Supervised Learning I

Fundamentals of Machine Learning

Xufeng Zhang

Inria
xufeng.zhang@inria.fr

December 10, 2025

Outline

What is Machine Learning?

- Machine Learning (ML) is the study of algorithms that improve their performance at some task through experience.
- Data contains patterns and regularities. ML methods aim to automatically discover and exploit them.
- We typically assume:
    - We have data: observations, measurements, features.
    - We have a task: prediction, classification, ranking, etc.
    - We have a performance measure: error, accuracy, loss.
- ML is widely used in:
    - Computer vision, natural language processing, recommender systems.
    - Medicine, finance, robotics, and many other domains.

Main Types of Machine Learning

- **Supervised Learning**
    - Training data comes with input–output pairs.
    - Goal: learn mapping from inputs to outputs.
- **Unsupervised Learning**
    - Only inputs are given; no explicit labels.
    - Goal: find structure (clusters, latent factors, density).
- **Other paradigms**
    - Semi-supervised learning: few labels $+$ many unlabeled examples.
    - Reinforcement learning: learn from interaction with environment.

What is Supervised Learning?

- We observe a dataset of pairs:

$$\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N},$$

  where $x^{(i)}$ are inputs (features) and $y^{(i)}$ are outputs (labels or targets).
- Goal: learn a function f such that $f(x)$ predicts y well for new, unseen x.
- Two main problem types:
  - **Regression:** y is numeric (continuous).
  - **Classification:** y is categorical (finite set of labels).
- Typical learning strategy:
  - Choose a model family (linear, tree-based, etc.).
  - Define a loss function.
  - Minimize the loss on training data.

Regression vs. Classification

Regression

- Output variable is continuous.
- Examples:
  - Predicting house prices from size, location, age.
  - Forecasting temperature, sales volume, or stock returns.

Classification

- Output variable takes one of a finite set of classes.
- Examples:
  - Classifying an email as spam or not spam.
  - Diagnosing a tumor as benign vs malignant.

Supervised Learning Notation

- Input (feature) vector:
$$x = (x_1, x_2, \dots, x_d)^\top \in \mathbb{R}^d.$$
- Output:
  - Regression: $y \in \mathbb{R}$.
  - Binary classification: $y \in \{0, 1\}$ (or $\{-1, +1\}$).
  - Multiclass classification: $y \in \{1, 2, \dots, K\}$.
- Prediction function:
$$\hat{y} = f_\theta(x),$$
  where θ are model parameters (weights, splits, etc.).
- Learning: choose θ to minimize a loss function, e.g.,
$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell\big(y^{(i)}, f_\theta(x^{(i)})\big).$$

Example Applications of Regression

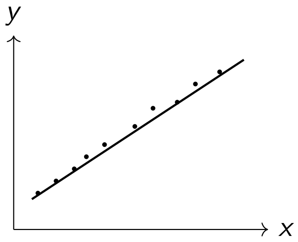
- Predicting:
  - House price based on size, number of rooms, and location.
  - Patient's blood glucose level from recent measurements and lifestyle variables.
  - Electricity demand as a function of temperature and time of day.
- Evaluation metrics:
  - Mean Absolute Error (MAE).
  - Mean Squared Error (MSE).
  - Root Mean Squared Error (RMSE).
- Regression models considered here:
  - Linear regression.
  - Decision trees for regression.
  - k Nearest Neighbours (kNN) regression.

Example Applications of Classification

- Email spam detection.
- Image recognition (cat vs dog, digit 0–9, etc.).
- Medical diagnosis (disease vs no disease).
- Credit scoring (good vs bad borrower).
- For classification we evaluate models using:
    - Accuracy.
    - Precision, Recall, F-score.
    - ROC curve and AUC.

Linear Regression: Intuition

- Linear regression models the relationship between a numeric target y and one or more features x_j.
- Assumes that y can be approximated as a linear combination of the features.
- In simple linear regression with one feature, we fit a straight line to data points.



