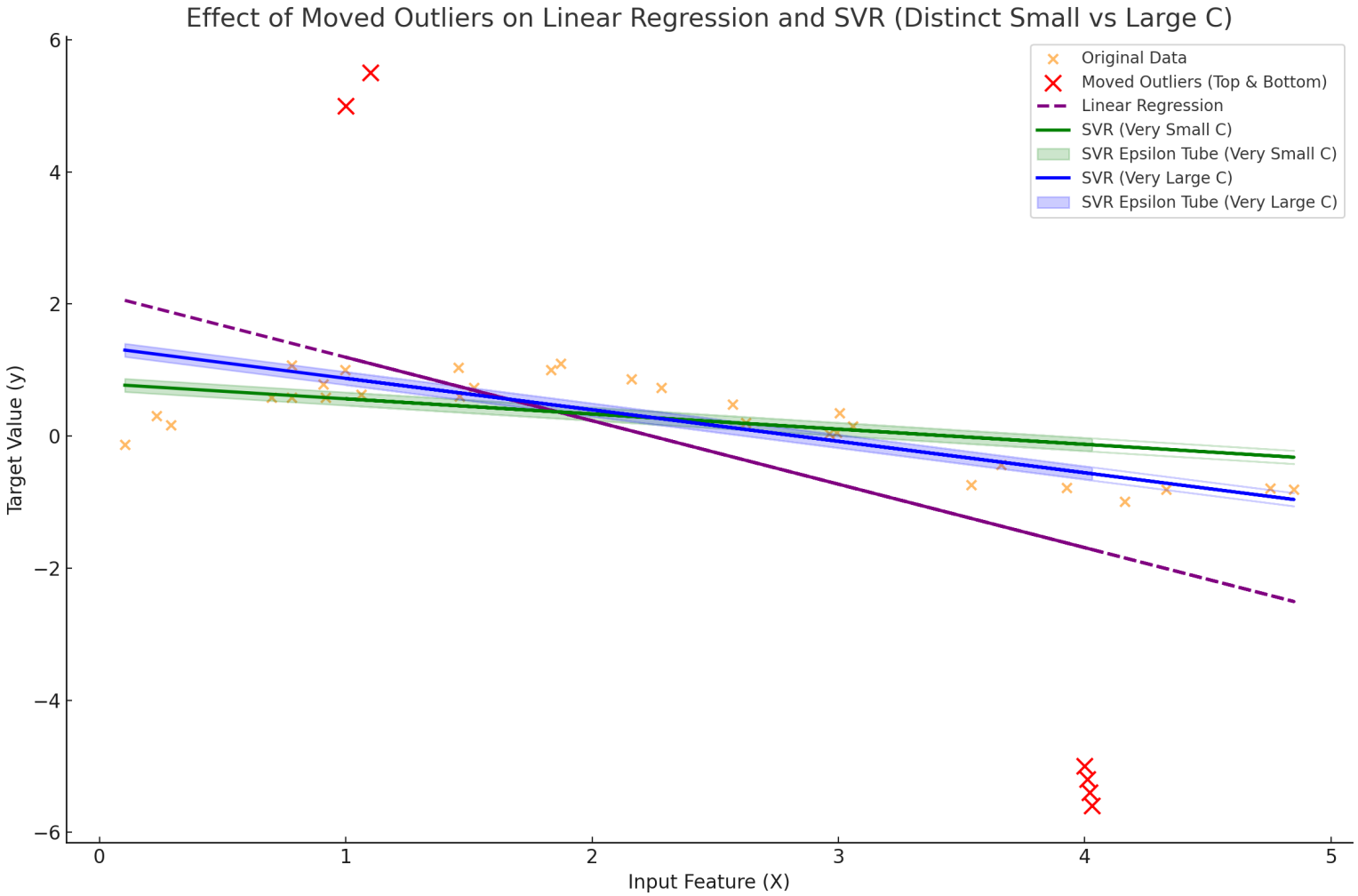
$$L(w) = \frac{1}{2}\|w\|^2 + C\sum(\xi_i + \xi_i^*)$$

  - $\|w\|^2$: Regularization term to control model complexity.
  - C: Penalty for errors outside the tube.

Formulation for linear function $f(x) = w \cdot x + b$



Resistance Against Outliers



Optimization Goal:

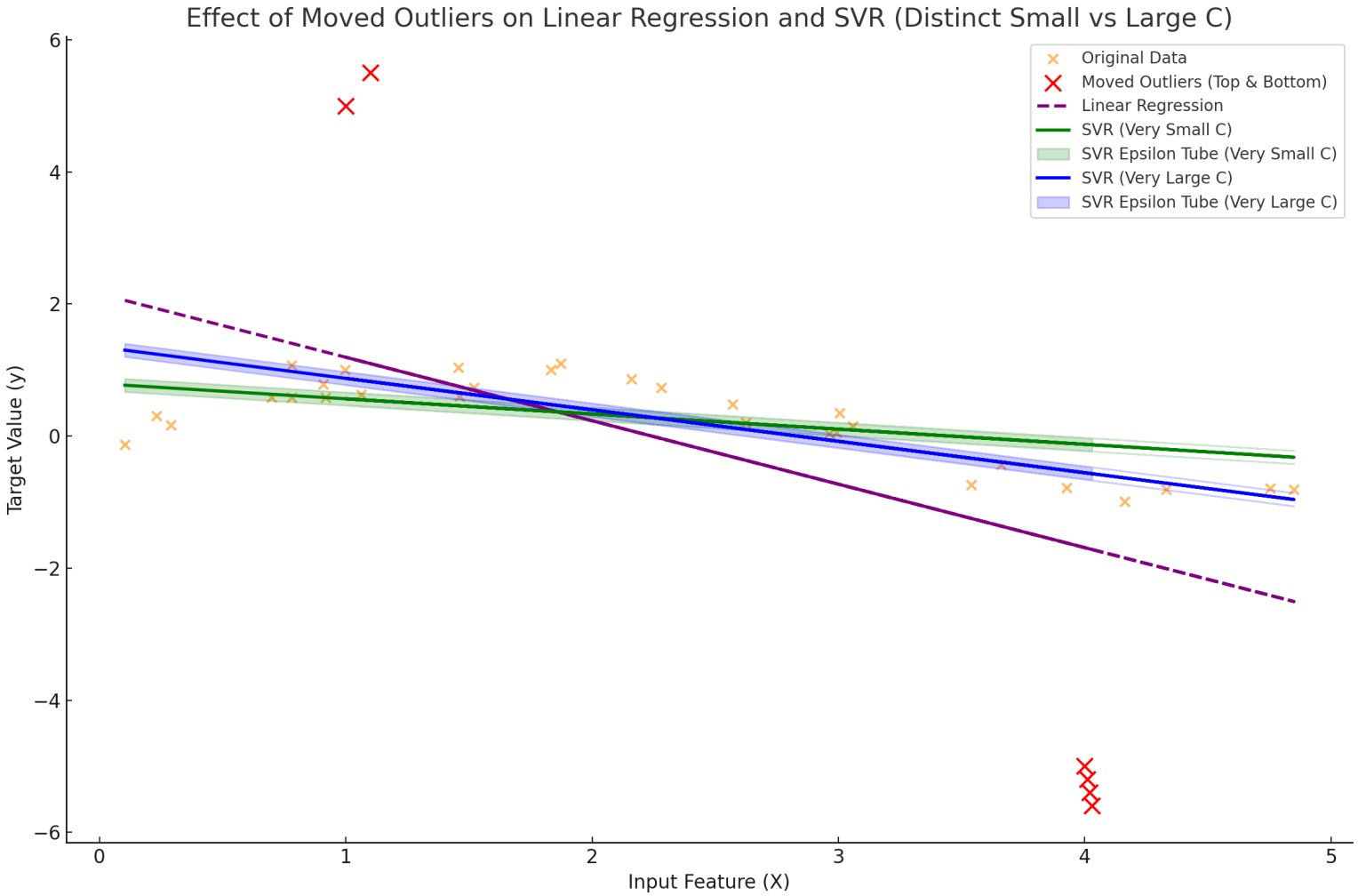
- Minimize:

$$L(w) = \frac{1}{2}\|w\|^2 + C\sum(\xi_i + \xi_i^*)$$

  - $\|w\|^2$: Regularization term to control model complexity.
  - C: Penalty for errors outside the tube.

- **Green line (Very Small C)**:
  - Produces a smoother regression line that minimizes the influence of outliers.
- **Blue line (Very Large C)**:
  - Attempts to fit tightly to all data, including outliers.
- **Purple dashed line**: Linear Regression, heavily skewed by outliers.

Resistance Against Outliers



Optimization Goal:

- Minimize:

$$L(w) = \frac{1}{2}\|w\|^2 + C\sum(\xi_i + \xi_i^*)$$

  - $\|w\|^2$: Regularization term to control model complexity.
  - C: Penalty for errors outside the tube.

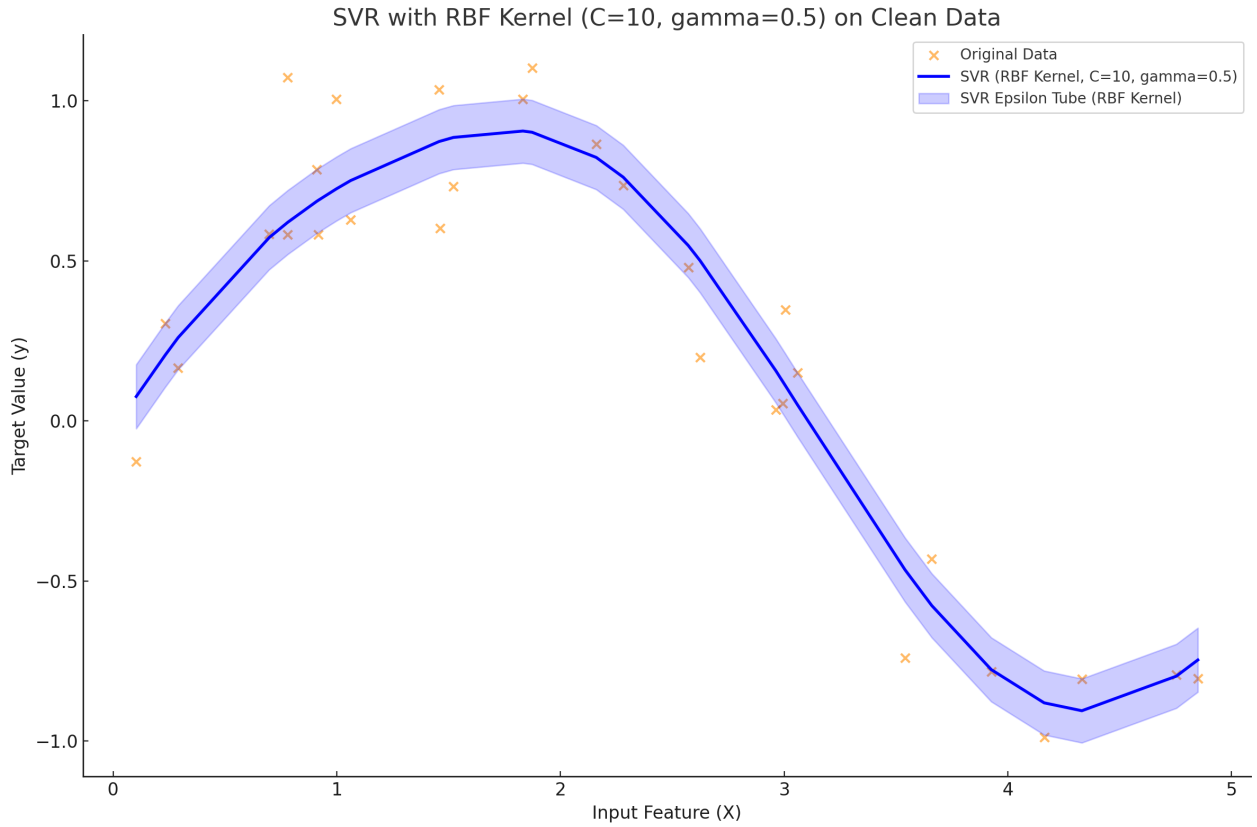
- **Green line (Very Small C)**:
  - Produces a smoother regression line that minimizes the influence of outliers.
- **Blue line (Very Large C)**:
  - Attempts to fit tightly to all data, including outliers.
- **Purple dashed line**: Linear Regression, heavily skewed by outliers.

Radial Basis Function (RBF)

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right)$$

- $K(x_i, x_j)$: Measures similarity between points x_i and x_j.
- $\|x_i - x_j\|^2$: Squared Euclidean distance between the two points.
- γ: Kernel parameter controlling the "spread" of the similarity function.

- **Close Points**: If two points are close in feature space, the kernel value K will be close to 1, indicating high similarity.
- **Far Points**: If two points are far apart, K approaches 0, indicating low similarity.
- γ determines how quickly this similarity drops off with distance.



Some Famous Regression Algorithms

- **Gradient Boosting Machines (GBM)**
  - **Concept**: Builds models sequentially, where each model corrects errors of the previous ones.
  - **Examples**: XGBoost, LightGBM, CatBoost.
  - **Use Cases**: Competitive modeling tasks.
- **Boosting Concept**: Combines multiple weak learners (usually decision trees) to form a strong model.
  - **Gradient Descent**: Optimizes the model by minimizing a loss function iteratively.

Steps:

1. **Start with Initial Prediction**: Often the mean value for regression or log-odds for classification.

Some Famous Regression Algorithms

- **Gradient Boosting Machines (GBM)**
  - **Concept**: Builds models sequentially, where each model corrects errors of the previous ones.
  - **Examples**: XGBoost, LightGBM, CatBoost.
  - **Use Cases**: Competitive modeling tasks.
- **Boosting Concept**: Combines multiple weak learners (usually decision trees) to form a strong model.
  - **Gradient Descent**: Optimizes the model by minimizing a loss function iteratively.

Steps:

1. **Start with Initial Prediction**: Often the mean value for regression or log-odds for classification.
2. **Compute Residuals**: Measure the difference between true values and current predictions.

Some Famous Regression Algorithms

- **Gradient Boosting Machines (GBM)**
  - **Concept**: Builds models sequentially, where each model corrects errors of the previous ones.
  - **Examples**: XGBoost, LightGBM, CatBoost.
  - **Use Cases**: Competitive modeling tasks.
- **Boosting Concept**: Combines multiple weak learners (usually decision trees) to form a strong model.
  - **Gradient Descent**: Optimizes the model by minimizing a loss function iteratively.

Steps:

1. **Start with Initial Prediction**: Often the mean value for regression or log-odds for classification.
2. **Compute Residuals**: Measure the difference between true values and current predictions.
3. **Fit a Weak Learner**: Train a decision tree on the residuals.

Some Famous Regression Algorithms

- **Gradient Boosting Machines (GBM)**
  - **Concept**: Builds models sequentially, where each model corrects errors of the previous ones.
  - **Examples**: XGBoost, LightGBM, CatBoost.
  - **Use Cases**: Competitive modeling tasks.
- **Boosting Concept**: Combines multiple weak learners (usually decision trees) to form a strong model.
  - **Gradient Descent**: Optimizes the model by minimizing a loss function iteratively.

Steps:

1. **Start with Initial Prediction**: Often the mean value for regression or log-odds for classification.
2. **Compute Residuals**: Measure the difference between true values and current predictions.
3. **Fit a Weak Learner**: Train a decision tree on the residuals.
4. **Update Prediction**: Add the scaled predictions of the new tree to the model.

Some Famous Regression Algorithms

- **Gradient Boosting Machines (GBM)**
  - **Concept**: Builds models sequentially, where each model corrects errors of the previous ones.
  - **Examples**: XGBoost, LightGBM, CatBoost.
  - **Use Cases**: Competitive modeling tasks.
- **Boosting Concept**: Combines multiple weak learners (usually decision trees) to form a strong model.
  - **Gradient Descent**: Optimizes the model by minimizing a loss function iteratively.

Steps:

1. **Start with Initial Prediction**: Often the mean value for regression or log-odds for classification.
2. **Compute Residuals**: Measure the difference between true values and current predictions.
3. **Fit a Weak Learner**: Train a decision tree on the residuals.
4. **Update Prediction**: Add the scaled predictions of the new tree to the model.
5. **Iterate**: Repeat until the desired number of iterations or convergence.

