Hamedan University of Technology

Special Topics in the Industrial Applications of Machine Learning

Dr. Ghasem Alipoor

Electrical Engineering Department

Hamedan University of Technology

E-mail: g.alipoor@gmail.com, alipoor@hut.ac.ir

Fall 2025

Chapter 4: Basic ML Algorithms

Machine Learning is a field of study that gives computers the capability to learn from data and improve performance without being explicitly programmed.

By the end of this chapter, students should be able to:

- Understand the main categories of ML algorithms.
- Learn the principles behind training and optimization of models.
- Implement and evaluate **basic and commonly used ML algorithms.**
- Compare models on industrial datasets using appropriate evaluation metrics.
- Recognize the strengths, limitations, and suitable applications of each algorithm in industrial contexts.

Chapter 4: Basic ML Algorithms

- Section 1: ML Algorithms & Training Basics
  Overview of ML types and core training concepts.
- Section 2: Regression Models
  Predicting continuous outputs from industrial process data.
- Section 3: Classification Models
  Assigning discrete labels for fault detection and decision tasks.
- Section 4: Unsupervised Learning
  Discovering hidden patterns and structure without labeled data.

Introduction to ML Algorithms

An ML algorithm is a procedure that learns patterns from data to make predictions or decisions.

- **General workflow:**

Algorithms differ in:

- The kind of data they need (labeled or unlabeled)
- How they express the relation between inputs and outputs
- How they learn and adjust their parameters

Examples of industrial applications:

- Forecasting energy demand (Regression)
- Detecting faults in machines (Classification)
- Clustering consumers by load profiles (Clustering)
- Reducing sensor data for monitoring (Dimensionality Reduction)

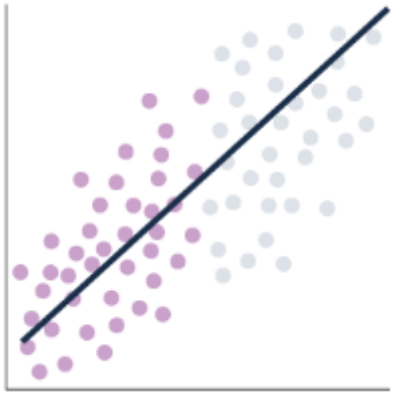
Categories of ML Algorithm

Supervised
(Train on labeled data)

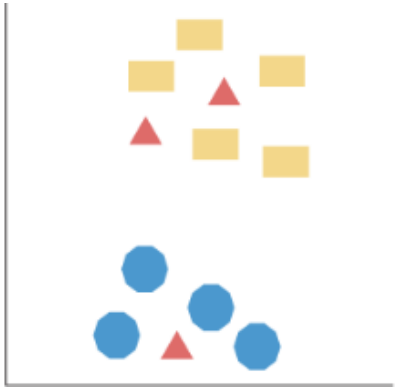
Unsupervised
(Train on unlabeled data)

Regression

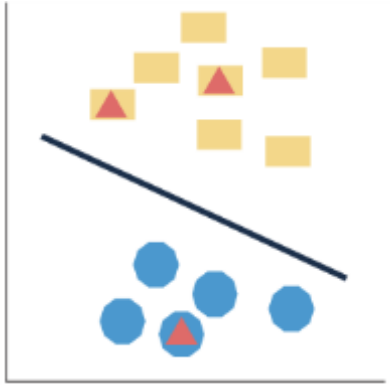
predict
continuous values



Classification



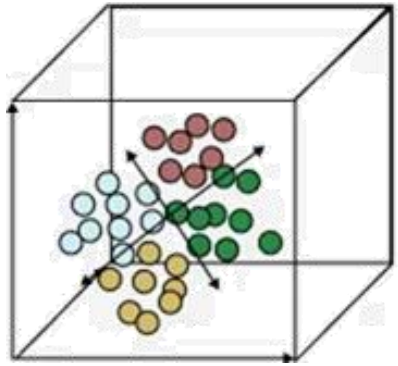
predict
categories



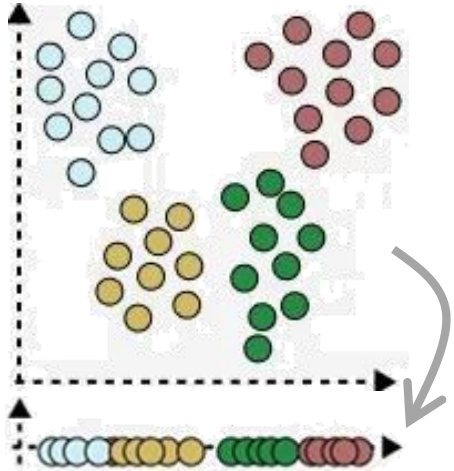
Clustering

group
similar data

Dim. Reduction



simplify data



What Does "Training" Mean?

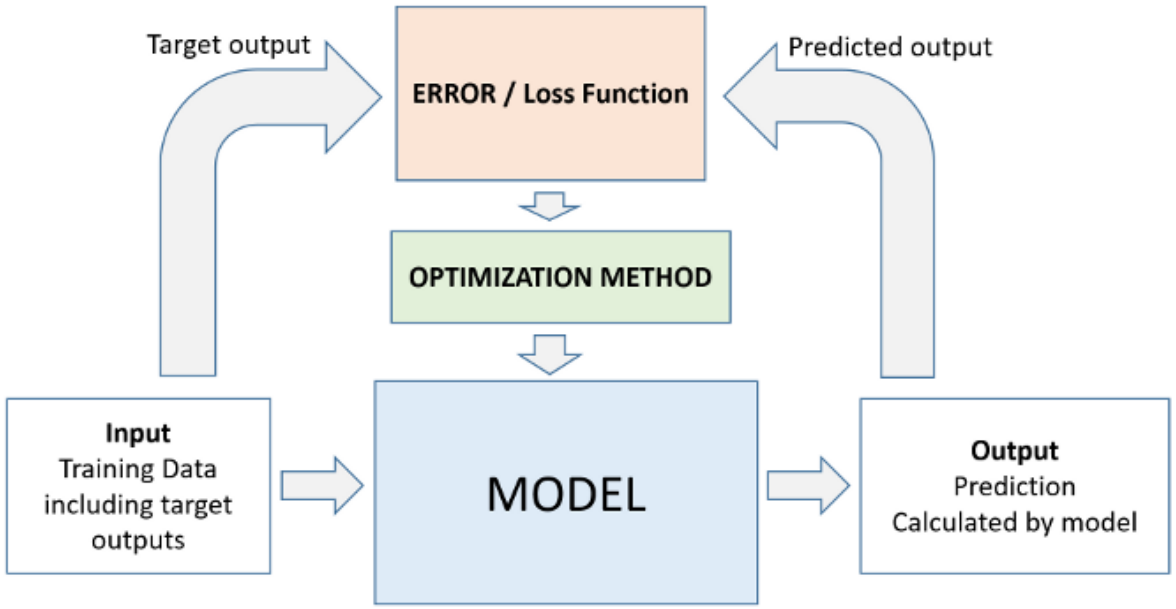
Training in machine learning means adjusting model parameters to minimize a **loss function**.

The model prediction is $\hat{y} = f_\theta(x)$, and we aim to solve:

$$\theta^* = \arg mean_\theta \ \mathcal{L}(y, \hat{y})$$

where

- θ: model parameters (e.g., weights, bias)
- $f(x)$: input features
- $\hat{y}$: model's predicted output
- $\mathcal{L}$: loss function measuring prediction error



Optimization algorithms (like Gradient Descent) update parameters step-by-step toward lower loss.

Key Terminology

Term	Meaning (Concise Definition)
Model	A mathematical function (e.g., $\hat{y} = f_\theta(x)$) that maps inputs to outputs; its parameters θ are learned from data
Feature (Input)	Measurable variable(s) describing each sample (e.g., temperature, voltage)
Target / Label (Output)	True value or category we want the model to predict
Sample / Example	One data point: a pair (x_i, y_i)
Training	Adjusting model parameters to minimize error on training data
Validation	Testing model performance on unseen data to tune hyperparameters
Prediction / Inference	Using the trained model to estimate outputs for new inputs
Learning Rate	Controls how big each parameter update step is during training
Epoch	One complete pass through the entire training dataset
Batch Size	Number of samples processed before updating the model parameters
Iteration	One parameter update step

Common Loss Functions

Category	Loss Function	Formula	Typical Use / Behavior
Regression	Mean Squared Error (MSE)	$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$	Linear regression; penalizes large errors strongly
	Mean Absolute Error (MAE)	$L = \frac{1}{N}\sum_{i=1}^{N}\lvert y_i - \hat{y}_i\rvert$	Robust to outliers; linear penalty
Classification	Binary Cross-Entropy (Log Loss)	$\sum_{i=1}^{N}[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)]$	Binary classification; compares probabilities
	Categorical Cross-Entropy	$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{c} y_{i,c} \log \hat{y}_{i,c}$	Multi-class (softmax) classification
Regularization	$L_{total} = L_{data} + \lambda L_{reg}$ L_{data}: Any loss L_{reg}: Penalty Term	Ridge (L2): $L_{reg} = \sum_i w_i^2$	Penalizes large weights; reduces variance
		Lasso (L1): $L_{reg} = \sum_i \lvert w_i\rvert$	Promotes sparsity; implicit feature selection

The Gradient Descent Algorithm

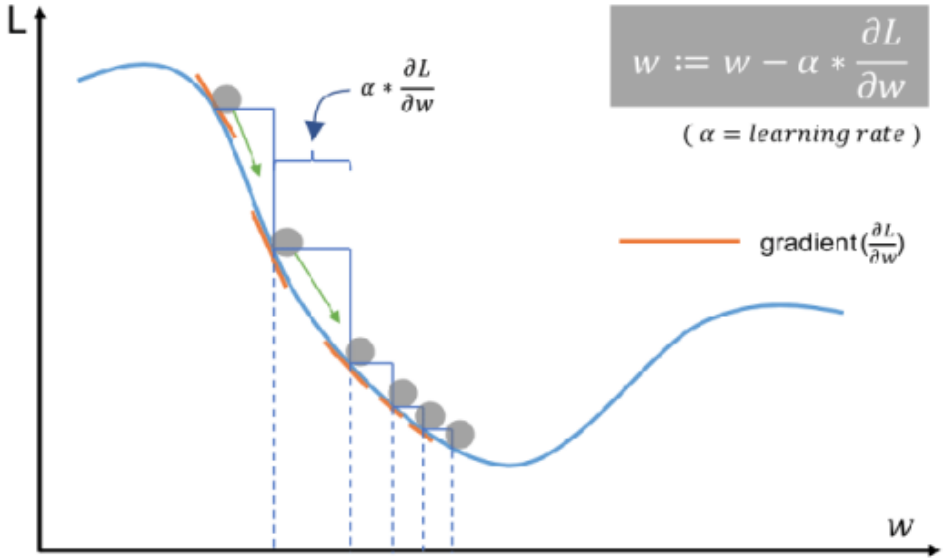
Gradient Descent (GD) is an iterative algorithm that minimizes a loss function $L(\theta)$ by updating parameters **opposite to the gradient**.

