- Maskinlæring H25

November 22, 2025

Contents

1 Supervised Learning: A First Approach **3**
- 1.1 From Traditional Programming to Machine Learning . . . 3
- 1.2 Learning from Data . . . 3
  - 1.2.1 Example: ECG Data . . . 3
  - 1.2.2 Defining Learning . . . 4
- 1.3 Supervised Datasets . . . 4
- 1.4 Types of Supervised Problems . . . 4
- 1.5 The Hypothesis Space . . . 4
- 1.6 Loss Function and Model Selection . . . 4
  - 1.6.1 Expected Loss . . . 5
  - 1.6.2 No Free Lunch Theorem . . . 5
- 1.7 Supervised Learning Workflow . . . 5
- 1.8 Underfitting and Overfitting . . . 5
- 1.9 k-Nearest Neighbors (k-NN) . . . 5
  - 1.9.1 Normalization . . . 6
- 1.10 Summary . . . 6

2 Generative Models **7**
- 2.1 Gaussian Mixture Models (GMMs) . . . 7
- 2.2 Supervised, Unsupervised, and Semi-supervised Learning . . . 8
- 2.3 Choosing the Number of Clusters (Elbow Method) . . . 8
- 2.4 Anomaly Detection with GMMs . . . 9
- 2.5 Synthetic Data Generation . . . 10
- 2.6 Summary and Exam Notes . . . 10

3 Classification **10**
- 3.1 Reinforcement Learning (RL) . . . 11
- 3.2 RL Interaction Loop . . . 11
- 3.3 Markov Decision Process (MDP) . . . 11
- 3.4 Value Functions and Q-Learning . . . 12
- 3.5 Example: Robot, Banana Peel, Apple . . . 12
- 3.6 Exploration vs Exploitation . . . 12
- 3.7 K-Armed Bandit Problem . . . 12

3.7.1 Upper Confidence Bound (UCB) 13
3.7.2 Gradient Bandits 13

1 Supervised Learning: A First Approach

1.1 From Traditional Programming to Machine Learning

In traditional computer science, a program is explicitly written to map inputs to outputs:

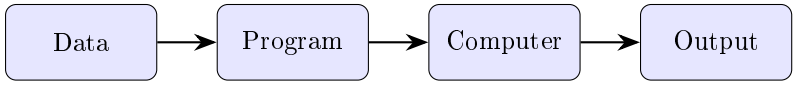
$$\text{Data} + \text{Program} \rightarrow \text{Computer} \rightarrow \text{Output}$$

Here, the program defines deterministic rules, such as `if-else` statements.

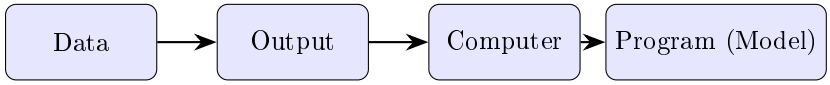
Machine learning inverts this paradigm:

$$\text{Data} + \text{Output} \rightarrow \text{Computer} \rightarrow \text{Program (Model)}$$

The computer now learns the mapping from examples rather than explicit programming. The more representative the data, the better the learned model.



Traditional Programming: Rules are explicitly coded.



Machine Learning: The computer learns the program from data.

1.2 Learning from Data

1.2.1 Example: ECG Data

Electrocardiogram (ECG) signals vary in time and across individuals. Each signal segment can be labeled as one of several heartbeat types:

$$y = \begin{cases} \text{NA} & \text{Normal Activity} \\ \text{AF} & \text{Atrial Fibrillation} \\ \text{RB} & \text{Resting Beat} \end{cases}$$

The task is to learn a model that maps input signals to the correct label.

1.2.2 Defining Learning

An **agent** learns from experience E with respect to a task T and a performance measure P if its performance on T, measured by P, improves with experience E.

1.3 Supervised Datasets

A supervised dataset consists of labeled examples:

$$T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

where x_i is a feature vector and y_i is the known output.

Examples:

- **Housing Prices:** y_i = price, x_i = location, size, year, etc.
- **Stock Prediction:** y_i = up/down, x_i = time-based features.