Boosting

$$\hat{y} = F_0(x) + \sum_{m=1}^{M} \eta \cdot h_m(x)$$

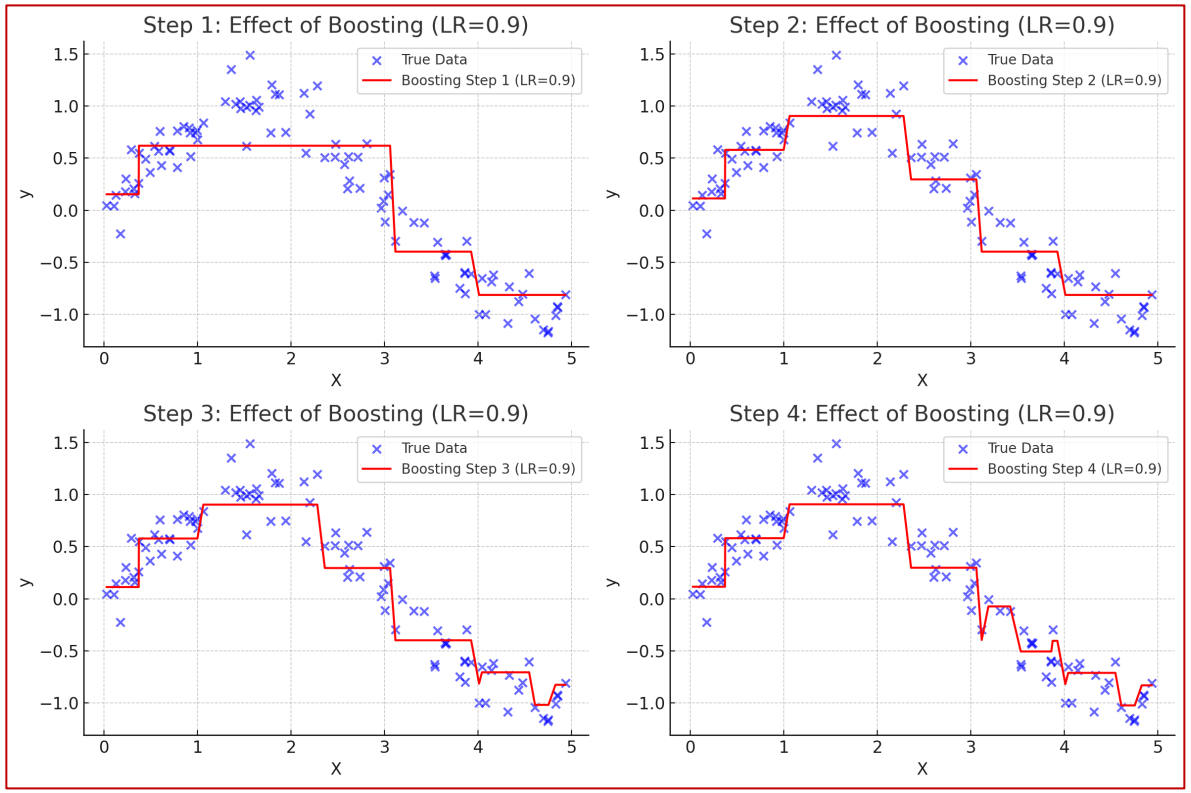
- $F_0(x)$: Initial prediction (e.g., mean of the target in regression).
- $h_m(x)$: Prediction of the m-th weak learner.
- η: Learning rate (scales each learner's contribution).
- M: Number of predictors (trees).

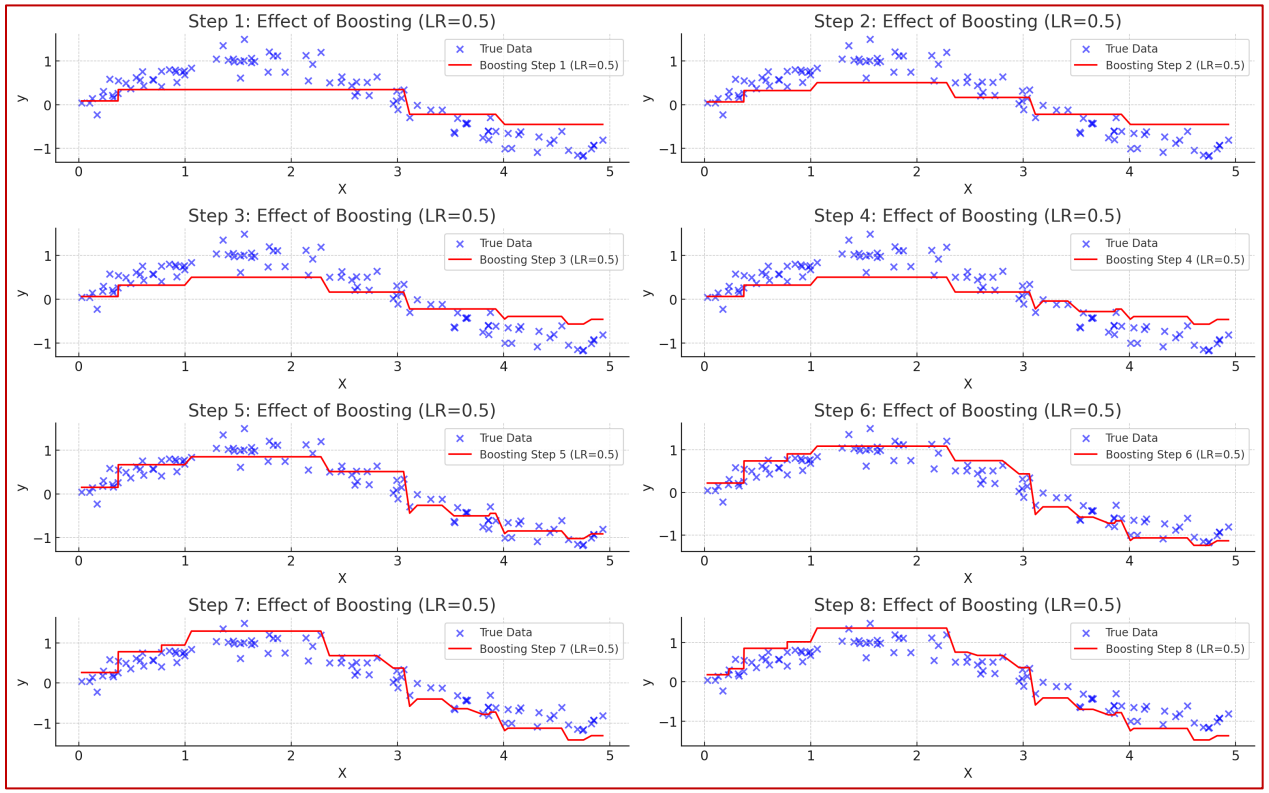
Example:

- Suppose you have 4 predictors and a learning rate of 0.1.
- Each predictor outputs corrections like: $h_1(x) = 0.3$, $h_2(x) = 0.2$, $h_3(x) = -0.1$, $h_4(x) = 0.05$.
- The final prediction is:

$$\hat{y} = F_0(x) + 0.1 \cdot (0.3 + 0.2 - 0.1 + 0.05)$$

Boosting





Learning Rate = 0.9
Need less steps

Learning Rate = 0.5
Need more steps

Random Forest

Example:

- Original dataset: $[A, B, C, D, E]$
- Bootstrap sample (with replacement): $[B, D, B, A, C]$

- **Based on** Bagging (Bootstrap Aggregation)
- **Steps:**
  - **Data Sampling**: Randomly sample the training data (with replacement) for each tree (bootstrap sampling).
  - **Tree Training**: Train each decision tree independently on its bootstrap sample.
  - **Prediction by Trees**: Each tree generates a numerical prediction for the target value.
  - **Aggregation**: Combine the predictions from all trees by **averaging** their outputs to produce the final prediction.

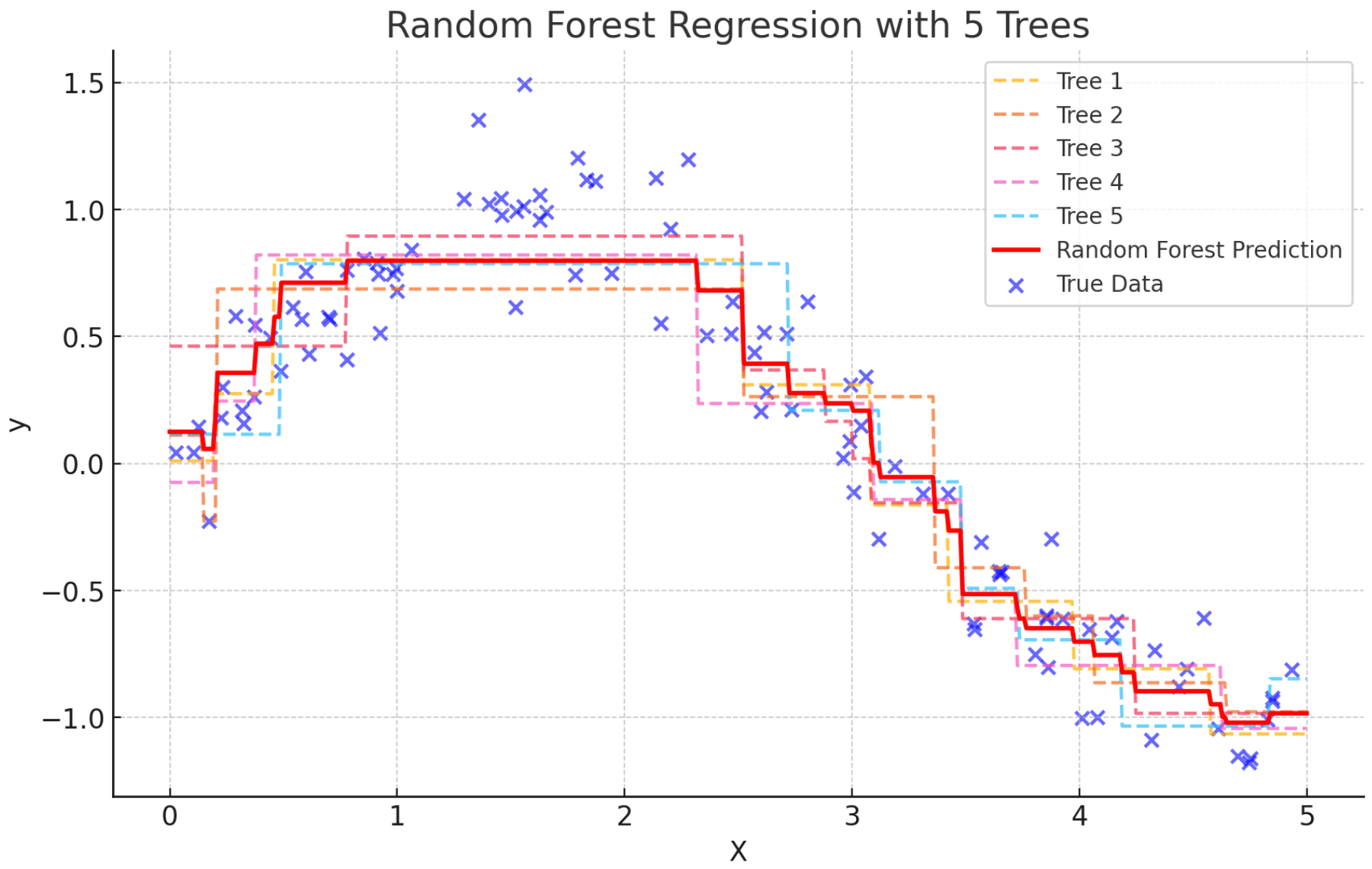
Final prediction = Average of predictions from all trees:

$$\hat{y} = \frac{1}{N} \sum_{i=1}^{N} T_i(x)$$

Where $T_i(x)$ is the prediction of the i-th tree, and N is the number of trees.

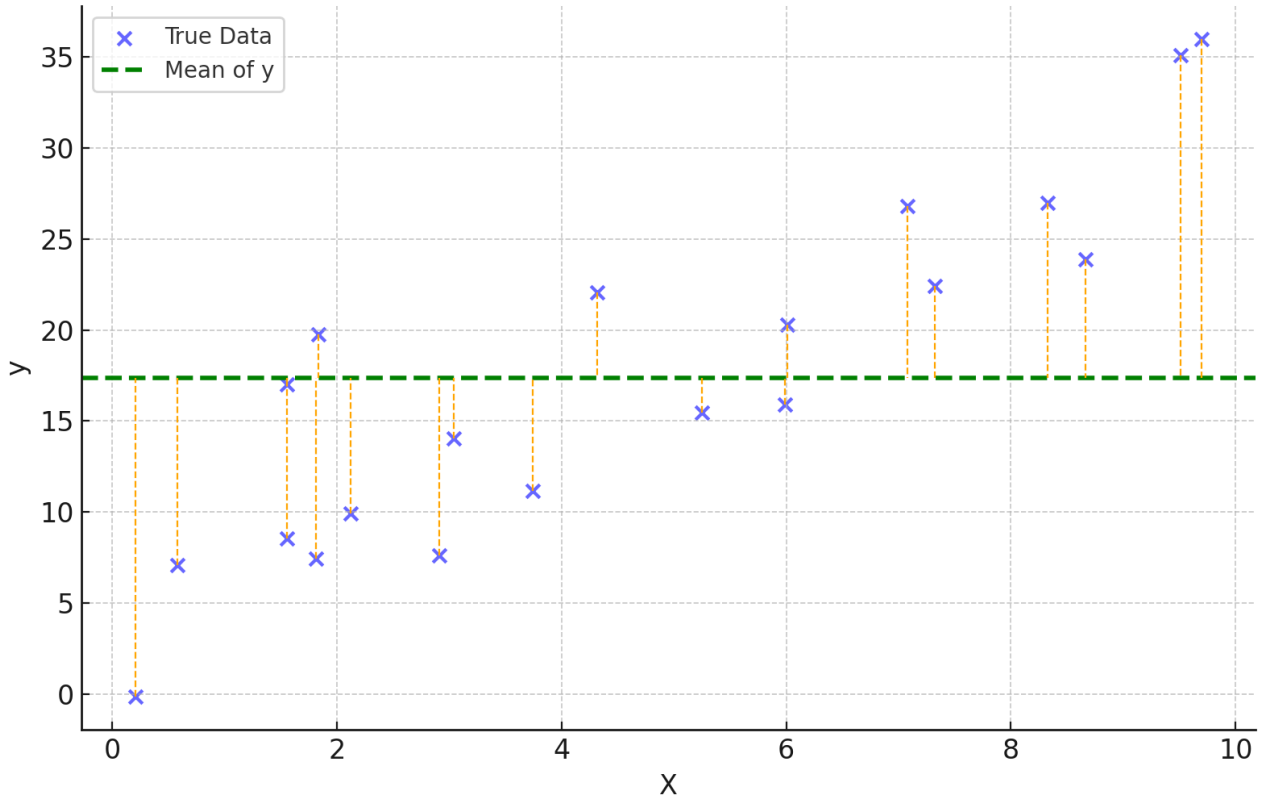
- Random Forest reduces **overfitting** by averaging multiple trees, making it robust.
- Random selection of features ensures **diversity** among trees, improving generalization.