Simple Linear Regression Model

- Model with a single explanatory variable x:

$$y = \beta_0 + \beta_1 x + \varepsilon,$$

  where
  - β_0 is the intercept (value at $x = 0$).
  - β_1 is the slope (change in y per unit change in x).
  - ε is a random noise term with mean zero.
- Given data $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, we estimate β_0, β_1.
- Prediction for new input $x^\star$:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x^\star.$$

Least Squares Estimation

- Ordinary Least Squares (OLS) chooses parameters that minimize the sum of squared residuals:

$$\min_{\beta_0,\beta_1} \sum_{i=1}^{N} \left(y^{(i)} - \beta_0 - \beta_1 x^{(i)}\right)^2.$$

- The residual for data point i is:

$$r^{(i)} = y^{(i)} - \hat{y}^{(i)}.$$

- OLS has a closed-form solution for linear regression.
- Intuition: we choose the line that lies "as close as possible" (in squared distance) to all training points.

Geometric View of OLS

- Let $X \in \mathbb{R}^{N\times(d+1)}$ be the design matrix (with a column of ones for the intercept).
- Let $y \in \mathbb{R}^N$ be the vector of targets.
- Linear model: $y \approx X\beta$.
- OLS solution:

$$\hat{\beta} = (X^\top X)^{-1}X^\top y,$$

  assuming $X^\top X$ is invertible.
- Interpretation:
  - $X\hat{\beta}$ is the orthogonal projection of y onto the column space of X.
  - Residual vector $r = y - X\hat{\beta}$ is orthogonal to all columns of X.

Multiple Linear Regression

- With multiple features $x_1, x_2, \ldots, x_d$:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_d x_d + \varepsilon.$$

- Vector form:

$$y = \beta_0 + \boldsymbol{\beta}^\top \mathbf{x} + \varepsilon,$$

  where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_d)^\top$.
- Interpretation of β_j:
  - Change in y caused by a one-unit increase in x_j while holding other features fixed (under the model assumptions).

Example: House Price Prediction

- Suppose we model house price y using features:
  - x_1: living area (m^2)
  - x_2: number of bedrooms
  - x_3: distance to city center (km)
  - x_4: age of building (years)
- A linear regression model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \varepsilon.$$

- After fitting the model, we can:
  - Interpret which features are most important.
  - Predict the price of a new house with given attributes.

Assumptions of Linear Regression (1)

Linear regression is based on several assumptions:

- **Linearity**
  - Relationship between each feature and the target is approximately linear.
  - Scatter plots of (x_j, y) should not show strong non-linear patterns.
- **Additivity**
  - Effects of different features add up.
  - Interaction terms (e.g., $x_i x_j$) may be needed if this is violated.

Assumptions of Linear Regression (2)

- **Independence of errors**
  - Residuals should be uncorrelated across observations.
  - Time series data often violate this assumption (autocorrelation).
- **Homoscedasticity**
  - Error variance should be constant across levels of the predictors.
  - Residual plots vs. fitted values can reveal heteroscedasticity.

Assumptions of Linear Regression (3)

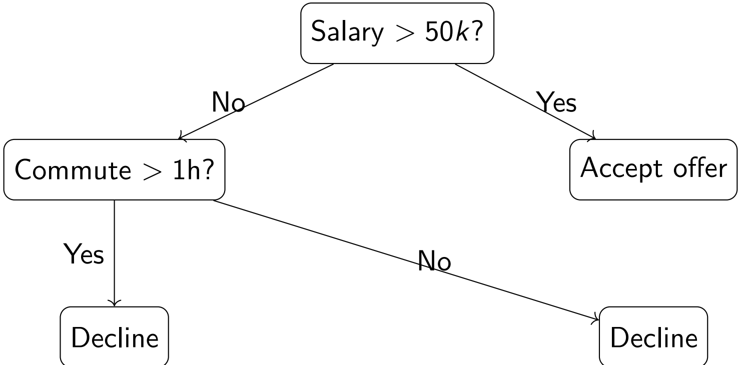
- **Normality of errors**
  - Residuals are assumed to be approximately normally distributed.
  - Mainly important for confidence intervals and hypothesis tests.
- **No multicollinearity**
  - Features should not be highly correlated with each other.
  - Severe multicollinearity makes individual coefficients unstable.
- When assumptions are violated, consider:
  - Transformations of features or target.
  - Regularization or other model families (trees, kernels, neural networks).

Limitations of Linear Regression

- Cannot capture complex non-linear patterns without feature engineering.
- Sensitive to outliers:
    - Large residuals have a strong influence on the fitted line.
- Performance deteriorates if:
    - Model assumptions are strongly violated.
    - Relevant variables are omitted or irrelevant variables are included.
- Nevertheless, linear regression is:
    - Simple, interpretable, and often a strong baseline.