1.4 Types of Supervised Problems

- **Binary Classification:** $y \in \{0, 1\}$
- **Multiclass Classification:** $y \in \{1, 2, \ldots, K\}$
- **Regression:** $y \in \mathbb{R}$

The goal is to find a function $h(x)$ that approximates y:

$$h(x_i) \approx y_i, \quad \forall i = 1, \ldots, n$$

1.5 The Hypothesis Space

Let $\mathcal{H}$ denote the set of candidate functions (hypotheses). Each $h \in \mathcal{H}$ represents one possible model:

$$h(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

The objective is to select the hypothesis that best fits the data.

1.6 Loss Function and Model Selection

The **loss function** quantifies how well a hypothesis performs:

$$L_{\text{MSE}}(h, T) = \frac{1}{n} \sum_{i=1}^{n} (h(x_i) - y_i)^2$$

$$L_{\text{MAE}}(h, T) = \frac{1}{n} \sum_{i=1}^{n} |h(x_i) - y_i|$$

A perfect fit ($L = 0$) may indicate overfitting — the model memorizes the training data and fails to generalize.

1.6.1 Expected Loss

We aim to minimize the expected loss:

$$\mathbb{E}_{(x,y)\sim P}[L(h(x), y)]$$

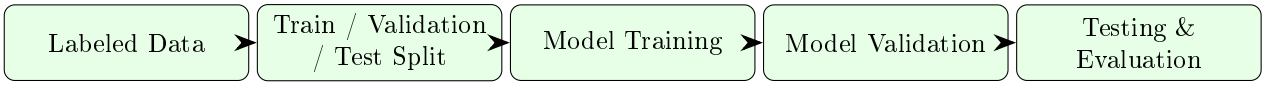
Since the population distribution P is unknown, we approximate it with train/validation/test splits:

$$80\% \text{ train},\ 10\% \text{ validation},\ 10\% \text{ test}.$$

1.6.2 No Free Lunch Theorem

No single model works for every problem — each dataset has its own structure.
Model choice must depend on the problem context.

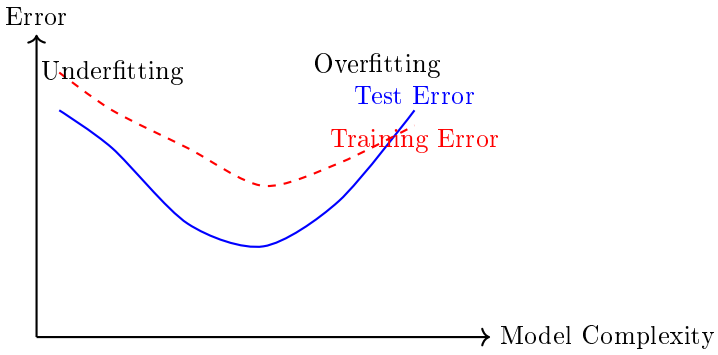
1.7 Supervised Learning Workflow



1.8 Underfitting and Overfitting

- **Underfitting:** Model too simple; fails to capture structure.
- **Overfitting:** Model too complex; captures noise.

Balancing bias and variance is crucial for generalization.



1.9 k-Nearest Neighbors (k-NN)

In k-NN, predictions are based on the labels of the k closest data points to a query x_*:

$$\|x_i - x_*\| = \sqrt{\sum_{j=1}^{p} (x_{ij} - x_{*j})^2}$$

Hyperparameter: k

- Low k: possible overfitting.
- High k: possible underfitting.

Input: Training set $T = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, query x_*, number of neighbors k
Output: Predicted label $\hat{y}_*$

foreach $(x_i, y_i) \in T$ **do**
| Compute distance $d_i = \|x_i - x_*\|$ (e.g., Euclidean)
end
Sort d_i in ascending order;
Select k nearest neighbors;
if *classification* **then**
| $\hat{y}_* \leftarrow$ majority label among k neighbors
else
| $\hat{y}_* \leftarrow \frac{1}{k} \sum_{i=1}^{k} y_i$ (mean for regression)
end

Algorithm 1: k-Nearest Neighbors Algorithm

1.9.1 Normalization

Since distances depend on scale, normalize each feature:

$$x'_j = \frac{x_j - \mu_j}{\sigma_j}$$

where μ_j and σ_j are the mean and standard deviation of feature j.