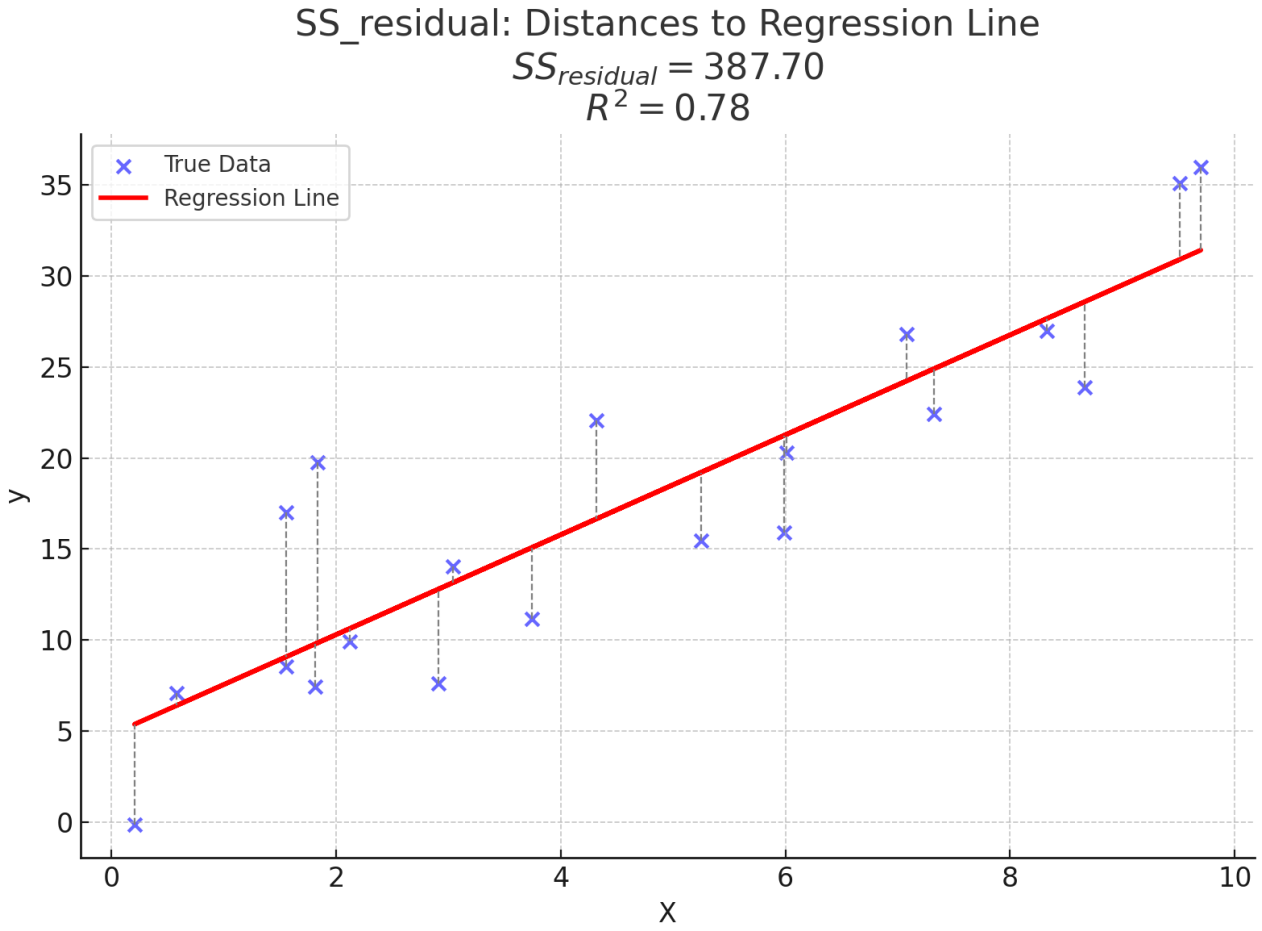
Random Forest Example



The Quality of Regression

R^2

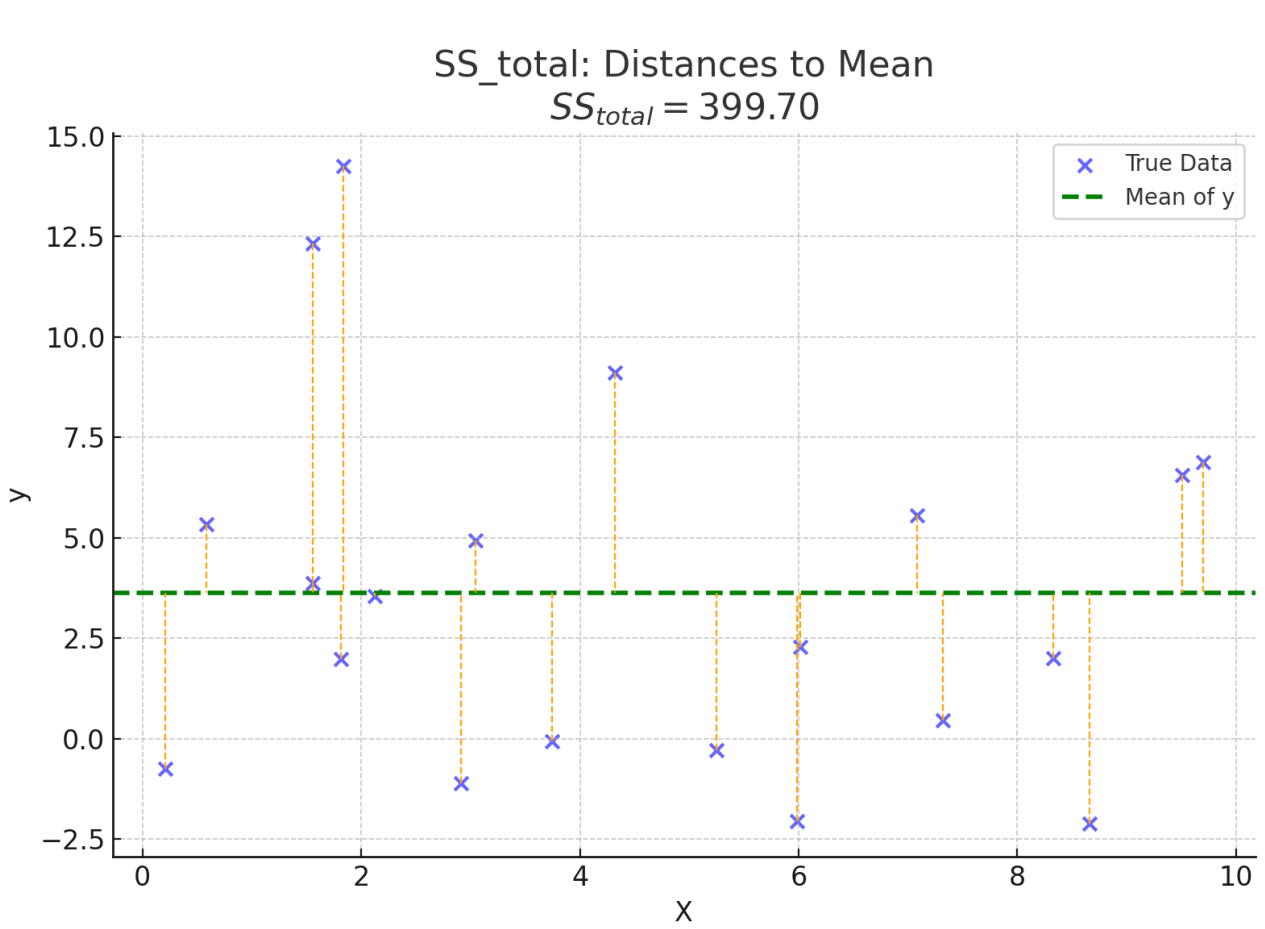


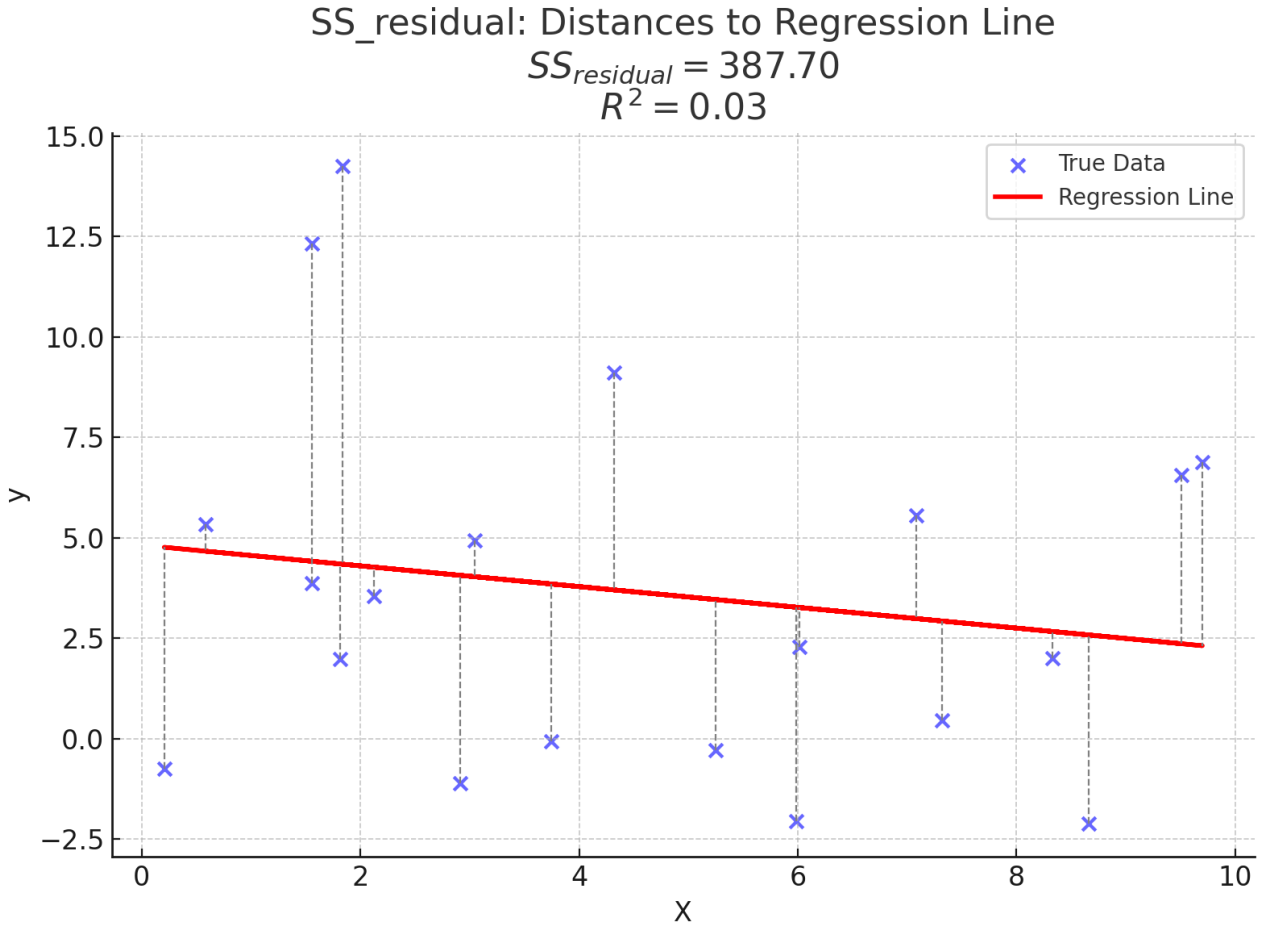


$$R^2 = 1 - \frac{\text{SS}_{\text{residual}}}{\text{SS}_{\text{total}}}$$

How much better the regression model is compared to the **average line** (horizontal line at the mean of the target values

R^2 for a Bad Model





$$R^2 = 1 - \frac{\mathrm{SS}_{\mathrm{residual}}}{\mathrm{SS}_{\mathrm{total}}}$$

Evaluate Regression Performance

How much better the regression model is compared to the **average line** (horizontal line at the mean of the target values

$$R^2 = 1 - \frac{\text{SS}_{\text{residual}}}{\text{SS}_{\text{total}}}$$

Where:

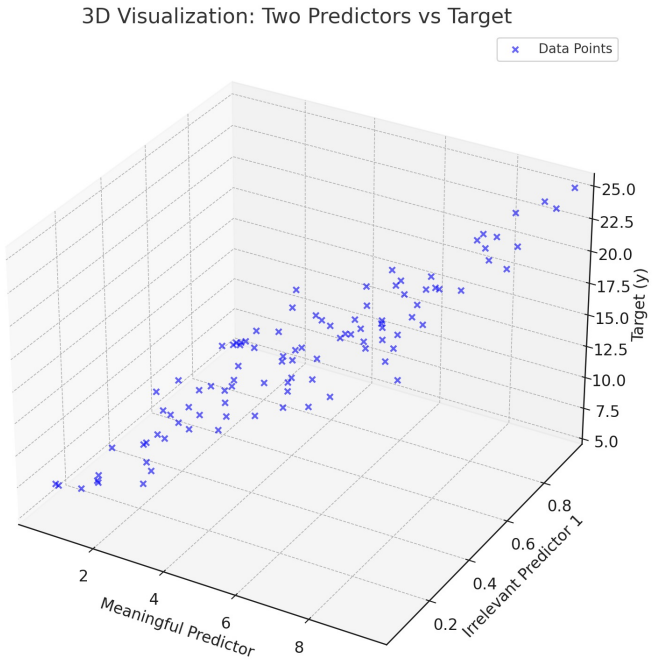
- $\text{SS}_{\text{residual}}$: Sum of squared residuals (errors) = $\sum(y_i - \hat{y}_i)^2$
- SS_{total}: Total sum of squares = $\sum(y_i - \bar{y})^2$
- y_i: Actual value of the target.
- $\hat{y}_i$: Predicted value of the target.
- $\bar{y}$: Mean of the actual target values.

Interpretation:

1. $R^2 = 1$: Perfect model, explains all the variance in the target variable.
2. $R^2 = 0$: The model explains none of the variance, equivalent to using the mean of y as the prediction.
3. $R^2 < 0$: The model is worse than a horizontal line at $\bar{y}$, indicating poor performance.

Adjusted R^2

Meaningful Predictor	Irrelevant Predictor 1	Target (y)
0.06	0.14	5.63
0.21	0.89	6.33
0.25	0.94	5.79
0.34	0.08	6.83
0.45	0.59	4.99
0.46	0.32	6.33
0.58	0.62	7.64
0.64	0.89	6.93
0.65	0.99	6.56
0.74	0.92	6.71



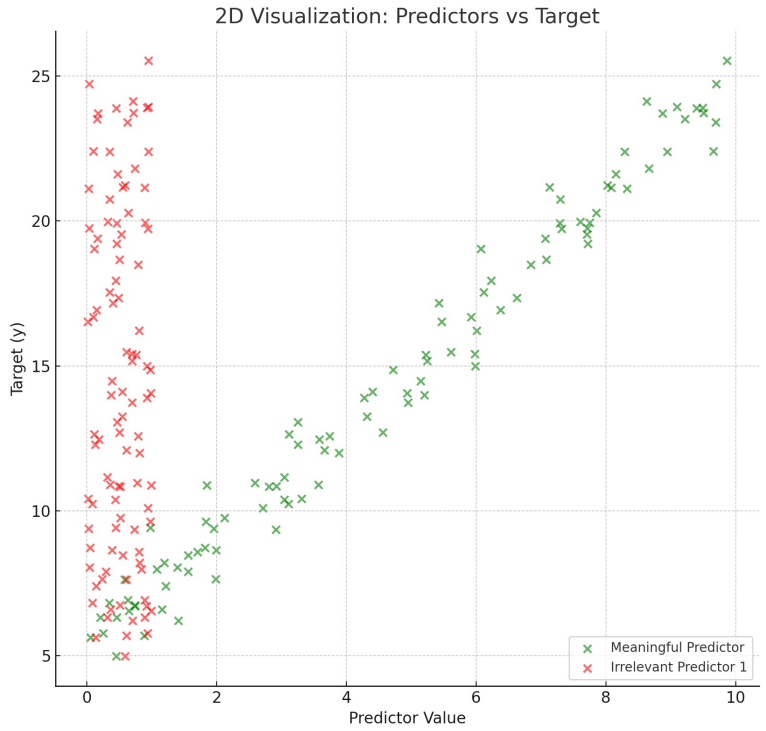
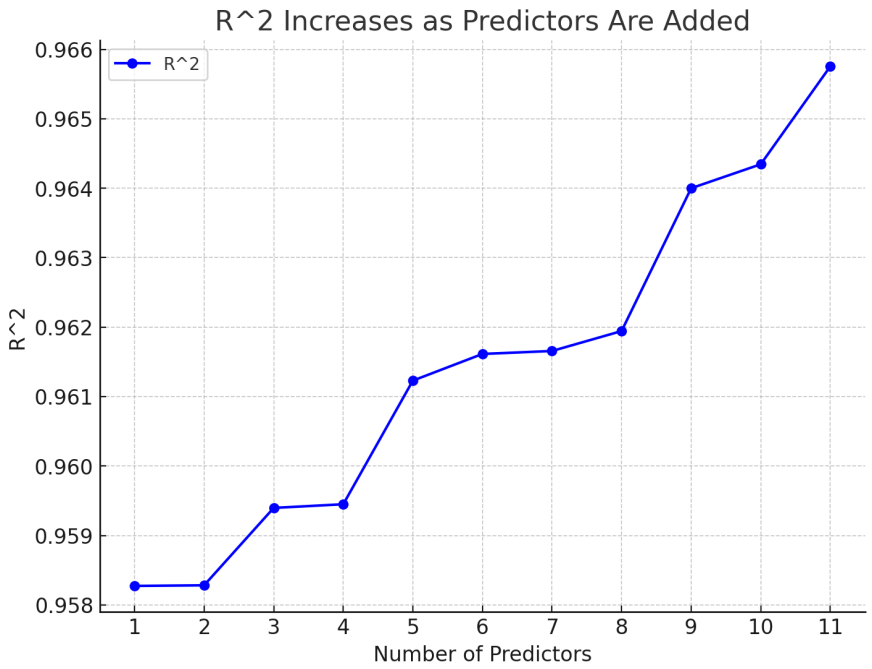


Table on the left shows predictors and target values. Plots on the right illustrate the strong influence of the meaningful predictor and the irrelevance of the noise predictor. Adjusted R^2 addresses this by penalizing irrelevant predictors, ensuring only meaningful ones improve the model's performance.

Adjusted R^2



$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

Decreases

Does not change

Where:

- $SS_{\text{res}} = \sum(y_i - \hat{y}_i)^2$: Residual sum of squares (unexplained variance).
- $SS_{\text{tot}} = \sum(y_i - \bar{y})^2$: Total sum of squares (total variance in y).

$$\hat{y} = b_0 + b_1X_1 + b_2X_2 + \cdots + b_pX_p$$

Where:

- b_0: Intercept.
- $b_1, b_2, \ldots, b_p$: Coefficients for the predictors $X_1, X_2, \ldots, X_p$.

Adding a new predictor X_{p+1} leads to:

$$\hat{y} = b_0 + b_1X_1 + b_2X_2 + \cdots + b_pX_p + b_{p+1}X_{p+1}$$

- Original model: $\hat{y} = b_0 + b_1X_1 + b_2X_2$
- After adding a new feature: $\hat{y} = b_0 + b_1X_1 + b_2X_2 + b_3X_3$
- Adding X_3 allows the model to adjust $\hat{y}$ to better fit the data, reducing the residuals.

Adjusted R^2

Formula:

$$R^2_{\text{adj}} = 1 - \left(\frac{(1 - R^2)(n - 1)}{n - p - 1} \right)$$

$$R^2_{\text{adj}} \in (-\infty, 1]$$

Where:

- R^2: Regular R^2, which measures the proportion of variance explained by the model.
- n: Number of observations (data points).
- p: Number of predictors (features).

When n is Too Large ($n \gg p$):

- The term $n - p - 1 \approx n$ because p is much smaller than n.
- The penalty for adding predictors becomes negligible, and Adjusted R^2 is very close to R^2:

$$R^2_{\text{adj}} \approx R^2$$

This happens because the model has enough data to reliably estimate the effects of predictors.

- If $p \approx n$, the denominator $n - p - 1$ approaches **0**, making the adjustment term very large:

$$R^2_{\text{adj}} = 1 - \left((1 - R^2) \cdot \frac{n - 1}{\text{small value}} \right)$$

Adjusted R^2

Formula:

$$R^2_{\text{adj}} = 1 - \left(\frac{(1 - R^2)(n - 1)}{n - p - 1} \right)$$

$$R^2_{\text{adj}} \in (-\infty, 1]$$

Where:

- R^2: Regular R^2, which measures the proportion of variance explained by the model.
- n: Number of observations (data points).
- p: Number of predictors (features).

- **Prevents Overfitting**: Penalizes irrelevant predictors, increasing only for meaningful improvements.
- **Balances Complexity**: Accounts for dataset size and predictor count in its formula.
- **Fair Model Comparison**: Compares models with different numbers of predictors effectively.
- **Supports Larger Datasets**: Allows more predictors as dataset size increases.
- **Favors Simplicity**: Rewards models with genuine explanatory value, discouraging unnecessary complexity.

Classification

Probability vs Odds

1. Probability (p):

- The likelihood of getting heads or tails.
  - For a fair coin:

$$p = 0.5 \quad (\text{probability of heads})$$

Probability vs Odds

1. Probability (p):

- The likelihood of getting heads or tails.
  - For a fair coin:

$$p = 0.5 \quad (\text{probability of heads})$$

2. Odds:

- **Definition**: The ratio of the probability of getting heads (p) to the probability of not getting heads ($1 - p$).
- **Formula**:

$$\text{Odds} = \frac{p}{1 - p}$$

- For a fair coin:

$$\text{Odds} = \frac{0.5}{1 - 0.5} = 1$$

  - Interpretation: The odds of getting heads are **1:1** (equal chance of heads and tails).
- If the coin were biased, with $p = 0.8$ (80% chance of heads):

$$\text{Odds} = \frac{0.8}{1 - 0.8} = 4$$

  - Interpretation: The odds of getting heads are **4:1** (4 times more likely to get heads than tails).