Parameter vector $\theta = [w_1, w_2, \cdots, w_n, b]$ is updated as:

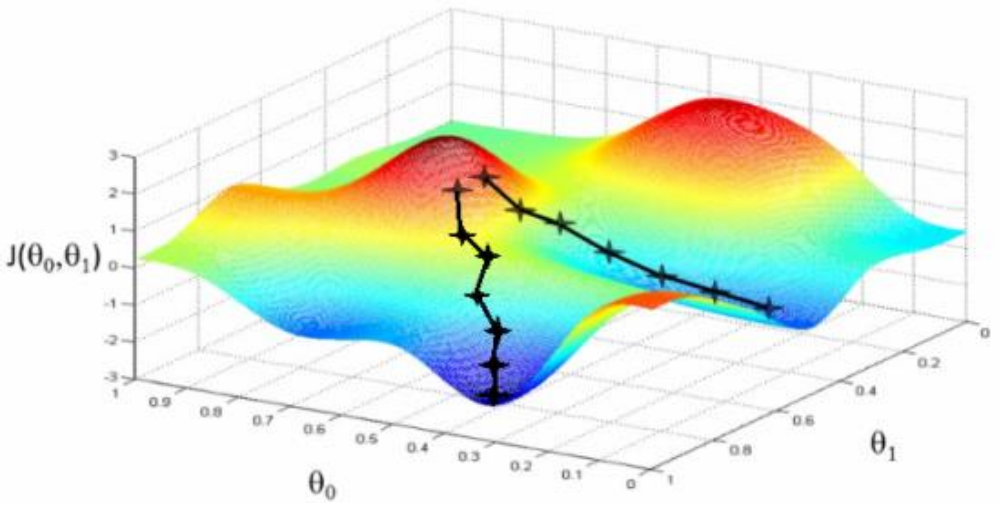
$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$$

- η: the learning rate
- $\nabla_\theta L(\theta) = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \cdots, \frac{\partial L}{\partial w_n}, \frac{\partial L}{\partial b}\right]$: gradient of the loss

1D model: *Single parameter moves downhill along the loss parabola.*

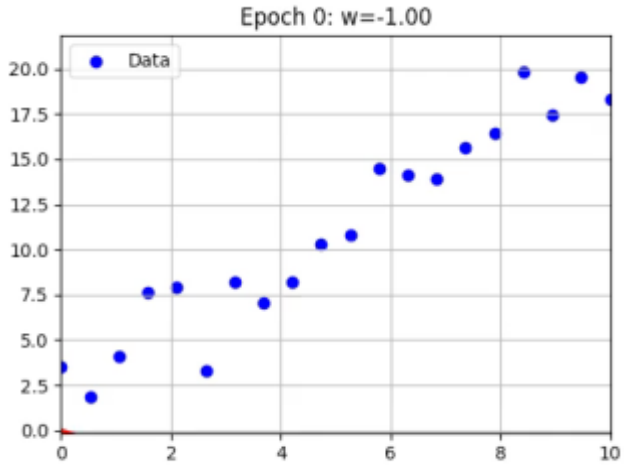


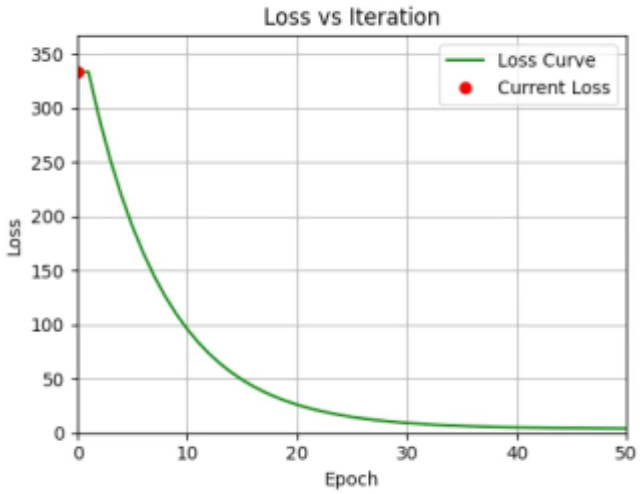
2D model: *Parameters descend the 3D loss bowl toward the minimum.*

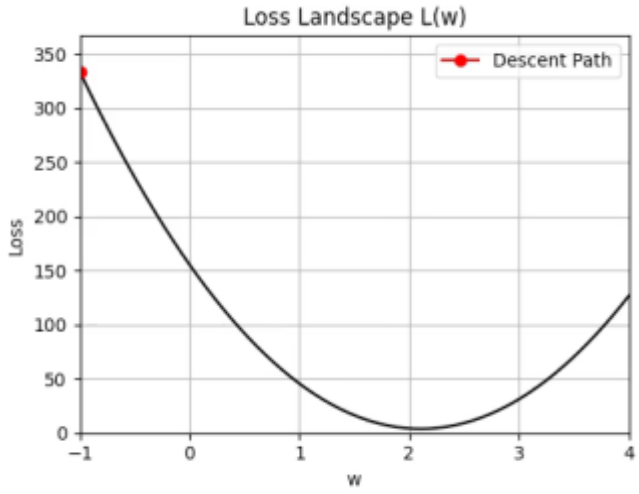


Example: GD on Linear Regression

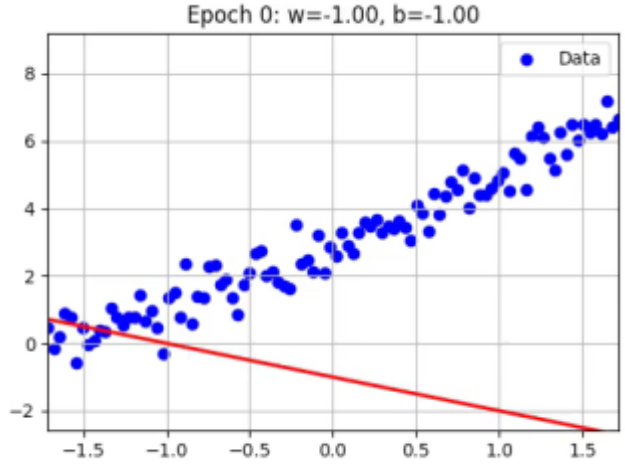
- 1D Linear Regression (Only w): $L(w) = \frac{1}{N}\sum_{i=1}^{N}(y_i - wx_i)^2$

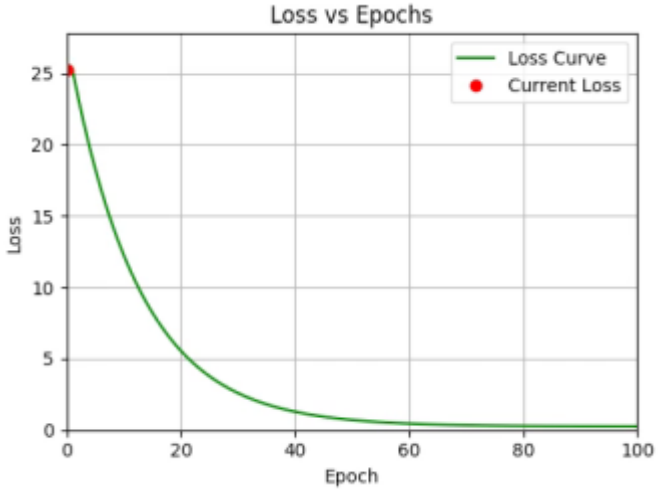


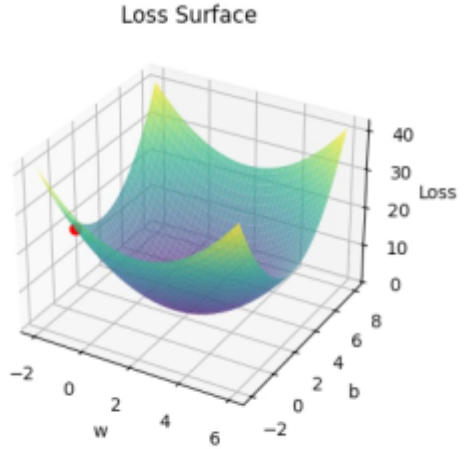




- 2D Linear Regression (w and b): $L(w) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (wx_i + b))^2$

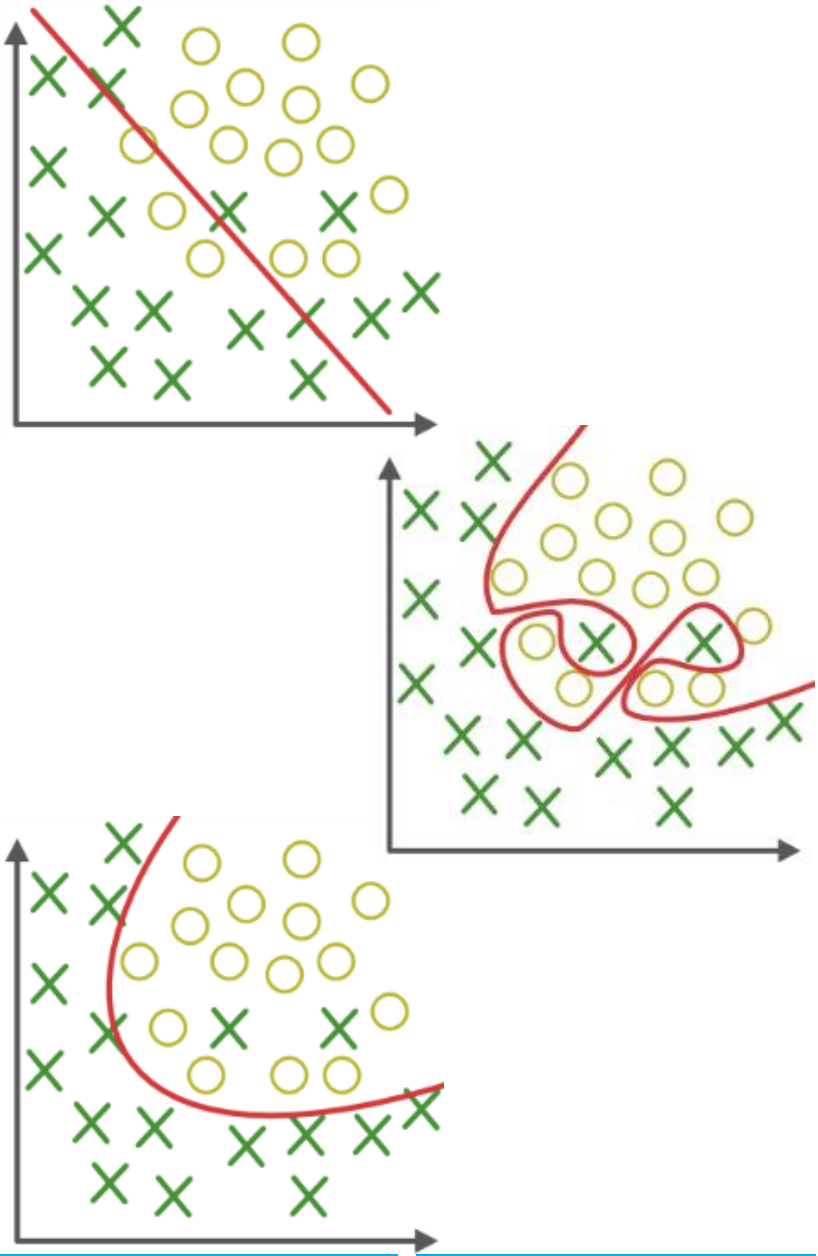






Underfitting and Overfitting

- **Underfitting**
  - Model is too simple → misses real patterns
  - Poor on both training & test data
  - **Example:** Forecasting demand using only daily averages, ignoring seasons
- **Overfitting**
  - Model is too complex → memorizes data, not patterns
  - Excellent on training but poor on new data
  - **Example:** Forecasting demand by following every random fluctuation, failing next year
- **Appropriate Fitting**
  - Model captures key patterns without memorizing noise
  - Generalizes well to new conditions (new equipment, customers, or weather)



