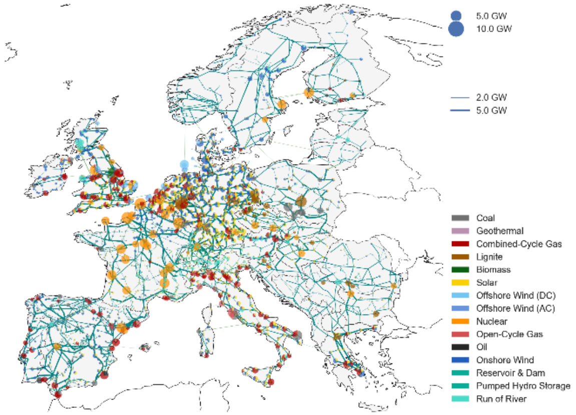
# Energy System Modeling with Python

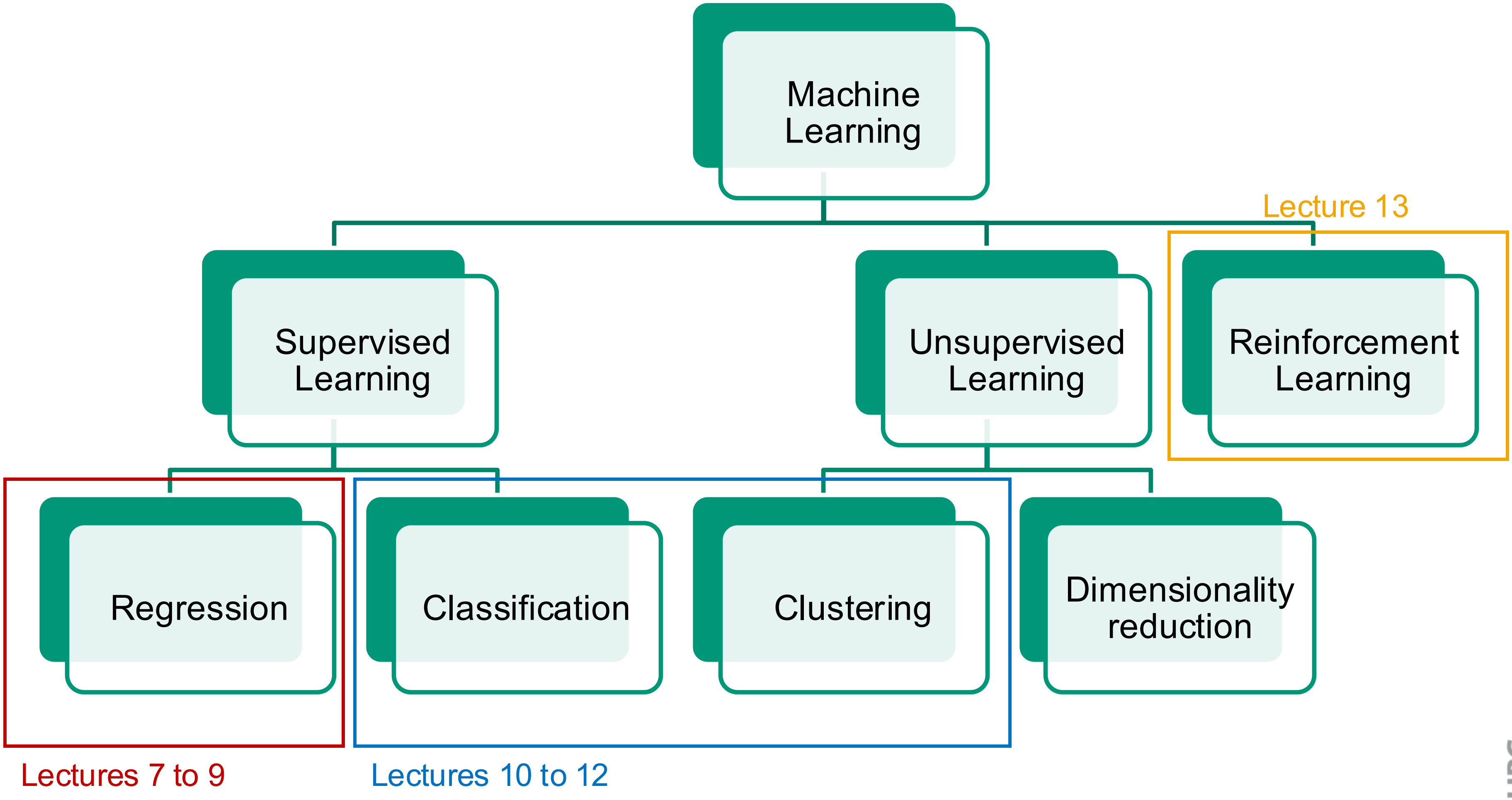
python




University of Freiburg (Germany) | Faculty of Engineering
Department of Sustainable Systems Engineering | INATECH
**Chair for Control and Integration of Grids**