Log-Odds

$$\text{Log-Odds} = \ln\left(\frac{p}{1-p}\right)$$

- For a fair coin ($p = 0.5$):

$$\text{Log-Odds} = \ln(1) = 0$$

  - Interpretation: The log-odds of getting heads are **0**, meaning equal chance.

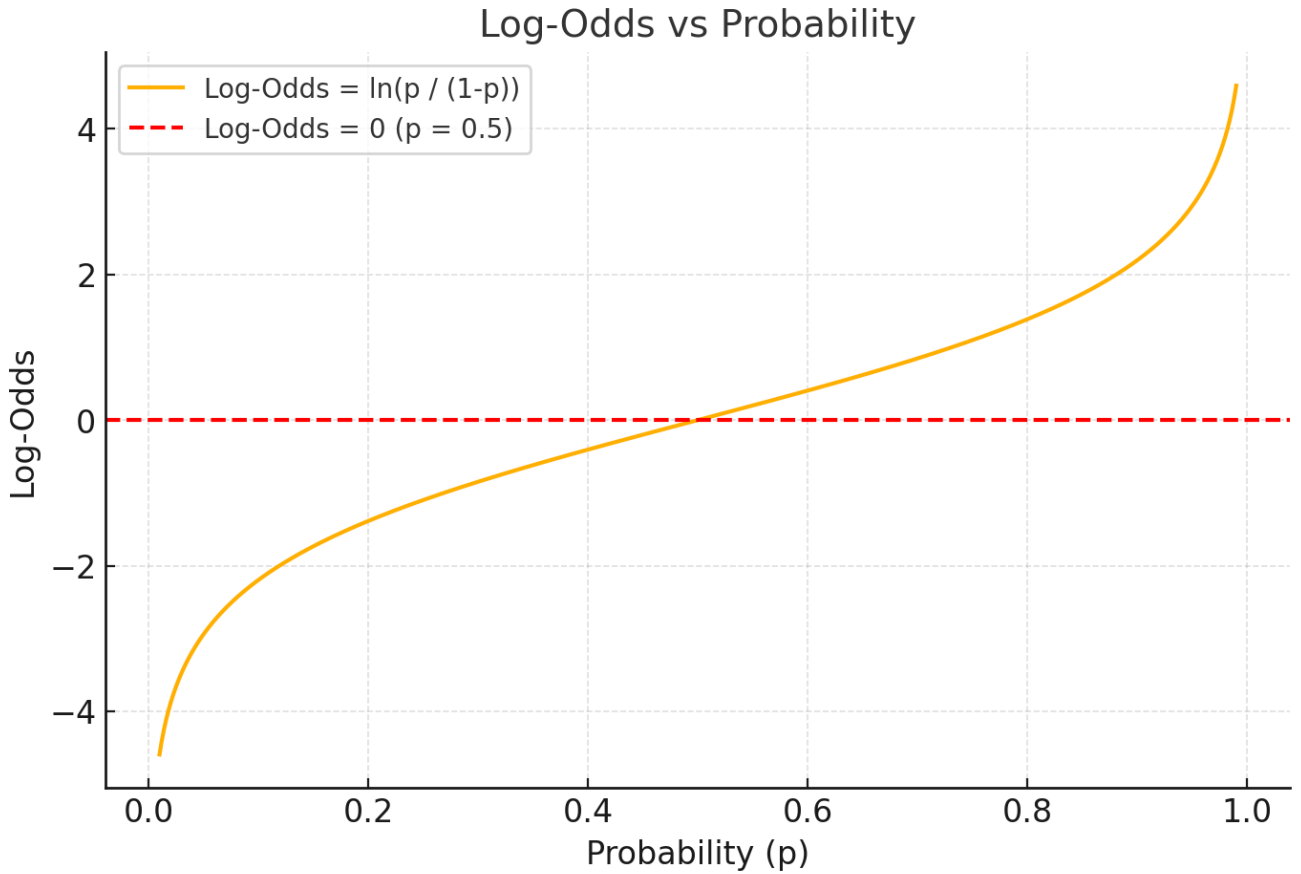
- For a biased coin ($p = 0.8$):

$$\text{Log-Odds} = \ln(4) \approx 1.386$$

  - Interpretation: Positive log-odds indicate heads are more likely than tails.

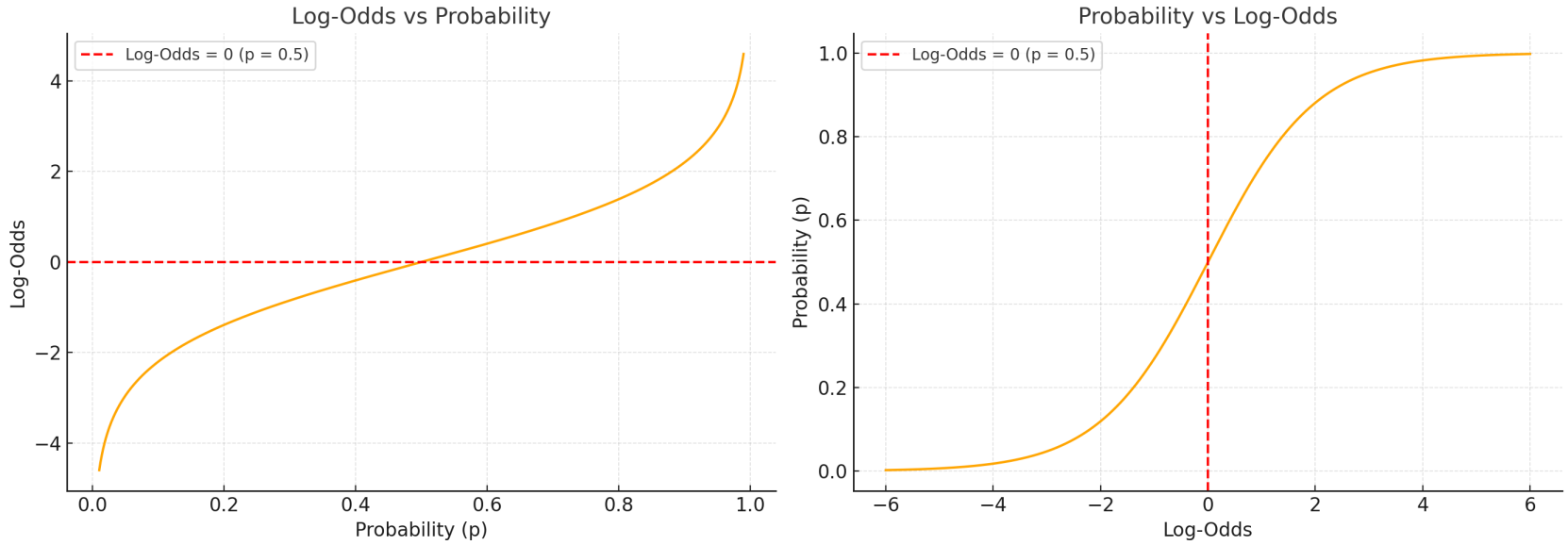
Probability (p)	**Odds**	**Log-Odds**
0.50	1	0
0.80	4	1.386
0.20	0.25	−1.386

Log-Odds



- **Log-odds increases** as probability (p) approaches 1.
- **Log-odds decreases** as p approaches 0.
- At $p = 0.5$, log-odds is 0 (equal likelihood for both outcomes).

Logistic Regression



1. **Model the Log-Odds**:
   - Fit a linear relationship between the features X and the log-odds of $y = 1$.

$$\text{Log-Odds} = \ln\left(\frac{p}{1-p}\right) = b_0 + b_1X_1 + b_2X_2 + \cdots + b_nX_n$$

2. **Convert Log-Odds to Probabilities**:
   - Apply the sigmoid function to the log-odds to get probabilities.

$$p = \frac{1}{1 + e^{-(b_0 + b_1X_1 + \cdots + b_nX_n)}}$$

3. **Classification**:
   - Assign class labels based on a probability threshold (e.g., $p > 0.5$).

Logistic Regression

- **Purpose**: Primarily used for binary classification but can be extended to multiclass problems.
- **Output**: Predicts the probability of class membership (values between 0 and 1) using the **sigmoid function**.

1. Computes a weighted sum of input features and adds a bias term.

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

2. Applies the **sigmoid function** to convert z into a probability:

$$P(y = 1|x) = \frac{1}{1 + e^{-z}}$$

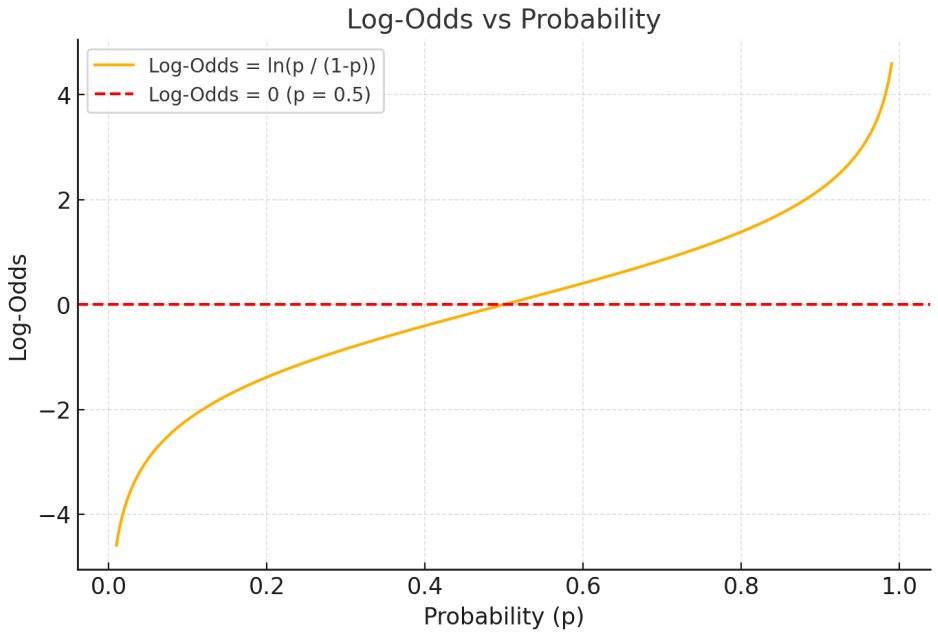
3. Classifies the output based on a threshold (e.g., $P(y = 1) > 0.5$).

Logistic Regression

1. **Log-Odds Transformation**:
   - Logistic regression models the relationship between X and the **log-odds** (logit) of the target variable y.
   - The log-odds formula:

$$\text{Log-Odds} = \ln\left(\frac{p}{1-p}\right) = b_0 + b_1X_1 + b_2X_2 + \cdots + b_nX_n$$

   - Here:
     - p: Probability of $y = 1$
     - $b_0, b_1, \ldots, b_n$: Model parameters (coefficients)
     - $X_1, X_2, \ldots, X_n$: Input features.

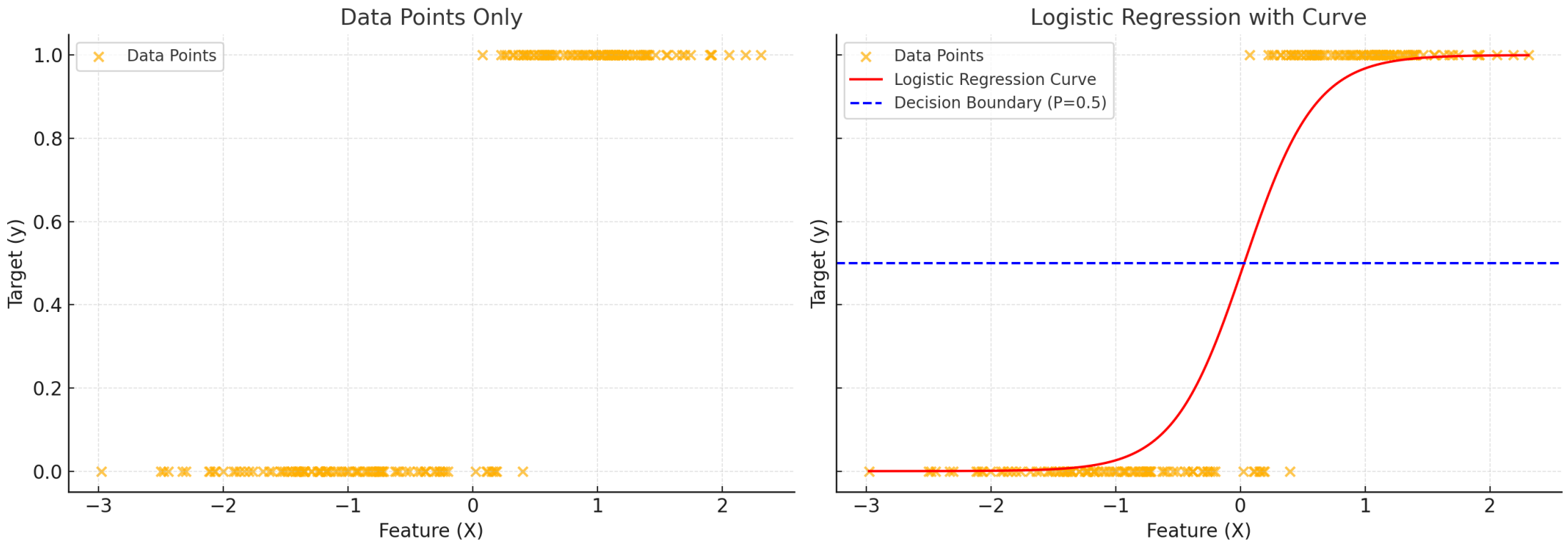


2. **Mapping to Probability**:
   - Once the log-odds are calculated, they are converted into probabilities using the **sigmoid function**:

$$p = \frac{1}{1 + e^{-(b_0 + b_1X_1 + \cdots + b_nX_n)}}$$

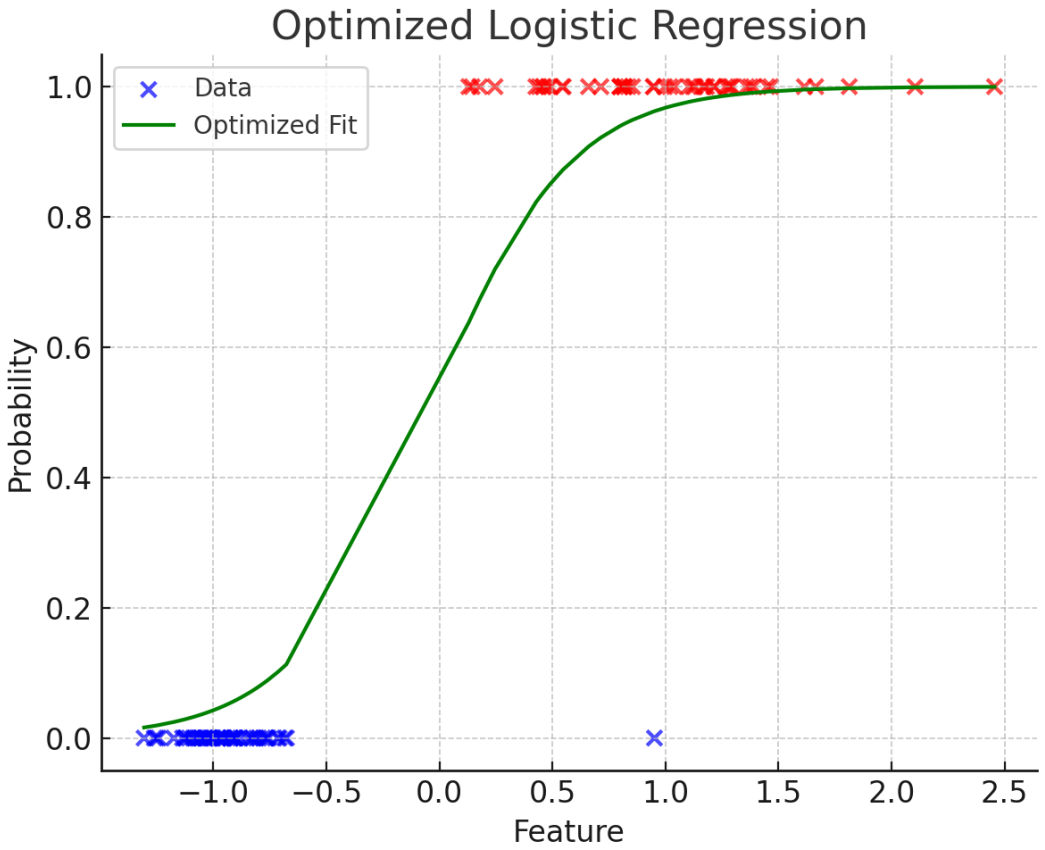
   - This maps the log-odds (which range from $-\infty$ to $+\infty$) to a probability range $[0, 1]$.