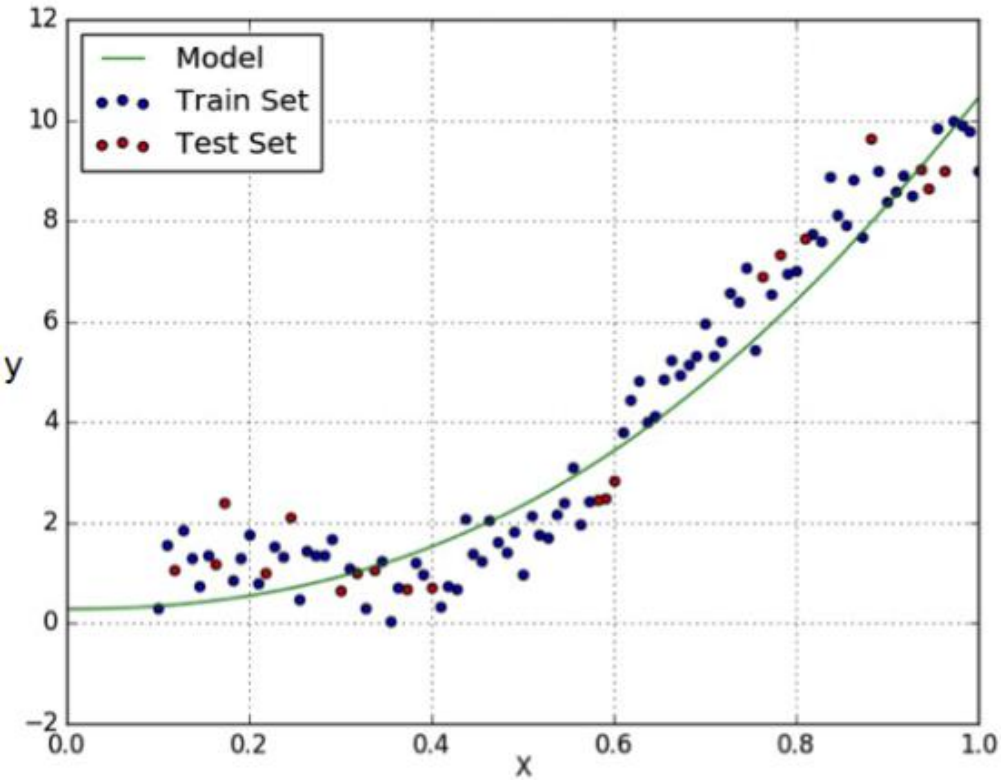
Chapter 4: Basic ML Algorithms

- Section 1: ML Algorithms & Training Basics
  Overview of ML types and core training concepts.
- Section 2: Regression Models
  Predicting continuous outputs from industrial process data.
- Section 3: Classification Models
  Assigning discrete labels for fault detection and decision tasks.
- Section 4: Unsupervised Learning
  Discovering hidden patterns and structure without labeled data.

Introduction to Regression Models

What is regression?

- Regression models estimate a **continuous output variable** y from one or more inputs $x_1, x_2, \cdots, x_n$.
- Goal: Learn a function $f(X)$ that best approximates the relationship between inputs and outputs.
- Used widely in **industrial systems** for prediction, control, and monitoring.



Examples:

- Predict **energy consumption** from temperature, humidity, and time.
- Estimate **machine wear** from vibration, current, and operation time.
- Forecast **process output quality** from sensor readings.

Main idea: Find parameters that minimize prediction error (e.g., MSE, MAE).

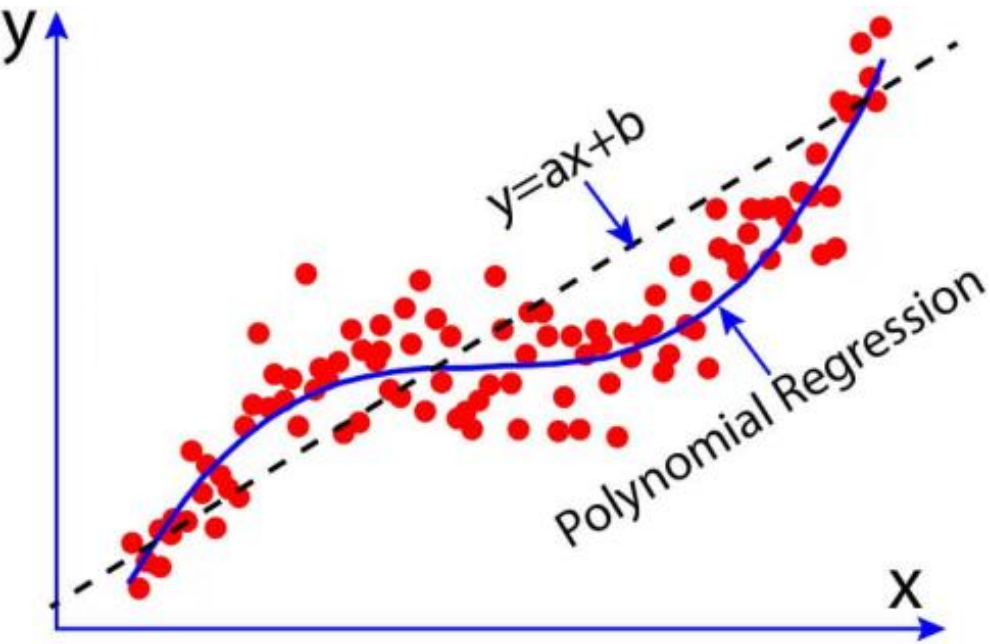
Linear & Polynomial Regression

- **Linear regression** models the relationship between the input features $x_1, x_2, \cdots, x_n$ and a target variable y as a **linear combination** of inputs:

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

- **Polynomial regression** extends linear regression by including **cross-terms and powers** of the inputs:

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2 \cdots$$



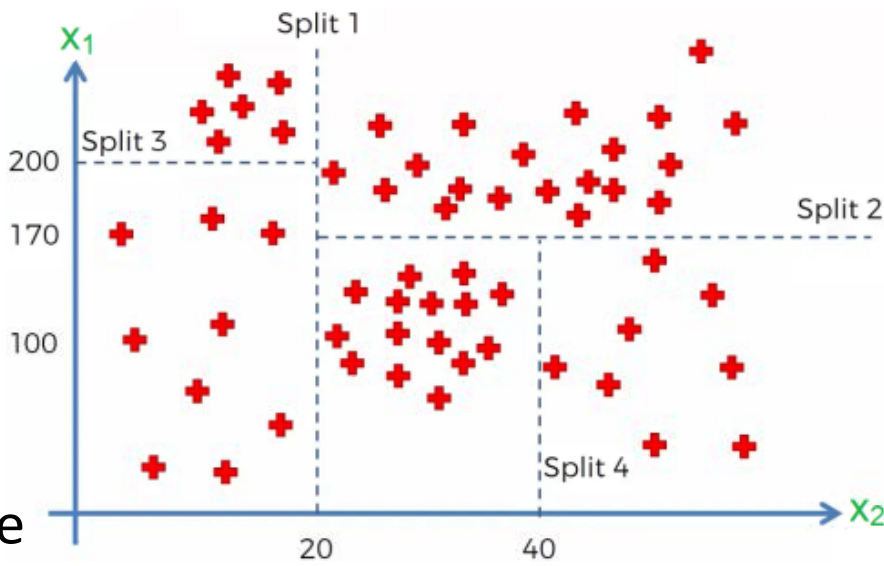
Polynomial regression is often implemented by first expanding input features and then applying ordinary linear regression to the transformed features.

Regularized forms are derived by adding the penalty terms (e.g., L1 and L2) to the MSE loss.

Tree-Based Regression

Basic Idea: Partition the data space into regions with similar target values, and predict y as the average in the region R_m containing the input:

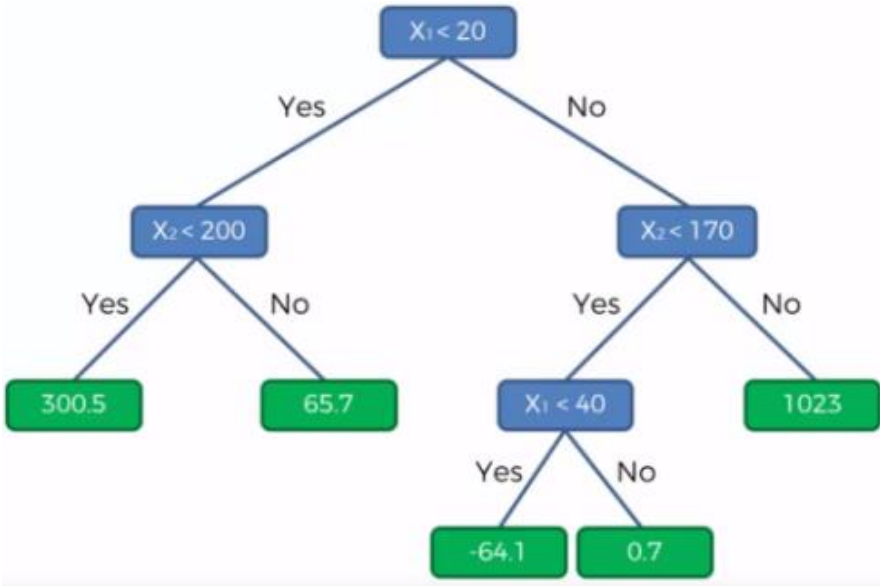
$$\hat{y} = \bar{y}_m = \frac{1}{|R_m|} \sum_{i \in R_m} y_i$$



Data Splitting: Splits are chosen to minimize the Residual Sum of Squares (RSS):

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} \left(y_i - \hat{y}_{R_j} \right)$$

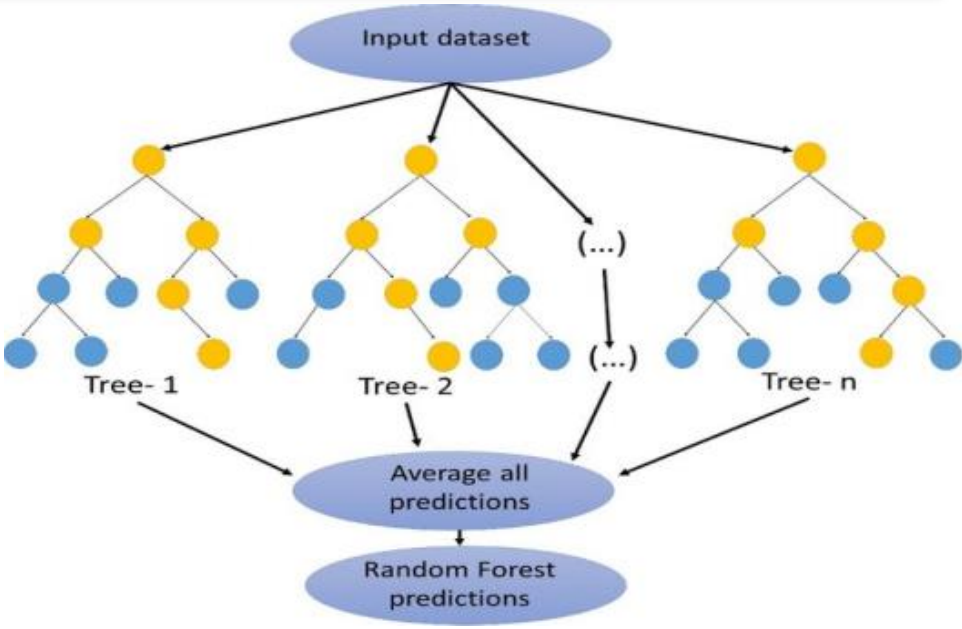
Tree Construction: At each node, the algorithm selects the feature and threshold that minimize error, recursively splitting data until stopping criteria (e.g., max depth or min samples per leaf) are met.