Tuesday, 3. June 2025

# Branches of ML



# Theoretical part

# From Un- to Supervised Learning

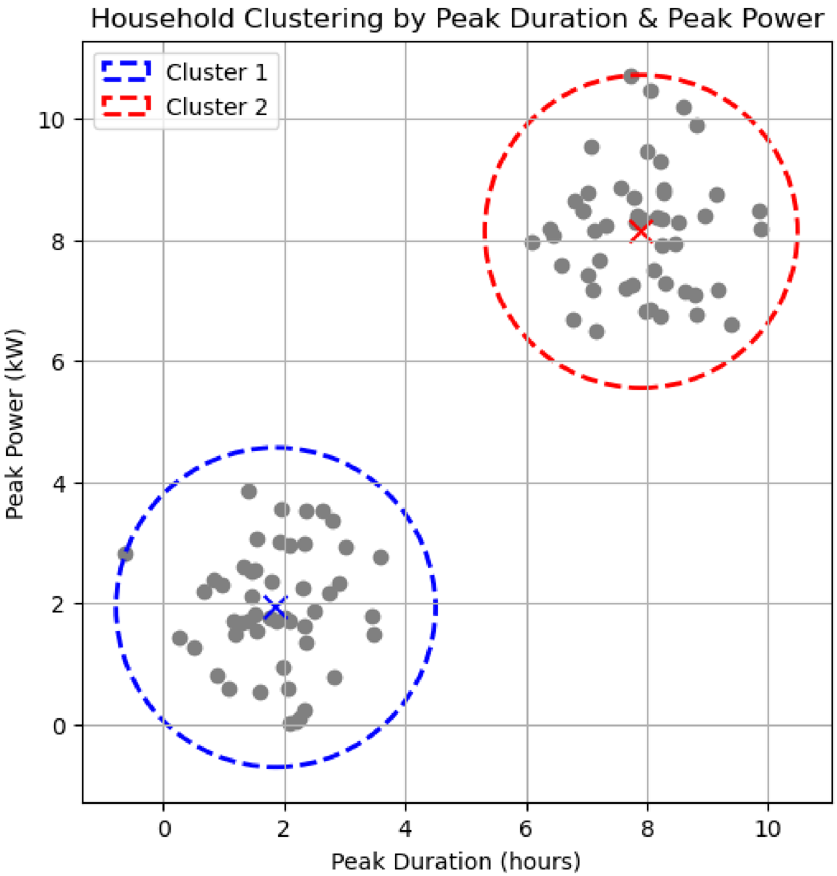
- **Last lecture:** labels were not available → **Unsupervised learning** to discover *structure* in data without labels (e.g., clustering similar load profiles)

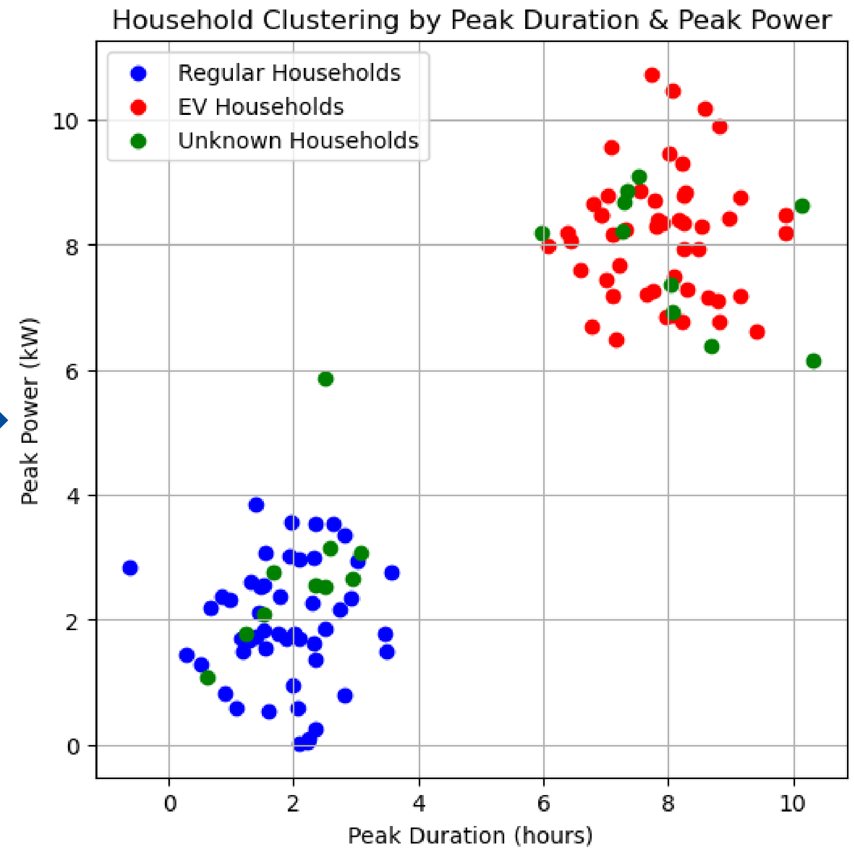
- **Now**: labels are available → move to **supervised learning** to learn a function $h: \mathcal{X} \to \mathcal{Y}$, where
  - $\mathcal{X}$ = input features,
  - $\mathcal{Y}$ = discrete class labels

- **Task:** predict class $y \in \{0, 1, \dots, K-1\}$ from features $x \in R^d$

$$\boxed{\text{Find } h(x) \approx y \quad \text{with } y \in \{\text{class labels}\}}$$

# Problem Formulation



# Regression vs Classification

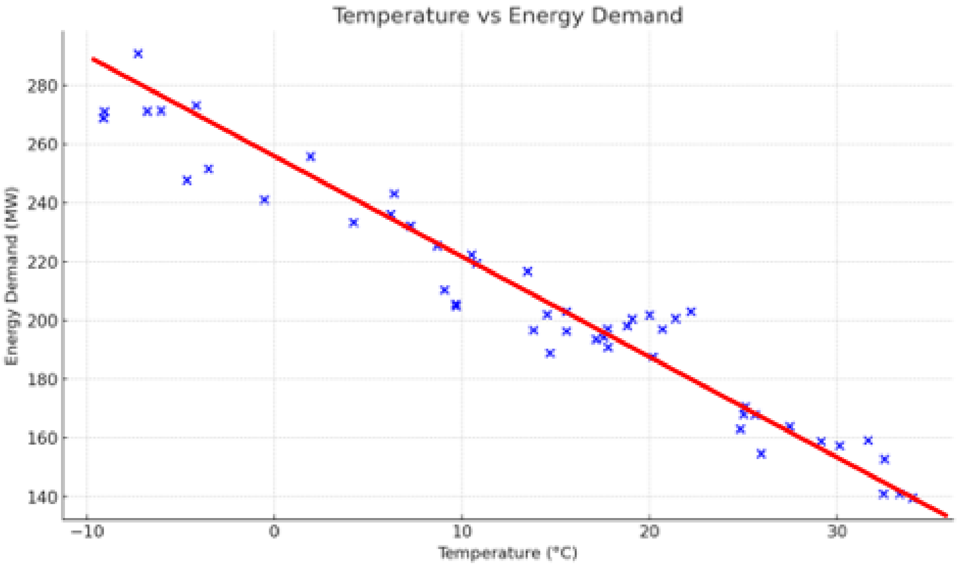
- **What changes when we add labels?** → Depends on the **type** of label