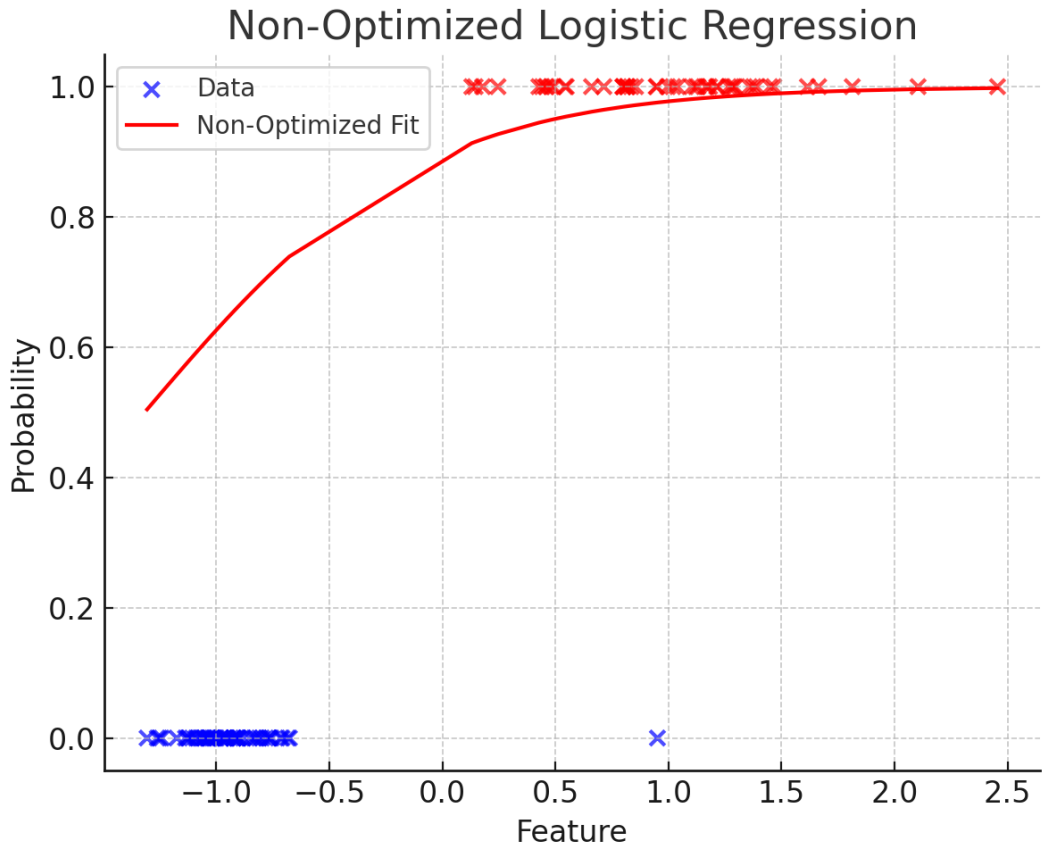
Logistic Regression



- **Data Points**: The scatterplot represents the raw data (feature X vs binary outcome y).
- **Logistic Regression Curve**: The red curve shows the predicted probability of $y = 1$ as a function of X, using the sigmoid function.
- **Decision Boundary**: The blue dashed line at $P = 0.5$ separates the two classes. If $P > 0.5$, the prediction is $y = 1$, and if $P \leq 0.5$, the prediction is $y = 0$.

Logistic Regression





Maximum Likelihood

- **Maximum Likelihood Estimation (MLE)** is a method for estimating the parameters of a statistical model. It selects the parameter values that **maximize the likelihood** of observing the given data.

  - **Likelihood Function**: Represents the probability of observing the data D given the model parameters θ.

    $$L(\theta|D) = P(D|\theta)$$

  - **Log-Likelihood**: For computational simplicity, the logarithm of the likelihood is often used:

    $$\ln L(\theta|D)$$

    Logarithms convert products into sums, making calculations easier, especially for large datasets.

Maximum Likelihood in Logistic Regression

Logistic regression models the probability of a binary outcome y (0 or 1) as:

$$P(y=1|X) = \frac{1}{1+e^{-(b_0+b_1X_1+\cdots+b_nX_n)}}$$

Objective: Find the parameters $b_0, b_1, \ldots, b_n$ that maximize the likelihood of the observed outcomes y.

Likelihood Function for Logistic Regression

For n observations:

$$L(\mathbf{b}) = \prod_{i=1}^{n} P(y_i|X_i) = \prod_{i=1}^{n} \left(p_i^{y_i} \cdot (1-p_i)^{1-y_i} \right)$$

Where:

- $p_i = \frac{1}{1+e^{-(b_0+b_1X_{i1}+\cdots+b_nX_{in})}}$
- y_i: Observed outcome (0 or 1)
- If $y_i = 1$: $P(y_i = 1|X_i) = p_i$.
- If $y_i = 0$: $P(y_i = 0|X_i) = 1 - p_i$.

Log-Likelihood Function

Taking the logarithm:

$$\ln L(\mathbf{b}) = \sum_{i=1}^{n} \left[y_i \ln(p_i) + (1-y_i) \ln(1-p_i) \right]$$

This function is maximized to estimate the model parameters $\mathbf{b}$.

- The term $p_i^{y_i} \cdot (1-p_i)^{1-y_i}$ elegantly represents the probabilities for both $y_i = 1$ and $y_i = 0$:
  - When $y_i = 1$:

$$P(y_i|X_i) = p_i^1 \cdot (1-p_i)^0 = p_i$$

  - When $y_i = 0$:

$$P(y_i|X_i) = p_i^0 \cdot (1-p_i)^1 = 1 - p_i$$

Maximum Likelihood in Logistic Regression

- The objective in logistic regression is to find the parameters $\mathbf{b}$ that maximize this likelihood function. This is done by taking the **logarithm** (to simplify computation) and maximizing the **log-likelihood**:

$$\ln L(\mathbf{b}) = \sum_{i=1}^{n} \left(y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i) \right)$$

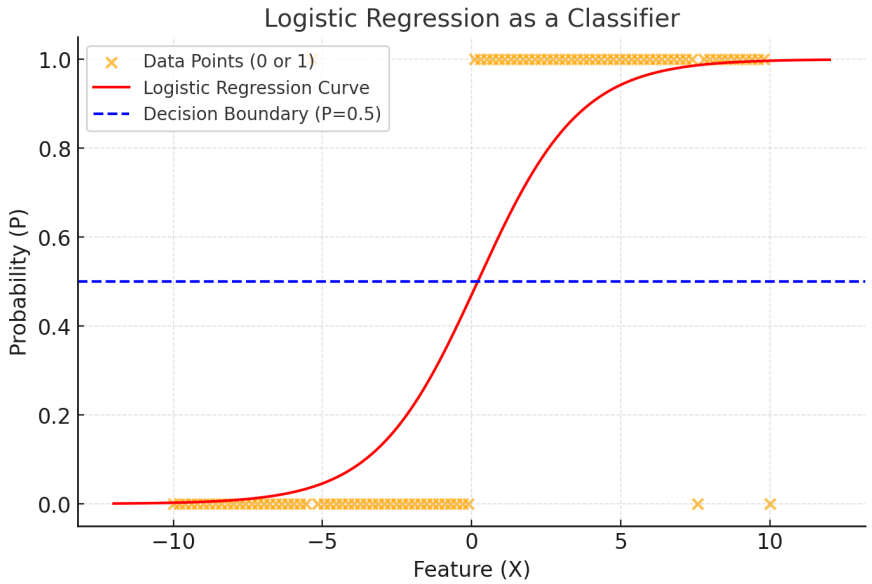
By maximizing this log-likelihood, the model learns the parameters $\mathbf{b}$ that make the observed data most probable under the logistic regression model.

Steps in Logistic Regression Using MLE

1. **Define the Likelihood Function**:
   - Based on the logistic model and observed data.
2. **Maximize the Log-Likelihood**:
   - Use optimization algorithms (e.g., gradient descent or Newton-Raphson) to find the parameter values $\mathbf{b}$ that maximize the log-likelihood.
3. **Fit the Model**:
   - With the optimized parameters, the logistic regression model predicts probabilities for new data.

Why is it called "Logistic Regression"?

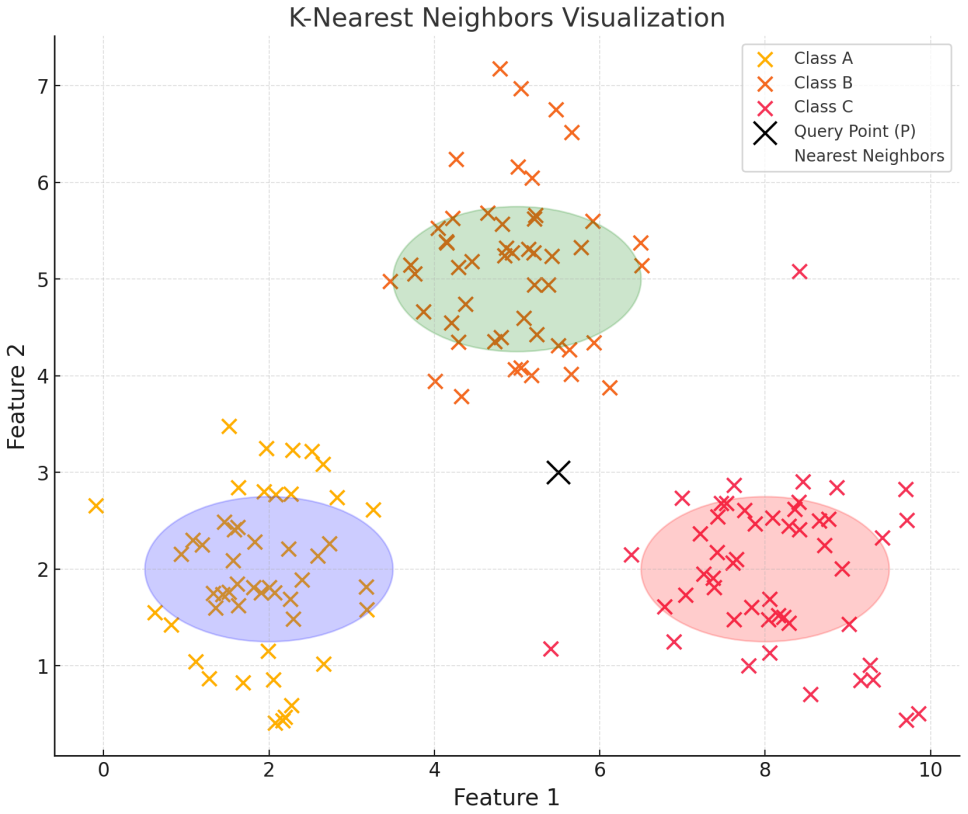
- It is called "regression" because it models a relationship between the input features and the target variable using a linear equation.
- However, instead of predicting a continuous output like traditional regression, logistic regression applies the **sigmoid function** to the linear equation to output probabilities.
  - These probabilities are then converted into class labels (e.g., P>0.5→Class 1,P≤0.5→Class 0



$$y = \begin{cases} 1, & \text{if } P(y=1|X) > 0.5 \\ 0, & \text{if } P(y=1|X) \leq 0.5 \end{cases}$$

K-Nearest Neighbors (KNN)

- Predicts the class of a data point based on the majority class of its nearest neighbors in the feature space.
- Example:
  - Suppose K=3:
    - If the 3 nearest neighbors have classes [1,0,1], the predicted class is **1** (majority vote).
    - If the 3 nearest neighbors have classes [0,0,1], the predicted class is **0**.



Given a dataset D with n labeled data points and a query point q:

1. **Input**:
   - $D = \{(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n)\}$: Dataset with feature vectors X_i and corresponding class labels y_i.
   - q: Query (test) point.
   - K: Number of nearest neighbors to consider (user-defined).

K-Nearest Neighbors (KNN)

Given a dataset D with n labeled data points and a query point q:

1. **Input**:
   - $D = \{(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n)\}$: Dataset with feature vectors X_i and corresponding class labels y_i.
   - q: Query (test) point.
   - K: Number of nearest neighbors to consider (user-defined).

Input: Dataset D, query point q, number of neighbors K
Output: Predicted class for q

1. Compute distances between q and all points in D
2. Sort points in D by ascending distance
3. Select the K nearest neighbors
4. Perform majority voting on the class labels of the K neighbors
5. Return the predicted class

K-Nearest Neighbors (KNN)

Example

- **Dataset**:

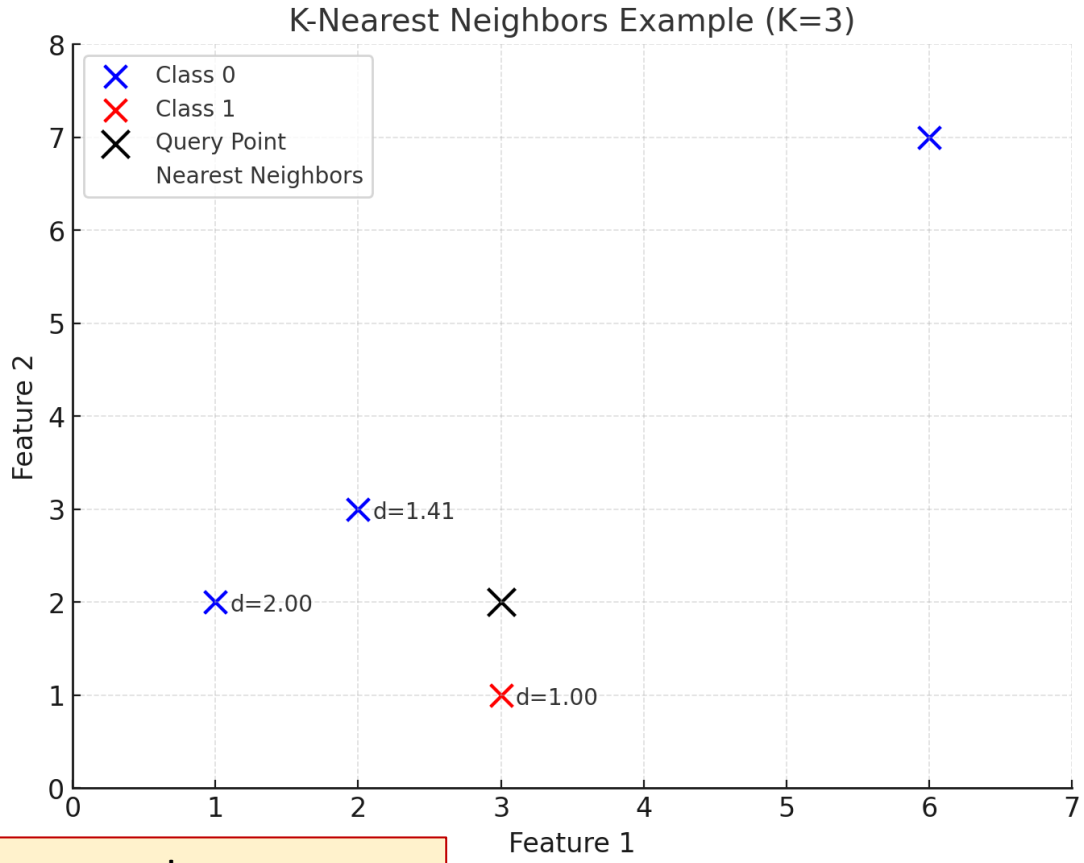
$$D = \{(X_1, y_1), (X_2, y_2), (X_3, y_3), (X_4, y_4)\}, \quad y \in \{0, 1\}$$

  Where $X_1 = [1, 2], X_2 = [2, 3], X_3 = [3, 1], X_4 = [6, 7]$.

- **Query Point**: $q = [3, 2]$, $K = 3$.

- **Steps**:

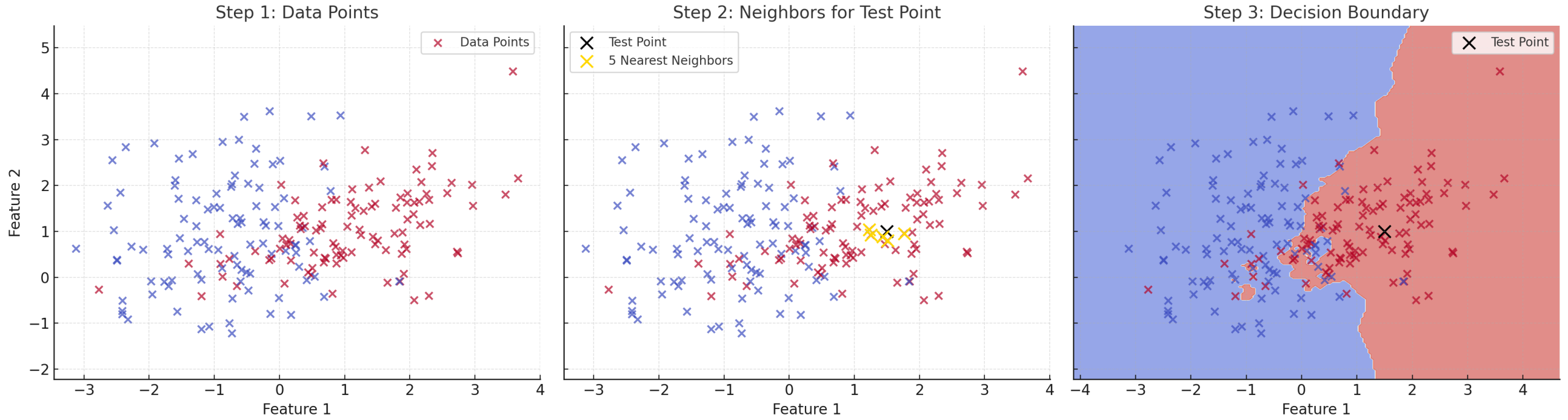
  1. Calculate distances:
     $$d(q, X_1) = 2.24,\ d(q, X_2) = 1.41,\ d(q, X_3) = 1.00,\ d(q, X_4) = 6.40$$
  2. Sort and select nearest neighbors: X_3, X_2, X_1 (labels: $1, 0, 0$).
  3. Majority vote: Predicted class = 0.



- **KNN assumes labeled data**: It relies on labeled neighbors to make predictions.
- If no data is labeled, KNN cannot classify directly, and alternative methods like clustering or semi-supervised learning are needed.