Ensemble Tree Models

Random Forest (Bagging)

- Trains many trees on bootstrap samples (data with replacement).
- Each split considers a random subset of features.
- Prediction = average of all tree outputs.
- Reduces variance and overfitting.



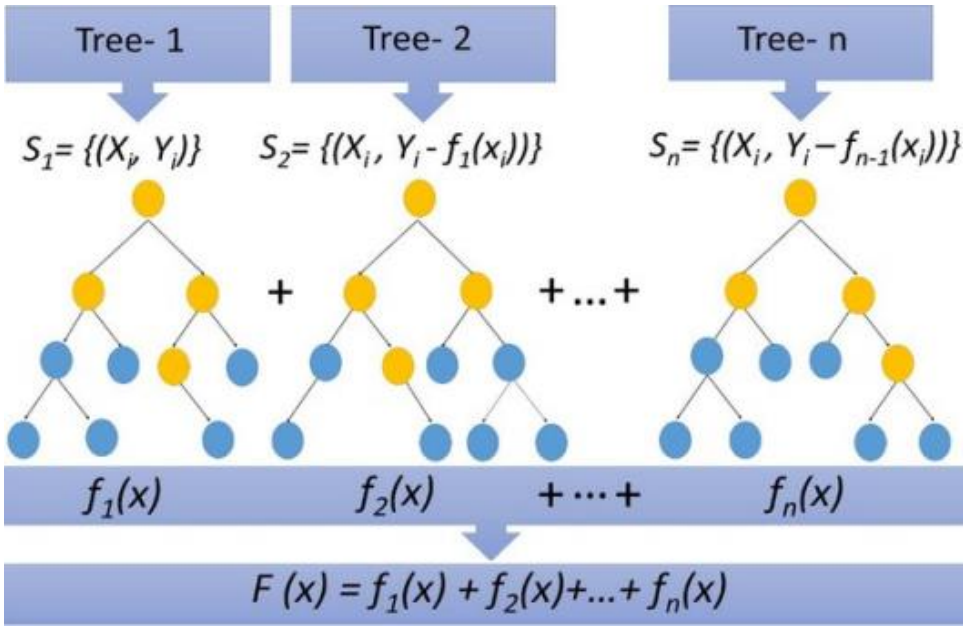
Gradient Boosting

- Builds trees sequentially, each predicting the residuals of the previous model.

$$\hat{y}^{(m)}(x) = \hat{y}^{(m-1)}(x) + \gamma_m h_m(x)$$

where $\hat{y}^{(m)}(x)$ is the new tree and γ_m is the learning rate.

- Gradually corrects errors, improving accuracy.
- Requires tuning to prevent overfitting.



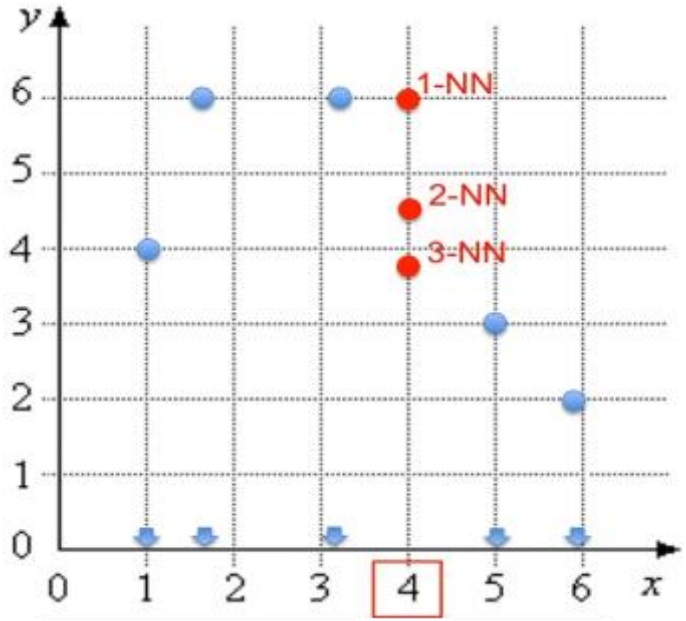
K-Nearest Neighbors (KNN) Regression

Idea:

- Predict the output for a new input by averaging the targets of its *k* nearest neighbors.

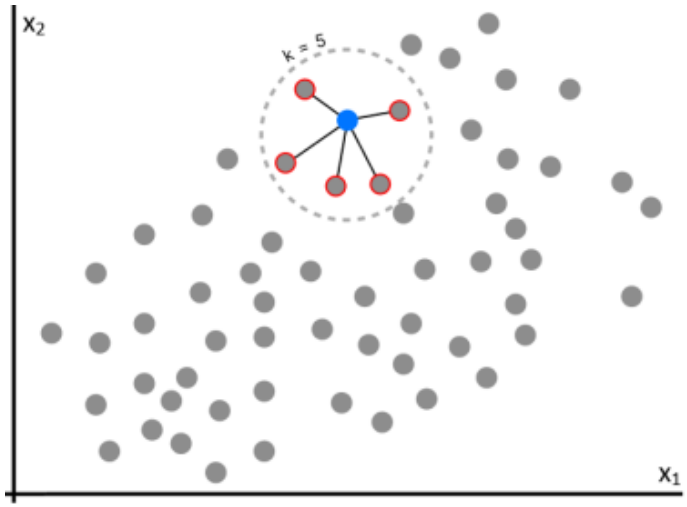
$$\hat{y}(x) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(x)} y_i$$

$\mathcal{N}_k(x)$ is the set of the k closest samples to x.



Key Points:

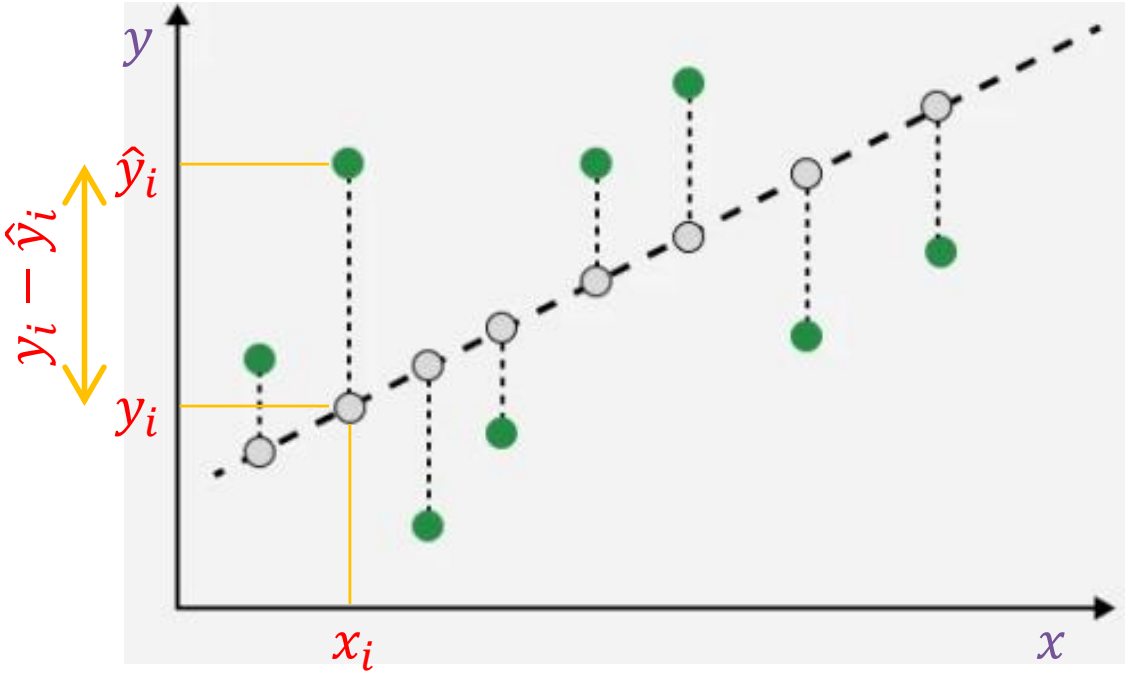
- **Non-parametric:** No model is trained; predictions are based directly on data.
- **Distance metric:** Usually Euclidean; defines the neighborhood.
- **Effect of *k*:**
  - Small *k* → flexible but sensitive to noise.
  - Large *k* → smoother but may underfit.
- **Works best** in low-dimensional spaces.



Note: Regression can also be performed using Neural Networks — introduced in the next chapter.

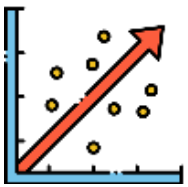
Evaluation & Diagnostics

- **Common metrics:**
  - $\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
    - robust to outliers
  - $MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i}$
    - unstable near zero
  - $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
    - penalizes large errors
  - *Coefficient of Determination:* $R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y}_i)^2}$
    - fraction of variance explained



- **Diagnostics & Industrial Notes:**
  - Check predicted vs. actual (identity line), residual and learning curves
  - Select metrics reflecting operational or business costs
  - Use k-fold CV; rolling/blocked CV for temporal data

Regression - Summary

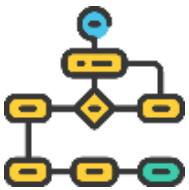


- **Linear Regression**
  - Fast, interpretable baseline; assumes linearity.
  - Struggles with nonlinearity and multicollinearity.

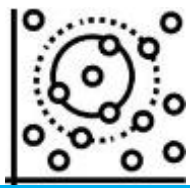
- **Polynomial Regression**
  - Extends linear models to nonlinear patterns.
  - Risk of overfitting with high-degree terms.