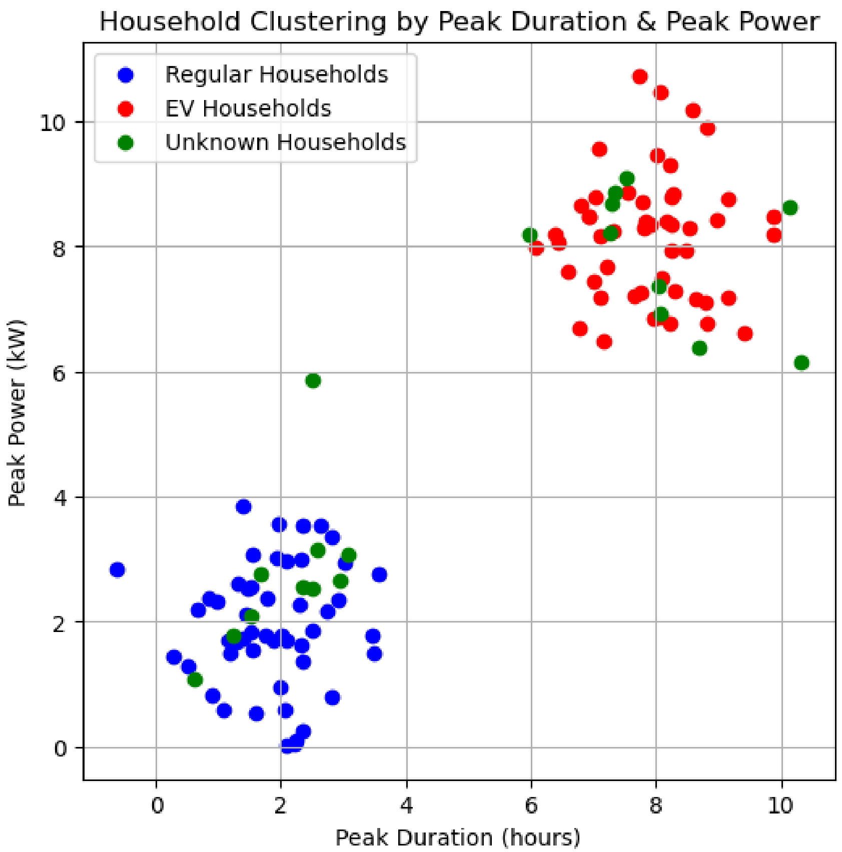
- **Regression**:
  - Target variable $y \in R$ (continuous)
  - Goal: predict precise numeric output
  - Typical loss: Mean Squared Error (MSE)

- **Classification**:
  - Target variable $y \in \{0, 1, \dots, K-1\}$ (discrete)
  - Goal: assign input to one of K classes
  - Typical loss: Cross-Entropy or 0-1 loss (more later)

- **Consequences for model behavior**:
  - Regression → continuous output
  - Classification → probability or discrete label

# Regression vs Classification



# Energy-System Examples: Regression vs Classification

| Task | Label Type | Goal | Method Type |
|---|---|---|---|
| Demand Forecasting | Continuous (kWh) | Predict numeric load | Regression |
| Device Type Classification | Multi-class (e.g., heat pump, boiler, ...) | Identify type | Classification |
| Anomaly Detection | Binary or multi-class | Detect and classify faults | Classification |
| Price Estimation | Continuous (€/MWh) | Predict future price | Regression |

- ✓ Choosing the correct ML formulation depends on the *nature of the label*
- ✓ Common tasks in energy systems involve **both** regression and classification

# Loss & Cost Refresher

- You've already seen this in **regression**:
  - *Loss* = error for one example
  - *Cost* = average loss over training set

- Same structure holds for classification, but with different loss functions

- Classification requires **losses suitable for discrete outputs**

- We'll look at:
  - 0-1 Loss (ideal, non-differentiable)
  - Hinge Loss (Support Vector Machine SVM)
  - **Cross-Entropy Loss** (used today)

# Classification Loss Functions – Overview

## 0–1 Loss

$$\mathcal{L}(y,\hat{y}) = \begin{cases} 0 \text{ if } y = \hat{y} \\ 1 \text{ otherwise} \end{cases}$$

- Directly measures classification accuracy
- Not differentiable → not usable for gradient descent