1.10 Summary

Supervised learning aims to infer a mapping from inputs to outputs based on labeled data. The process involves:

1. Defining the hypothesis space $\mathcal{H}$
2. Choosing a suitable loss function L
3. Optimizing $h \in \mathcal{H}$ to minimize L
4. Validating and testing to ensure generalization

A good model balances complexity and performance, avoiding both underfitting and overfitting.

2 Generative Models

Generative models aim to model the joint probability distribution $P(x, y)$ over inputs x and labels y. By Bayes' theorem:

$$P(x, y) = P(x \mid y)P(y)$$

They attempt to learn the underlying structure of the data — allowing inference of missing labels, anomaly detection, and generation of new (synthetic) samples.

Given training data:

$$T = \{(x_i, y_i)\}_{i=1}^{n}$$

we aim to generalize from the observed data to find a hypothesis $h(x) \in \mathcal{H}$ that minimizes a suitable loss function.

—

2.1 Gaussian Mixture Models (GMMs)

A **Gaussian Mixture Model (GMM)** assumes that the data are generated from a mixture of K Gaussian distributions:

$$P(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k)$$

where:

- π_k is the mixture coefficient ($\sum_k \pi_k = 1$),
- μ_k is the mean of cluster k,
- Σ_k is the covariance of cluster k,
- $\mathcal{N}(x \mid \mu_k, \Sigma_k)$ is the multivariate Gaussian PDF.

Each class or cluster is represented as a multivariate Gaussian distribution. The model can describe both separate and overlapping clusters, as shown below.

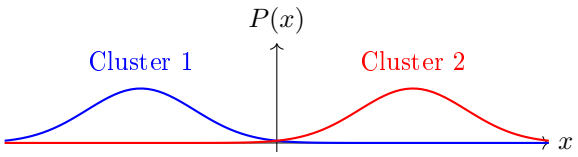


Figure 1: Two clusters represented by Gaussian components in a GMM. Overlap between clusters indicates probabilistic (soft) boundaries.

—

2.2 Supervised, Unsupervised, and Semi-supervised Learning

GMMs and K-Means can both be used for clustering:

- **K-Means:** hard assignments — each point belongs to one cluster.
- **GMM:** soft assignments — each point has a probability of belonging to each cluster.

GMMs can be used for:

- **Unsupervised learning:** to discover latent clusters in unlabeled data.
- **Supervised learning:** to estimate class-conditional distributions $P(x \mid y)$.
- **Semi-supervised learning:** to combine labeled and unlabeled data.

Semi-supervised GMM Algorithm:

1. Learn the initial GMM from available (labeled + unlabeled) data.
2. Predict missing labels using the current model (e.g., via QDA or posterior probability).
3. Update the GMM using all data (including predicted labels).
4. Repeat until convergence (the loss no longer decreases).

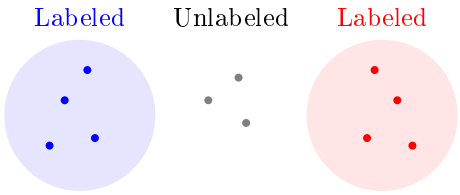


Figure 2: Semi-supervised data distribution: labeled and unlabeled points.

—

2.3 Choosing the Number of Clusters (Elbow Method)

The **elbow method** is used to select the optimal number of clusters K. It plots the within-cluster sum of squares (WCSS) versus K. The "elbow" point indicates where adding more clusters provides diminishing returns.

—

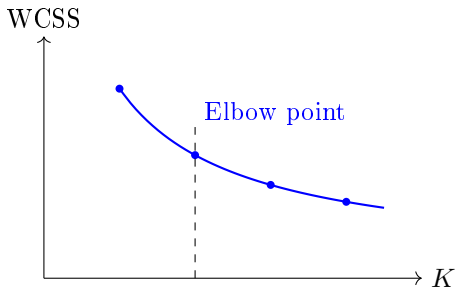


Figure 3: Elbow method: the optimal K occurs at the elbow point where reduction in error slows down.

2.4 Anomaly Detection with GMMs

GMMs can detect anomalies by identifying points with very low probability density.

Let $P(x_{\text{test}})$ be the probability density of a test observation. If

$$P(x_{\text{test}}) < \epsilon$$

for some threshold ϵ, the observation is classified as an **anomaly**.