- **Decision Tree Regression**
  - Captures nonlinearities and interactions.
  - Easy to visualize, but prone to overfitting.

- **Random Forest (Bagging)**
  - Reduces variance via averaging multiple trees.
  - Robust, less interpretable.



- **Gradient Boosting**
  - Sequentially corrects prior errors; very accurate.
  - Needs careful tuning (learning rate, number of trees).



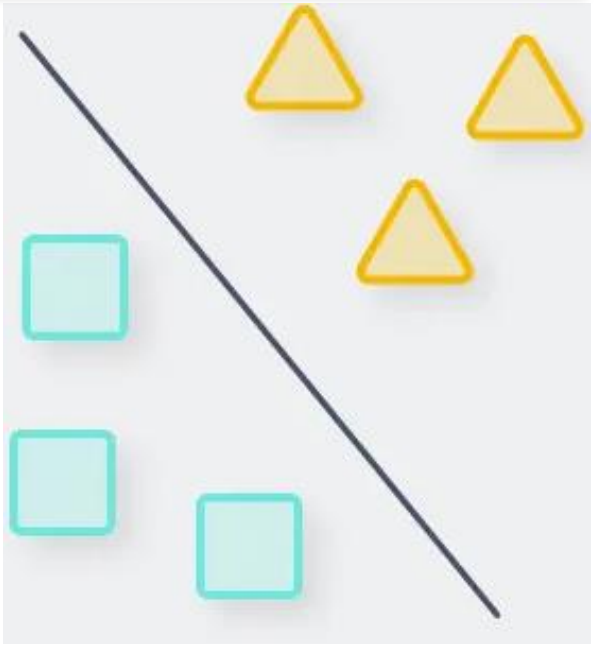
- **KNN Regression**
  - Simple, non-parametric; adapts to local data.
  - Sensitive to noise and scaling; slow on large data.

Chapter 4: Basic ML Algorithms

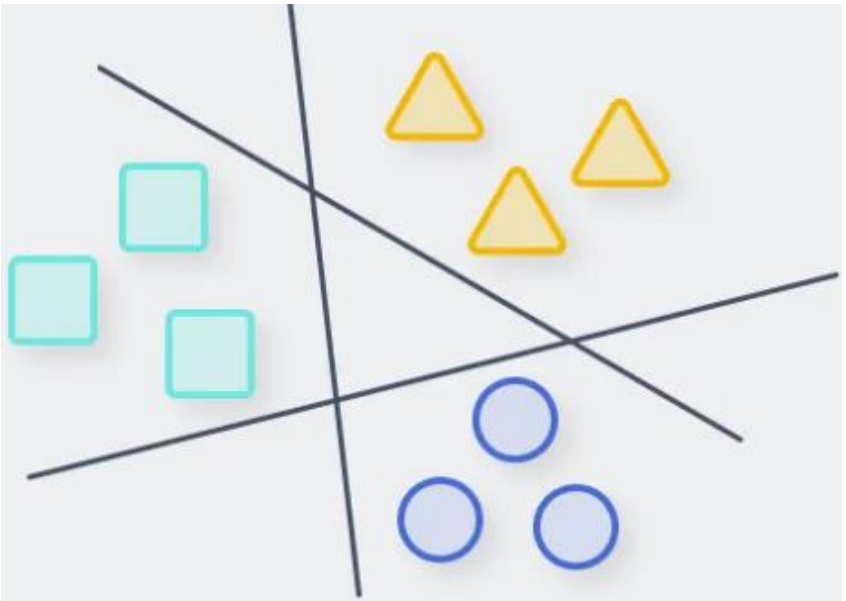
- Section 1: ML Algorithms & Training Basics
  Overview of ML types and core training concepts.
- Section 2: Regression Models
  Predicting continuous outputs from industrial process data.
- Section 3: Classification Models
  Assigning discrete labels for fault detection and decision tasks.
- Section 4: Unsupervised Learning
  Discovering hidden patterns and structure without labeled data.

Introduction to Classification

- **Goal:** Predict **discrete categories** (labels) instead of continuous values.
  - Example: Machine status → {Normal, Warning, Fault}
  - Example: Product quality → {Pass, Fail}



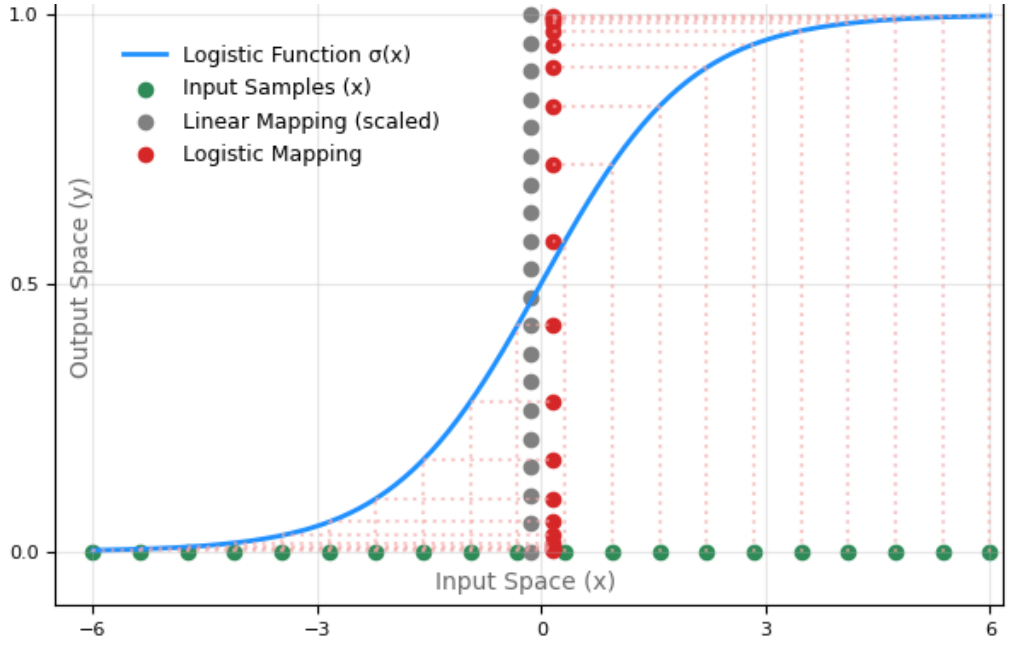
- **Types of Classification:**
  - **Binary:** Two classes (e.g., defect vs. no defect)
  - **Multi-class:** More than two classes (e.g., machine states, fault location)
- **Core Concepts:**
  - Decision boundaries separate classes in feature space.
  - Loss functions differ from regression (e.g., Cross-Entropy for classification).
  - Evaluation metrics: accuracy, precision, recall, F1-score (detailed later).



Logistic Regression

- Extends Linear Regression to **binary classification**, by predicting the probability of Class 1 (vs. Class 0) via the sigmoid function:

$$\hat{y} = \sigma(\boldsymbol{w}^T \boldsymbol{x} + b) = \frac{1}{1 + e^{(\boldsymbol{w}^T \boldsymbol{x} + b)}}$$



- Decision rule:

$$\hat{y} \geq 0.5 \Rightarrow Class1,$$
$$\hat{y} < 0.5 \Rightarrow Class0$$

- Use Binary Cross-Entropy Loss.

- **Key Points (Industrial Note):**
  - Highly **interpretable** — useful when understanding feature impact matters.
  - Works well for **linearly separable** data or as a **baseline classifier**.

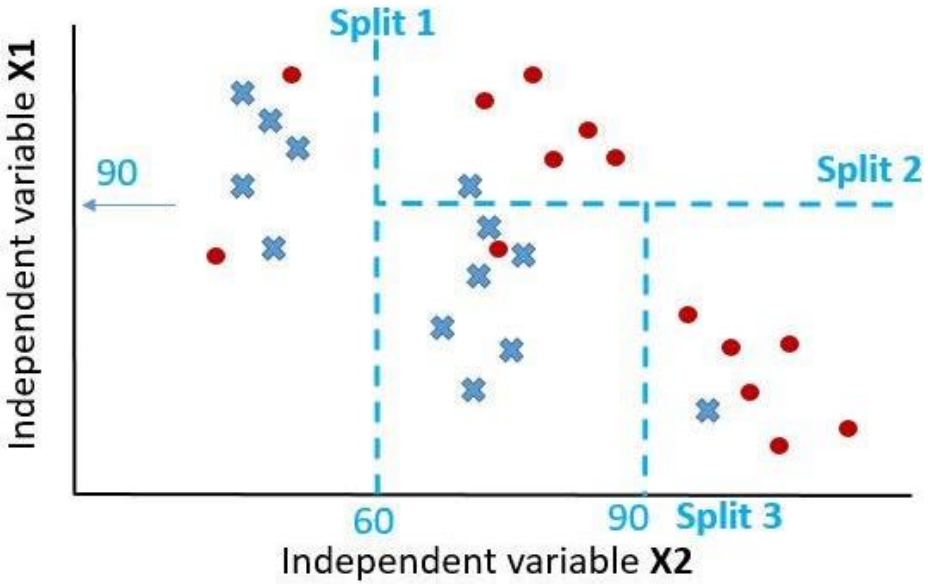
- **For multi-class** problems use **softmax** to predict probabilities:

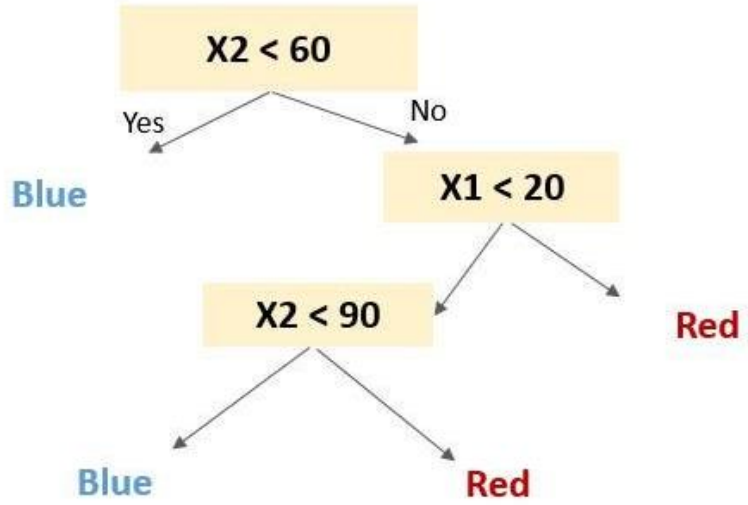
$$\hat{y}_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

Decision: class with **highest probability**

Tree-Based Classifiers

- **Basic idea:** Split feature space; assign **majority class** in each leaf.
- **Key points:**
  - Handles **nonlinear boundaries**.
  - Works with **mixed feature types**.
  - **Interpretable** — visualize splits and leaf classes.
  - Useful for **baseline classifiers** or when interpretability is required.
  - Ensemble Tree Models (Random Forest, Gradient Boosting) improve accuracy and reduce overfitting.