Decision Trees: Intuition

- A decision tree is a flowchart-like structure:
  - Internal nodes: tests on feature values (e.g., $x_j \leq t$).
  - Branches: outcomes of tests.
  - Leaves: predicted outputs (class labels or numeric values).
- Prediction is obtained by following the path from root to leaf.
- Trees are easy to interpret and visualize.



Decision Tree Terminology

- **Root node**: top node, all training samples start here.
- **Internal node**: performs a test on an attribute.
- **Leaf node**: outputs prediction; no further splits.
- **Path**: sequence of tests from root to leaf.
- **Depth**: length of the longest path (number of splits).
- **Splitting criterion**: function measuring quality of a split (information gain, Gini impurity, variance reduction, etc.).

Entropy for Classification

- Let K be the number of classes.
- For a node containing N samples, let p_k be the fraction of samples of class k.
- **Shannon entropy** of that node is:

$$H = -\sum_{k=1}^{K} p_k \log_2 p_k .$$

- Properties:
  - $H = 0$ when all samples are from one class (pure node).
  - H is maximal when classes are equally likely.
- Goal: choose splits that reduce entropy (increase purity).

Information Gain

- Assume we split a node S into subsets $S_1, S_2, \ldots, S_q$ using a test on feature Q.
- Let $H(S)$ be the entropy before the split, and $H(S_i)$ entropy of subset S_i.
- Weighted average entropy after the split:

$$H_{\text{after}} = \sum_{i=1}^{q} \frac{|S_i|}{|S|} H(S_i).$$

- **Information Gain** (IG) is:

$$\text{IG}(Q) = H(S) - H_{\text{after}}.$$

- We choose the split (test Q) that maximizes information gain.

Toy Example: Colored Balls (Setup)

- We have 20 balls on a line: 9 blue and 11 yellow.
- Task: predict the color of a randomly chosen ball based on its position x.
- Before any split:
  - Probability of blue: $p_{\text{blue}} = 9/20$.
  - Probability of yellow: $p_{\text{yellow}} = 11/20$.
- Root entropy:

$$H_{\text{root}} = -\frac{9}{20}\log_2\frac{9}{20} - \frac{11}{20}\log_2\frac{11}{20} \approx 1.$$

Toy Example: First Split

- Consider split by $x \leq 12$ vs $x > 12$.
- Left subset ($x \leq 12$) contains 13 balls, 5 blue and 8 yellow:

$$H_1 = -\frac{5}{13}\log_2\frac{5}{13} - \frac{8}{13}\log_2\frac{8}{13} \approx 0.96.$$

- Right subset ($x > 12$) contains 7 balls, 1 blue and 6 yellow:

$$H_2 = -\frac{1}{7}\log_2\frac{1}{7} - \frac{6}{7}\log_2\frac{6}{7} \approx 0.60.$$

Toy Example: Information Gain

- Weighted entropy after split:

$$H_{\text{after}} = \frac{13}{20} H_1 + \frac{7}{20} H_2.$$

- Information gain:

$$\text{IG}(x \leq 12) = H_{\text{root}} - H_{\text{after}} \approx 0.16.$$

- We can compare this IG with IG from other candidate splits (e.g., $x \leq 8$, $x \leq 15$, etc.) and choose the best.
- Repeating this process recursively builds the whole tree.

Tree Construction Algorithm (High-Level)

Recursive Procedure

1. Start with all training samples at the root.
2. At each node:
    - If stopping criterion is met (pure node, max depth, too few samples), make a leaf with majority class (or mean value).
    - Otherwise, search over all candidate splits and choose the best one according to a criterion (information gain, Gini, variance reduction).
3. Split the data into child nodes and recurse on each child.

Example: Playing Tennis (Concept)

- Input features:
    - Outlook: Sunny, Overcast, Rain.
    - Humidity: High, Normal.
    - Wind: Weak, Strong.
- Target:
    - Play tennis? Yes or No.
- Decision tree might:
    - Use Outlook at the root.
    - For Sunny days, look at Humidity.
    - For Rainy days, look at Wind strength.
- This yields simple interpretable rules for deciding when to play.