Applications:

- Fraud detection (e.g., credit card transactions)
- Aircraft health monitoring (heat, vibration)
- Data center metrics:
  - x_1: memory usage
  - x_2: disk space
  - x_3: CPU load
  - x_4: network traffic

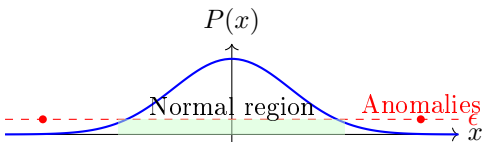


Figure 4: Anomaly detection using GMM: data points below the probability threshold ϵ are marked as anomalies.

—

2.5 Synthetic Data Generation

Generative models can also create new synthetic data points from the learned distribution $P(x)$. This can enhance model performance when labeled data are scarce, helping improve cluster separation and classification accuracy.

—

2.6 Summary and Exam Notes

- Generative models learn $P(x, y) = P(x \mid y)P(y)$.
- GMMs assume data are generated from a mixture of Gaussians.
- GMMs can be used for supervised, unsupervised, and semi-supervised learning.
- Use the elbow method to determine the number of clusters.
- Anomalies are detected when $P(x_{\text{test}}) < \epsilon$.
- Example question: *"What method can handle semi-supervised data?"* → **GMM (Gaussian Mixture Model)**.

3 Classification

Classification is a supervised learning task where the goal is to assign input x to discrete labels y. Examples include:

- Image classification
- Spam detection
- Medical diagnosis

Training involves:

1. Labeled dataset $\{(x_i, y_i)\}_{i=1}^{n}$
2. Model $f(x)$ predicting labels
3. Loss function (e.g., cross-entropy)
4. Optimization (e.g., gradient descent)

Classification assumes a known "correct action" for each input, which may not be feasible in complex real-world problems. This motivates **Reinforcement Learning (RL)**.

3.1 Reinforcement Learning (RL)

RL differs from supervised learning:

- No correct action labels; the agent learns from interaction
- Behavior shaped through rewards and punishments
- Active exploration required

Example: *"Good dog gets treats; bad dog does not."*
RL is useful for:

- Agents controlling robots, cars, helicopters
- Dynamic state systems (position, battery, time)
- Learning through trial and error

3.2 RL Interaction Loop

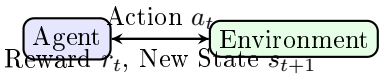


Figure 5: RL agent-environment interaction loop

3.3 Markov Decision Process (MDP)

An MDP is defined by:

- States S
- Actions A
- Transition probabilities $P(s'|s,a)$
- Reward function $R(s,a)$
- Discount factor $\gamma \in [0,1]$

The goal: maximize cumulative discounted reward

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

3.4 Value Functions and Q-Learning

State-value function:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t \mid s_t = s]$$

Action-value (Q-function):

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t \mid s_t = s, a_t = a]$$

Optimal policy:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

Q-learning update:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\big(r_t + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)\big)$$

3.5 Example: Robot, Banana Peel, Apple

- R: Robot, B: Banana peel, A: Apple
- Rewards: $R_A = 100$, $R_{\text{alive}} = 40$, $R_B = -100$
- Agent chooses actions (left/right) to maximize cumulative reward

3.6 Exploration vs Exploitation

- **Exploitation:** Choose best-known action
- **Exploration:** Try new actions to find better outcomes

ε-Greedy policy:

- With probability ε, select a random action
- With probability $1-\varepsilon$, select $a_t = \arg\max_a Q_t(a)$

Optimistic Initial Values (OIV): Start $Q_1(a)$ high to encourage exploration.

3.7 K-Armed Bandit Problem

- k slot machines with unknown reward distributions
- Goal: maximize total reward over time
- Update rule:

$$Q_{t+1}(a) = Q_t(a) + \frac{1}{N_t(a)}(R_t - Q_t(a))$$

- Balance exploration and exploitation

3.7.1 Upper Confidence Bound (UCB)

Action selection:

$$a_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

3.7.2 Gradient Bandits

Use preference $H_t(a)$ and softmax policy:

$$\pi_t(a) = \frac{e^{H_t(a)}}{\sum_b e^{H_t(b)}}$$

Update $H_t(a)$ in the direction of higher reward.