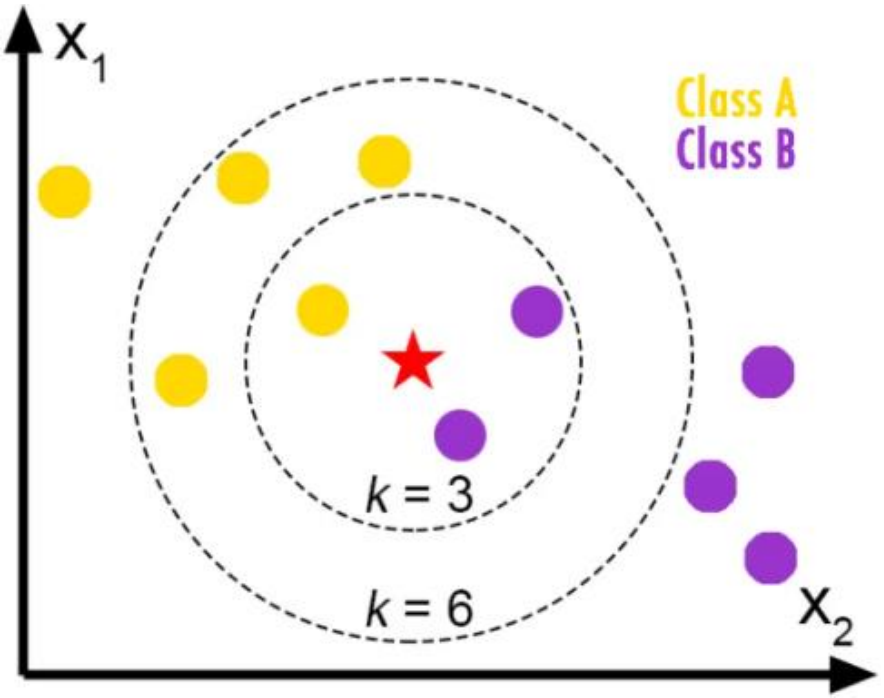




K-Nearest Neighbors (KNN) Classifiers

- Classifies a sample by the **majority label** among its *k* nearest neighbors in feature space.
- Simple and intuitive; effective for small or low-dimensional datasets but **computationally heavy** for large ones.
- Sensitive to data scaling and the choice of *k*.

- **Distance measures:**
  - Euclidean: $d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$
  - Manhattan: $d(p,q) = \sqrt{\sum_i |p_i - q_i|}$
  - Other metrics (e.g., cosine, Minkowski) can be used depending on data characteristics.



- **Choosing K:**
  - Small K → captures local detail but noisy
  - large K → smoother decision boundary, less sensitive to outliers

Naïve Bayes Classifier

- Based on **Bayes' theorem**:

$$P(C_k|x) \propto P(x|C_k)P(C_k)$$

  - predict the class C_k with the highest posterior probability
- **Probabilities are estimated as:**
  - Feature independence (Naïve): $P(x|C_k) = \prod_i P(x_i|C_k)$
  - $P(C_k)$: frequency of each class in the training data
  - $P(x_i|C_k)$: estimated from data distribution of feature x_i under class C_k
- **Industrial example** (fault detection):

Feature	Value	P(x_i | Normal)	P(x_i | Faulty)
Temperature > 60 °C	Yes	0.2	0.8
Vibration > 1.0 m/s²	Yes	0.1	0.7
Prior P(C_1), P(C_2)	-	0.6	0.4

$P(Normal \mid x) \propto 0.2 \times 0.1 \times 0.6 = 0.012$

$P(Faulty \mid x) \propto 0.8 \times 0.7 \times 0.4 = 0.224$

Predicted class: Faulty

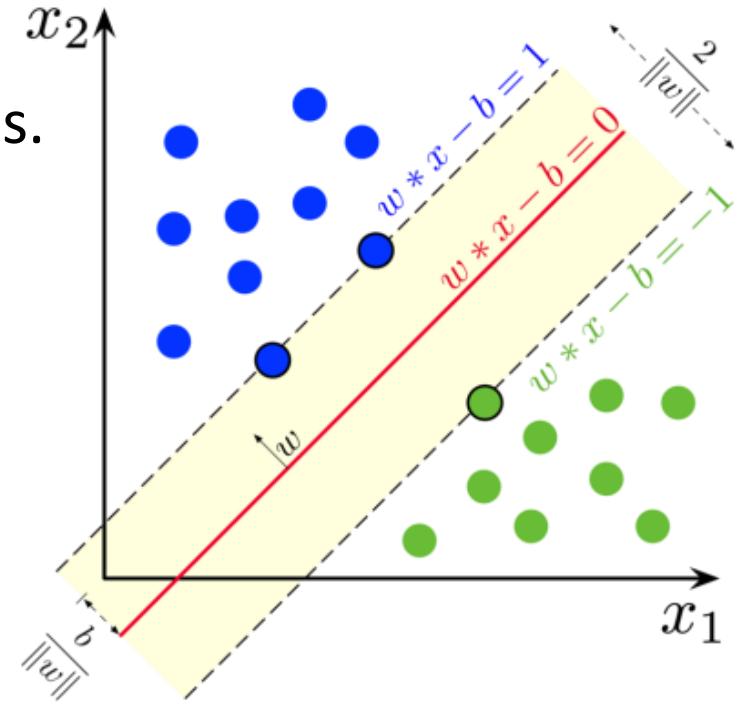
- **Common variants:**
  - Gaussian NB – continuous features (e.g., temperature, vibration, voltage levels)
  - Multinomial NB – discrete/count features (e.g., number of sensor triggers)
  - Bernoulli NB – binary features (e.g., alarm on/off, switch status)

Support Vector Machines (SVM)

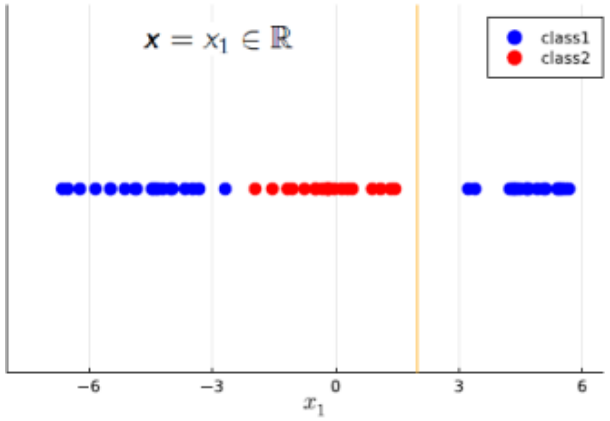
- **Concept:** Find the hyperplane that **maximizes the margin** between two classes.
- **Decision function:** $f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$
- Optimization problem:

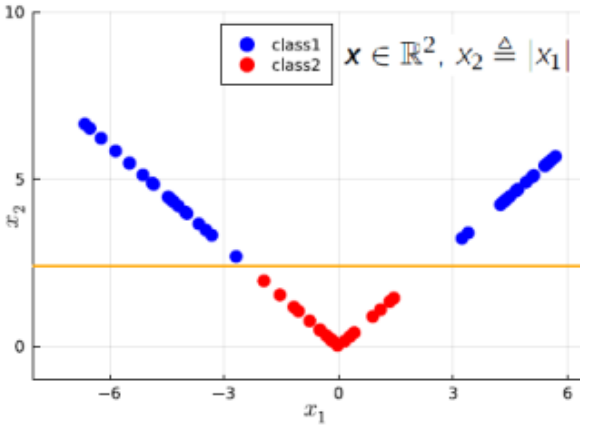
$$\min_{\boldsymbol{w},b} \frac{1}{2} \|\boldsymbol{w}\|^2 \quad s.t. \quad y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1$$

  - Only **support vectors** (points on the margins) affect the solution



- **Extensions:**
  - **Soft-Margin SVM:** Permits some misclassification for noisy data
  - **Kernel SVM:** Maps inputs to higher dimensions for non-linear separation

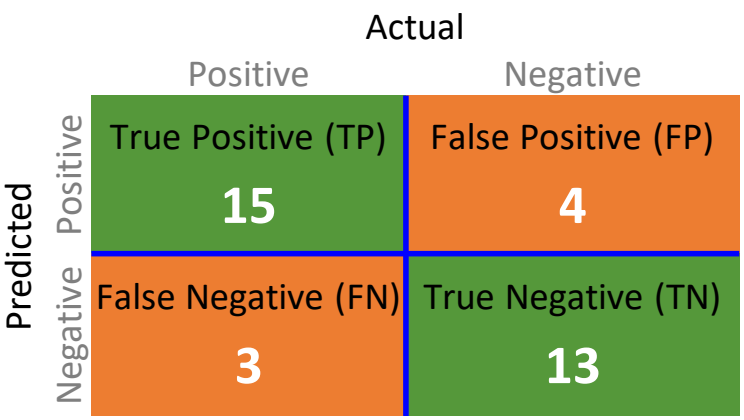




  - **SVR** (Support Vector Regression): An SVM variant for regression tasks

Evaluation Metrics

Core Metrics



Confusion Matrix

$$\text{Precision} = \frac{TP}{TP + FP} = 0.833$$

$$\text{Specificity} = \frac{TN}{TN + FP} = 0.813$$

$$FPR = \frac{FP}{FP + TN} = 0.188$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = 0.800$$

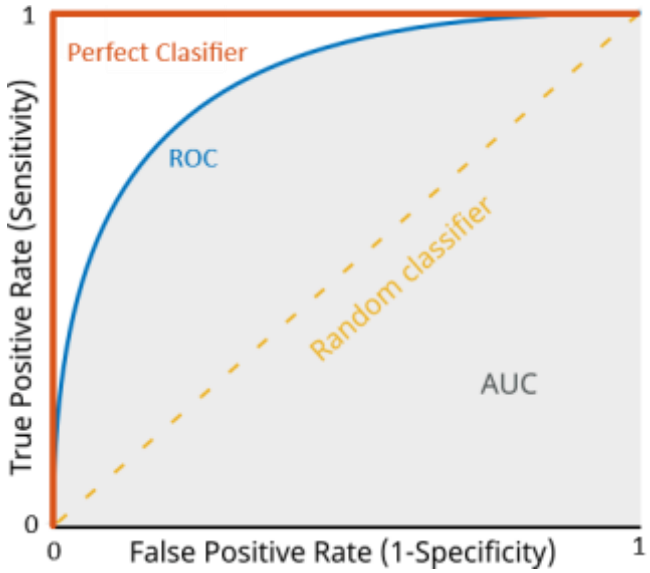
$$Recall\ (Sens., TPR) = \frac{TP}{TP + FN} = 0.789$$

$$F1 = \frac{2 * \text{Precision} * Recall}{\text{Precision} + Recall} = 0.810$$

- **Accuracy:** Overall correct predictions
- **Precision:** True positives among predicted positives
- **Recall (Sensitivity):** True positives among actual positives
- **F1-Score:** Balance of Precision & Recall

ROC/AUC

- **ROC:** Plot of TPR vs. FPR for different thresholds.
- **AUC:** Area under ROC