Example: Playing Tennis (Rules)

- Example rules from the tree:
  - If Outlook is Overcast $\Rightarrow$ Play.
  - If Outlook is Sunny and Humidity is High $\Rightarrow$ Do not play.
  - If Outlook is Sunny and Humidity is Normal $\Rightarrow$ Play.
  - If Outlook is Rain and Wind is Strong $\Rightarrow$ Do not play.
  - If Outlook is Rain and Wind is Weak $\Rightarrow$ Play.
- Each path from root to leaf corresponds to one such rule.

Regression Trees

- For predicting numeric targets, we build **regression trees**.
- At each leaf, prediction is usually the mean target value of training points in that leaf.
- Instead of entropy or Gini, we use variance or mean squared error (MSE) as splitting criterion:
$$\mathrm{Var}(S) = \frac{1}{|S|} \sum_{i \in S} (y^{(i)} - \bar{y}_S)^2,$$
where $\bar{y}_S$ is the mean target value in node S.
- We choose splits that reduce variance:
$$\Delta\mathrm{Var} = \mathrm{Var}(S) - \sum_i \frac{|S_i|}{|S|} \mathrm{Var}(S_i).$$

MSE and Piecewise Constant Prediction

- Regression tree partitions feature space into rectangles (or more general regions).
- Within each region (leaf), the prediction is constant:

$$\hat{y}(x) = \bar{y}_{\text{leaf}(x)}.$$

- This yields a piecewise constant approximation to the regression function.
- Trees handle non-linear relationships and interactions between features naturally.

Overfitting in Decision Trees

- If we allow trees to grow very deep:
  - Each leaf may contain very few training examples.
  - Training error becomes extremely low (even zero).
  - Generalization error can be high (overfitting).
- Symptoms:
  - Complex tree structure with many branches.
  - Very different trees obtained by small changes in training data.

Deep vs. Shallow Trees

- **Deep tree**
  - Many levels, many leaves.
  - Very flexible, low bias, high variance.
  - High risk of overfitting.
- **Shallow tree**
  - Few levels, fewer leaves.
  - Less flexible, higher bias, lower variance.
  - May underfit complex data.
- Trade-off controlled by hyperparameters (max depth, min samples per leaf, etc.).

Pruning and Regularization for Trees

- **Pre-pruning** (early stopping):
  - Limit maximum depth of the tree.
  - Require a minimum number of samples to split a node.
  - Require a minimum improvement in splitting criterion.
- **Post-pruning**:
  - Grow a large tree.
  - Then iteratively remove subtrees that do not improve validation performance.
- Both strategies aim to improve generalization.

Ensemble Methods with Trees

- Single trees can be unstable and prone to overfitting.
- **Random Forests**:
  - Train many trees on bootstrap samples of the data.
  - Use random subsets of features at each split.
  - Aggregate predictions (majority vote or averaging).
- **Gradient Boosting**:
  - Build trees sequentially.
  - Each new tree focuses on correcting errors of previous ones.
- Ensembles often achieve much higher predictive accuracy than a single tree.

Decision Trees: Advantages

- High interpretability:
  - Rules can be expressed in human-readable form.
  - Easy to visualize and explain to non-technical stakeholders.
- Can handle both numerical and categorical features.
- Non-linear decision boundaries.
- Fast training and prediction for moderate tree sizes.
- Can handle multi-output problems (multiple targets).

Decision Trees: Drawbacks

- Unstable: small changes in the data can lead to very different trees.
- Tendency to overfit without pruning or ensemble methods.
- Decision boundaries are axis-aligned and piecewise constant:
    - May be suboptimal for some tasks.
- Handling missing values and rare categories can be tricky.
- Optimal decision tree search is computationally hard; algorithms use greedy heuristics.

k Nearest Neighbours: Intuition

- Under the **compactness** or **locality** hypothesis:

  "Nearby points in feature space tend to have similar labels."

- kNN predicts the label of a new point by looking at labels of its nearest neighbours in the training set.
- No explicit training phase: all computation is deferred to prediction time.

kNN Classification Algorithm

Given a test point $x^\star$:

1. Compute distance $d(x^\star, x^{(i)})$ to each training point $x^{(i)}$.
2. Select the k closest training points.
3. For classification:
    - Predict the majority class among these k neighbours.
    - Optionally weight neighbours by inverse distance.