## Hinge Loss (SVMs)

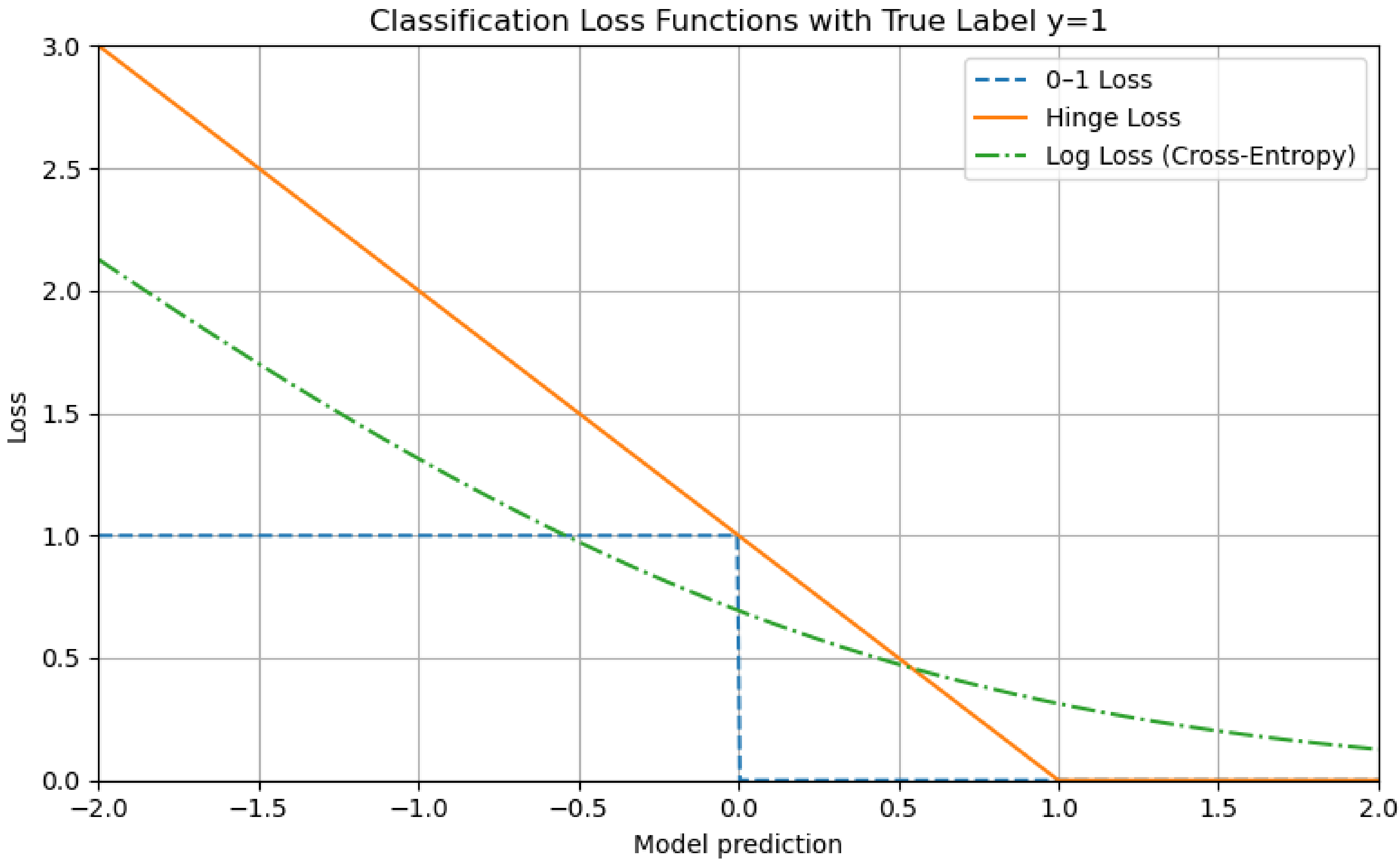
$$\mathcal{L}(y,\hat{y}) = \max(0{,}1 - y \cdot \hat{y})$$

- Promotes large margins
- Used in Support Vector Machines

## **Log Loss / Cross-Entropy** (our focus)

- Smooth, convex
- Measures probability error
- Used in logistic regression & neural networks

# Classification Loss Functions – Overview



# Cross-Entropy Loss (Binary Case)

- Used in **binary classification**, especially with **logistic regression**

- Prediction using **sigmoid function**:

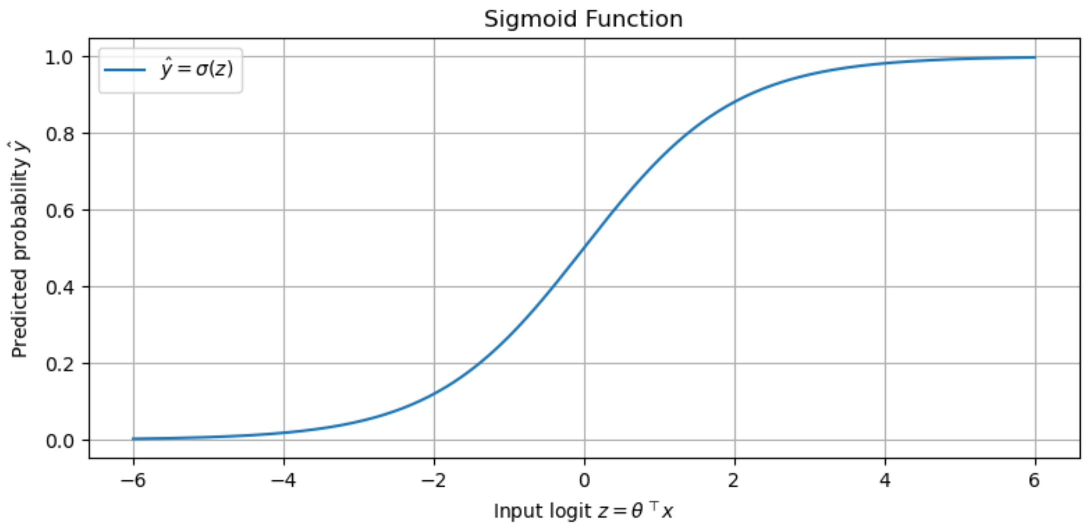
$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}, \quad z = \theta^{\top} x$$