- **Multi-Class Evaluation**

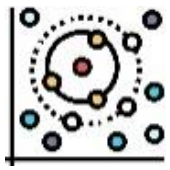
  Compute metrics **for each class separately** (treat one as "positive," others as "negative"), and then average:

  - **Macro:** Equal weight to all classes - good for balanced data
  - **Weighted:** Weighted by class frequency - handles imbalance
  - **Micro:** Aggregates TP, FP, FN globally - reflects overall performance

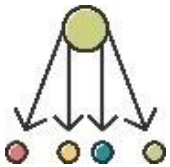
Classification - Summary

- **Logistic Regression**
  - Fast, interpretable baseline for linear separations
  - Struggles with nonlinear boundaries

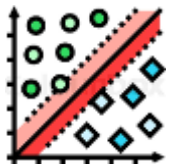
- **Decision Tree Classifier**
  - Captures nonlinearities and interactions
  - Easy to visualize, but prone to overfitting



- **KNN (k-Nearest Neighbors)**
  - Simple, non-parametric; adapts to data shape
  - Sensitive to scaling and k choice



- **Naive Bayes**
  - Probabilistic; fast and effective for text or event data
  - Assumes feature independence



- **Support Vector Machine (SVM)**
  - Maximizes margin; strong for high-dimensional data
  - Kernel trick handles nonlinear cases; slower on large sets

- **Evaluation Metrics**
  - Accuracy, Precision, Recall, F1, ROC–AUC
  - Macro/Micro averaging for multi-class tasks

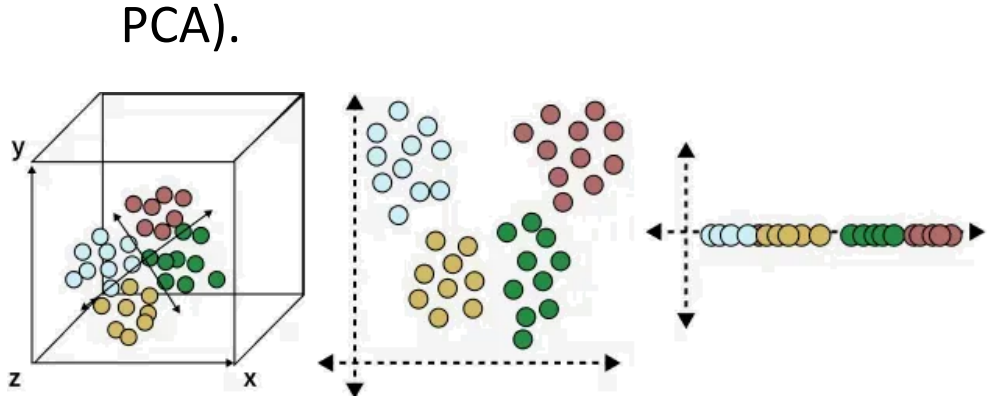
➢ **Note:** NNs also perform classification — covered in *Deep Learning* chapter.

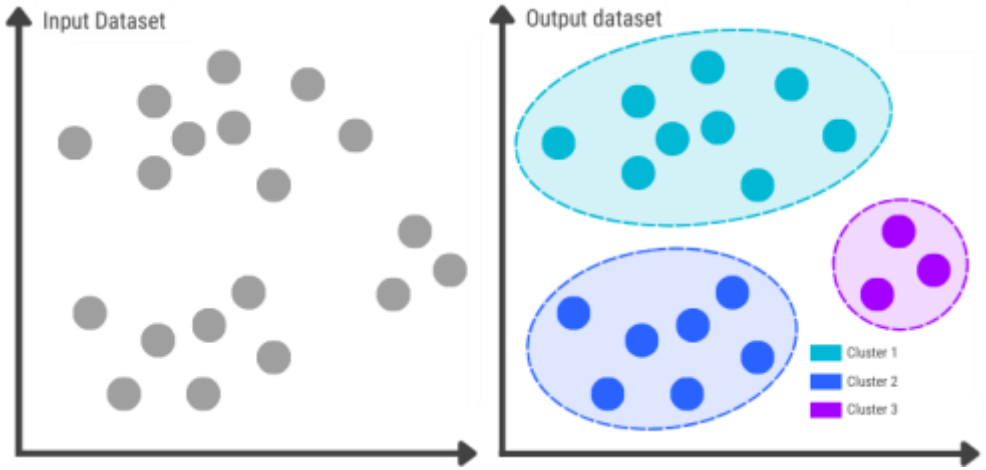
Chapter 4: Basic ML Algorithms

- Section 1: ML Algorithms & Training Basics
  Overview of ML types and core training concepts.
- Section 2: Regression Models
  Predicting continuous outputs from industrial process data.
- Section 3: Classification Models
  Assigning discrete labels for fault detection and decision tasks.
- Section 4: Unsupervised Learning
  Discovering hidden patterns and structure without labeled data.

Introduction to Unsupervised Learning

- **Definition:**
  Learning from **unlabeled data** to **discover hidden structures or patterns** without predefined outputs.
- **Main Approaches:**
  - **Clustering:** Groups similar samples (e.g., *k*-Means).
  - **Dim. Reduction:** Compresses data while preserving important information (e.g., PCA).

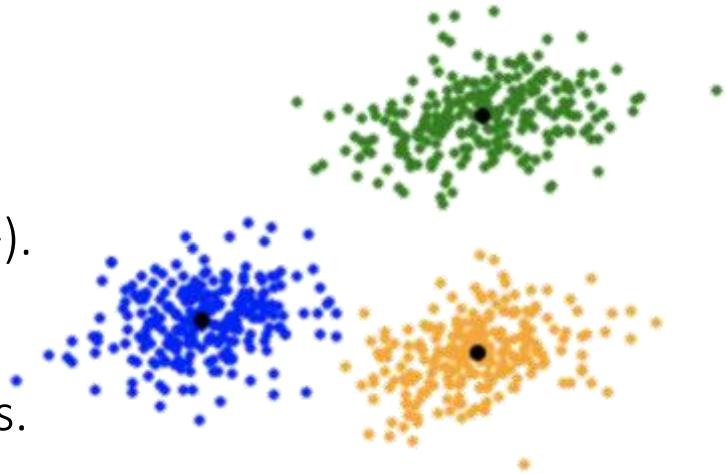




- **Industrial Context:**
  Industrial systems often produce large volumes of **unlabeled sensor data**.
  Unsupervised methods support:
  - Identifying **operating modes** of machines.
  - **Anomaly detection** in production systems.
  - **Data compression** for real-time monitoring.

K-Means: Concept

- **Idea:**
  k-Means partitions data into *k* groups so that samples within each group are **similar**, and groups are **distinct** from one another.
- **Key Intuition:**
  Each cluster is represented by its center (centroid) — data points are assigned to the nearest centroid, and centroids shift to the mean of their assigned points.
- **Algorithm Steps:**
  1. Choose number of clusters (*k*).
  2. Initialize *k* centroids (randomly or with *k*-Means++).
  3. Assign each sample to the **nearest centroid**.
  4. Update centroids as the **mean** of assigned samples.
  5. Repeat steps 3–4 until assignments stop changing.

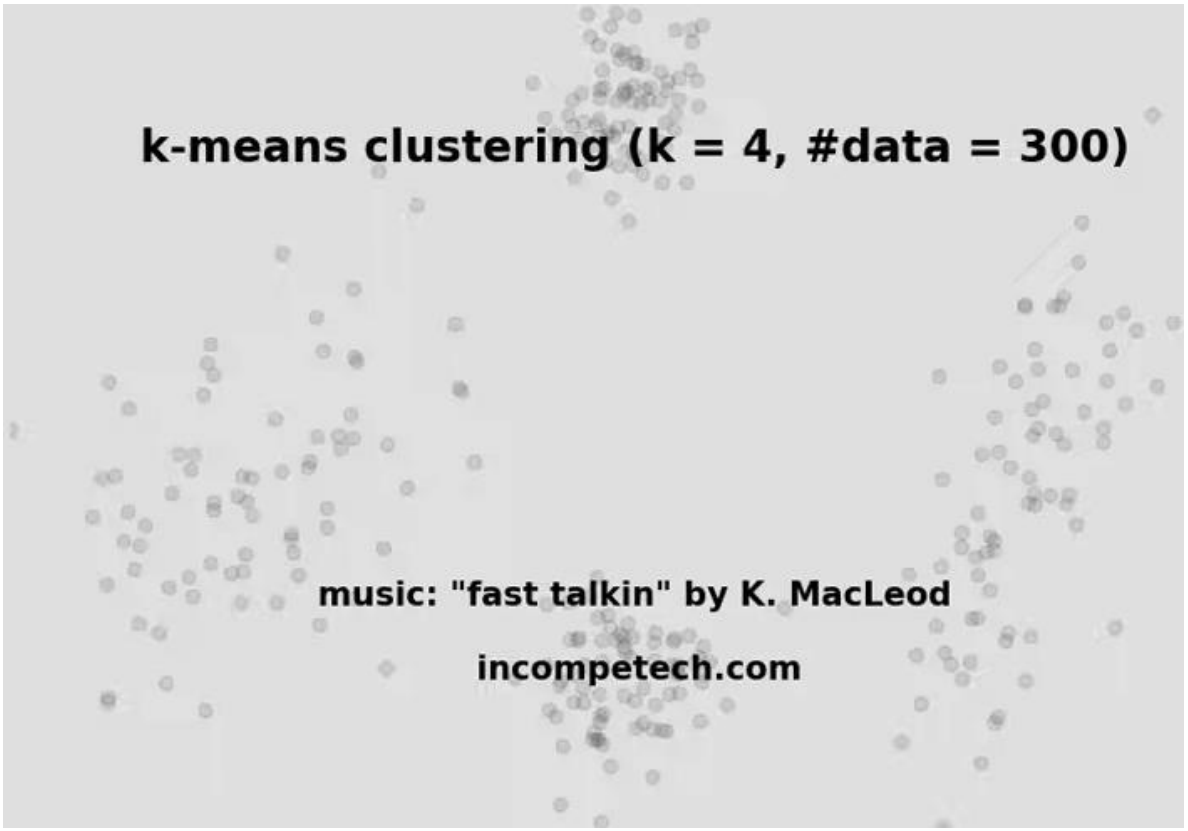


- **Distance Measure (default):**

$$d(x, \mu_i) = \|x - \mu_i\|^2$$

This quantifies the closeness of data point to cluster center

K-Means: Visualization & Practical Notes

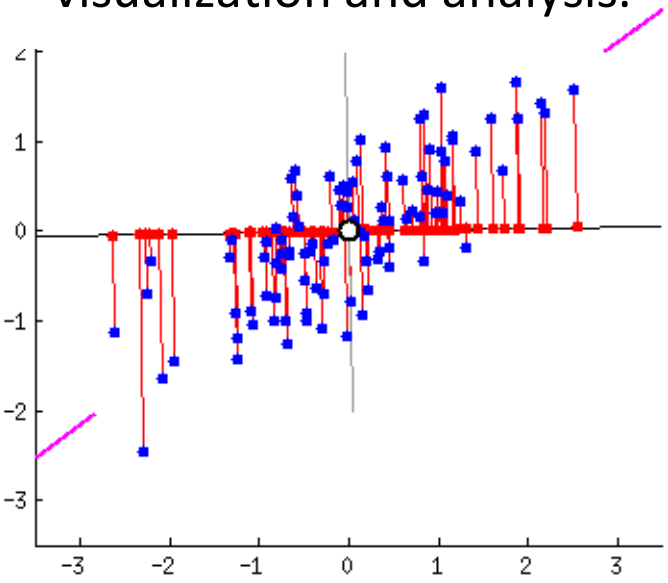


- Other distance measures:
  - Manhattan (L1): $|x - \mu_i|$
  - Cosine distance: $1 - \frac{x \cdot \mu_i}{\|x\|\|\mu_i\|}$
- Stopping Criteria:
  - Cluster assignments stop changing
  - Centroid shifts below a small threshold
  - Maximum iteration limit reached