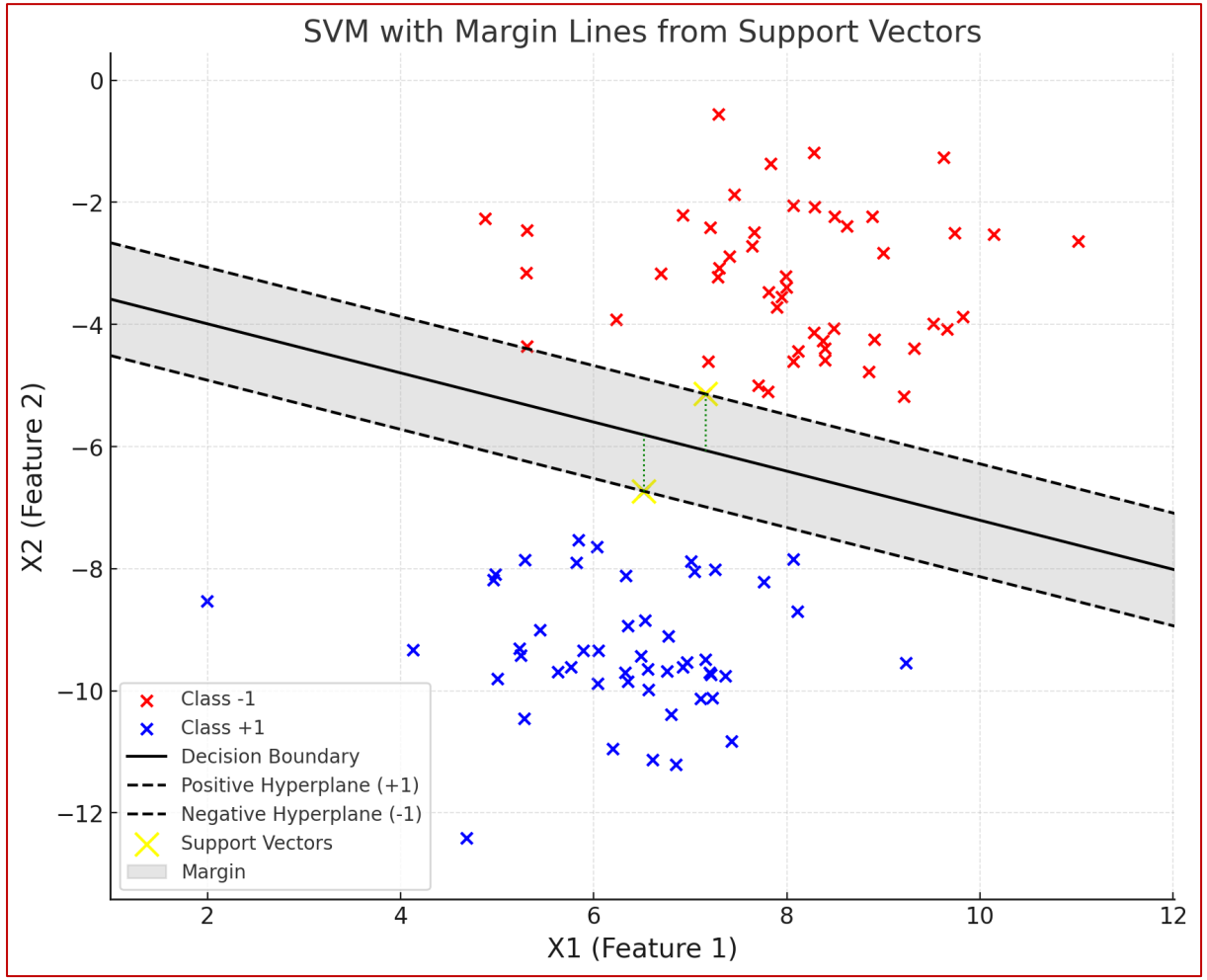
K-Nearest Neighbors (KNN)

- Predicts the class of a data point based on the majority class of its nearest neighbors in the feature space.



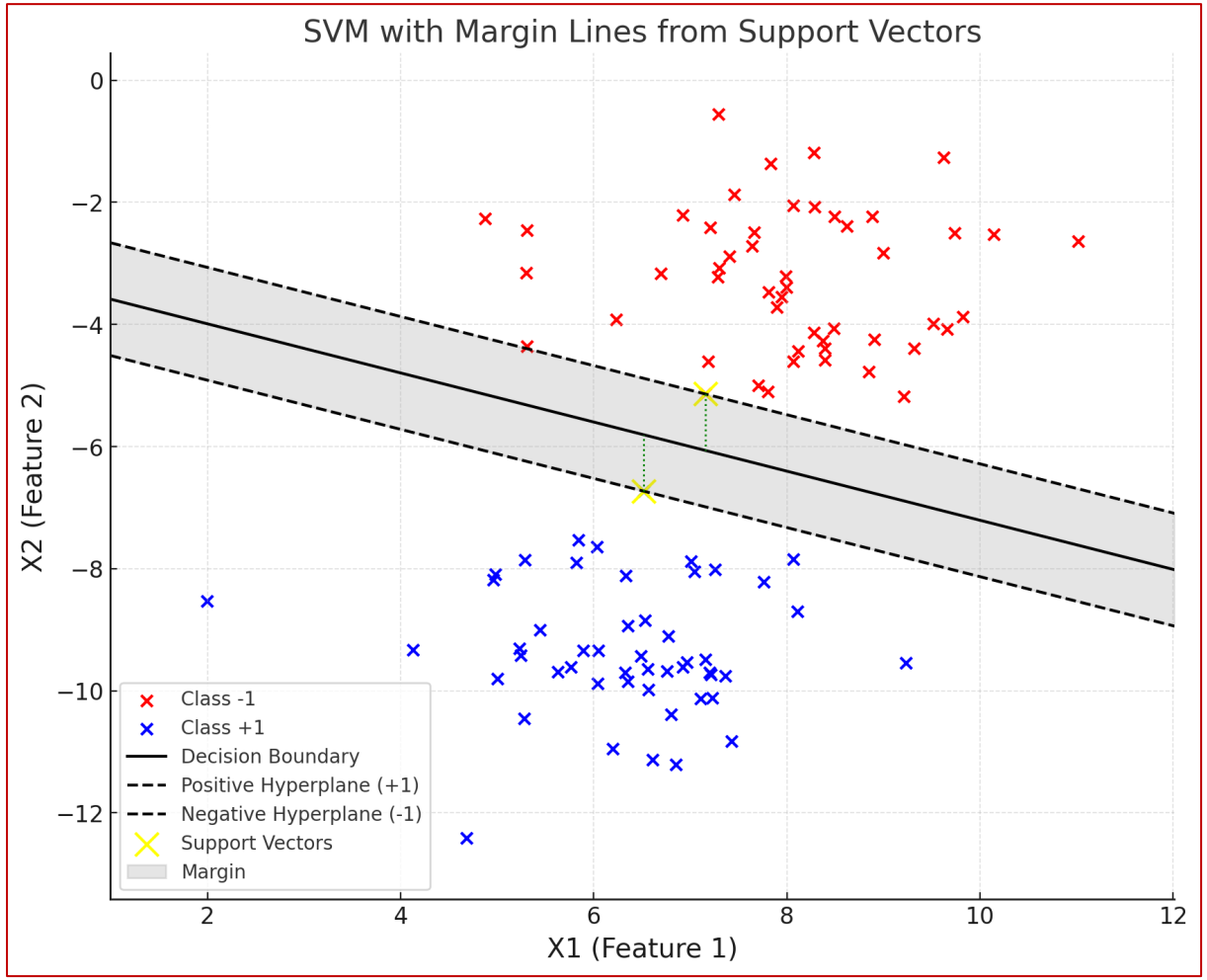
SVM for Binary Classification

- **Decision Boundary (Hyperplane)**:
  - A hyperplane is a line (in 2D), plane (in 3D), or higher-dimensional surface that separates the data into two classes.
  - SVM finds the **optimal hyperplane** that maximizes the margin.



SVM for Binary Classification

- **Decision Boundary (Hyperplane)**:
  - A hyperplane is a line (in 2D), plane (in 3D), or higher-dimensional surface that separates the data into two classes.
  - SVM finds the **optimal hyperplane** that maximizes the margin.
- **Margin**:
  - The margin is the distance between the hyperplane and the nearest data points from either class.
  - **Support Vectors** are the data points that lie closest to the hyperplane and define the margin.
  - SVM maximizes this margin.



SVM for Binary Classification

1. **Objective**:
   - Find a hyperplane $w \cdot x + b = 0$ that separates the classes such that the margin is maximized.
2. **Optimization Problem**:
   - Minimize:

$$\frac{1}{2}\|w\|^2$$

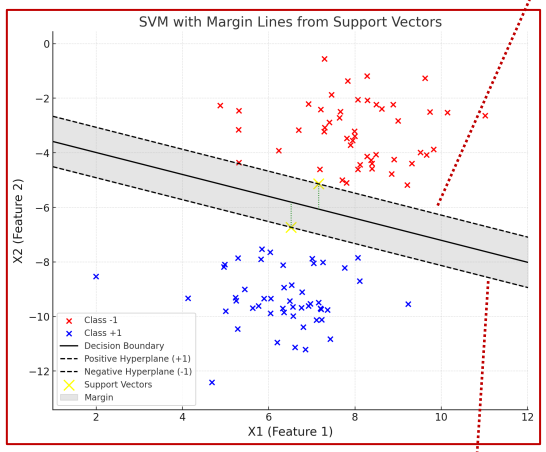
Subject to:

$$y_i(w \cdot x_i + b) \geq 1 \quad \forall i$$

Where:

- w: Weight vector (defines the hyperplane).
- b: Bias term (offset of the hyperplane).
- y_i: Class labels ($+1$ or -1).

$w \cdot x + b = +1$ (positive class boundary)



$w \cdot x + b = -1$ (negative class boundary)

The distance between these two parallel hyperplanes is:

$$\frac{2}{\|w\|}$$

$$\text{Distance} = \frac{|w \cdot x_i + b|}{\|w\|}$$

- $|w \cdot x_i + b|$ represents how far the point is from the hyperplane.
- $\|w\|$ normalizes the distance (scaling the direction of the weight vector).

To **maximize the margin**, we need to maximize the distance between the closest data points and the hyperplane. This is where the weight vector w comes in.

Minimizing $\|w\|$ is the key idea. The smaller the weight vector w, the **wider the margin**, leading to a more generalizable model.

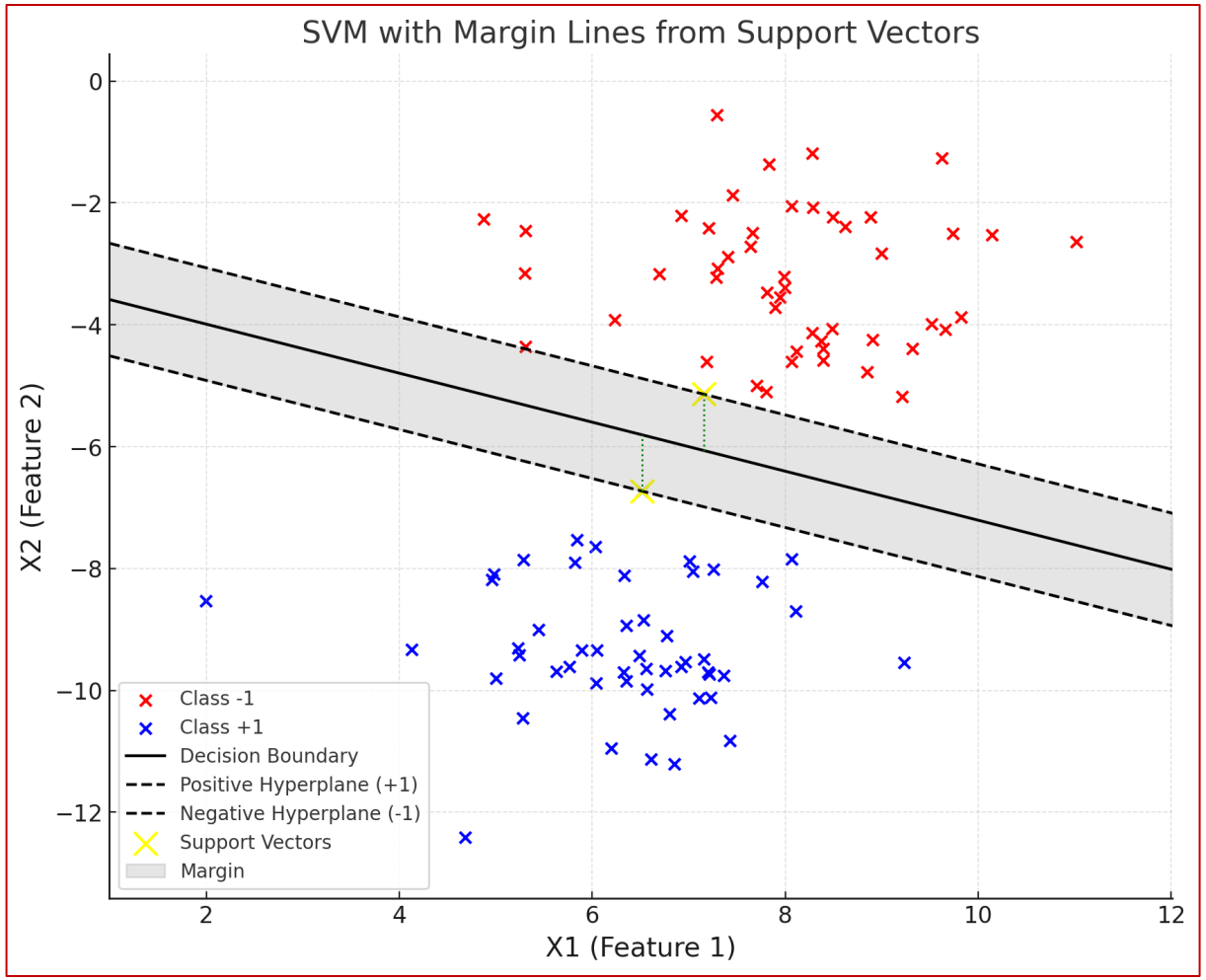
What's so Special About SVM?

Usually in classification, we put the separation line where the difference is very noticeable

What's so Special About SVM?

In hard margin SVM, even if the data points are **blended** between the classes, SVM **cannot** tolerate any misclassification.

What's so Special About SVM?



Soft Margin SVM

1. **Objective function:**

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

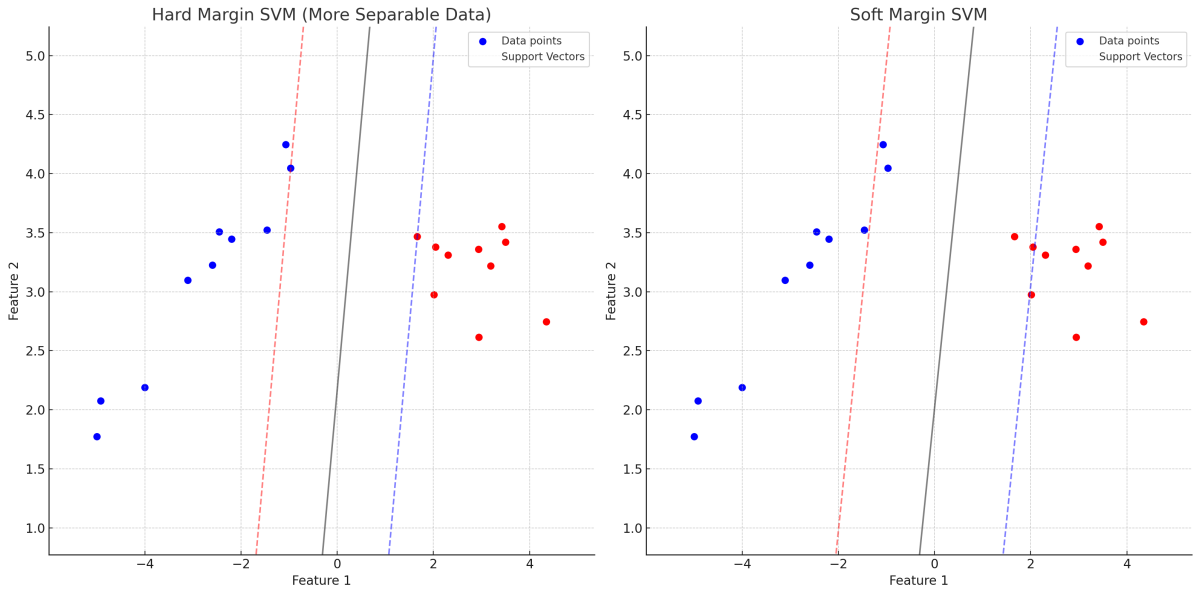
- $\frac{1}{2}\|w\|^2$: Maximizes the margin.
- $C\sum_{i=1}^{n}\xi_i$: A penalty term for misclassified points, where ξ_i is the slack variable for each data point i. Larger values of C result in fewer misclassifications but a smaller margin, and smaller values of C allow a larger margin but more misclassifications.