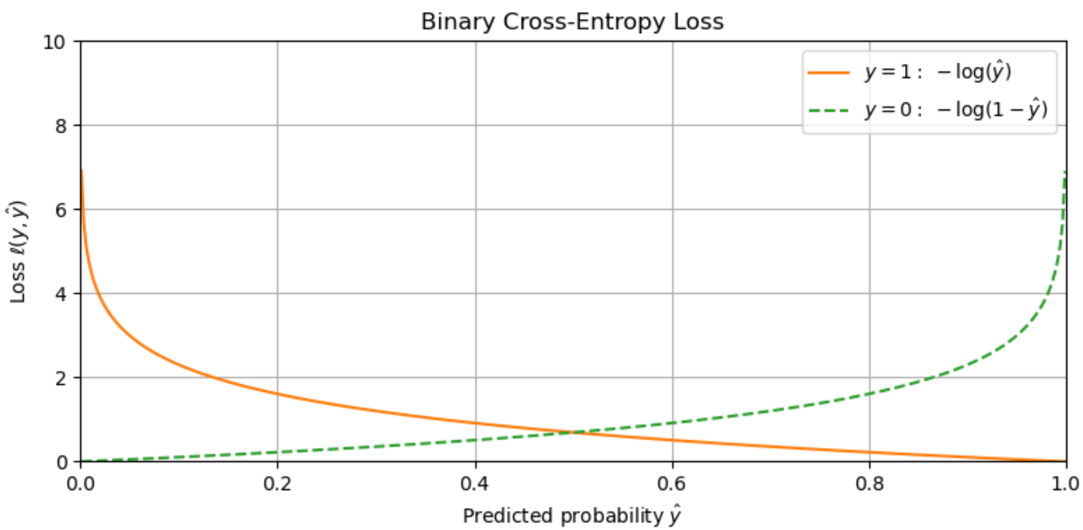
- **Binary Cross-Entropy Loss:**

$$\mathcal{L}(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

- **Intuition:**
  - Penalizes confident wrong predictions heavily
  - Encourages well-calibrated probabilities

# Cross-Entropy Loss (Binary Case)






INATECH

UNI FREIBURG

# Classification Models

# Logistic Regression – Probabilistic Classifier

- **Model form**:

$$\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}} \text{ where } z = \theta^{\top} x$$

- **Training objective**: minimize **binary cross-entropy**

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)]$$

- Optimized via **gradient-based methods** (e.g. gradient descent)

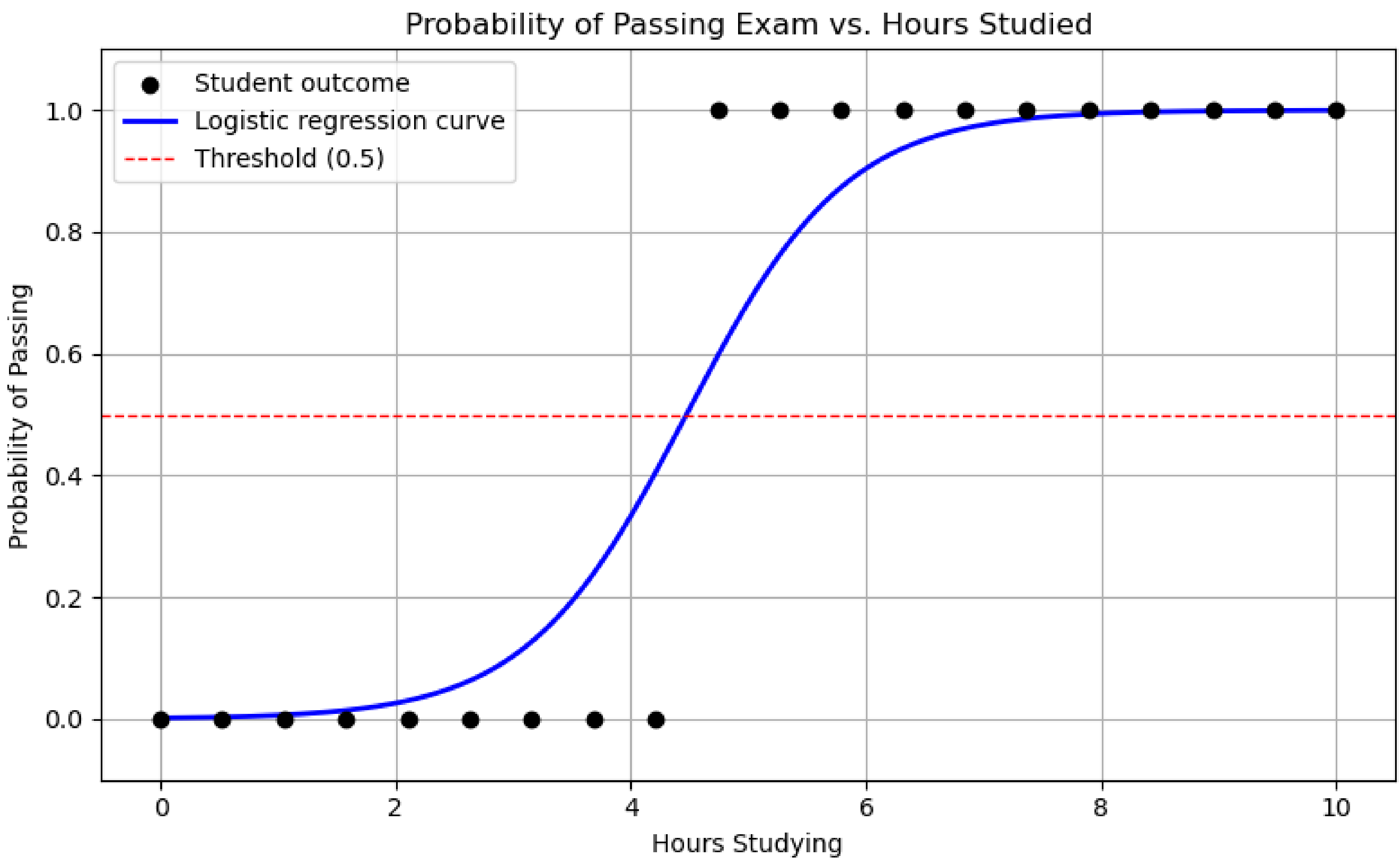
- Output $\hat{y} \in (0,1)$ is a **probability** estimate for class 1