- Typical choices: $k = 3, 5, 7, \ldots$.
- k is a hyperparameter chosen via validation.

kNN for Regression

- Same idea as classification, but we predict a numeric value.
- After selecting k neighbours:
    - Prediction is the average (or median) of their target values:

$$\hat{y}(x^\star) = \frac{1}{k} \sum_{j=1}^{k} y_{(j)}.$$

    - Weighted versions use weights w_j depending on distance:

$$\hat{y}(x^\star) = \frac{\sum_{j=1}^{k} w_j y_{(j)}}{\sum_{j=1}^{k} w_j}.$$

- Produces a locally smoothed approximation of the true regression function.

Choice of k

- **Small** k (e.g., $k = 1, 2$):
  - Very flexible, low bias.
  - High variance, sensitive to noise and outliers.
- **Large** k:
  - Smoother decision boundary (higher bias).
  - Lower variance, more robust to noise.
- We typically choose k by:
  - Evaluating performance on a validation set.
  - Or using cross-validation to pick k with best average performance.

Distance Metrics for *k*NN

- **Euclidean distance** (most common):

$$d(x, z) = \sqrt{\sum_{j=1}^{d} (x_j - z_j)^2}.$$

- **Manhattan distance**:

$$d(x, z) = \sum_{j=1}^{d} |x_j - z_j|.$$

- **Minkowski distance**:

$$d(x, z) = \left(\sum_{j=1}^{d} |x_j - z_j|^p \right)^{1/p}.$$

- For categorical variables, we can use:
  - Hamming distance or kernel functions.
  - One-hot encoding combined with Euclidean distance (with care).

Curse of Dimensionality

- In high dimensions, distances become less informative:
    - All points tend to be far from each other.
    - Nearest and farthest neighbours have similar distances.
- Consequences for kNN:
    - Local neighbourhoods may not be truly "local".
    - Performance can degrade significantly.
- Remedies:
    - Feature selection or dimensionality reduction (PCA, etc.).
    - Domain-specific distance metrics.

kNN: Advantages

- Simple and intuitive algorithm.
- No explicit training phase; model is fully determined by data.
- Flexible: can approximate complex decision boundaries with enough data.
- Naturally supports multi-class classification.
- Can be adapted to different problems using customized distance metrics or kernels.

kNN: Drawbacks

- Prediction can be slow for large datasets:
  - Need to compute distances to many training points.
- Memory intensive:
  - Must store all training data.
- No clear theory for choosing k and the distance metric.
- Sensitive to irrelevant features and feature scaling.
- Performance suffers in high-dimensional spaces (curse of dimensionality).

Why Evaluate Models Carefully?

- Our goal is not to fit training data perfectly, but to generalize to unseen data.
- We need quantitative metrics to compare models and select hyperparameters.
- For classification:
  - Accuracy, precision, recall, F-score, ROC and AUC.
- For regression:
  - MAE, MSE, RMSE, R^2, etc.
- Choice of metric should reflect the application and its costs (e.g., cost of false positives vs false negatives).

Confusion Matrix (Binary Classification)

	Predicted	
Actual	Positive	Negative
Positive	TP	FN
Negative	FP	TN

- TP: true positives (correct positive predictions).
- TN: true negatives (correct negative predictions).
- FP: false positives (type I error).
- FN: false negatives (type II error).

Accuracy

- Accuracy is the fraction of correctly classified samples:

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

- Easy to understand and widely used.
- However, it can be misleading in imbalanced datasets.
- Example:
    - Medical test where only 0.2% of patients have a tumor.
    - A trivial classifier that always predicts “no tumor” achieves 99.8% accuracy, but is useless.

Precision and Recall

- **Precision** (positive predictive value):

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

- **Recall** (sensitivity, true positive rate):

$$\text{Recall} = \frac{TP}{TP + FN}.$$

- Interpretation:
    - Precision: of all predicted positives, how many are truly positive?
    - Recall: of all true positives, how many did we correctly detect?
- There is usually a trade-off between precision and recall.