2. **Constraints:**

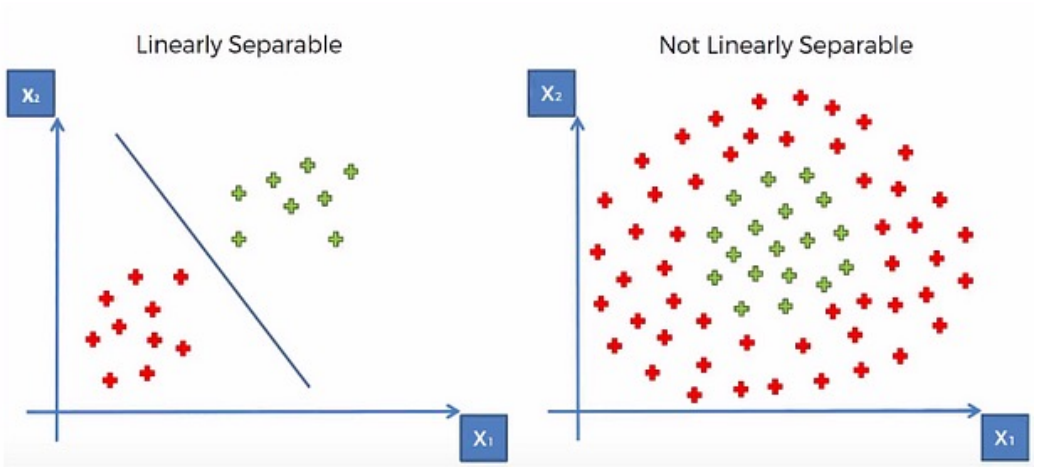
$$y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad \forall i$$

$$\xi_i \geq 0 \quad \forall i$$

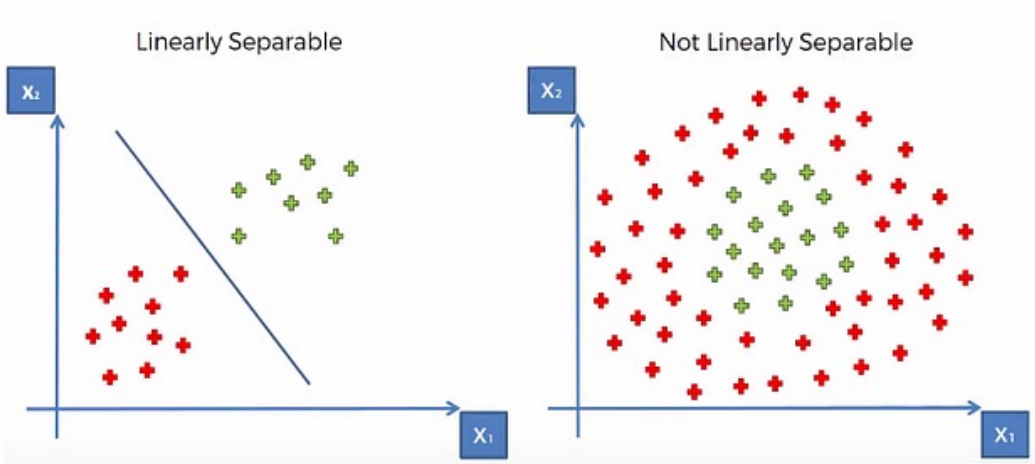
- The slack variable ξ_i allows the **i-th data point** to violate the margin constraint. If $\xi_i > 0$, it means the data point lies within the margin or is misclassified.
- The goal is to minimize the slack variables while still maximizing the margin, allowing a few violations for a more robust decision boundary.

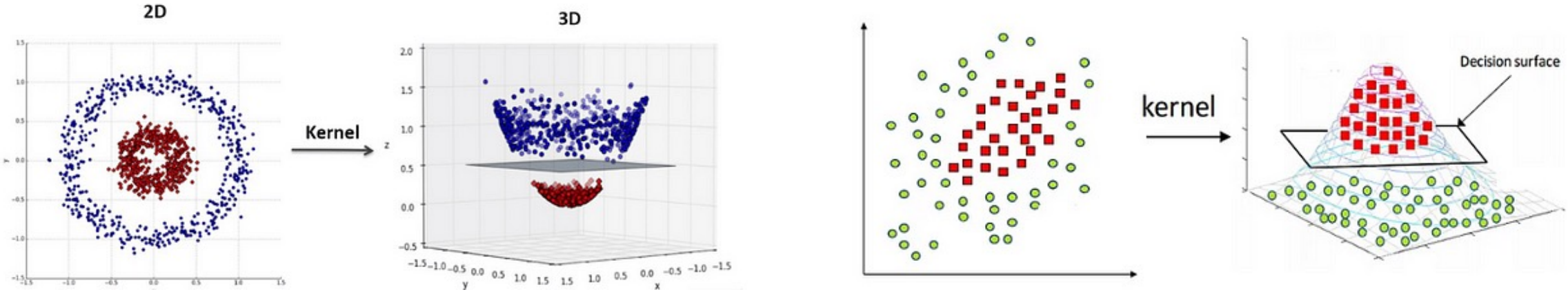


Kernel SVM



Kernel SVM

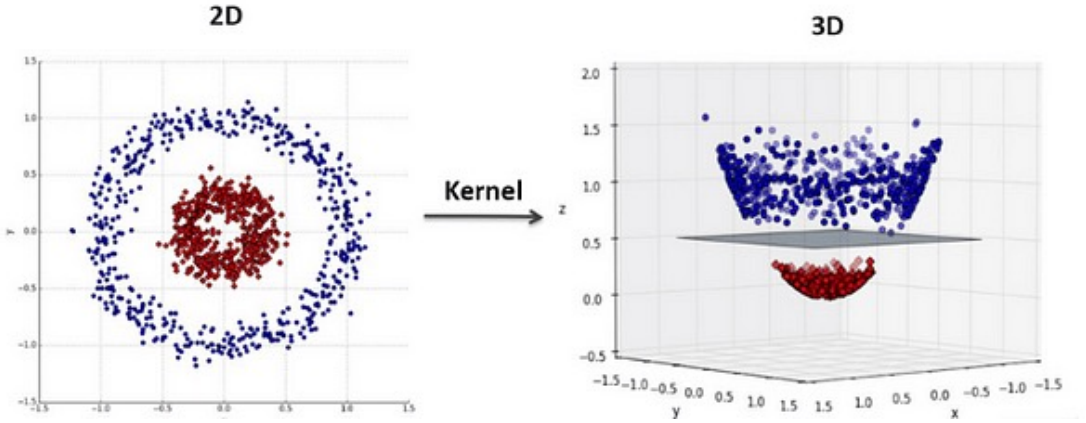




Transform the data to the higher dimension to make it linearly separable

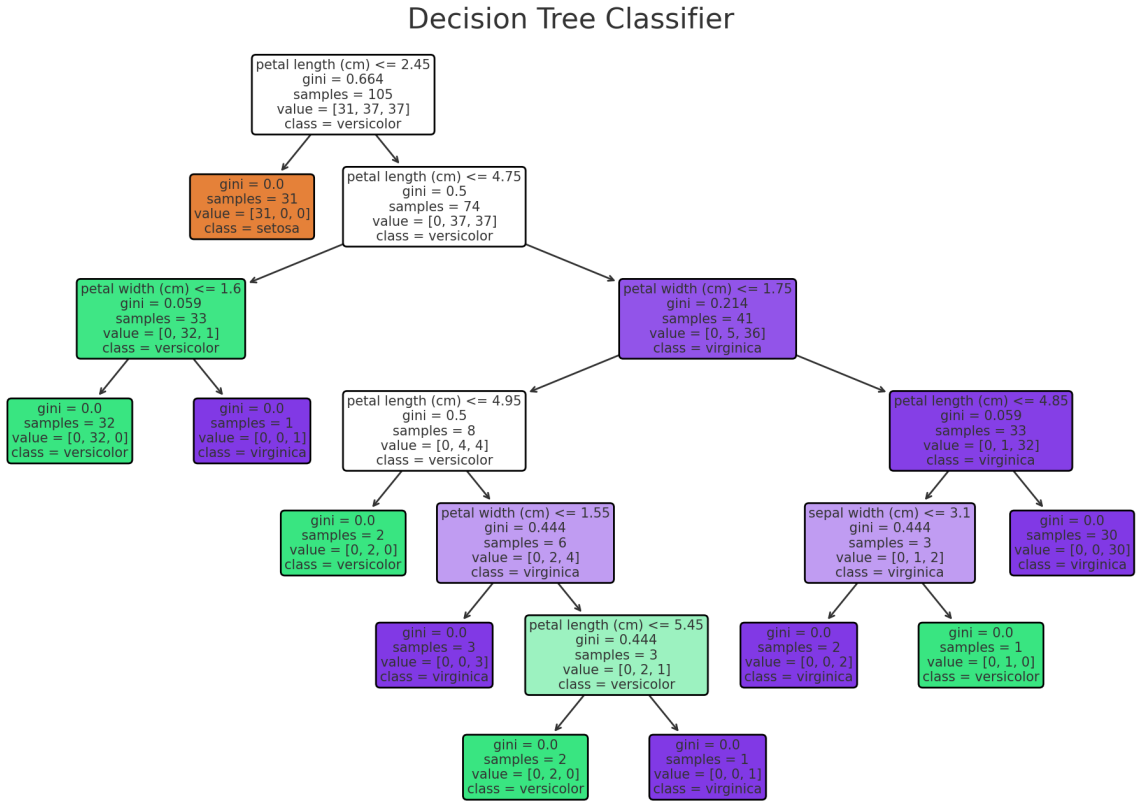
Kernel SVM

- **Kernel SVM** transforms the data into a **higher-dimensional space** where it becomes **linearly separable**.
- The kernel function helps implicitly map the data from its original space to this higher-dimensional feature space without explicitly performing the transformation.
- This allows SVM to apply a **linear decision boundary** in the transformed space, even if the data is **non-linearly separable** in its original space.

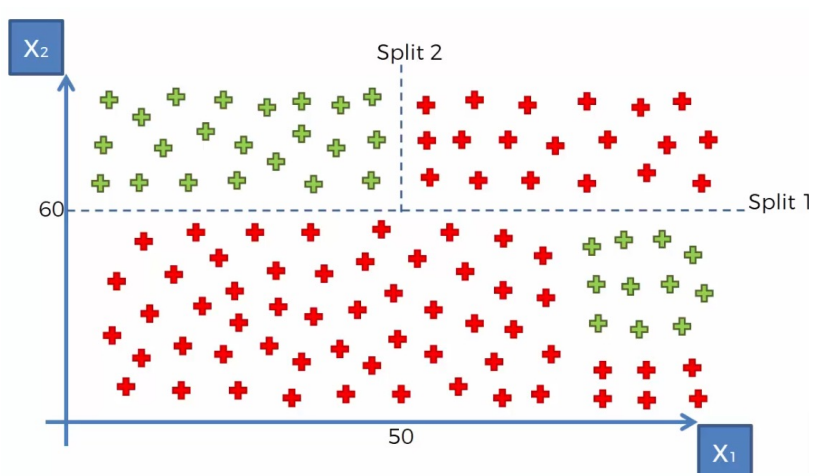


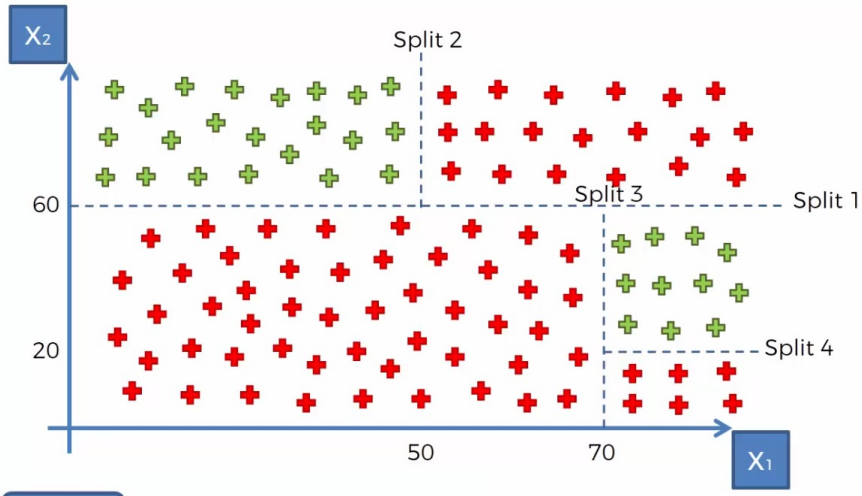
Decision Tree Classification

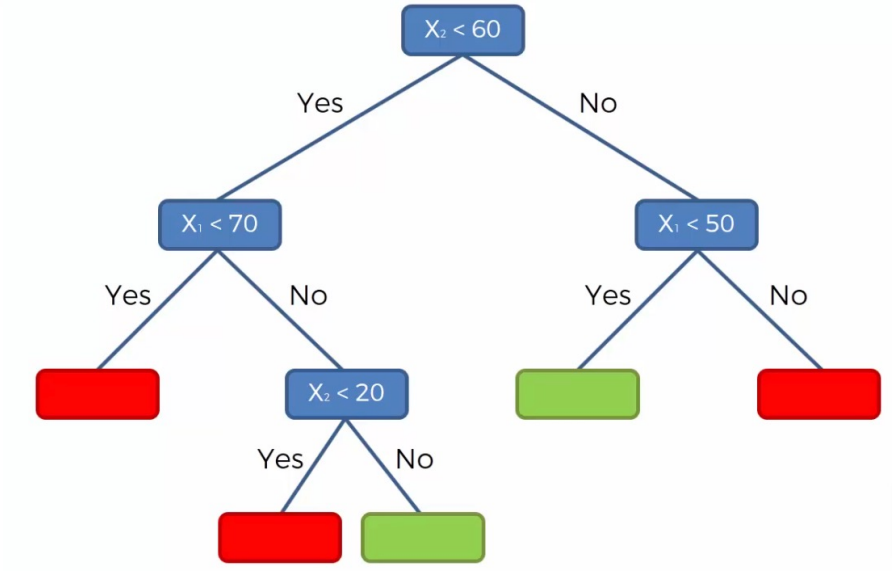
- **Goal**: Split the data into subsets based on feature values to make predictions.
- **How it works**:
  - Start at the root node (entire dataset).
  - Split the dataset based on the feature that results in the best separation (using metrics like Gini impurity or entropy).
  - Recursively split the subsets until a stopping condition is met (max depth, minimum samples, etc.).
  - Each leaf node represents a class label, and the path taken to reach it represents the decision rules.
- **Advantages**: Easy to interpret, handles both numerical and categorical data.
- **Disadvantages**: Prone to overfitting, sensitive to small changes in data.



Decision Tree Classification







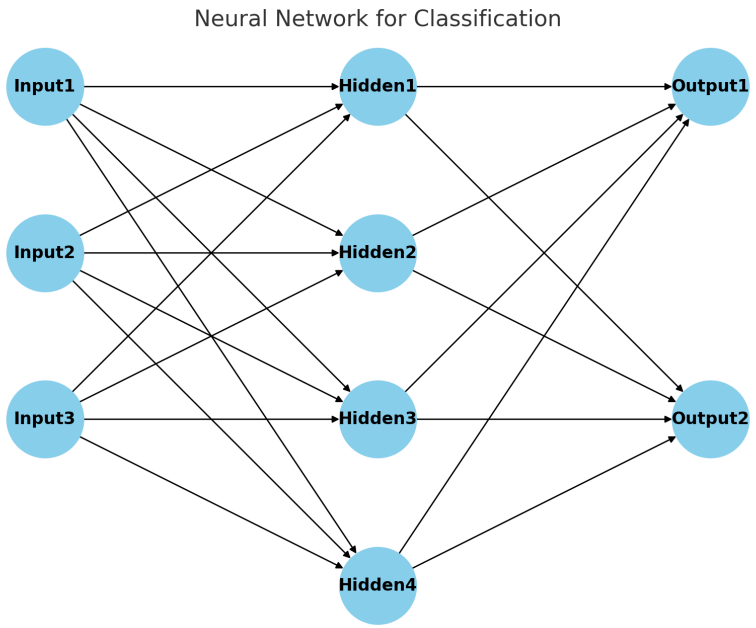
- **Advantages**: Easy to interpret, handles both numerical and categorical data.
- **Disadvantages**: Prone to overfitting, sensitive to small changes in data.

Random Forest

1. **Bootstrap Sampling**: Generate multiple subsets of the data by sampling with replacement.
2. **Train Multiple Trees**: For each subset, train a decision tree. Each tree is trained on a different data subset and a random subset of features.
3. **Voting**: Once all trees are trained, make predictions by having each tree vote. The class with the most votes is chosen as the final prediction.

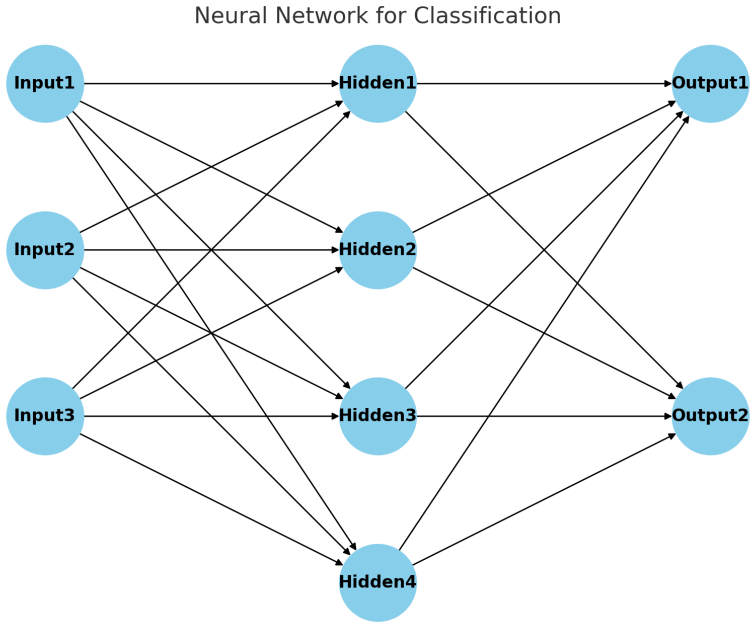
- **Advantages**:
  - Reduces overfitting (compared to a single decision tree).
  - Can handle large datasets and maintain high accuracy.
  - Handles both numerical and categorical data.
- **Disadvantages**:
  - More computationally expensive than a single decision tree.
  - Less interpretable than a single decision tree.

Neural Network for Classification



- **Input layer**: 3 nodes representing features.
- **Hidden layer**: 4 nodes where non-linear transformations take place.
- **Output layer**: 2 nodes for binary classification (e.g., class 1 and class 2).