- **Decision rule:**

$$\text{Predict class } = \begin{cases} 1 \text{ if } \hat{y} \geq \tau \\ 0 \text{ otherwise} \end{cases}$$

- Logistic regression learns a **linear decision boundary** in feature space

# Logistic Regression – Probabilistic Classifier



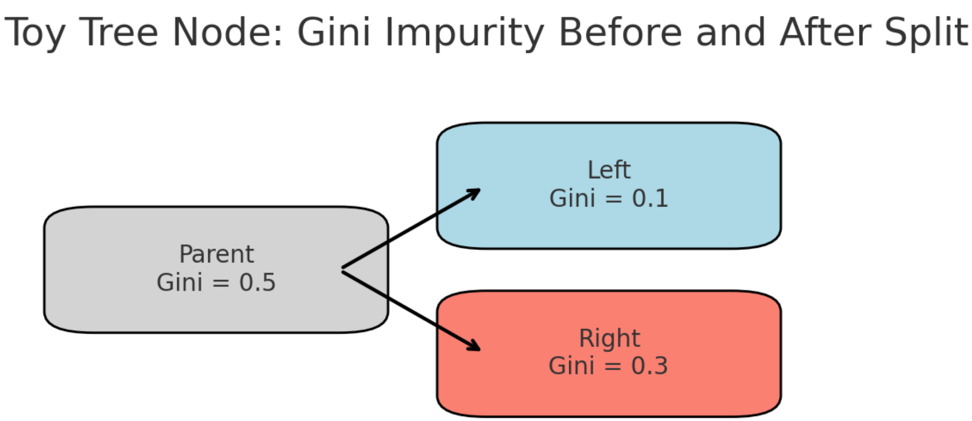
# Random Forest Classifier – Splits & Impurity

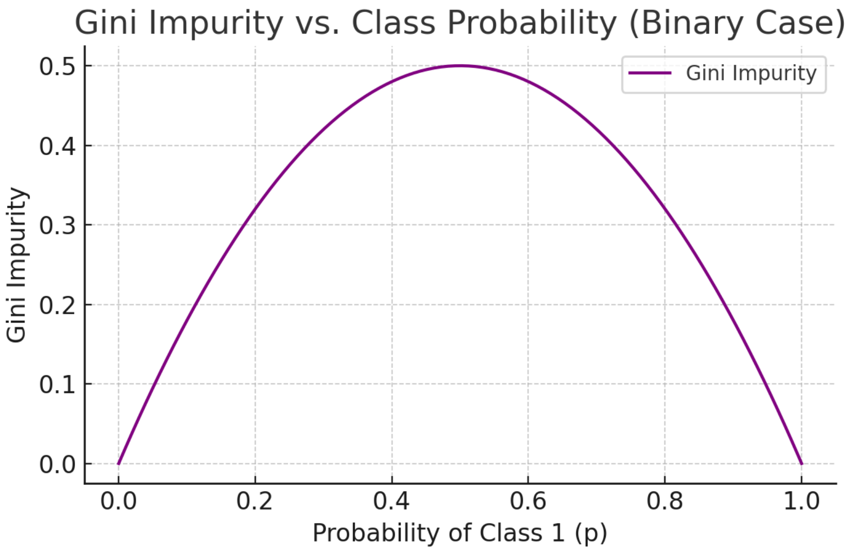
- **Ensemble of trees** built on bootstrapped samples & random feature subsets

- Trees choose splits that **maximize impurity reduction**

- **Node impurity** quantifies class mix:
  - **Gini**: $1 - \sum_k p_k^2$
  - **Entropy**: $-\sum_k p_k \log_2 p_k$

- **Regression trees** instead minimize **MSE** at each split






INATECH

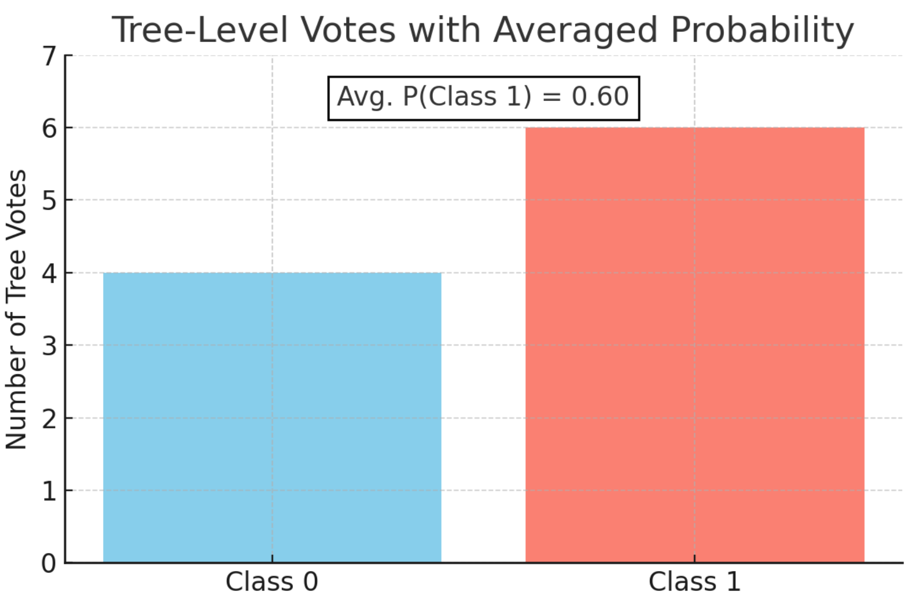
UNI FREIBURG

# Random Forest Classifier – Voting & Probabilities

- **Hard vote**: final class = majority of tree predictions
- **Soft vote**: class k probability

$$P(y = k \mid x) = \frac{1}{T}\sum_{t=1}^{T} 1\{h_t(x) = k\}$$

- Probabilities enable thresholding and confidence analysis
- **Regression trees** instead use **average** of the trees