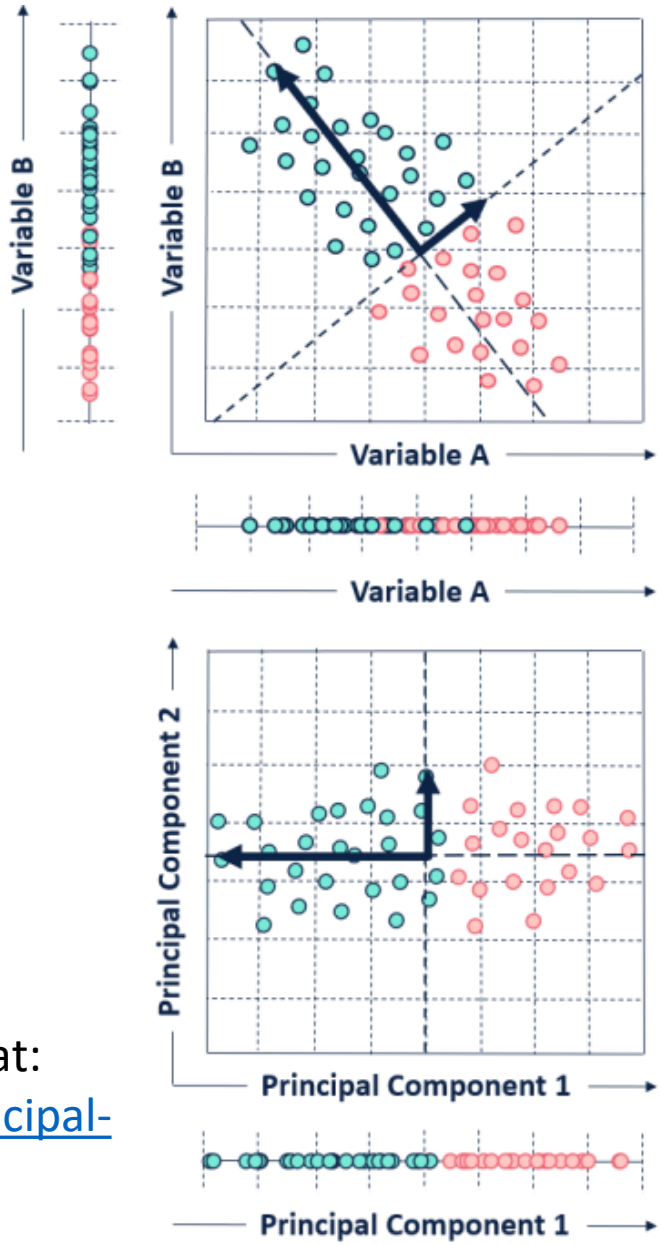
- Choosing Number of Clusters (k):
  - **Elbow Method:** Plot total variance vs. k; look for "bend".
- **Initialization Strategy:**
  - *k-Means++* improves stability and convergence speed.
  - Run algorithm multiple times with different seeds → choose lowest loss.

PCA: Intuition and Goal

- **Objective:**
  Reduce high-dimensional data to a few principal components capturing main patterns and variations.
- **Key Ideas:**
  - Industrial data often have correlated variables (e.g., temperatures, currents).
  - PCA finds uncorrelated axes capturing maximum variance, ordered by importance.
  - Projecting onto top components enables simpler visualization and analysis.

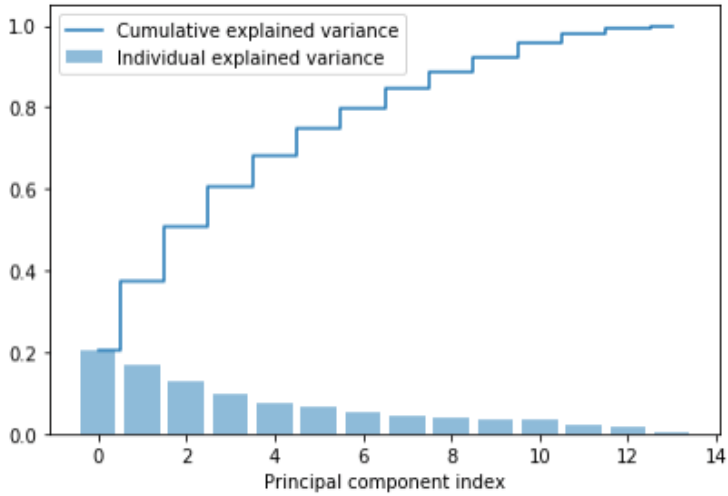


Explore interactive GIFs at:
https://setosa.io/ev/principal-component-analysis/



PCA: Procedure & Key Notes

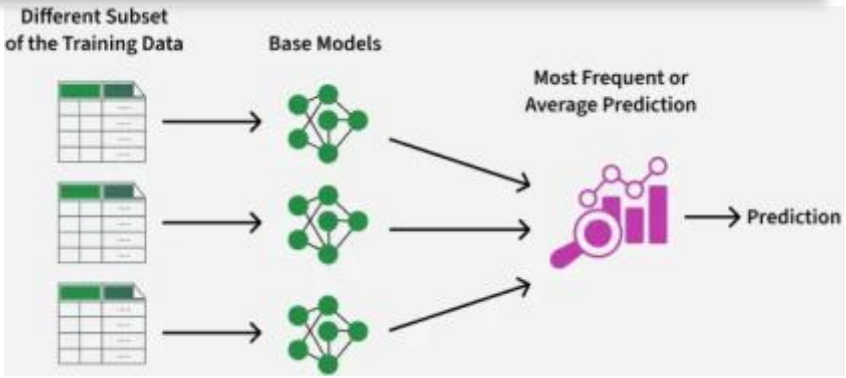
- **Procedure:**
  1. Center/standardize data.
  2. Compute covariance: $\Sigma = \frac{1}{n-1} X^T X$
  3. Eigen-decomposition: $\Sigma \boldsymbol{v}_i = \boldsymbol{v}_i \lambda_i \rightarrow \boldsymbol{v}_i = PC\ direction, \lambda_i = variance$
  4. Sort PCs by λ_i, select top k.
  5. Project data: $Z = XV_k$.

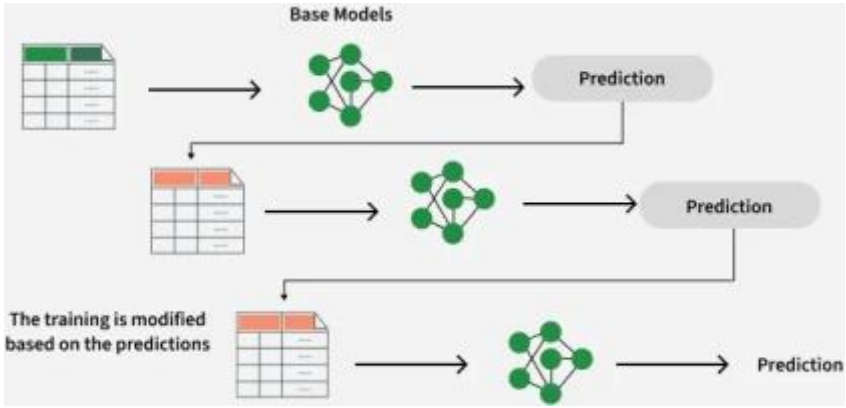


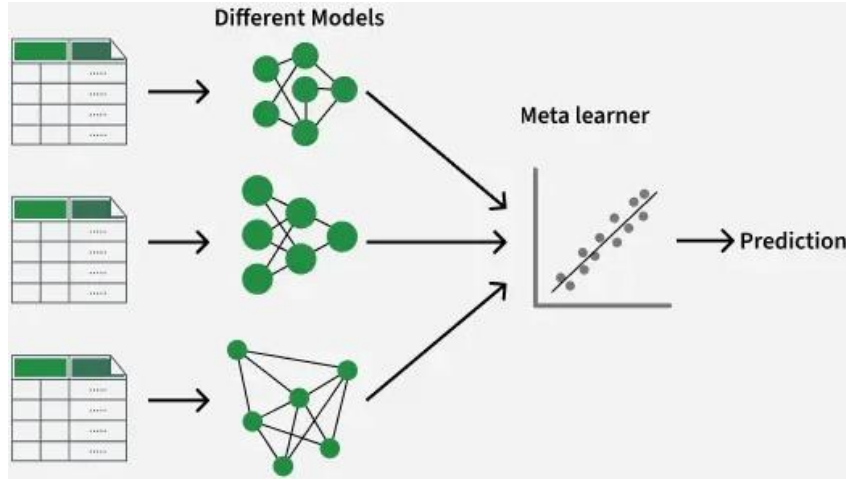
- **Notes & Practical Tips:**
  - PCs = linear combinations of original features.
  - For nonlinear patterns, use kernel PCA or t-SNE.
  - PCs are **orthogonal** (uncorrelated).
  - PCA **rotates axes** to align with directions of maximum variance.
  - Keep only top PCs to reduce dimensionality and filter noise.
  - **Industrial Monitoring:**
    - Sudden jumps in PC scores indicate faults.
    - Slow drift may indicate sensor aging or process changes.
    - Applications: sensor monitoring, fault detection, visualization, predictive maintenance.

Ensemble Methods

- **Idea:**
  Combine multiple models to improve accuracy, stability, and generalization.
- **Techniques:**
  - **Bagging:** parallel models on bootstrap samples → aggregate predictions.
    Good for high-variance learners (trees).
  - **Boosting:** sequential models correct previous errors.
    Best with simple, fast learners.
  - **Stacking:** combine diverse models via a meta-learner.
    Leverages complementary strengths.
- **Industrial Benefits:**
  - Outperforms single models on high-dimensional data.
  - Robust to noise and overfitting.
  - Strong baseline before deep learning.







Summary & Next Steps

- **Summary:**
  - **Supervised vs. Unsupervised:** understand task type and data labels.
  - **Regression & Classification:** key models, loss functions, and evaluation metrics.
  - **Unsupervised Learning:** K-Means for clustering; PCA for dimensionality reduction and process monitoring.
  - **Ensemble Methods:** bagging, boosting, stacking improve accuracy, robustness, and generalization.
  - **Neural Networks:** versatile models for supervised and unsupervised tasks; foundation for deep learning.
- **Next Steps:**
  - **Deep Learning Models:** handling complex and unstructured industrial data