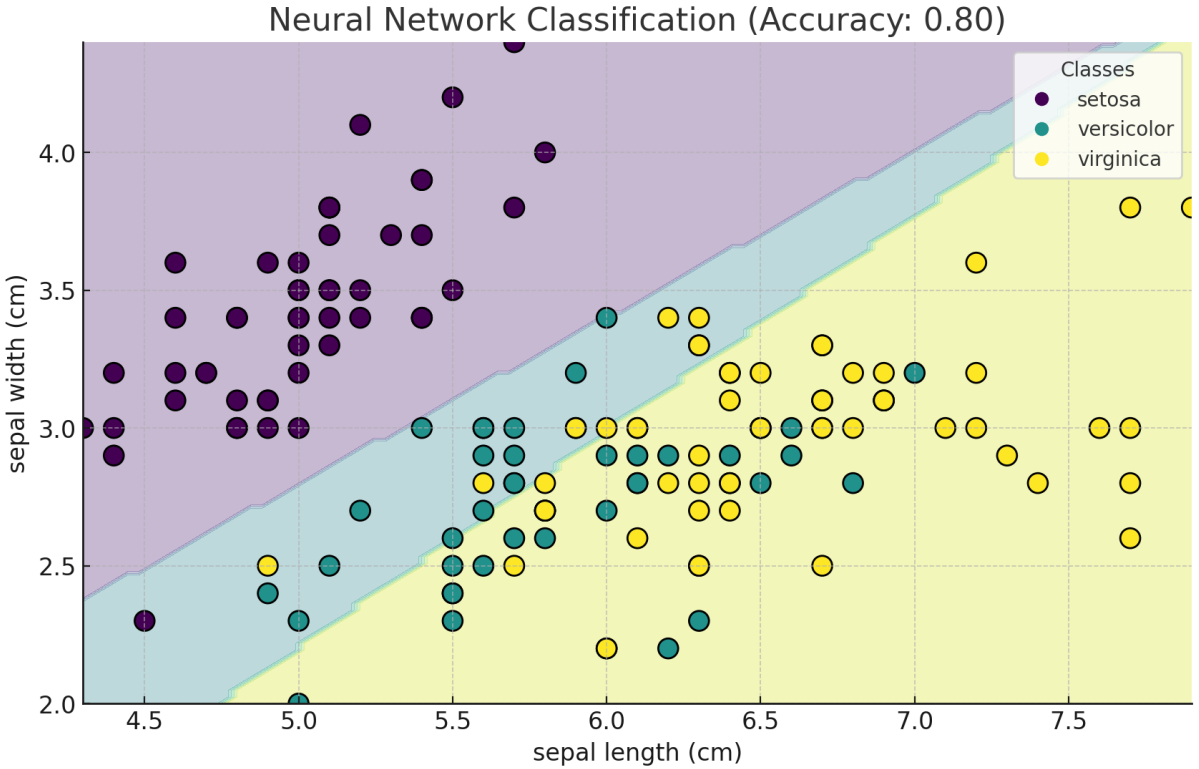
Neural Network for Classification



- **Advantages**:
  - Can model complex relationships.
  - Automatically extracts features.
  - Captures non-linear patterns.
  - Scalable for large datasets.

- **Disadvantages**:
  - Computationally expensive.
  - Long training time.
  - Prone to overfitting.
  - Difficult to interpret.
  - Requires large datasets.

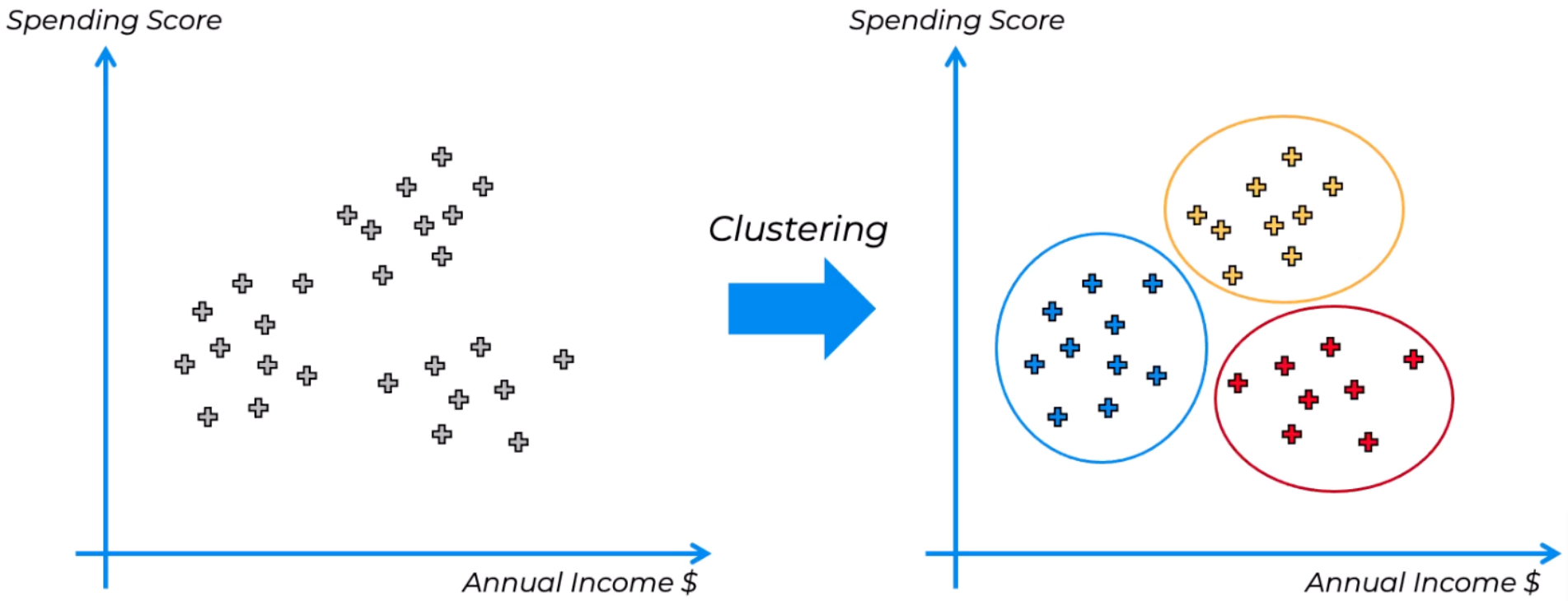
Neural Network for Classification



Neural network classification of the Iris dataset, which includes three species of Iris flowers (Setosa, Versicolor, and Virginica) based on their petal and sepal measurements. The plot shows the decision boundary and class predictions.

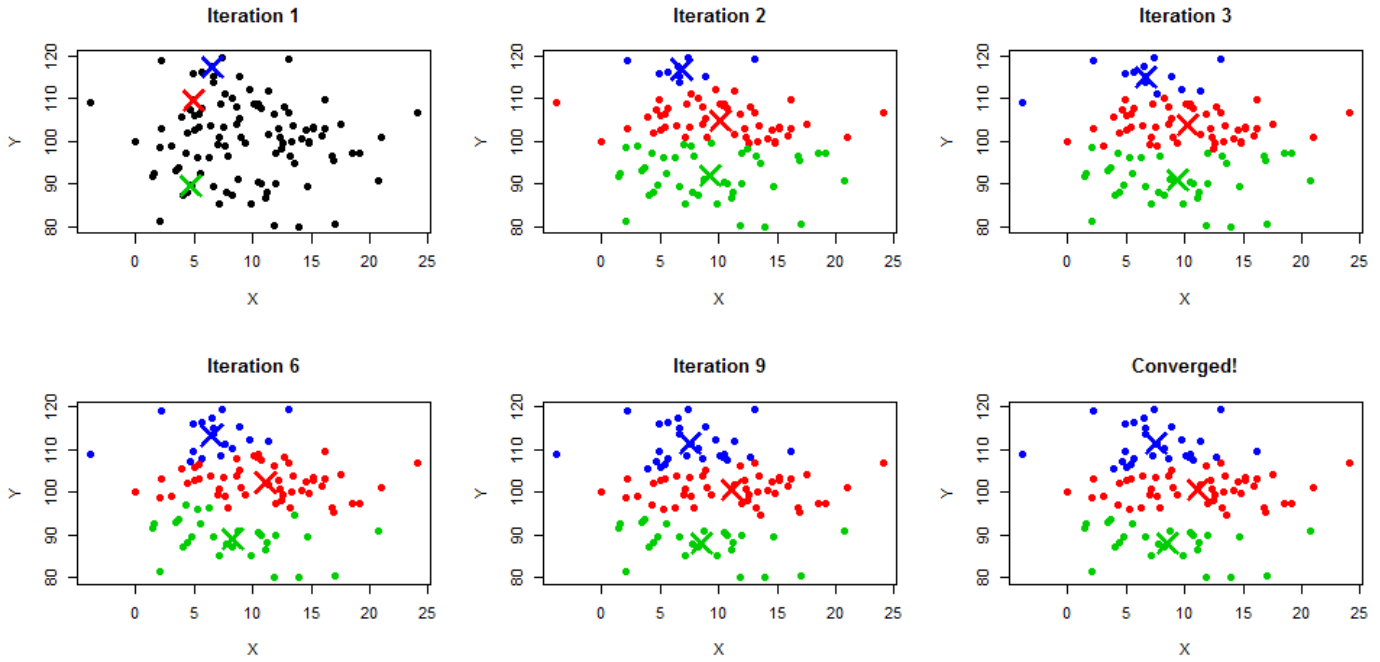
Clustering

Clustering


Spending Score
Annual Income $
Clustering
Spending Score
Annual Income $

K-Means Algorithm

1. **Initialization**: Randomly select k centroids (cluster centers).
2. **Assignment**: Assign each data point to the nearest centroid.
3. **Update**: Recalculate centroids as the mean of points in each cluster.
4. **Repeat**: Iterate steps 2 and 3 until centroids do not change significantly (convergence).



What should be k? The Elbow Method

The elbow method helps find the optimal k by evaluating the **Within-Cluster Sum of Squares (WCSS)**, which measures the total variance within clusters.

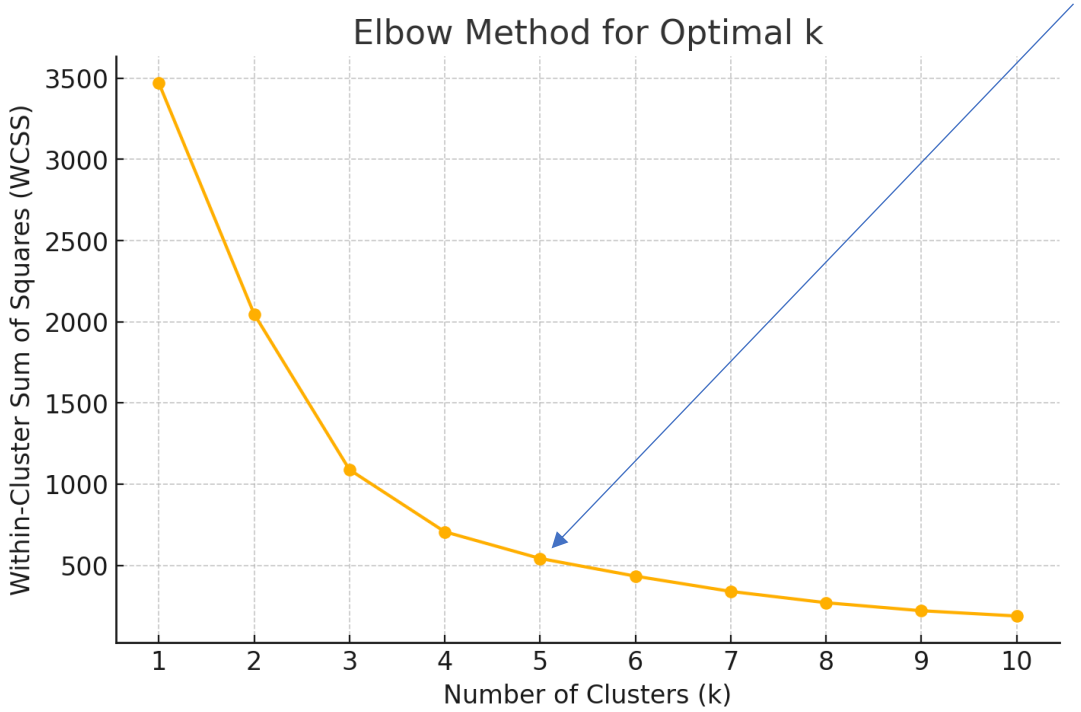
Steps:

1. **Run K-Means for different k values** (e.g., 1 to 10).
2. **Calculate WCSS** for each k.
3. Plot k vs. WCSS.
4. Identify the "elbow point," where the rate of decrease sharply changes (like an elbow).

WCSS Formula:

$$WCSS = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

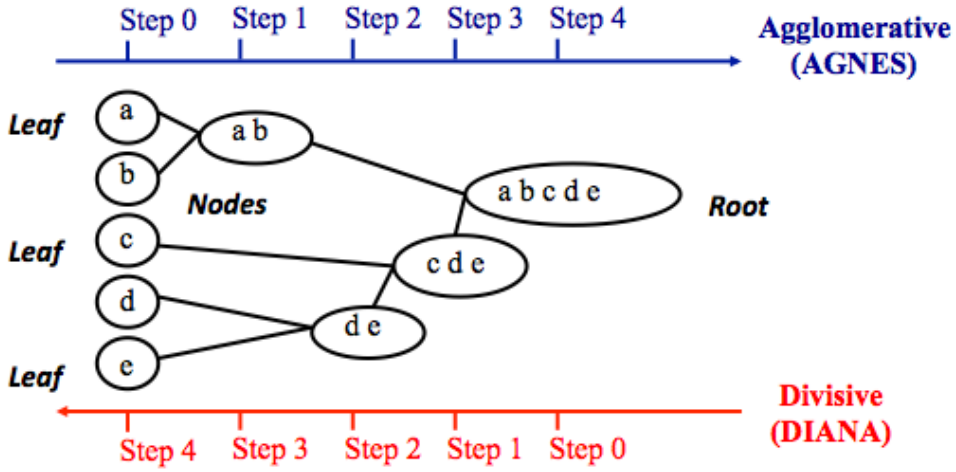
- C_i: Cluster i
- μ_i: Centroid of cluster i



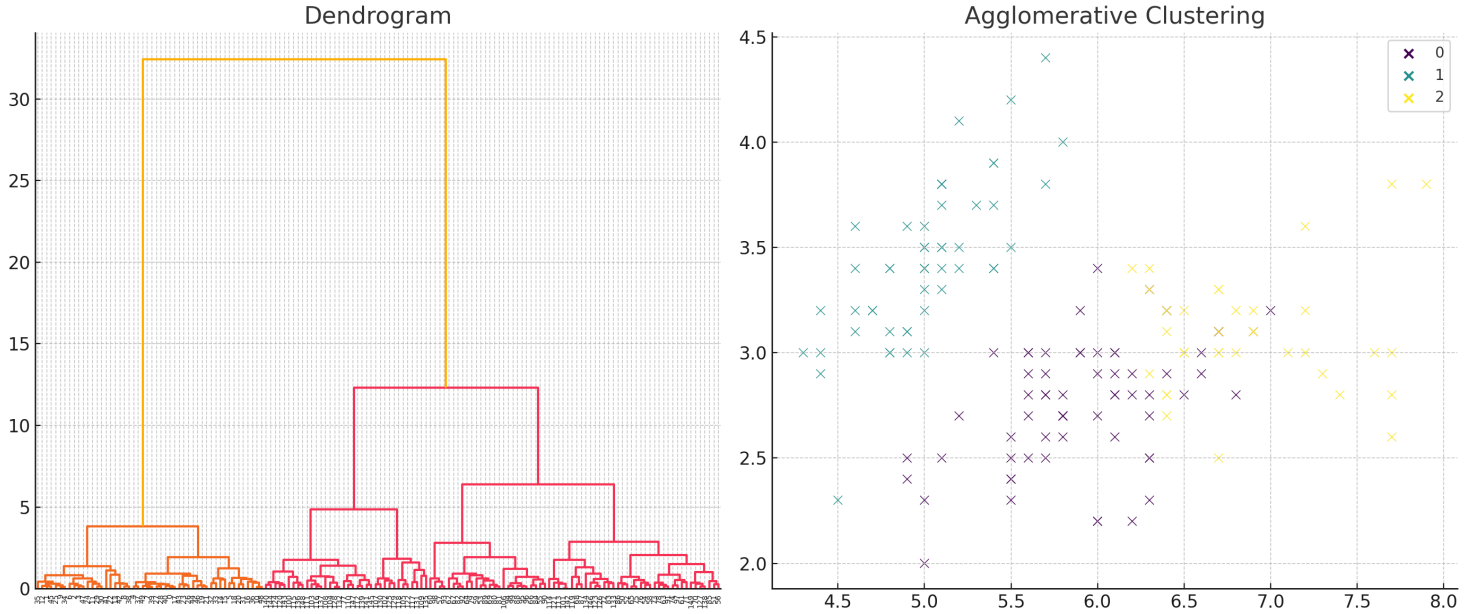
The "elbow point" is where the WCSS curve starts to flatten, suggesting the optimal number of clusters. In this case, you can visually identify it from the plot.

Hierarchical Clustering

1. **Initialize**: Treat each data point as its own cluster.
2. **Compute distances**: Calculate the distance between all pairs of clusters.
3. **Merge clusters**: Merge the two closest clusters.
4. **Update distances**: Recalculate distances between the new cluster and remaining clusters.
5. **Repeat**: Continue until all points are in one cluster or desired clusters are formed.
6. **Result**: A hierarchical tree (dendrogram) showing the merging process.



https://uc-r.github.io/hc_clustering



Dendrogram: x axis are the points, y axis is the distance