INATECH

UNI FREIBURG

# LightGBM for Classification – What Changes?

- LightGBM = **gradient boosting** → models built sequentially, each new tree fits the **gradient of the loss**

- You've seen this for regression:
  - Loss: **MSE**
  - Gradients: residuals $y_i - \hat{y}_i$

- For **classification**:
  - Loss = **binary cross-entropy**

    $$\mathcal{L}(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$
  - Gradients = from this loss, based on predicted probabilities

- Output is a **logit score**, converted via sigmoid:

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

- Supports:
  - **Hard predictions** via thresholding (e.g., 0.5)
  - **Probabilistic outputs** for metric evaluation

# Performance Metrics

# Confusion Matrix Fundamentals

- **Definition of terms**
  - **True Positive (TP):** model predicts 1, actual = 1
  - **False Positive (FP):** model predicts 1, actual = 0
  - **False Negative (FN):** model predicts 0, actual = 1
  - **True Negative (TN):** model predicts 0, actual = 0

- **Purpose:** all classification metrics derive from these four counts

- **Cost implications:** different errors carry different real-world costs (e.g., missed fault vs. false alarm)

| | Predicted 1 | Predicted 0 |
|---|---|---|
| **Actual 1** | **TP** | FN |
| **Actual 0** | FP | **TN** |

# Precision & Recall Trade-off

- **Precision (P):** also known as positive predictive value

$$P = \frac{TP}{TP + FP}$$

  - ✓ "Of all predicted positives, how many were correct?"

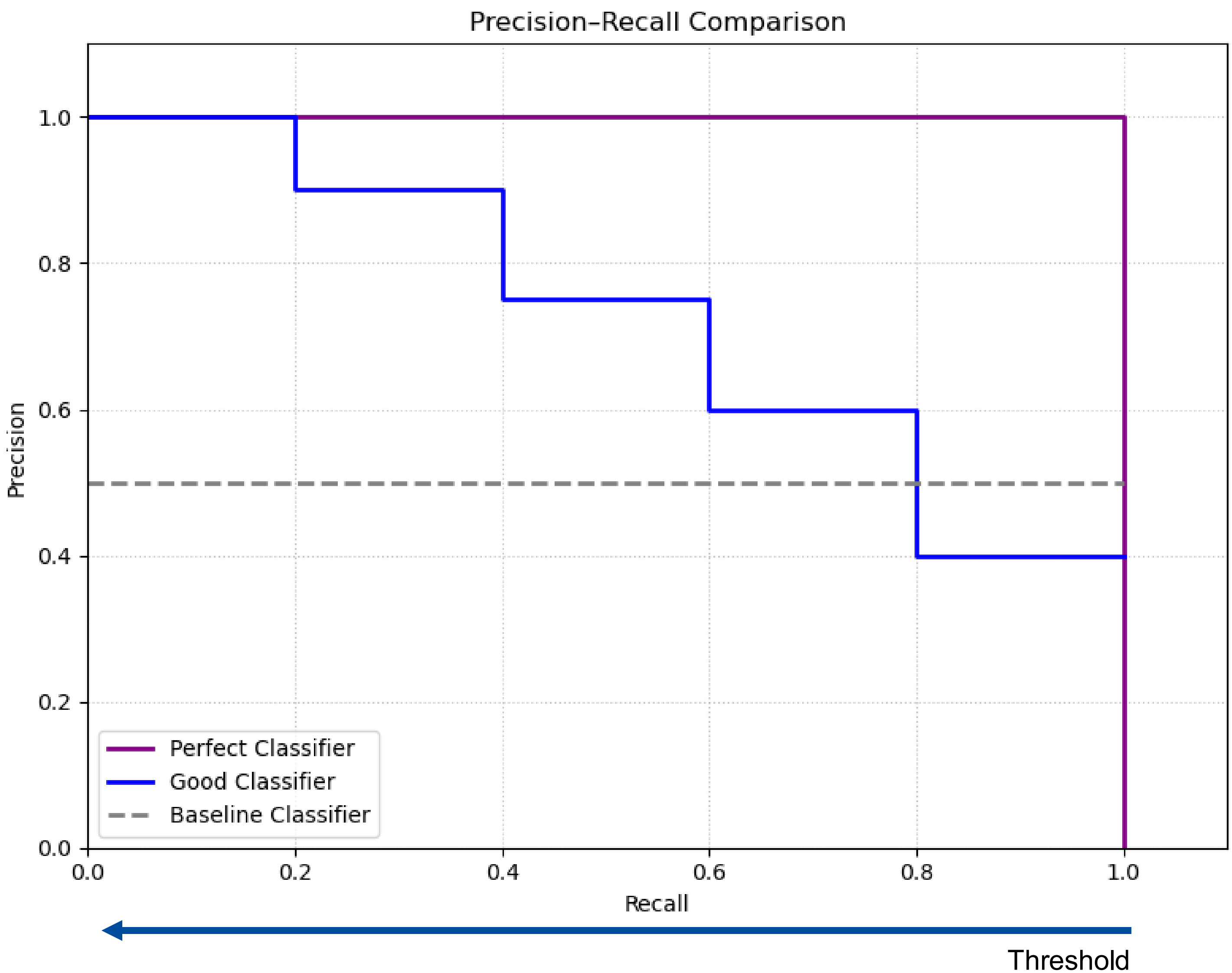
- **Recall (R):** also known as sensitivity

$$R = \frac{TP}{TP + FN}$$

  - ✓ "Of all actual positives, how many did we catch?"