Medical Diagnosis Example

- Suppose we test 10 000 patients:
  - 20 truly have a tumor.
  - Our model predicts 20 patients as "tumor".
  - Among those, 10 truly have a tumor (TP = 10), 10 do not (FP = 10).
  - We also miss 10 cases (FN = 10) and correctly classify the rest as no tumor (TN = 9 970).
- Metrics:

$$\begin{aligned} \text{Accuracy} &= \frac{TP + TN}{N} = \frac{10 + 9\,970}{10\,000} = 99.8\%, \\ \text{Precision} &= \frac{10}{10 + 10} = 0.5, \\ \text{Recall} &= \frac{10}{10 + 10} = 0.5. \end{aligned}$$

- High accuracy, but only half of positive predictions are correct and half of tumors are missed.

F-score and F_β

- F-score combines precision and recall into a single number.
- General F_β score:
$$F_\beta = (1+\beta^2)\frac{\text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}},$$
where $\beta > 0$.
- β controls importance of recall vs precision:
    - $\beta > 1$: recall is more important.
    - $\beta < 1$: precision is more important.
- F_1 **score** $(\beta = 1)$ is the harmonic mean of precision and recall:
$$F_1 = 2\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Comparing Two Systems with F-score

- System A:
  - Precision $= 0.70$, Recall $= 0.60$.
- System B:
  - Precision $= 0.80$, Recall $= 0.50$.
- Which is better?
  - Compute F_1 for both systems.
  - When recall is more important (e.g., tumor detection), use F_β with $\beta > 1$ or with β tuned to the application.
- Conclusion depends on the relative cost of false negatives vs false positives.

ROC Curves

- Many classifiers output scores or probabilities.
- By varying the decision threshold, we trade off:
    - True Positive Rate (TPR) $=$ Recall.
    - False Positive Rate (FPR) $= \dfrac{FP}{FP + TN}$.
- **ROC curve** plots TPR vs FPR as the threshold varies.
- Useful for:
    - Comparing different models.
    - Selecting an operating point that balances TPR and FPR.

Area Under the ROC Curve (AUC)

- AUC is the area under the ROC curve:
    - Ranges from 0 to 1.
    - 0.5 corresponds to a random classifier.
    - 1 corresponds to a perfect classifier.
- Interpretation:
    - Probability that the classifier ranks a randomly chosen positive example higher than a randomly chosen negative example.
- AUC is insensitive to class imbalance and is widely used in many domains.

Why Do We Need Model Validation?

- Training error underestimates the true error on unseen data.
- We need an independent estimate of generalization performance.
- Strategy:
    - Split data into training and validation sets.
    - Use validation set to evaluate models and choose hyperparameters.
    - Optionally keep a separate test set for final evaluation.

Hold-out Validation

- Simple and popular approach:
    1. Randomly split the dataset into:
        - Training set (e.g., 60%–80% of data).
        - Validation set (e.g., 20%–40% of data).
    2. Train the model on the training set.
    3. Evaluate performance on the validation set.
- Pros:
    - Easy to implement and fast.
- Cons:
    - Performance estimate can be noisy, especially for small datasets.

K-fold Cross-Validation

- Procedure:
  1. Split data into K approximately equal folds.
  2. For each fold k:
     - Train model on $K-1$ folds.
     - Evaluate on the remaining fold k.
  3. Average the K performance scores.
- Advantages:
  - More stable and reliable estimate of performance.
  - Efficient use of all data points for both training and validation.

Using Cross-Validation for Model Selection

- We often have several candidate models:
    - Different algorithms (linear regression, trees, kNN).
    - Different hyperparameters (tree depth, k in kNN, etc.).
- For each candidate:
    - Perform K-fold cross-validation.
    - Compute mean validation score (and maybe standard deviation).
- Choose the model with the best cross-validation performance.
- Finally, retrain the chosen model on the full training data and evaluate it on a held-out test set.

Summary of Supervised Methods

- Supervised learning uses labeled data to learn input–output relationships.
- We studied three core families of models:
    - Linear regression (simple and multiple).
    - Decision trees (for classification and regression).
    - *k* Nearest Neighbours (instance-based learning).
- We discussed:
    - Model assumptions, strengths, and weaknesses.
    - Evaluation metrics: accuracy, precision, recall, F-scores, ROC/AUC.
    - Validation methods: hold-out sets and cross-validation.

Further Reading

- Introductory textbooks:
    - T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*.
    - C. Bishop, *Pattern Recognition and Machine Learning*.
- Online courses:
    - Open machine learning courses and MOOCs on supervised learning.
- Practice:
    - Implement linear regression, decision trees, and kNN on simple datasets.
    - Experiment with evaluation metrics and validation strategies.