- **Trade-offs:**
  - ✓ raising threshold → ↑precision, ↓recall;
  - ✓ lowering threshold → ↑recall, ↓precision
- **When to prioritize:**
  - ✓ High precision → costly false positives (e.g., dispatching crews)
  - ✓ High recall → costly misses (e.g., undetected faults)

# Precision & Recall Trade-off



# F1 Score & Averaging

- **F1 score** combines precision and recall into one metric:

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

- **Why F1?** Balances false positives and false negatives when both matter.

**Averaging for multi-class**:

- **Macro-F1**
  - Compute F1 for each class independently and take the **unweighted mean**:

$$\mathrm{Macro-F1} = \frac{1}{K}\sum_{k=1}^{K} F1_k$$

  - Treats all classes equally → highlights poor performance on rare classes
- **Weighted-F1**
  - Compute F1 for each class, then weight by support $n_k$:

$$\text{Weighted−F1} = \frac{1}{N}\sum_{k=1}^{K} n_k\, F1_k, \text{ where } N = \sum_k n_k$$

  - Reflects dataset composition → gives more influence to common classes

# F1 Score & Averaging

**Averaging for multi-class**:

- **Macro-F1**
  - Compute F1 for each class independently and take the **unweighted mean**:

$$\mathrm{Macro-F1} = \frac{1}{K}\sum_{k=1}^{K} F1_k$$

  - Treats all classes equally → highlights poor performance on rare classes
- **Weighted-F1**
  - Compute F1 for each class, then weight by support $n_k$:

$$\text{Weighted−F1} = \frac{1}{N}\sum_{k=1}^{K} n_k\, F1_k, \text{ where } N = \sum_k n_k$$

  - Reflects dataset composition → gives more influence to common classes

| Class | F1 | Support $n_k$ |
|---|---|---|
| **A** | 0.90 | 100 |
| **B** | 0.60 | 20 |
| **Macro-F1** | (0.90+0.60)/2 = 0.75 | — |
| **Weighted-F1** | (100·0.90+20·0.60)/120 = 0.85 | — |

# Example – Confusion Matrix & Classification Report

## Confusion Matrix

| Predicted ↓ \ True → | Baseline | EV |
|---|---|---|
| **Baseline** | 191 | 9 |
| **EV** | 9 | 41 |

## Classification Report

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Baseline | 0.95 | 0.95 | 0.95 | 200 |
| EV | 0.82 | 0.82 | 0.82 | 50 |
| **Accuracy** | | | **0.93** | 250 |
| **Macro avg** | 0.89 | 0.89 | 0.89 | 250 |
| **Weighted avg** | 0.93 | 0.93 | 0.93 | 250 |

**Key Takeaways from Classification Report**

- **High overall accuracy (0.93) hides class imbalance effects**
- **Baseline class (n=200):** precision & recall = 0.95 → very reliable
- **EV class (n=50):** precision & recall = 0.82 → notably weaker performance
- **Macro-F1 = 0.89** < overall accuracy → equal weighting reveals minority-class challenges
- **Weighted-F1 = 0.93** ≈ accuracy → majority class dominates the aggregate metric
- **Next steps:** focus on improving EV detection (e.g., resampling, class weights, tailored features)

# Unbalanced Datasets

# Imbalanced Datasets – Why It’s a Problem

### Confusion Matrix

| Predicted ↓ \ True → | Baseline | EV |
|---|---|---|
| **Baseline** | 90 | 10 |
| **EV** | 0 | 0 |

### Classification Report

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Baseline | 0.90 | 1.0 | 0.95 | 90 |
| EV | 0.00 | 0.00 | 0.00 | 10 |
| **Accuracy** | | | **0.90** | 100 |
| **Macro avg** | 0.45 | 0.50 | 0.47 | 100 |
| **Weighted avg** | 0.81 | 0.90 | 0.85 | 100 |

- **Definition:** one class (majority) far outnumbers the other(s) (minority)
- **Naïve accuracy trap:**
  - Suppose Baseline:EV = 90:10 ratio
  - Always predicting “Baseline” → 90% accuracy, but **zero** detection of EV
- **Real-world stakes:**
  - In **fault detection**, rare faults may be only 1–5% of data
  - Missing every fault → catastrophic consequences despite high accuracy
- **Key insight:** accuracy alone is **misleading** under imbalance

# Coffee Break

# Time to put everything into code

# What You'll Do in Code Today

## Goal:

Assign missing households their correct demand-response device type.

**Main Steps:**

1. **Load data** (household profiles + known device labels)
2. **Feature extraction** (reuse last exercise's time-series summary pipeline)
3. **Split** into training and test sets (stratified by device type)
4. **Train** three classifiers:
   a. **Logistic Regression**
   b. **Random Forest Classifier**
   c. **LightGBM Classifier**
5. **Evaluate** on test set:
   a. **Confusion matrix** for each model
   b. **Classification report** (precision, recall, F1, support)
   c. **Macro-F1** and **weighted-F1** comparisons
6. **Compare** model behavior and decision making

# Takeaways

- ✓ Classification tasks require **appropriate loss functions** (cross-entropy) and **evaluation metrics** beyond accuracy
- ✓ **Logistic Regression**, **Random Forest**, and **LightGBM** offer different trade-offs in interpretability, non-linearity, and handling of rare classes
- ✓ **Macro-F1** vs **Weighted-F1** reveal performance on minority classes
- ✓ **Unbalanced datasets** can often be an issue and require special treatment

**Further Questions to Think About**

- How might you **improve detection** of the rarest device class?
- When would you choose **one classifier** over another in production?
- How could **threshold tuning** or **class weighting** further boost performance?
- What additional **features** might capture device-specific behavior more effectively?